25th International Symposium on Temporal Representation and Reasoning

TIME 2018, October 15–17, 2018, Warsaw, Poland

Edited by
Natasha Alechina
Kjetil Nørvåg
Wojciech Penczek

Editors

Natasha Alechina
University of Nottingham
UK
natasha.alechina@nottingham.ac.uk

Kjetil Nørvåg
NTNU
Norway
noervaag@ntnu.no

Wojciech Penczek
ICS PAS and UPH
Poland
penczek@ipipan.waw.pl

ACM Classification 2012
Theory of computation → Logic, Information systems → Temporal data, Computing methodologies → Knowledge representation and reasoning

ISBN 978-3-95977-089-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-089-7.

Publication date
October, 2018

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 http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.TIME.2018.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The 25th International Symposium on Temporal Representation and Reasoning (TIME 2018), held 15–17 October 2018 in Warsaw, Poland, was hosted by the Institute of Computer Science of the Polish Academy of Sciences.

TIME is a well-established symposium series which brings together researchers interested in reasoning about temporal aspects of information in all areas of computer science. The symposium has a wide remit and is devoted to both theoretical aspects and well-founded applications. One of the key aspects of the symposium is its interdisciplinarity, with attendees from different areas such as artificial intelligence, database management, logic and verification, and beyond.

Following the call for papers of TIME 2018, a total of 27 full papers were submitted. Each submitted paper was reviewed by at least three, in exceptional circumstances two, members of the program committee, and the reviews were followed by an additional discussion to select among those papers. The members of the program committee and the external reviewers did an excellent job that enabled a high-quality selection process, and we thank them for their commitment and dedication. In the end, 20 papers were selected for publication in the proceedings and presentation at the symposium. In addition to the contributed talks, this year's program featured three invited speakers: Stéphane Demri (LSV, CNRS & ENS de Cachan, France), Johann Gamper (Free University of Bozen-Bolzano, Italy), and Wojciech Jamroga (Institute of Computer Science, PAS, Warsaw, Poland). We are delighted that they were able to accept our invitation, and grateful for their contribution.

These are the second TIME proceedings published in the Dagstuhl/LIPIcs series. We would like to thank dr. Wagner and the LIPIcs team for all the help and support. Finally, we would like to thank the following organizations for patronage and sponsoring the event: Institute of Computer Science PAS, Committee of Informatics of the Polish Academy of Sciences, and NaviParking.

Natasha Alechina
Kjetil Nørvåg
Wojciech Penczek

25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Program Committee Chairs

- Natasha Alechina (University of Nottingham, UK)
- Kjetil Nørvåg (Norwegian University of Science and Technology, Norway)
- Wojciech Penczek (ICS PAS, Poland)

Program Committee

- Patricia Bouyer (CNRS, Cachan, France)
- Christos Doulkeridis (University of Piraeus, Greece)
- Rajeev Gore (Australian National University, Australia)
- Fabio Grandi (University of Bologna, Italy)
- Keijo Heljanko (Aalto University, Finland)
- Adam Jatowt (Kyoto University, Japan)
- Marcin Jurdziński (University of Warwick, UK)
- Roman Kontchakov (Birkbeck, University of London, UK)
- Martin Lange (University of Kassel, Germany)
- Francois Laroussinie (LIAFA, Univ. Paris 7, CNRS, France)
- Aniello Murano (University of Napoli "Federico II", Italy)
- Doron Peled (Bar Ilan University, Israel)
- R. Ramanujam (Institute of Mathematical Sciences, Chennai, India)
- Peter Z. Revesz (University of Nebraska-Lincoln, USA)
- Mark Reynolds (Univeristy of Western Australia, Australia)
- Sven Schewe (University of Liverpool, UK)
- Renate A. Schmidt (University of Manchester, UK)
- Kostas Stefanidis (University of Tampere, Finland)
- Andrzej Szałas (Warsaw University, Poland, and University of Linköping, Sweden)
- Kristian Torp (Aalborg University, Denmark)
- Bożena Woźna-Szcześniak (Jan Długosz University, Częstochowa, Poland)
- Bin Yang (Aalborg University, Denmark)
- Neil Yorke-Smith (TU Delft, The Netherlands)

Local Organizers

- Wojciech Penczek (ICS PAS, Poland) - Chair
- Michał Ciesiołka (ICS PAS, Poland)
- Waldemar Słonina (ICS PAS, Poland)
- Teofil Sidoruk (ICS PAS, Poland)
- Jan Ziółkowski (ICS PAS, Poland)

25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Alexander Bolotov
University of Westminster
United Kingdom
A.Bolotov@wmin.ac.uk

Hamad Al Marri
University of Regina
Canada
almarrih@uregina.ca

Eisa Alanazi
Um Al Qura University
Saudi Arabia
eaanazi@uqu.edu.sa

Natasha Alechina
University of Nottingham
United Kingdom
natasha.alechina@nottingham.ac.uk

Alexander Artikis
University of Piraeus and NCSR
"Demokritos"
Greece
a.artikis@unipi.gr

George Athanasopoulos
University of Athens
Greece
gathanas@di.uoa.gr

Andreas Behrend
University of Bonn
Germany
behrend@cs.uni-bonn.de

Michael H. Böhlen
University of Zurich
Switzerland
boehlen@ifi.uzh.ch

Laura Bozzelli
University of Napoli "Federico II"
Italy
lr.bozzelli@gmail.com

Davide Bresolin
University of Padova
Italy
davide.bresolin@unipd.it

Massimo Cairo
University of Trento
Italy
massimo.cairo@unitn.it

Marta Cialdea Mayer
University of Roma Tre
Italy
cialdea@ing.uniroma3.it

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

Carlo Comin
University of Verona
Italy
carlo.comin.86@gmail.com

Enrico Cominato
University of Udine
Italy
enrico.cominato@gmail.com

Stéphane Demri
CNRS
France
demri@lsv.fr

Anton Dignös
Free University of Bozen-Bolzano
Italy
dignoes@inf.unibz.it

Andreas Dohr
University of Bonn
Germany
andreas.dohr@uni-bonn.de

Christos Doulkeridis
University of Piraeus
Greece
cdoulk@unipi.gr

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

25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Christiane Engels
University of Bonn
Germany
engelsc@cs.uni-bonn.de

Tim French
The University of Western Australia
Australia
tim.french@uwa.edu.au

Johann Gamper
Free University of Bozen-Bolzano
Italy
gamper@inf.unibz.it

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

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

Simone Gnani
University of Ferrara
Italy
simone.gnani@student.unife.it

Gunther Heidemann
University of Osnabrueck
Germany
gheidema@uni-osnabrueck.de

Montserrat Hermo
University of the Basque Country
Spain
montserrat.hermo@ehu.eus

Luke Hunsberger
Vassar College, Poughkeepsie
USA
hunsberger@vassar.edu

Wojciech Jamroga
Institute of Computer Science, Polish Academy of Sciences
Poland
w.jamroga@ipipan.waw.pl

Christian S. Jensen
Aalborg University
Denmark
csj@cs.aau.dk

Catholijn M. Jonker
Delft University of Technology
The Netherlands
c.m.jonker@tudelft.nl

Nikos Katzouris
NCSR “Demokritos”
Greece
nkatz@iit.demokritos.gr

Daniel Kernberger
University of Kassel
Germany
daniel.kernberger@uni-kassel.de

Malte S. Kließ
Delft University of Technology
The Netherlands
m.s.kliess@tudelft.nl

Julius Köpke
University of Klagenfurt
Austria
julius.koepke@aau.at

Ioannis Kontopoulos
NCSR “Demokritos”
Greece
ikon@iit.demokritos.gr

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

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

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

Malek Mouhoub
University of Regina
Canada
mouhoubm@uregina.ca

Emilio Muñoz-Velasco
University of Malaga
Spain
ejmunoz@uma.es

Aniello Murano
University of Napoli "Federico II"
Italy
murano@na.infn.it

Kjetil Nørvåg
Norwegian University of Science and Technology
Norway
noervaag@ntnu.no

Andrea Orlandini
National Research Council - ISTC
Italy
andrea.orlandini@istc.cnr.it

George Paliouras
NCSR "Demokritos"
Greece
paliourg@iit.demokritos.gr

Alessandro Passantino
University of Ferrara
Italy
alessandr.passantino@student.unife.it

Wojciech Penczek
Institute of Computer Science, Polish Academy of Sciences and Siedlce University
Poland
penczek@ipipan.waw.pl

Roberto Posenato
University of Verona
Italy
roberto.posenato@univr.it

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

M. Birna van Riemsdijk
Delft University of Technology
The Netherlands
m.b.vanriemsdijk@tudelft.nl

Romeo Rizzi
University of Verona
Italy
romeo.rizzi@univr.it

Georgios M. Santipantakis
University of Piraeus
Greece
gsant@unipi.gr

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

Loredana Sorrentino
University of Napoli "Federico II"
Italy
loredana.sorrentino@unina.it

Jianwen Su
University of California at Santa Barbara
USA
su@cs.ucsb.edu

Pattreeya Tanisaro
University of Osnabrueck
Germany
pattanisaro@uni-osnabrueck.de

Lewis Tolonen
The University of Western Australia
Australia

Grigorios Tzortzis
NCSR "Demokritos"
Greece
gtzortzi@iit.demokritos.gr

Akrivi Vlachou
University of Piraeus
Greece
avlachou@unipi.gr

Dimitrios Vogiatzis
NCSR "Demokritos"
Greece
dimitrv@iit.demokritos.gr

Przemysław A. Wałęga
University of Warsaw and University of Oxford
Poland, United Kingdom
p.a.walega@gmail.com

On Temporal and Separation Logics

Stéphane Demri
LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay, France
demri@lsv.fr
https://orcid.org/0000-0002-3493-2610

Abstract

There exist many success stories about the introduction of logics designed for the formal verification of computer systems. Obviously, the introduction of temporal logics to computer science has been a major step in the development of model-checking techniques. More recently, separation logics extend Hoare logic for reasoning about programs with dynamic data structures, leading to many contributions on theory, tools and applications. In this talk, we illustrate how several features of separation logics, for instance the key concept of separation, are related to similar notions in temporal logics. We provide formal correspondences (when possible) and present an overview of related works from the literature. This is also the opportunity to present bridges between well-known temporal logics and more recent separation logics.

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

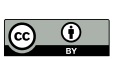
Keywords and phrases separation logics, temporal logics, expressive power

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.1

Category Invited Paper

1 Separation Logics

Separation logic has been introduced as an extension of Hoare logic [24] to verify programs with mutable data structures [28, 39, 41]. A major feature is to be able to reason locally in a modular way, which can be performed thanks to the separating conjunction $*$ that allows one to state properties in disjoint parts of the memory. The companion connective $-\!\!*$ corresponding to separating implication (a.k.a the magic wand) happens to be also helpful for program verification. So, the study of separation logics is currently very active, with works ranging from foundations to formal verification of programs. For instance, since the evidence that the method is scalable [3, 46], many tools supporting separation logic as an assertion language have been developped [3, 20, 46, 9, 10, 21]. Moreover, many variants of separation logics have been considered, leading to many interesting problems related to decidability/complexity of reasoning tasks, expressive power, relationships with other logical formalisms, proof systems, etc. It is not reasonable to enumerate herein all the existing variants and research directions. By way of example, decidability results about separation logic with general inductive predicates can be found in [27, 7]: notably in [7], the satisfiability problem for the symbolic heap fragment [2] with general inductively defined predicates is shown decidable. Furthermore, as already advocated in [8, 43, 42, 26, 37], dealing with the separating implication $-\!\!*$ is a desirable feature for program verification and several semi-automated or automated verification tools support it in some way, see e.g. [43, 42, 37], going beyond separation logics built over the symbolic heap fragment. Nevertheless, the combination of the magic wand $-\!\!*$ and the list segment predicate `ls` (a simple inductive



predicate) may lead to undecidability [19]. First-order separation logics have been also been considered in [2, 6, 16]. So, the first part of the talk is dedicated to basics on separation logics.

2 Relating Modal/Temporal Logics with Separation Logics

As the first versions of separation logic can be understood as a concretisation of the logic of bunched implication BI [38, 28, 40], it is not surprising that separation logics can be related to other logics, see also [15]. For instance, the concept of separation can be found in interval temporal logics (see e.g. [44, 45, 25, 35]), in graph logics (see e.g. [14, 1]), or in other formalisms [23, 22, 4]. Besides, as for temporal logics, the relationships between separation logic, and first-order or second-order logics have been the source of many characterizations and works. This is particularly true since the separating connectives are second-order in nature, see e.g. [32, 29, 11, 6, 17]. Moreover, separation logics can be shown to have close relationships with hybrid modal logics (see e.g. [8, 18]), with relevance logics (see e.g. [13, 12]) or with logics equipped with associative binary modalities (see e.g. [30, 4]).

In this talk, we illustrate how several features of separation logics are related to similar notions in temporal logics. We provide formal correspondences (when possible) and present an overview of related works from the literature. It is worth noting that temporal logics and separation logics can be related in many ways. At the semantical level, memory states from separation logics can be understand as tree-like models or as linear structures, see e.g. [16, 18] leading to explicit relationships with temporal logics on similar structures. Nevertheless, the correspondence is not always immediate. At the level of the operators, separation is a key concept that has been already introduced in interval temporal logic PITL [36]. Relationships between interval temporal logics and separation logics can be formally stated, see e.g. [16, 18, 34] and we shall show how complexity results about separation logics can be concluded. Typically, the TOWER-hardness of the satisfiability problem for first-order separation logics restricted to the separation conjunction and to two individual variables with one record field, can be established by reduction the satisfiability problem for PITL [16].

Similarly to the links between separation logics are (weak) second-order logics, ongoing investigations[1] relating separation logics with quantified temporal logics [31] shall be also evoked. So, apart from the analogies between temporal logics and separation logics and cross-fertilising results, we also motivate the introduction of formalisms that combine modal/temporal logics and separation logics, see e.g. [5, 33, 18], in order to reason about resources in a temporal framework.

So, the talk is the opportunity to present bridges between well-known temporal logics and more recent separation logics.

References

1 T. Antonopoulos and A. Dawar. Separating graph logic from MSO. In *FOSSACS'09*, volume 5504 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2009.

2 J. Berdine, C. Calcagno, and P. O'Hearn. A decidable fragment of separation logic. In *FST&TCS'04*, volume 3328 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2004.

[1] Bartosz Bednarczyk's internship at LSV (2018) is dedicated to related issues.

3 J. Berdine, C. Calcagno, and P. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO'05*, volume 4111 of *Lecture Notes in Computer Science*, pages 115–137. Springer, 2005.

4 J. Boudou. Decidable logics with associative binary modalities. In *CSL'17*, volume 82 of *LIPIcs*, pages 1–15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

5 R. Brochenin, S. Demri, and E. Lozes. Reasoning about sequences of memory states. *Annals of Pure and Applied Logic*, 161(3):305–323, 2009.

6 R. Brochenin, S. Demri, and E. Lozes. On the almighty wand. *Information and Computation*, 211:106–137, 2012.

7 J. Brotherston, C. Fuhs, N. Gorogiannis, and J. Navarro Perez. A decision procedure for satisfiability in separation logic with inductive predicates. In *CSL-LICS'14*, 2014.

8 J. Brotherston and J. Villard. Parametric completeness for separation theories. In *POPL'14*, pages 453–464. ACM, 2014.

9 C. Calcagno and D. Distefano. Infer: An automatic program verifier for memory safety of C programs. In *NASA Formal Methods*, volume 6617 of *Lecture Notes in Computer Science*, pages 459–465. Springer, 2011.

10 C. Calcagno, D. Distefano, P.W. O'Hearn, and H. Yang. Compositional shape analysis by means of bi-abduction. *Journal of the ACM*, 58(6):26:1–26:66, 2011.

11 C. Calcagno, Ph. Gardner, and M. Hague. From separation logic to first-order logic. In *FOSSACS'05*, volume 3441 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2005.

12 J. Spring D. Pym and P. O'Hearn. Why separation logic works. Manuscript, 2017.

13 M. Dams. *Relevance logic and concurrent composition*. PhD thesis, University of Edinburgh, 1989.

14 A. Dawar, Ph. Gardner, and G. Ghelli. Expressiveness and complexity of graph logic. *Information and Computation*, 205(3):263–310, 2007.

15 S. Demri and M. Deters. Separation logics and modalities: A survey. *Journal of Applied Non-Classical Logics*, 25(1):50–99, 2015.

16 S. Demri and M. Deters. Two-variable separation logic and its inner circle. *ACM Transactions on Computational Logics*, 2(16), 2015.

17 S. Demri and M. Deters. Expressive completeness of separation logic with two variables and no separating conjunction. *ACM Transactions on Computational Logics*, 17(2):12, 2016.

18 S. Demri and R. Fervari. On the complexity of modal separation logics. In *AiML'18*, 2018. to appear.

19 S. Demri, E. Lozes, and A. Mansutti. The effects of adding reachability predicates in propositional separation logic. In *FOSSACS'18*, volume 10803 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2018.

20 D. Distefano, P. O'Hearn, and H. Yang. A local shape analysis based on separation logic. In *TACAS'06*, volume 3920 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006.

21 C. Haase, S. Ishtiaq, J. Ouaknine, and M. Parkinson. SeLoger: A tool for graph-based reasoning in separation logic. In *CAV'13*, volume 8044 of *Lecture Notes in Computer Science*, pages 790–795. Springer, 2013.

22 L. Hella, K. Luosto, K. Sano, and J. Virtema. The expressive power of modal dependence logic. In *AIML'14*, pages 294–312. College Publications, 2014.

23 A. Herzig. A simple separation logic. In *WoLLIC'13*, volume 8071 of *Lecture Notes in Computer Science*, pages 168–178. Springer, 2013.

24 C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

25 I. Hodkinson, A. Montanari, and G. Sciavicco. Non-finite axiomatizability and undecidability of interval temporal logics with C, D, and T. In *CSL'08*, volume 5213 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 2008.

26 Z. Hou, R. Goré, and A. Tiu. Automated theorem proving for assertions in separation logic with all connectives. In *CADE'15*, volume 9195 of *Lecture Notes in Computer Science*, pages 501–516. Springer, 2015.

27 R. Iosif, A. Rogalewicz, and J. Simacek. The tree width of separation logic with recursive definitions. In *CADE'13*, volume 7898 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2013.

28 S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *POPL'01*, pages 14–26. ACM, 2001.

29 V. Kuncak and M. Rinard. On spatial conjunction as second-order logic. Technical Report MIT–CSAIL–TR–2004–067, MIT CSAIL, October 2004.

30 A. Kurucz, I. Németi, I. Sain, and A. Simon. Decidable and undecidable logics with a binary modality. *Journal of Logic, Language, and Information*, 4:191–206, 1995.

31 F. Laroussinie and N. Markey. Quantified CTL: Expressiveness and complexity. *Logical Methods in Computer Science*, 10(4:17), 2014.

32 E. Lozes. *Expressivité des Logiques Spatiales*. Phd thesis, ENS Lyon, 2004.

33 Xu Lu, Cong Tian, and Zhenhua Duan. Temporalising separation logic for planning with search control knowledge. In *IJCAI'17*, pages 1167–1173, 2017.

34 A. Mansutti. Extending propositional separation logic for robustness properties, July 2018. Manuscript.

35 D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. Interval temporal logics: a journey. *Bulletin of the EATCS*, 105:73–99, 2011.

36 B. Moszkowski. Reasoning about digital circuits. Technical Report STAN-CS-83-970, Dept. of Computer Science, Stanford University, Stanford, CA, 1983.

37 P. Müller, M. Schwerhoff, and A.J. Summers. Viper: A verification infrastructure for permission-based reasoning. In *VMCAI'16*, volume 9583 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2016.

38 P. O'Hearn and D. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

39 P.W. O'Hearn, J.C. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *CSL'01*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2001.

40 D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic*. Kluwer Academic Publishers, 2002.

41 J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*, pages 55–74. IEEE, 2002.

42 M. Schwerhoff and A. Summers. Lightweight support for magic wands in an automatic verifier. In *ECOOP'15*, pages 999–1023. Leibniz-Zentrum für Informatik, LIPICS, 2015.

43 A. Thakur, J. Breck, and T. Reps. Satisfiability modulo abstraction for separation logic with linked lists. In *SPIN'14*, pages 58–67. ACM, 2014.

44 Y. Venema. Expressiveness and completeness of an interval tense logic. *NDJFL*, 31(4):529–547, 1990.

45 Y. Venema. A modal logic for chopping intervals. *Journal of Logic and Computation*, 1(4):453–476, 1991.

46 H. Yang, O. Lee, J. Berdine, C. Calcagno, B. Cook, D. Distefano, and P. O'Hearn. Scalable shape analysis for systems code. In *CAV'08*, volume 5123 of *Lecture Notes in Computer Science*, pages 385–398. Springer, 2008.

Database Technology for Processing Temporal Data

Michael H. Böhlen
University of Zurich, Switzerland
boehlen@ifi.uzh.ch
https://orcid.org/0000-0003-3694-9026

Anton Dignös
Free University of Bozen-Bolzano, Italy
dignoes@inf.unibz.it
https://orcid.org/0000-0002-7621-967X

Johann Gamper
Free University of Bozen-Bolzano, Italy
gamper@inf.unibz.it
https://orcid.org/0000-0002-7128-507X

Christian S. Jensen
Aalborg University, Denmark
csj@cs.aau.dk
https://orcid.org/0000-0002-9697-7670

Abstract

Despite the ubiquity of temporal data and considerable research on processing such data, database systems largely remain designed for processing the current state of some modeled reality. More recently, we have seen an increasing interest in processing historical or temporal data. The SQL:2011 standard introduced some temporal features, and commercial database management systems have started to offer temporal functionalities in a step-by-step manner. There has also been a proposal for a more fundamental and comprehensive solution for sequenced temporal queries, which allows a tight integration into relational database systems, thereby taking advantage of existing query optimization and evaluation technologies. New challenges for processing temporal data arise with multiple dimensions of time and the increasing amounts of data, including time series data that represent a special kind of temporal data.

2012 ACM Subject Classification Information systems → Data management systems, Information systems → Temporal data

Keywords and phrases Temporal databases, temporal query processing, sequenced semantics, SQL

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.2

Category Invited Paper

1 Introduction and Background

The storage and querying of temporal data in database management systems (DBMSs) has been researched for decades, evolving the field and covering multiple aspects of time, the design of SQL-based query languages, the development of efficient storage and index structures and algorithms, as well as standardization efforts. The last few years have seen a renewed interest in studying temporal data management.

To facilitate the formulation of temporal queries, various temporal query languages have been proposed [5]. The earliest proposals extended *SQL with new data types* with associated predicates and functions, e.g., as consolidated in TSQL2 [30]. Though simple, this approach makes it difficult to be comprehensive and to avoid unintended interactions among different temporal features. To overcome such problems, more systematic approaches were proposed that, conceptually, adopt a point-based view of data in combination with timestamp normalization to transform between an interval-based representation and the point-based conceptual model. Representative examples include IXSQL [22] that uses *fold* and *unfold* functions, SQL/TP [31] that uses a *normalization* function, and an approach [1] that extends normalization to bitemporal relations by means of a *split* operator. For a systematic construction of temporal SQL queries from nontemporal SQL queries, so-called *statement modifiers* were proposed in ATSQL [6].

To make the processing of temporal queries efficient, various query processing algorithms have been studied, primarily for temporal aggregations and temporal joins over interval timestamped relations. Different ways of grouping data along the time dimension yield different forms of temporal aggregation: instant, moving-window, and span temporal aggregation. Unified frameworks to express these forms of temporal aggregation were proposed [21, 4]. Prominent examples of index structures and algorithms for temporal aggregation include the *aggregation tree* algorithm [19] and the *balanced tree* algorithm [24] for instant temporal aggregation, the *SB-tree* [32], a disk-based index structure for the incremental maintenance of instant and moving-window temporal aggregates, and its extension, the *MVSB-tree* [34], which additionally supports nontemporal range predicates.

Joins are the second class of operators for which efficient evaluation algorithms have been studied intensively. An overview and classification of temporal join algorithms is available in the literature [13]. Recent research results include the *timeline index* [17, 18], a main memory index, which was further developed in the context of the *lazy endpoint-based interval* join algorithm [27]. The *overlap interval partition* join [10] partitions the input relations such that the percentage of matching tuples in corresponding partitions is maximized. This yields a robust algorithm that is not affected by the temporal distribution of the data. Another partition-based approach, the *disjoint interval partitioning* join algorithm [8], ensures that all tuples in a partition are temporally disjoint to avoid expensive backtracking. The forward-scan based *plane sweep* algorithm [7] tries to minimize the number of comparisons and provides also a parallel evaluation strategy based on a temporal partitioning of the two input relations.

Recently, we have seen a renewed interest in providing support for temporal data in database management systems in both academia and industry. This has several reasons: abundant storage has made long term archival of historical data feasible, and it has been recognized in many application areas that temporal data holds the potential to reveal valuable insights that cannot be found by analyzing only a snapshot of the data. This has been witnessed by the SQL:2011 standard [33, 20], which for the first time provides support for storing temporal data, and by commercial DBMSs that have started to offer temporal functionalities [26]. Recently, a comprehensive solution for sequenced queries has been integrated into the kernel of PostgreSQL [11].

2 Temporal Support in SQL:2011 and Commercial DBMSs

SQL:2011 Standard. The SQL:2011 standard [33, 20] is arguably the first SQL standard to introduce explicit support for the storage and manipulation of temporal data. A core extension concerns the possibility to specify one or two time periods associated with tables,

representing *application time* and *system time*, which are commonly known as valid time and transaction time, respectively [15]. Valid time is the time when a tuple is true in the modeled reality, whereas transaction time is the time when the tuple was current in the database. While the valid time is specified by users, the transaction time is maintained by the DBMS when a tuple is created, updated, or deleted.

SQL:2011 adopts an interval-based data model with tuple timestamping. Time periods can be added as metadata to the table schema, specifying a start time attribute and an end time attribute. As start and end time attributes are often already present, a time period can be added without modifying the table schema. This approach achieves backward compatibility, keeping old schemas, queries, and tools running.

The behavior of temporal tables in the case of updates and deletions is different for valid time tables and transaction time tables. Conventional update and delete operations on valid time tables work in the same way as for nontemporal tables. Additionally, tuples can be modified over parts of the associated time period, which might cause tuples to be split or cut. For transaction time tables, the user can only modify nontemporal attributes of the current tuples. The timestamp attributes are maintained automatically by the system whenever nontemporal attributes of current tuples are modified.

The SQL:2011 standard also specifies primary and foreign keys. A primary key on a valid time table can be used to ensure that only one value at a time exists for the nontemporal key attributes [16]. Foreign keys enforce the existence of certain tuples in a referenced table. Similarly, primary and foreign key constraints can be specified for transaction time tables.

The support for querying temporal relations is limited to simple range restrictions and predicates. For valid time tables, the usual SQL syntax can be used to specify constraints on the start and end time points of the periods. For transaction time tables, three new SQL extensions are provided to retrieve tuples in a given time range. There is no explicit support for more advanced operations, such as various forms of temporal aggregations or temporal joins.

Commercial DBMSs. Following the SQL:2011 standard, major database vendors have started to offer temporal support in their database management systems [26].

IBM offers the temporal features from SQL:2011 in version 10 of their DB2 database system [29], supporting both valid time and transaction time tables. Transaction time tables are implemented by means of a current table and a history table; queries over transaction time tables are automatically rewritten into queries over one or both of the two tables. The Oracle DBMS supports temporal features from SQL:2011 as of version 12c. The temporal features are implemented using the Oracle flashback technology [25] and adopt a syntax that differs slightly from the SQL standard. PostgreSQL version 9.2 introduces a new range data type together with associated predicates and functions into the language to support the SQL:2011 standard [28]. For efficient query processing over range predicates, two index structures have been provided, the Generalized Search Tree (GiST) [14] and the space-partitioned Generalized Search Tree (SP-GiST) [12]. The Teradata DBMS supports temporal features from the SQL:2011 standard from version 13.10 onwards [2]. For querying temporal tables, so-called temporal statement modifiers [6] are used in combination with query rewriting where temporal queries are translated into equivalent standard SQL queries. In terms of querying, Teradata is the most advanced database management system, supporting sequenced aggregation and coalescing. Since 2016, Microsoft's SQL Server [23] offers limited support for transaction time tables. For more general temporal query support, user-defined functions have to be used.

3 Native Support for Sequenced Temporal Queries

While the SQL:2011 standard provides limited support for querying temporal data, the *temporal alignment* framework [11] is the first approach to achieve systematic and comprehensive support for so-called sequenced temporal queries in relational database engines without limiting the use of queries with so-called nonsequenced semantics. The approach allows a tight integration into the database kernel, thus making it possible to leverage existing query optimization and evaluation strategies for processing temporal queries.

The key idea of the temporal alignment approach is to reduce temporal queries to nontemporal queries in a two-step process: (1) Adjust the timestamps of the input tuples such that they are aligned. This yields an intermediate relation, where all tuples that together contribute to a result tuple have the same timestamp. This intermediate relation can conceptually be considered as a sequence of snapshots, each of which lasts for one or more time points. Two interval adjustment operators are needed: a *temporal normalizer* for the operators π, ϑ, $-$, $\cap$, and $\cup$, and a *temporal aligner* for the operators $\times$, $\bowtie$, $⟕$, $⟖$, $⟗$, and $\triangleright$. (2) The corresponding nontemporal operator is applied to the intermediate relations. By treating the adjusted timestamps as nontemporal, atomic values and adding an equality constraint over the adjusted timestamps (e.g., as a grouping attribute for aggregation or an equality predicate in joins), it is ensured that tuples that contribute to the same result tuple are processed together, yielding the correct result of the original temporal query. Conceptually, the nontemporal query is applied on each snapshot of the intermediate relations.

The temporal alignment framework features two optional steps. First, it allows the replication of the original timestamp attribute as a nontemporal attribute before the interval adjustment step. This is necessary if a subsequent operation needs information about the original timestamp, e.g., a query predicate over the original timestamp. Second, it allows attribute values of a tuple to be "scaled" in response to changes to the duration of the tuple's timestamp, which may occur during the interval adjustment step.

The temporal alignment approach is systematic and separates interval adjustment from the evaluation of the operators. This strategy renders it possible to fully leverage the query optimization and evaluation engine of a DBMS for sequenced temporal query processing. An implementation of the temporal alignment framework in the kernel of the PostgreSQL database system is available at `tpg.inf.unibz.it` [9, 11].

4 Conclusion and Outlook

The processing of temporal data is receiving renewed attention in the database community. In this work, we provide a brief overview of current results, covering both research results and commercial database management systems (a more detailed version is available [3]). Starting with SQL:2011 the SQL standard offers support for storing and updating temporal data; query support remains limited. These temporal features have been implemented in a step-by-step fashion in prominent database management systems. In the research field, a number of new index structures and query algorithms, mainly for aggregation and join, have been proposed. Further, the first comprehensive framework for sequenced temporal queries has been implemented in a relational database management system.

Future work in temporal databases points in various directions. While temporal alignment provides a framework for implementing temporal query support in relational database systems, open issues remain that require further investigation. First, in order to achieve scalability to very large datasets in the framework, some operators need substantial performance improvements. This can be achieved, for example, by providing additional and more targeted

temporal alignment primitives that produce smaller intermediate relations. Also, as current cost estimates are very conservative, it is of interest to study more accurate and optimistic cost estimates for the query optimizer. This might require the maintenance of statistics about the temporal distribution of data in the dictionary. Integration of specialized query algorithms and equivalence rules may also be pertinent.

Second, it is of interest to broaden the applicability of the framework. For instance, it is of interest to extend the temporal alignment framework to relations with temporal duplicates. This extension is relevant since duplicates occur in many practical applications, and they are also permitted in the SQL:2011 standard. Another extension concerns the support for two or more time dimensions, such as valid time and transaction time. Currently, only valid time is supported, while the SQL:2011 standard supports both valid time and transaction time. One problem there is that the adjustment of bitemporal timestamps becomes much more complex. In time series data, a special type of temporal data, each value is timestamped with a time point rather than a time period. It is of interest to study how existing technologies from temporal databases can be adopted for the processing of such data.

Finally, more research in SQL-based temporal query languages is needed to facilitate the formulation of complex temporal queries; this aspect is not covered in the SQL:2011 standard.

References

1 Mikkel Agesen, Michael H. Böhlen, Lasse Poulsen, and Kristian Torp. A split operator for now-relative bitemporal databases. In *Proceedings of the 17th International Conference on Data Engineering, ICDE 2001*, pages 41–50, 2001. doi:10.1109/ICDE.2001.914812.

2 Mohammed Al-Kateb, Ahmad Ghazal, Alain Crolotte, Ramesh Bhashyam, Jaiprakash Chimanchode, and Sai Pavan Pakala. Temporal query processing in teradata. In *Proceedings of the 16th International Conference on Extending Database Technology, EDBT 2013*, pages 573–578, 2013. doi:10.1145/2452376.2452443.

3 Michael H. Böhlen, Anton Dignös, Johann Gamper, and Christian S. Jensen. Temporal data management - an overview. In Esteban Zimányi, editor, *Business Intelligence and Big Data - 7th European Summer School, eBISS 2017, Bruxelles, Belgium, July 2-7, 2017, Tutorial Lectures*, volume 324 of *Lecture Notes in Business Information Processing*, pages 51–83. Springer, 2017. doi:10.1007/978-3-319-96655-7_3.

4 Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Multi-dimensional aggregation for temporal data. In *Proceedings of the 10th International Conference on Extending Database Technology, EDBT 2006*, volume 3896 of *Lecture Notes in Computer Science*, pages 257–275. Springer, 2006. doi:10.1007/11687238_18.

5 Michael H. Böhlen and Christian S. Jensen. Temporal data model and query language concepts. In *Encyclopedia of Information Systems*, pages 437–453. Elsevier, 2003. doi:10.1016/B0-12-227240-4/00184-2.

6 Michael H. Böhlen, Christian S. Jensen, and Richard T. Snodgrass. Temporal statement modifiers. *ACM Trans. Database Syst.*, 25(4):407–456, 2000. URL: http://portal.acm.org/citation.cfm?id=377674.377665.

7 Panagiotis Bouros and Nikos Mamoulis. A forward scan based plane sweep algorithm for parallel interval joins. *PVLDB*, 10(11):1346–1357, 2017. URL: http://www.vldb.org/pvldb/vol10/p1346-bouros.pdf, doi:10.14778/3137628.3137644.

8 Francesco Cafagna and Michael H. Böhlen. Disjoint interval partitioning. *The VLDB J.*, 26(3):447–466, 2017. doi:10.1007/s00778-017-0456-7.

9 Anton Dignös, Michael H. Böhlen, and Johann Gamper. Temporal alignment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012*, pages 433–444, 2012. doi:10.1145/2213836.2213886.

10 Anton Dignös, Michael H. Böhlen, and Johann Gamper. Overlap interval partition join. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2014*, pages 1459–1470, 2014. doi:10.1145/2588555.2612175.

11 Anton Dignös, Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Extending the kernel of a relational DBMS with comprehensive support for sequenced temporal queries. *ACM Trans. Database Syst.*, 41(4):26:1–26:46, 2016. doi:10.1145/2967608.

12 Mohamed Y. Eltabakh, Ramy Eltarras, and Walid G. Aref. Space-partitioning trees in postgresql: Realization and performance. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 100. IEEE Computer Society, 2006. doi:10.1109/ICDE.2006.146.

13 Dengfeng Gao, Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. Join operations in temporal databases. *The VLDB J.*, 14(1):2–29, 2005. doi:10.1007/s00778-003-0111-3.

14 Joseph M. Hellerstein. Generalized search tree. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 1222–1224. Springer US, 2009. doi:10.1007/978-0-387-39940-9_743.

15 Christian S. Jensen, Curtis E. Dyreson, Michael H. Böhlen, James Clifford, Ramez Elmasri, Shashi K. Gadia, Fabio Grandi, Patrick J. Hayes, Sushil Jajodia, Wolfgang Käfer, Nick Kline, Nikos A. Lorentzos, Yannis G. Mitsopoulos, Angelo Montanari, Daniel A. Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria Rita Scalas, Arie Segev, Richard T. Snodgrass, Michael D. Soo, Abdullah Uz Tansel, Paolo Tiberio, and Gio Wiederhold. The consensus glossary of temporal database concepts. In *Temporal Databases, Dagstuhl*, pages 367–405, 1997. doi:10.1007/BFb0053710.

16 Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. Extending existing dependency theory to temporal databases. *IEEE Trans. Knowl. Data Eng.*, 8(4):563–582, 1996. doi:10.1109/69.536250.

17 Martin Kaufmann, Amin Amiri Manjili, Panagiotis Vagenas, Peter M. 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 ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*, pages 1173–1184, 2013. doi:10.1145/2463676.2465293.

18 Martin Kaufmann, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, and Franz Färber. Comprehensive and interactive temporal query processing with SAP HANA. *PVLDB*, 6(12):1210–1213, 2013. URL: http://www.vldb.org/pvldb/vol6/p1210-kaufmann.pdf, doi:10.14778/2536274.2536278.

19 Nick Kline and Richard T. Snodgrass. Computing temporal aggregates. In *Proceedings of the 11th International Conference on Data Engineering, ICDE 1995*, pages 222–231, 1995. doi:10.1109/ICDE.1995.380389.

20 Krishna G. Kulkarni and Jan-Eike Michels. Temporal features in SQL: 2011. *SIGMOD Record*, 41(3):34–43, 2012. doi:10.1145/2380776.2380786.

21 Inés Fernando Vega López, Richard T. Snodgrass, and Bongki Moon. Spatiotemporal aggregate computation: a survey. *IEEE Trans. Knowl. Data Eng.*, 17(2):271–286, 2005. doi:10.1109/TKDE.2005.34.

22 Nikos A. Lorentzos and Yannis G. Mitsopoulos. SQL extension for interval data. *IEEE Trans. Knowl. Data Eng.*, 9(3):480–499, 1997. doi:10.1109/69.599935.

23 Microsoft. SQL Server 2016 - temporal tables. `https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables`, 2016.

24 Bongki Moon, Inés Fernando Vega López, and Vijaykumar Immanuel. Efficient algorithms for large-scale temporal aggregation. *IEEE Trans. Knowl. Data Eng.*, 15(3):744–759, 2003. `doi:10.1109/TKDE.2003.1198403`.

25 Oracle. Database development guide - temporal validity support. `https://docs.oracle.com/database/121/ADFNS/adfns_design.htm#ADFNS967`, 2016.

26 Dusan Petkovic. Temporal data in relational database systems: A comparison. In *New Advances in Information Systems and Technologies - Volume 1*, volume 444 of *Advances in Intelligent Systems and Computing*, pages 13–23. Springer, 2016. `doi:10.1007/978-3-319-31232-3_2`.

27 Danila Piatov, Sven Helmer, and Anton Dignös. An interval join optimized for modern hardware. In *Proceedings of the 32nd International Conference on Data Engineering, ICDE 2016*, pages 1098–1109, 2016. `doi:10.1109/ICDE.2016.7498316`.

28 PostgreSQL Global Development Group. Documentation manual PostgreSQL - range types. `http://www.postgresql.org/docs/9.2/static/rangetypes.html`, 2012.

29 Cynthia Saracco, Matthias Nicola, and Lenisha Gandhi. A matter of time: Temporal data management in DB2 10. `http://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/dm-1204db2temporaldata-pdf.pdf`, 2012.

30 Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

31 David Toman. Point vs. interval-based query languages for temporal databases. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1996*, pages 58–67, 1996. `doi:10.1145/237661.237676`.

32 Jun Yang and Jennifer Widom. Incremental computation and maintenance of temporal aggregates. *The VLDB J.*, 12(3):262–283, 2003. `doi:10.1007/s00778-003-0107-z`.

33 Fred Zemke. Whats new in SQL: 2011. *SIGMOD Record*, 41(1):67–73, 2012. `doi:10.1145/2206869.2206883`.

34 Donghui Zhang, Alexander Markowetz, Vassilis J. Tsotras, Dimitrios Gunopulos, and Bernhard Seeger. Efficient computation of temporal aggregates with range predicates. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2001*, 2001. `doi:10.1145/375551.375600`.

Model Checking Strategic Ability Why, What, and Especially: How?

Wojciech Jamroga[1]
Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
w.jamroga@ipipan.waw.pl

Abstract

Automated verification of discrete-state systems has been a hot topic in computer science for over 35 years. Model checking of temporal and strategic properties is one of the most prominent and most successful approaches here. In this talk, I present a brief introduction to the topic, and mention some relevant properties that one might like to verify this way. Then, I describe some recent results on approximate model checking and model reductions, which can be applied to facilitate verification of notoriously hard cases.

2012 ACM Subject Classification Computing methodologies → Multi-agent systems

Keywords and phrases model checking, strategic ability, alternating-time temporal logic, imperfect information games, approximate verification, model reductions

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.3

Category Invited Paper

Acknowledgements This extended abstract reports joint research with Francesco Belardinelli, Rodica Condurache, Piotr Dembiński, Cătălin Dima, Michał Knapik, Damian Kurpiewski, Antoni Mazurkiewicz, Łukasz Mikulski, and Wojciech Penczek. I gratefully acknowledge their contribution in what is presented below.

1 Multi-Agent Systems and Strategic Ability

More and more systems involve social as much as technological aspects, and even those that focus on technology are often based on distributed components exhibiting self-interested, goal-directed behavior. Moreover, the components act in environments characterized by incomplete information and uncertainty. The field of *multi-agent systems* studies the whole spectrum of phenomena ranging from agent architectures to communication and coordination in agent groups to agent-oriented software engineering. The theoretical foundations are mainly based on game theory and formal logic.

Many relevant properties of multi-agent systems refer to *strategic abilities* of agents and their groups. In particular, most functionality requirements can be specified as the ability of the authorized users to achieve their legitimate goals. At the same time, many security properties can be phrased in terms of the inability of unauthorized users to compromise the system. Properties of this kind can be conveniently specified in *modal logics of strategic ability*, of which alternating-time temporal logic (**ATL**) [2] is probably the most popular.

ATL modalities, possibly in combination with epistemic operators, allow e.g. to capture different flavors of coercion-resistance in voting systems [23, 17]. A simple hierarchy of

[1] The author acknowledges the support of the National Centre for Research and Development (NCBR), Poland, under the PolLux project VoteVerif (POLLUX-IV/1/2016).

$$
\begin{aligned}
CR_1 &\equiv \neg\langle\!\langle Coercer\rangle\!\rangle \,\mathrm{G}\,\big((\mathsf{closed} \wedge \bigwedge_{v\in A} \neg\mathsf{vote}_{v,1}) \rightarrow \mathsf{K}_{Coercer}(\bigvee_{v\in A} \neg\mathsf{vote}_{v,1})\big)\\
CR_2 &\equiv \neg\langle\!\langle Coercer\rangle\!\rangle \,\mathrm{G}\,\big((\mathsf{closed} \wedge \bigwedge_{v\in A} \neg\mathsf{vote}_{v,1} \rightarrow \bigvee_{v\in A} \mathsf{K}_{Coercer}(\neg\mathsf{vote}_{v,1})\big)\\
CR_3 &\equiv \neg\langle\!\langle Coercer\rangle\!\rangle \,\mathrm{G}\,\big((\mathsf{closed} \wedge \bigvee_{v\in A} \neg\mathsf{vote}_{v,1} \rightarrow \mathsf{K}_{Coercer}(\bigvee_{v\in A} \neg\mathsf{vote}_{v,1})\big)\\
CR_4 &\equiv \neg\langle\!\langle Coercer\rangle\!\rangle \,\mathrm{G}\,\big((\mathsf{closed} \wedge \bigvee_{v\in A} \neg\mathsf{vote}_{v,1} \rightarrow \bigvee_{v\in A} \mathsf{K}_{Coercer}(\neg\mathsf{vote}_{v,1})\big)
\end{aligned}
$$

Figure 1 Variants of coercion-resistance for coercion of multiple voters [17].

coercion-related specifications is presented in Figure 1. CR_1 expresses that the coercer cannot force all the agents in A to vote for candidate 1, or else he will know, at the closing of the election, that at least one of them disobeyed. CR_2 captures a stronger property: if someone in A disobeyed, the coercer will know the identity of at least one such agent. We leave the interpretation of CR_3 and CR_4 to the interested reader.

2 Practical Verification of Strategic Abilities

In the last 15 years, there has been a large number of works that study the syntactic and semantic variants of **ATL** for agents with imperfect information, cf. [5, 1] for an overview. The contributions are mainly theoretical, and concern the conceptual soundness of a given semantics, meta-logical properties, and the complexity of model checking and satisfiability problems. However, there is relatively little research on practical algorithms for verification.

This is somewhat easy to understand, since model checking of **ATL** variants with imperfect information is $\mathbf{\Delta_2^P}$- to **PSPACE**-complete for memoryless strategies [22, 4] and **EXPTIME**-complete to undecidable for agents with perfect recall [13, 14]. Moreover, the imperfect information semantics of **ATL** does not admit fixpoint characterizations [6, 11, 12], which makes incremental synthesis of strategies difficult to achieve. Experimental studies based on heuristic approaches [8, 15, 21, 9, 7] have also confirmed that the problem is hard, and dealing with it requires innovative techniques.

In this talk, I present several very recent attempts at overcoming the complexity barrier for model checking of $\mathbf{ATL}_{\mathrm{ir}}$, i.e., the **ATL** variant for memoryless strategies with imperfect information. The new techniques include approximate verification by fixpoint computation of upper- and lower bounds, model reduction based on locally defined model equivalence, and partial order reduction for simple strategic abilities in asynchronous systems. The main ideas are presented in the following sections, in a rather rudimentary form. For technical details, the reader is referred to the original papers [16, 18, 3, 19].

3 Approximate Model Checking

The proposal in [16, 18] is based on the idea that, instead of the exact model checking, it may suffice to provide a lower and an upper bound for the output. Given a formula φ, we construct two translations $LB(\varphi)$ and $UB(\varphi)$ such that $LB(\varphi) \Rightarrow \varphi \Rightarrow UB(\varphi)$. That is, if $LB(\varphi)$ is verified as true, then the original formula φ must also hold in the given model. Conversely, if $UB(\varphi)$ evaluates to false, then φ must also be false.

Table 1 Experimental results: solving endplay in bridge, formula $\langle\langle \mathbf{S} \rangle\rangle_{\text{ir}}$ Fwin [18].

#cards	#states	Approximate verification				Exact verification
		tgen	lower	upper	match	
4	11	0.0007	0.00007	0.00004	100%	0.12
8	346	0.011	0.0008	0.0003	100%	2.42 h*
12	12953	0.73	0.07	0.01	100%	timeout
16	617897	35.19	348.37	0.72	100%	timeout
20*	2443467	132.00	8815.73	4.216	100%	timeout

The construction of the upper bound is straightforward: instead of checking existence of an imperfect information strategy, we look for a perfect information strategy that obtains the same goal. If the latter is false, the former must be false too. The lower bound is defined by a fixpoint expression in alternating epistemic mu-calculus, with a nonstandard next-step strategic modality $\langle A \rangle^\bullet$ such that: (i) agents in A look for a short-term strategy that succeeds from the "common knowledge" neighborhood of the current state (rather than in the "everybody knows" neighborhood), and (ii) they are allowed to "steadfastly" pursue their goal in a variable number of steps within the indistinguishability class. Formally, the upper and the lower bounds are derived from φ through the following translations:

$$
\begin{array}{ll}
LB(p) = p, & UB(p) = p,\\
LB(\neg\phi) = \neg UB(\phi), & UB(\neg\phi) = \neg LB(\phi),\\
LB(\phi \wedge \psi) = LB(\phi) \wedge LB(\psi), & UB(\phi \wedge \psi) = UB(\phi) \wedge UB(\psi),\\
LB(\langle A \rangle \phi) = \langle A \rangle LB(\phi), & UB(\langle A \rangle \phi) = E_A \langle\langle A \rangle\rangle_{\text{Ir}} \mathrm{X}\, UB(\phi),\\
LB(\langle\langle A \rangle\rangle \mathrm{G}\phi) = \nu Z.(C_A LB(\phi) \wedge \langle A \rangle^\bullet Z), & UB(\langle\langle A \rangle\rangle \mathrm{G}\phi) = E_A \langle\langle A \rangle\rangle_{\text{Ir}} \mathrm{G}\, UB(\phi),\\
LB(\langle\langle A \rangle\rangle \psi\, \mathrm{U}\, \phi) = \mu Z.\big(E_A LB(\phi) \vee (C_A LB(\psi) \wedge \langle A \rangle^\bullet Z)\big) & UB(\langle\langle A \rangle\rangle \psi\, \mathrm{U}\, \phi) = E_A \langle\langle A \rangle\rangle_{\text{Ir}} UB(\psi)\, \mathrm{U}\, UB(\phi)
\end{array}
$$

▶ **Theorem 1** ([16]). *For every pointed model (M, q) and* $\mathbf{ATL}_{\text{ir}}$ *formula φ:*

$$M, q \models LB(\varphi) \implies M, q \models \varphi \implies M, q \models UB(\varphi).$$

The effectiveness of the approximations has been evaluated experimentally on a number of benchmarks. The results for a scalable scenario of *Bridge endplay* are presented in Table 1. The results in each row are averaged over 20 randomly generated instances, except for ($\star$) where only 1 hand-crafted instance was used. All the tests were conducted on a computer with an Intel Core i7-6700 CPU with dynamic clock speed of 2.60–3.50 GHz, 32 GB RAM, running 64bit Ubuntu 16.04 Linux. The running times are given in seconds. *Timeout* indicates that the process did not terminate in 48 hours. The computation of the lower and upper approximations was done with a straightforward implementation (in Python 3) of the fixpoint model checking algorithm. The exact $\mathbf{ATL}_{\text{ir}}$ model checking was done with MCMAS 1.3.0 [20].

Notice that the results in Table 1 have been obtained by a completely straigghtforward implementation of the lower/upper bound algorithms. After a simple optimization of data structures (based on the technique of *merge-find sets*) and operations on the data, the algorithms were able to generate and verify a model with over 70 million states in less than 75 minutes, cf. Table 2. Perhaps more importantly, the verification of the handpicked $(5, 5)^\star$ model (which marked the limit of our capability without the optimization) ran almost 3000 times (!) faster than with the straightforward implementation. This strongly suggests that the potential for further improvement is still large.

Table 2 Experimental results for the optimised approximate verification [18].

#cards	#states	Approximate verification				Exact verification
		tgen	lower	upper	match	
4	11	<0.0001	<0.0001	<0.0001	100%	0.12
8	346	<0.0001	<0.0001	<0.0001	100%	2.42 h*
12	12953	0.06	<0.0001	<0.0001	100%	timeout
16	617897	4.64	0.56	0.26	100%	timeout
20*	2443467	34.00	3.0	2.0	100%	timeout
20	1.5 e7	124.00	8.5	6.0	100%	timeout
24*	7 e7	3779.00	667.0	78.0	100%	timeout

4 Bisimulation-Based Model Reduction

The main source of hardness in model checking for strategic ability is the size of the model. First, the space of available strategies is at least exponential in the number of states and transitions. To make it even worse, there is (as of now) no method to factorize the model so that the algorithm would look for optimal substrategies independently, and then combine them into a winning strategy. Secondly, the explicit state model is typically exponential in size with respect to a higher level description, e.g., by means of local automata or concurrent programs. This means that realistic models are huge, and their strategy spaces are enormous. In consequence, an effective model reduction can offer invaluable help in turning a hopeless verification task into a feasible one.

One way to obtain such reductions is to identify a suitable notion of model equivalence that preserves the logic. Then, whenever the need for verification arises, we can look for a smaller model that is provably equivalent to the input model. Locally definable model equivalences for logics of strategies are usually called *alternating bisimulations*. The first variant of alternating bisimulation for **ATL** with imperfect information has been recently proposed in [3]. The main definitions and preservation theorems are summarized below.

Strategy simulators. Let M, M' be two models of $\mathbf{ATL}_{\mathrm{ir}}$. A *simulator of partial strategies for coalition A* from M into M' is a family $\mathbf{ST}$ of functions $\mathbf{ST}_{Q,Q'} : PStr_A(Q) \to PStr_A(Q')$ for some subsets of states $Q \subseteq St$ in model M and $Q' \subseteq St'$ in model M'. Intuitively, every $\mathbf{ST}_{Q,Q'}$ maps each partial strategy σ_A defined on set Q in M into a "corresponding" strategy σ'_A defined on Q' in M'. Typically, we will map strategies between the common knowledge neighborhoods of "bisimilar" states in M and M'. We formalize this idea as follows. Let $R \subseteq St \times St'$ be a relation between states in M and M'. A *simulator of partial strategies for coalition A with respect to relation R* is a family $\mathbf{ST}$ of functions $\mathbf{ST}_{C_A(q),C'_A(q')} : PStr_A(C_A(q)) \to PStr_A(C'_A(q'))$ such that qRq'.

Simulation and bisimulation. Let M, M' be two models defined on the same set $\mathbb{A}\mathrm{gt}$ of agents, and $A \subseteq \mathbb{A}\mathrm{gt}$ be a coalition. A relation $\Rrightarrow_A \subseteq St \times St'$ is a *simulation for A* iff

1. There exists a simulator $\mathbf{ST}$ of partial strategies for A w.r.t. $\Rrightarrow_A$, such that $q \Rrightarrow_A q'$ implies that:
   - **a.** $\pi(q) = \pi'(q')$;
   - **b.** for every $i \in A$, $r' \in St'$, if $q' \sim'_i r'$ then for some $r \in St$, $q \sim_i r$ and $r \Rrightarrow_A r'$.
   - **c.** For every states $r \in C_A(q)$, $r' \in C'_A(q')$ such that $r \Rrightarrow_A r'$, for every partial strategy $\sigma_A \in PStr_A(C_A(q))$, and every state $s' \in succ(r', \mathbf{ST}(\sigma_A))$, there exists a state $s \in succ(r, \sigma_A)$ such that $s \Rrightarrow_A s'$.

2. If $q_1 \Rrightarrow_A q'$ and $q_2 \Rrightarrow_A q'$, then $C_A(q_1) = C_A(q_2)$.

That is, state q' can only simulate q if (1a) q and q' agree on the interpretation of atoms; (1b) q simulates the epistemic transitions from q'; and (1c) for every partial strategy σ_A, defined on the common knowledge neighborhood $C_A(q)$, we are able to find some partial strategy $\mathbf{ST}(\sigma_A)$ (the same for all states in $C_A(q)$) such that the transition relations $\xrightarrow{\mathbf{ST}(\sigma_A)}$ and $\xrightarrow{\sigma_A}$ commute with the simulation relation $\Rrightarrow_A$. Moreover, (2) multiple states simulated by the same q' must be in the same common knowledge neighborhood.

Relation $\Leftrightarrows_A$ is a *bisimulation* iff both $\Leftrightarrows_A$ and its converse $\Leftrightarrows_A^{-1} = \{(q', q) \mid q \Leftrightarrows_A q'\}$ are simulations.

▶ **Theorem 2** (Preservation Theorem for A-$\mathbf{ATL}_{\mathrm{ir}}$ [3]). *If $\Leftrightarrows_A$ is a bisimulation for A and $q \Leftrightarrows_A q'$, then for every formula φ that contains only agents from A inside strategic modalities, we have:*

$$M, q \models \varphi \quad \textit{if and only if} \quad M', q' \models \varphi.$$

▶ **Theorem 3** (Preservation Theorem for $\mathbf{ATL}_{\mathrm{ir}}$ [3]). *If $\Leftrightarrows$ is a bisimulation for every $A \subseteq \mathbb{A}\mathrm{gt}$, and $q \Leftrightarrows q'$, then for every formula φ we have that:*

$$M, q \models \varphi \quad \textit{if and only if} \quad M', q' \models \varphi.$$

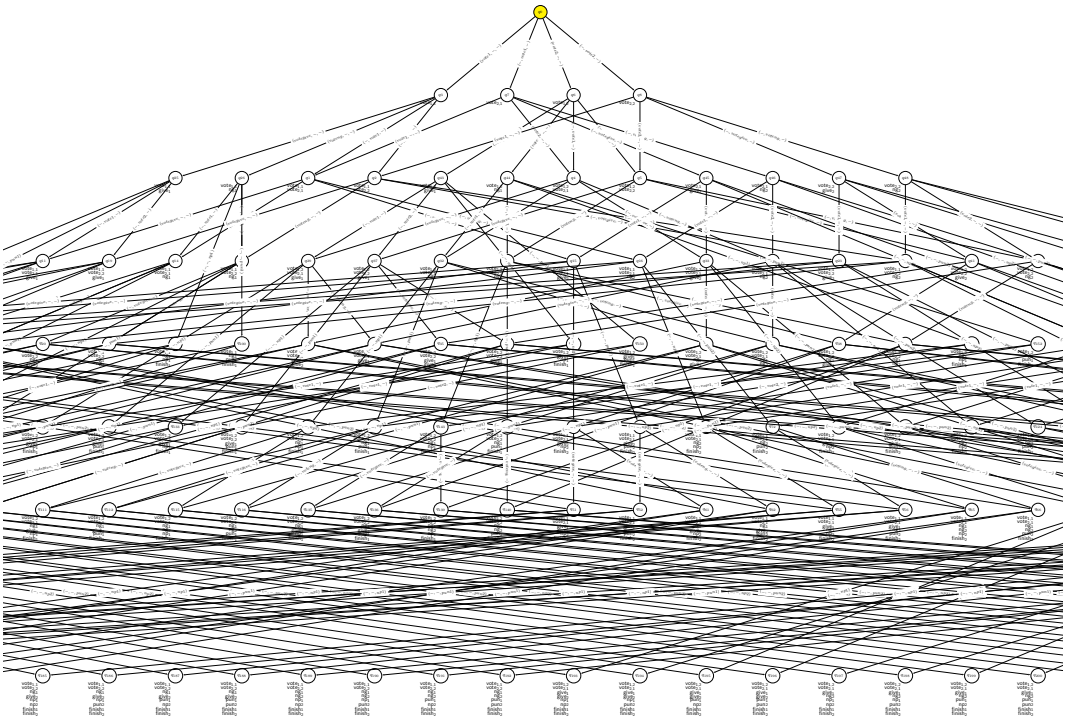
The bisimulation provides a strong notion of model equivalence, since it preserves the truth values of all $\mathbf{ATL}_{\mathrm{ir}}$ formulae. Moreover, it can lead to *very significant model reductions*. As an example, Figure 2a presents a fragment of the simple model of voting and coercion [18] for 1 coercer, 2 voters, and 2 candidates. A bisimilar, much sparser model is presented in Figure 2b.

5 Partial-Order Reduction

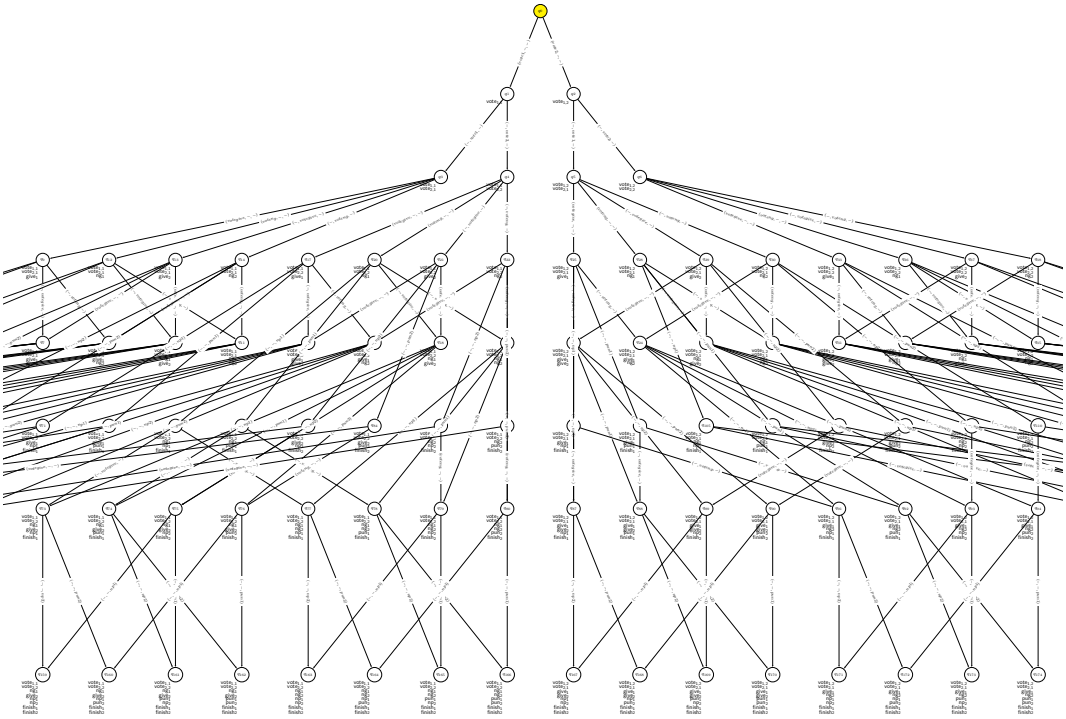
The bisimulation-based reduction can result in a significant trimming of states and transitions. However, the conditions of the bisimulation are quite strong, which means that the method has somewhat limited applicability. Moreover, the reduced model and the equivalence must be crafted and proved by hand. An automated method for reduction of models *that arise due to interleaving of asynchronously executed actions* has been recently proposed in [19].

Many real-life systems are inherently asynchronous, and do not operate on a global clock that perfectly synchronizes the atomic steps of all the components. As an example, consider robots interacting in an environment with faulty communication or non-negligible delays in execution of actions. No less importantly, many systems that are synchronous at the implementation level (say, the level of the virtual machine) can be more conveniently modeled as asynchronous on a more abstract level, because when we abstract away the implementation details it is not clear anymore how transitions initiated by different agents are exactly arranged in a particular temporal order. For instance, the actual implementation of a soccer match in the simulated RoboCup competition can be executed on a single computer with a global clock ticking every 0.3 ns, but the corresponding synchronous model would be huge and in consequence useless for analysis. Instead, one can remove a lot of unnecessary details by assuming that the players execute their actions asynchronously – without clear temporal relationship between their execution times – and synchronize only when a particular event has to be executed *jointly*.

In many scenarios, both aspects combine. For example, when modeling an election, one must take into account both the truly asynchronous nature of events happening at different



(a)



(b)

Figure 2 Bisimulation-based reduction of a simple voting model: before and after the reduction.

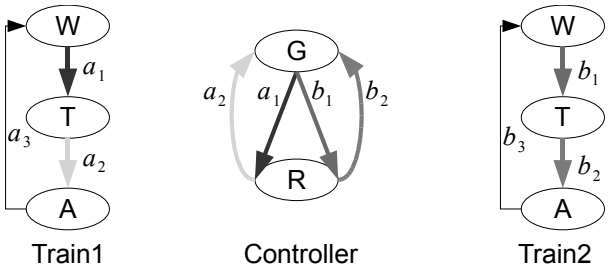


Figure 3 Asynchronous multi-agent system for the Trains, Gate and Controller benchmark.

polling stations, and the best level of granularity for modeling the events happening within a single polling station.

In [19], we have made the first step towards strategic analysis of such systems. Most importantly, we adapted *partial order reduction (POR)* to model checking of strategic abilities for agents with imperfect information. In fact, we showed that the most efficient variant of POR, defined for linear time logic **LTL**, can be applied almost directly. Interestingly, the scheme does *not* work for verification of agents with perfect information.

Algorithm for partial-order reduction. Given a collection of local automata $\mathcal{S}$ with its underlying model M, the reduction proceeds by iterative unfolding of $\mathcal{S}$ into its reduced model M'. The unfolding is done by means of the following Depth-First Search (DFS) algorithm:

1. Identify the set $en(q_n) \subseteq Act$ of enabled actions.
2. Heuristically select a subset $E(q_n) \subseteq en(q_n)$ of possible actions (see below).
3. For any action $a \in E(q_n)$, compute the successor state q' such that $q_n \xrightarrow{a} q'$, and add q' to the stack, thus generating the path $\pi' = q_0 a_0 q_1 a_1 \cdots q_n a q'$. Recursively proceed to explore the submodel originating at q'.
4. Remove q_n from the stack.

The algorithm begins with the stack consisting solely of the initial state of M, and terminates when the stack is empty.

Heuristics. Let $A \subseteq \mathbb{A}\mathrm{gt}$. The conditions **C1** $-$ **C3** below define a heuristics for the selection of $E(q) \subseteq en(q)$ in the DFS algorithm.

C1 Along each path π in M that starts at q, each action that is dependent on an action in $E(q)$ cannot be executed in π without an action in $E(q)$ is executed first in π. Formally, $\forall \pi \in \Pi_M(q)$ such that $\pi = q_0 a_0 q_1 a_1 \ldots$ with $q_0 = q$, and $\forall b \in Act$ such that $(b, c) \notin I_A$ for some $c \in E(q)$, if $a_i = b$ for some $i \geq 0$, then $a_j \in E(q)$ for some $j < i$.

C2 If $E(q) \neq en(q)$, then $E(q) \subseteq Invis_A$.

C3 For every cycle in M' there is at least one node q in the cycle for which $E(q) = en(q)$, i.e., for which all the successors of q are expanded.

Preservation theorems. Let $\mathcal{S}$ be a collection of local automata representing an asynchronous multi-agent system. $I_\emptyset$ is the standard independence of actions, used in partial-order reduction for **LTL** [10]. I_A is a slightly modified variant of $I_\emptyset$, that treats all the actions of agents in A as visible. The following theorems show that the **LTL** partial-order reduction can be directly (or almost directly) applied to $\mathbf{sATL}^*_{\mathrm{ir}}$, i.e., the fragment of $\mathbf{ATL}^*_{\mathrm{ir}}$ without nested strategic modalities and the temporal operator X.

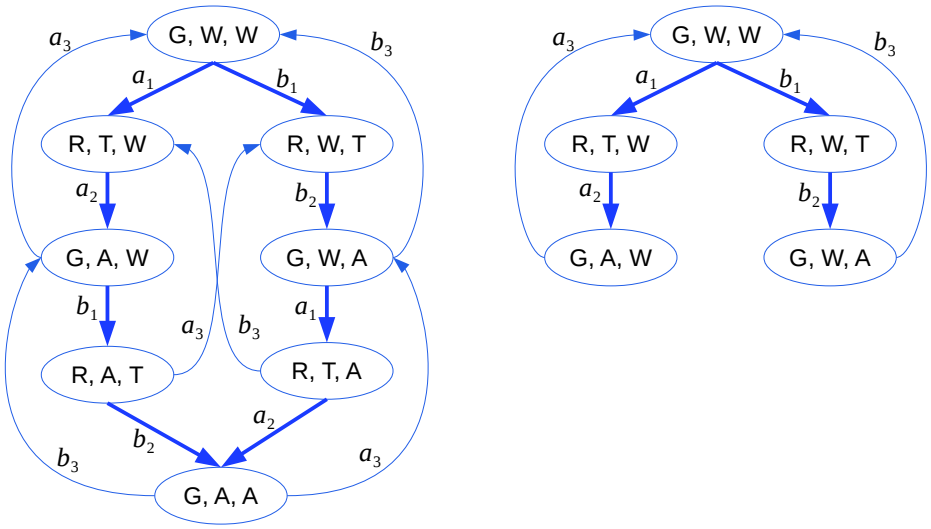


Figure 4 Interleaved interpreted systems for TGC: (a) full model, (b) reduced model. Visible transitions are depicted by bold arrows.

▶ **Theorem 4** ([19]). *Let M be the model of $\mathcal{S}$, and M' be the reduced model generated by DFS with the choice of $E(q')$ for $q' \in St'$ given by conditions* **C1, C3** *and the independence relation $I_\emptyset$. Then, M' satisfies exactly the same formulae of* $\mathbf{sATL}^*_{\mathrm{ir}}$ *as M under the concurrency-fairness assumption.*

▶ **Theorem 5** ([19]). *Let $A \subseteq \mathbb{A}\mathrm{gt}$, M be the model of $\mathcal{S}$, and M' the reduced model generated by DFS with the choice of $E(q')$ given by conditions* **C1, C2, C3** *and the independence relation I_A. Then, M' satisfies exactly the same formulae of A-*$\mathbf{sATL}^*_{\mathrm{ir}}$ *as M without the concurrency-fairness assumption.*[2]

How big is the gain? As an example, consider the well known TGC benchmark (Trains, Gate, and Controller). The local automata representing the system for $k = 2$ trains are shown in Figure 3, and the full logical model in Figure 4a. The reduced model obtained by POR is depicted in Figure 4b (same as for the **LTL** partial-order reduction). It is easy to see that the reduced state space is exponentially smaller than the size of the full model.

Of course, such optimistic outcomes are not guaranteed for all asynchronous agent systems. Still, it is important to note that $\mathbf{ATL}_{\mathrm{ir}}$ model checking is **NP**-hard *in the size of the model* (and not the size of the representation), and all the attempts at actual algorithms so far run in exponential time. So, even a linear reduction of the state space is likely to produce an exponential improvement of the performance.

Perfect information. The reduction scheme does not work for the standard variant of alternating-time logic, based on perfect information strategies:

▶ **Theorem 6** ([19]). *The analogues of Theorems 4 and 5 do not hold for* $\mathbf{sATL}_{\mathrm{Ir}}$.

This negative result is especially interesting because, until now, virtually all the results have been in favor of solving games with perfect information.

[2] where A-$\mathbf{sATL}^*_{\mathrm{ir}}$ is the fragment of $\mathbf{sATL}^*_{\mathrm{ir}}$ containing only agents from A in strategic modalities.

6 Conclusions

Verification by model checking is one of the top success stories in computer science and AI. Many important properties of multi-agent systems are underpinned by the ability of some agents (or groups) to achieve a given goal. However, model checking of strategic ability in realistic systems is a notoriously hard problem. In this short paper, I have tried to summarize some of the very recent developments that can contribute to overcoming the complexity barrier, and extending the scope of formal verification.

References

1 T. Ågotnes, V. Goranko, W. Jamroga, and M. Wooldridge. Knowledge and ability. In H.P. van Ditmarsch, J.Y. Halpern, W. van der Hoek, and B.P. Kooi, editors, *Handbook of Epistemic Logic*, pages 543–589. College Publications, 2015.

2 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002. doi:10.1145/585265.585270.

3 F. Belardinelli, R. Condurache, C. Dima, W. Jamroga, and A.V. Jones. Bisimulations for verification of strategic abilities with application to ThreeBallot voting protocol. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2017*, pages 1286–1295. IFAAMAS, 2017.

4 N. Bulling, J. Dix, and W. Jamroga. Model checking logics of strategic ability: Complexity. In M. Dastani, K. Hindriks, and J.-J. Meyer, editors, *Specification and Verification of Multi-Agent Systems*, pages 125–159. Springer, 2010.

5 N. Bulling, V. Goranko, and W. Jamroga. Logics for reasoning about strategic abilities in multi-player games. In J. van Benthem, S. Ghosh, and R. Verbrugge, editors, *Models of Strategic Reasoning. Logics, Games, and Communities*, volume 8972 of *Lecture Notes in Computer Science*, pages 93–136. Springer, 2015. doi:10.1007/978-3-662-48540-8.

6 N. Bulling and W. Jamroga. Alternating epistemic mu-calculus. In *Proceedings of IJCAI-11*, pages 109–114, 2011.

7 S. Busard. *Symbolic Model Checking of Multi-Modal Logics: Uniform Strategies and Rich Explanations*. PhD thesis, Universite Catholique de Louvain, 2017.

8 S. Busard, C. Pecheur, H. Qu, and F. Raimondi. Improving the model checking of strategies under partial observability and fairness constraints. In *Formal Methods and Software Engineering*, volume 8829 of *Lecture Notes in Computer Science*, pages 27–42. Springer, 2014. doi:10.1007/978-3-319-11737-9_3.

9 S. Busard, C. Pecheur, H. Qu, and F. Raimondi. Reasoning about memoryless strategies under partial observability and unconditional fairness constraints. *Information and Computation*, 242:128–156, 2015. doi:10.1016/j.ic.2015.03.014.

10 E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

11 C. Dima, B. Maubert, and S. Pinchinat. The expressive power of epistemic μ-calculus. *CoRR*, abs/1407.5166, 2014.

12 C. Dima, B. Maubert, and S. Pinchinat. Relating paths in transition systems: The fall of the modal mu-calculus. In *Proceedings of Mathematical Foundations of Computer Science (MFCS)*, volume 9234 of *Lecture Notes in Computer Science*, pages 179–191. Springer, 2015. doi:10.1007/978-3-662-48057-1_14.

13 C. Dima and F.L. Tiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225, 2011.

14 D.P. Guelev, C. Dima, and C. Enea. An alternating-time temporal logic with knowledge, perfect recall and past: axiomatisation and model-checking. *Journal of Applied Non-Classical Logics*, 21(1):93–131, 2011.

15 X. Huang and R. van der Meyden. Symbolic model checking epistemic strategy logic. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 1426–1432, 2014.

16 W. Jamroga, M. Knapik, and D. Kurpiewski. Fixpoint approximation of strategic abilities under imperfect information. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1241–1249. IFAAMAS, 2017.

17 W. Jamroga, M. Knapik, and D. Kurpiewski. Model checking the selene e-voting protocol in multi-agent logics. In *Proceedings of the 3rd International Joint Conference on Electronic Voting (E-VOTE-ID)*, Lecture Notes in Computer Science. Springer, 2018. To appear.

18 W. Jamroga, M. Knapik, D. Kurpiewski, and Łukasz Mikulski. Approximate verification of strategic abilities under imperfect information. *Artificial Intelligence*, 2018. To appear.

19 W. Jamroga, W. Penczek, P. Dembiński, and A. Mazurkiewicz. Towards partial order reductions for strategic ability. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 156–165. IFAAMAS, 2018.

20 A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: An open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, 2015. Availabe online. doi:10.1007/s10009-015-0378-x.

21 J. Pilecki, M.A. Bednarczyk, and W. Jamroga. Synthesis and verification of uniform strategies for multi-agent systems. In *Proceedings of CLIMA XV*, volume 8624 of *Lecture Notes in Computer Science*, pages 166–182. Springer, 2014.

22 P. Y. Schobbens. Alternating-time logic with imperfect recall. *Electronic Notes in Theoretical Computer Science*, 85(2):82–93, 2004.

23 M. Tabatabaei, W. Jamroga, and Peter Y. A. Ryan. Expressing receipt-freeness and coercion-resistance in logics of strategic ability: Preliminary attempt. In *Proceedings of the 1st International Workshop on AI for Privacy and Security, PrAISe@ECAI 2016*, pages 1:1–1:8. ACM, 2016. doi:10.1145/2970030.2970039.

Predicting the Evolution of Communities with Online Inductive Logic Programming

George Athanasopoulos

Department of Informatics and Telecommunications,
National and Kapodistrian University of Athens, Athens, Greece
geotha1995@hotmail.com

George Paliouras

Institute of Informatics and Telecommunications, NCSR "Demokritos", Athens, Greece
paliourg@iit.demokritos.gr

Dimitrios Vogiatzis

The American College of Greece, Deree, Athens, Greece; and
Institute of Informatics and Telecommunications, NCSR "Demokritos", Athens, Greece
dimitrv@iit.demokritos.gr

Grigorios Tzortzis

Institute of Informatics and Telecommunications, NCSR "Demokritos", Athens, Greece
gtzortzi@iit.demokritos.gr

Nikos Katzouris

Institute of Informatics and Telecommunications, NCSR "Demokritos", Athens, Greece
nkatz@iit.demokritos.gr

Abstract

In the recent years research on dynamic social network has increased, which is also due to the availability of data sets from streaming media. Modeling a network's dynamic behaviour can be performed at the level of communities, which represent their mesoscale structure. Communities arise as a result of user to user interaction. In the current work we aim to predict the evolution of communities, i.e. to predict their future form. While this problem has been studied in the past as a supervised learning problem with a variety of classifiers, the problem is that the "knowledge" of a classifier is opaque and consequently incomprehensible to a human. Thus we have employed first order logic, and in particular the event calculus to represent the communities and their evolution. We addressed the problem of predicting the evolution as an online Inductive Logic Programming problem (ILP), where the issue is to learn first order logical clauses that associate evolutionary events, and particular Growth, Shrinkage, Continuation and Dissolution to lower level events. The lower level events are features that represent the structural and temporal characteristics of communities. Experiments have been performed on a real life data set form the Mathematics StackExchange forum, with the OLED framework for ILP. In doing so we have produced clauses that model both short term and long term correlations.

2012 ACM Subject Classification Human-centered computing → Social network analysis, Computing methodologies → Machine learning, Computing methodologies → Online learning settings, Computing methodologies → Inductive logic learning

Keywords and phrases Social Network Analysis, Community Evolution Prediction, Machine Learning, Inductive Logic Programming, Event Calculus, Online Learning

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.4

25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek; Article No. 4; pp. 4:1–4:20

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

1 Introduction

A social network is a structure which contains individuals, who are linked to other individuals. The link among them states an interaction which has one or more types of interdependency such as friendship, kinship, common interest, financial exchange. Social networks are often represented as graphs, with nodes representing users and edge representing interactions. Usually a social network changes over time because new individuals join the network, new interactions are developed, or some individuals cease to be active for a short or a long period. This is actually the predominant behaviour especially in streaming social media, such as forums.

The social networks are often studied at the level of communities, which represent their meso-scale structure. A group of nodes forms a community if it is densely connected, and sparsely connected to other communities. The said communities are not explicitly formed but rather implicitly as a result of the actions of individual users, that are not random but tend to follow a certain pattern that is related to their similarity to other users. There are many algorithms that have been developed for the detection of communities in networks that are static [6].

In dynamic networks, the communities are influenced over the time by their users' interaction. This influence causes changes in the structure of the communities. Many researchers, consider that the structure of a community contains important information for network evolution as a whole. Thus, it is highly imperative to model the dynamic behavior in social networks and try to predict their evolution.

In this paper we study the problem of community evolution prediction in dynamic social networks. We address this problem as a supervised learning task where we predict four types of community evolutionary events, *growth*, *shrinkage*, *continuation* and *dissolution*. Various features were investigated in order to understand how they influence the results. Among them, are the structural and temporal characteristics of communities. What is unique in the current approach is that we use a first order logic formalism to represent the correlation between evolutionary events and the input features. Moreover Inductive Logic Programming (ILP) us used to learn event calculus clauses. Event calculus was chosen because it is human understandable, it can be used to model effect of actions in time, and the variation we have adopted can perform ILP in an online fashion which is especially useful in streaming media.

The rest of the paper is organised as follows. In Section 2 we refer to past work on community evolution prediction, in Section 3 we refer to the Event Calculus as a logic formalism but also to Inductive Logic Programming as a way of learning clauses, then in Section 4 we refer to the methodology we followed for community evolution prediction, in Section 5 we present experimental results, conclusions are drawn in Section 6. In appendix A we present samples of Event Calculus clauses, and an additional experiment with pruning.

2 Related Work

The literature in community evolution prediction is quite extensive in terms of features, classifier types and events predicted.

Patil et al. [14] predicted whether a community will disappear or will survive. They observed that both the level of member diversity and social activities are critical in maintaining the stability of communities. They also found that certain prolific members play an important role in maintaining the community's stability. Goldberg et al. [8] correlated the lifespan of a community with the structural parameters of its early stages. Brodka et al. [2],[7] tried to predict 6 evolutionary events of communities, i.e. growth, shrinkage, continuation,

dissolution, merging and split. They used as features the history of the events of a community in the three preceding timeframes, and the community size in these timeframes. They found that the prediction based on simple input features may be very accurate, while some classifiers are more precise than the others. Kairam et al. [11] tried to understand the factors contributing to the growth and longevity in a social network. They investigated the role that two types of growth (diffusion and non-diffusion) play during a community's formative stages. Diffusion growth is when a community attracts new members through ties to existing members. Non-diffusion growth occurs with individuals with no prior ties to the network. Diakidis et al. [4] studied on-line social networks as a supervised learning task with sequential and non-sequential classifiers. Structural, content and contextual features as well as the previous states of a community are considered as the features that are involved in the task of community evolution. The evolution phenomena they tried to predict are the continuation, shrinking, growth and dissolution.

Takaffoli et al. [16] quantified the events that may occur in a community as follows: survive:{true, false}, merge:{true, false}, split:{true, false}, size:{expand, shrink}, and cohesion:{cohesive, loose}. First they tried to predict whether a community will survive, followed by a separate predictor for each of the events.

Ilhan et al. [10] proposed a regression ARIMA model to predict values of community features based on the values of the past community instances. Then the predicted community features are used to train a classifier to predict the evolutionary events.

The classifiers proposed are quite opaque in terms of the model that is learnt. Our approach differs in trying build classifiers based on first order logic, and thus they can be inspected by humans.

3 Background: Event Calculus and Inductive logic programming

The Event Calculus (EC) is a temporal logic formalism for reasoning about actions and changes [13]. EC, that has been used as a basis in event recognition applications, provides among others, direct connections to machine learning, via Inductive Logic Programming (ILP) [3]. Its ontology comprises *time points*, represented by integers; time varying properties known as *fluents*; and actions known as *events*. The events occur in time and may affect the fluents by altering their value. The axioms of the EC incorporate the *law of inertia*, according to which fluents persist over time, unless they are affected by an event. Thus, if an event is initiated at time T, it will persist until another event will fire a termination rule. Also, if an event is terminated at time T, it will remain terminated until another event fires an initiation rule. The basic predicates are presented in Table 1, while the domain-independent axioms are in Table 2. Axiom (1) states that a high level event, represented as fluent F for convenience, is happening at time T if it has been initiated at the previous time point. While Axiom (2) states that F continues to happen unless it is terminated.

Let us examine how the evolution of a community could be represented in terms of EC; an evolutionary phenomenon such as the *growth* will be represented as a fluent, whereas the factors that contribute to the *growth* will be represented as events. For example in Table 3, it means that community $X0$ will grow in time T provided its size was 3 and its density was 4. Likewise, rules can be formed for the rest of the evolutionary phenomena as combinations of features (or events). The rules are also known as *clauses*, whereas the predicates *happensAt* are also known as *literals*. In addition, a rule B is a specialisation of rule A if the instances that satisfy rule B are a subset of the instances that satisfy rule A.

Table 1 The basic predicates of the EC.

Predicate	Meaning
happensAt(E,T)	Event E occurs at time T
initiatedAt(F,T)	At time T fluent F is initiated
terminatedAt(F,T)	At time T fluent F is terminated
holdsAt(F,T)	Fluent F holds at time T

Table 2 The domain-independent axioms.

Axioms
holdsAt(F,T +1) ← (1) initiatedAt(F,T).
holdsAt(F, T+1) ← (2) holdsAt(F,T), not terminatedAt(F,T).

Table 3 Theory Learnt by OLED.

```
initiatedAt(growth(X0),T) ←
        happensAt(size(X0,3),T),
        happensAt(density(X0,4),T).
```

The problem that we try to address is to learn such rules (or clauses) from data, rather than hand encode them; for that we can use inductive logic programming (ILP). In ILP, first order rules are learnt from relational data under a supervised learning scheme. Thus we have input data and class labels. The input data are often named the *narrative*, and the class label is known as the *annotation*.

Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a *hypothesised* logic program which entails all the positive and none of the negative examples. ILP provides various techniques for learning logical theories from examples. In Learning from Interpretations (LfI) [1] setting each training example is an interpretation, i.e. a set of narrative and annotation atoms (see Table 4). Given a set of training interpretations I and some background theory B, which consists of the domain-independent axioms of the EC, the goal in LfI is to find a theory.

In this paper, OLED (Online Learning of Event Definitions) [12] was used for learning rules that perform community evolution prediction. OLED is an online ILP system for learning logical theories from data streams. It has been designed having in mind the construction of knowledge bases for event recognition applications. These applications [5] process sequences of simple events, such as sensor data, and recognize complex events of interest, i.e. events that satisfy some pattern. Logic-based event recognition typically uses a knowledge base of first-order rules to represent complex event patterns and a reasoning engine to detect such patterns in the incoming data. In OLED this knowledge base is in the form of domain-specific axioms in the Event Calculus, i.e. rules that specify the conditions under which simple, low-level events initiate or terminate complex events.

OLED is using an online (single-pass) learning strategy. Online machine learning is a method of machine learning in which data becomes available in a sequential order and is used to update our best predictor for future data at each step, as opposed to batch learning techniques which generate the best predictor by learning on the entire training data set at once. To manage it, the Hoeffding bound [9] for evaluating clauses on a subset of the input stream, is used. With this approach, significant speed-ups are obtained in training time. Table 4 presents an example of input data that is provided to OLED. It consists of a narrative and an annotation list. Narratives are the simple (or low level) events in terms of happensAt/2, expressing the values of communities' features. i.e. happensAt(size(c1,3),1). denotes that community c1 has size 3 in time 1. Annotations are the complex (or high level

Table 4 Input of OLED.

Timeframe 1	Timeframe 2
Narrative happensAt(size(c1,3),1). happensAt(density(c1,4),1).	Narrative happensAt(size(c1,5),2). happensAt(density(c1,5),2).
Annotation	Annotation holdsAt(*growth*(c1),2).

events) events in terms of holdsAt/2, expressing the ground truth for our training set. i.e. holdsAt(*growth*(c1),2). denotes that community c1 grew in time 2. The non-existence of c1's annotation in time 1 states that growth event is terminated in time 1. Table 3 shows an example of the theory OLED learnt after training. It represents we will begin to have a growth event in time T+1 for any community, which has size 3 and density 4 in time T. This rule extracted because with these community features in time 1, we had a growth event in time 2 (Table 4).

Learning. OLED learns a clause in a top-down fashion, by gradually adding literals to its body. Instead of evaluating each candidate specialization on the entire input, it accumulates training data from the stream, until the Hoeffding bound allows to select the best specialization. The instances used to make this decision are not stored or reprocessed but discarded as soon as OLED extracts from them the necessary statistics for clause evaluation.

OLED relaxes the LfI requirement that a hypothesis H covers every training interpretation, and thus seeks a theory with a good fit in the training data. Let B consist of the domain-independent EC axioms, r be a clause and I an interpretation. We denote by narrative(I) and annotation(I) the narrative and the annotation part of I respectively (Table 4). We denote by Mr_I^r an answer set of $B \cup narrative(I) \cup r$. Given an annotation atom α we say that:

- α is a true positive (TP) atom clause r, iff $\alpha \in annotation(I) \cap Mr_I^r$.
- α is a false positive (FP) atom clause r, iff $\alpha \in Mr_I^r$ but $\alpha \notin annotation(I)$.
- α is a false negative (FN) atom clause r, iff $\alpha \in annotation(I)$ but $\alpha \notin Mr_I^r$.

We define a heuristic clause evaluation function G as follows:

$$G(r) = \begin{cases} \frac{TP_r}{TP_r+FP_r}, & \text{if } r \text{ is an initiatedAt clause} \\ \frac{TP_r}{TP_r+FN_r}, & \text{if } r \text{ is a terminatedAt clause} \end{cases}$$

where TP_r, FP_r and FN_r are the accumulated TP, FP and FN counts of clause r over the input stream and $G \in [0, 1]$.

On the arrival of new interpretations, OLED either expands H, by generating a new clause, or tries to expand (specialize) an existing clause. Clauses of low quality are pruned, after they have been evaluated on a sufficient number of examples. Below there is an example of OLED execution. Initially, processes Linit and Lterm start with two empty hypotheses, Hinit and Hterm. Assume that the annotation in one of the incoming interpretations dictates that the *growth* complex event holds at time 10, while it does not hold at time 9. Since no clause in Hinit yet exists to initiate *growth* at time 9, Linit detects the *growth* instance at time 10 as a FN and proceeds to theory expansion, generating an initiation clause for *growth*. Lterm is not concerned with initiation conditions, so it will take no actions in this case. Then, a new interpretation arrives, where the annotation dictates that *growth* holds

at time 20 but does not hold at time 21. In this case, since no clause yet exists in Hterm to terminate *growth* at time 20, Lterm will detect an FP instance at time 21. It will then proceed to theory expansion, generating a new termination condition for *growth*. At the same time, assume that the initiation clause in Hinit is over-general and erroneously re-initiates *growth* at time 20, generating an FP instance for the Linit process at time 21. In response to that, Linit will proceed to clause expansion, penalizing the over-general initiation clause by increasing its FP count, thus contributing towards its potential replacement by one of its specializations.

In EC the initiation/termination of complex events depends only on the simple events and contextual information of the previous time-point, therefore each interpretation is an independent training instance. This guarantees the independence of observations that is necessary for using the Hoeffding bound. The *Hoeffding bound* is a statistical tool that is used as a probabilistic estimator of the generalization error of a model (true expected error on the entire input), given its empirical error (observed error on a training subset). Given a random variable $X \in [0, 1]$ and an observed mean $\overline{X}$ of its values after n independent observations, the Hoeffding Bound states that, with probability $1 - \delta$, the true mean $\hat{X}$ of the variable lies in an interval $(\overline{X} - \varepsilon, \overline{X} + \varepsilon)$, where $\varepsilon = \sqrt{\frac{ln(1/\delta)}{2n}}$. In other words, the true average can be approximated by the observed one with probability $1 - \delta$, given an error margin ε that decreases with the number of observations n.

Let r be a clause and G a clause evaluation function. Assume also that after n training instances, $r1$ is r's specialization with the highest observed mean G-score $\overline{G}$ and $r2$ is the second best one, i.e. $\Delta\overline{G} = \overline{G}(r1) - \overline{G}(r2) > 0$. Then by the Hoeffding bound we have that for the true mean of the scores' difference $\Delta\hat{G}$ it holds $\Delta\hat{G} > \Delta\overline{G} - \varepsilon$, with probability $1 - \delta$. Hence, if $\Delta\overline{G} > \varepsilon$ then $\Delta\hat{G} > 0$, implying that $r1$ is indeed the best specialization to select at this point, with probability $1 - \delta$. In order to decide which specialization to select, it thus suffices to accumulate observations from the input stream until $\Delta\overline{G} > \varepsilon$. Also, because OLED allows to build decision models using only a small subset of the data, by relating the size of this subset to a user-defined confidence level on the error margin of not making a (globally) optimal decision, manages to consume small amounts of memory and time resources.

4 Proposed Methodology

A dynamic social network is time-stamped, and to be analysed it is *segmented* into time frames, with an overlap between them to allow for a smooth transition. The problem we are addressing is to predict the form of a community in the next frame, given some features of the existing form of a community. The model that perform the prediction is learnt through ILP and represented as clauses of Event Calculus.

4.1 Community Features

Pavlopoulou et al. [15] designed two types of features, the *structural* and the *temporal* ones. Structural features represented the physical characteristics of a community such as size, density etc. The temporal features included structural features and evolutionary events that were derived from the past instances of a community, and from relations between past instances of a community. In this work, we use the same features which describe below but first let us introduce some notation: C_t is the set of communities at time frame F_t; C_t^k is the community k of set C_t; $n(F_t) = |V_t|$ is the size of set V_t; $m(F_t) = |E_t|$ is the size of set E_t;

V_t and E_t are the sets of vertices and edges, respectively, at time frame F_t; $m_{out}(C_t^k)$ is the number of edges from community C_t^k to any other community in the same timeframe; and $C_{t_j}^{k_j}$ is the ancestor of $C_{t_i}^{k_i}$, where $j < i$.

The ancestors of a community do not necessarily belong to consecutive time frames. In the current experiments, each community is tracked in each timeframe until its dissolution, thus there are situations in which a community disappears at timeframe t_i but it reappears at timeframe t_j, where $j - i > 1$. Thus, the i-th ancestor of a community is the i-th appearance of the community in the past, counting from the present. Next are the features we used in detail:

Structural Features.

Size is the normalized value for the size of a community C_t^k in time frame F_t:

$$Size(C_t^k) = \frac{n(C_t^k)}{n(F_t)}$$

Density is the number of C_t^k edges to the maximum number of edges the community could have:

$$Density(C_t^k) = \frac{m(C_t^k)}{n(C_t^k)(n(C_t^k) - 1)/2}$$

Cohesion is defined as:

$$Cohesion(C_t^k) = \frac{2m(C_t^k)(n(F_t) - n(C_t^k))}{m_{out}(C_t^k)(n(C_t^k) - 1)}$$

Normalised Association is defined as:

$$NormalizedAssociation(C_t^k) = \frac{2m(C_t^k)}{2m(C_t^k) + m_{out}(C_t^k)}$$

Ratio Association is the average internal degree of a community's members:

$$RatioAssociation(C_t^k) = \frac{2m(C_t^k)}{n(C_t^k)}$$

Ratio Cut is the average external degree of a community's members:

$$RatioCut(C_t^k) = \frac{m_{out}(C_t^k)}{n(C_t^k)}$$

Normalized Edges Number is defined as:

$$NormalizedEdgesNumber(C_t^k) = \frac{m(C_t^k)}{m(F_t)}$$

Average Path Length shows how close on average two random nodes are:

$$AveragePathLength(C_t^k) = \frac{\sum_{v,u \in V_t^k, v \neq u} dist(v,u)}{n(C_t^k)(n(C_t^k) - 1)}$$

where $dist(v, u)$ indicates the shortest distance between nodes v and u.

Diameter is the maximum shortest path between all pairs of nodes in community C_t^k:

$$Diameter(C_t^k) = \max_{u,v \in V_t^k, u \neq v} dist(u,v)$$

Clustering Coefficient We set as clustering coefficient of a community the average of the local clustering coefficient of each node. The local clustering coefficient for a vertex v in a community C_t^k is defined as,

$$ClusteringCoefficient(v) = \frac{2neigh(v)}{neigh(v)(neigh(v)-1)}$$

where $neigh(v) = |\{u : (u,v) \in E_t^k\}|$ is the number of neighbours of vertex v and $neighE(v) = |\{(u,w) \in E_t^k : (u,v) \in E_t^k, (w,v) \in E_t^k\}|$ is the number of edges among the neighbours of vertex v.The clustering coefficient of a community C_t^k is the average over all its members:

Centrality measures how central each node of a community C_t^k is. We used three centrality measures as features, namely closeness, betweenness and eigenvector centrality [17].

Temporal features.

Structural features and Evolutionary events of N ancestors: One group of temporal features is all the structural features, as described above, as well as the *evolutionary events* for the first n immediate ancestors of community $C_{t_p}^{k_p}$.

Another group of temporal features concerns pairs of communities and depict how a community has evolved compared to its previous instance in time. Using these pairs of communities for a given number of ancestors n to use, we compute the following temporal features:

Similarity of consecutive communities is the fraction between the nodes/edges that are common in both instances of the community and total nodes/edges of two instances.

$$Jacc_{Nodes}(C_{t_i}^{k_i}, C_{t_{i-1}}^{k_{i-1}}) = \frac{|V_{t_i}^{k_i} \cap V_{t_{i-1}}^{k_{i-1}}|}{|V_{t_i}^{k_i} \cup V_{t_{i-1}}^{k_{i-1}}|}, Jacc_{Edges}(C_{t_i}^{k_i}, C_{t_{i-1}}^{k_{i-1}}) = \frac{|E_{t_i}^{k_i} \cap E_{t_{i-1}}^{k_{i-1}}|}{|E_{t_i}^{k_i} \cup E_{t_{i-1}}^{k_{i-1}}|},$$

$$Jacc_{Nodes\&Edges}(C_{t_i}^{k_i}, C_{t_{i-1}}^{k_{i-1}}) = \frac{|V_{t_i}^{k_i} \cap V_{t_{i-1}}^{k_{i-1}}| + |E_{t_i}^{k_i} \cap E_{t_{i-1}}^{k_{i-1}}|}{|V_{t_i}^{k_i} \cup V_{t_{i-1}}^{k_{i-1}}| + |E_{t_i}^{k_i} \cup E_{t_{i-1}}^{k_{i-1}}|}$$

where $E_{t_i}^{k_i}$ is the set of edges and $V_{t_i}^{k_i}$ the set of nodes of community $C_{t_i}^{k_i}$.

Join nodes ratio is the percentage of nodes joining the dynamic community as it evolves.

$$JoinNodesRatio(C_{t_i}^{k_i}, C_{t_{i-1}}^{k_{i-1}}) = \frac{|V_{t_i}^{k_i} \setminus V_{t_{i-1}}^{k_{i-1}}|}{|V_{t_i}^{k_i}|}$$

Left nodes ratio is the percentage of nodes leaving the dynamic community as it evolves.

$$LeftNodesRatio(C_{t_i}^{k_i}, C_{t_{i-1}}^{k_{i-1}}) = \frac{|V_{t_{i-1}}^{k_{i-1}} \setminus V_{t_i}^{k_i}|}{|V_{t_{i-1}}^{k_{i-1}}|}$$

Activeness is the ratio of the number of edges in current community $C_{t_i}^{k_i}$ which also existed in its ancestor community $C_{t_{i-1}}^{k_{i-1}}$, to the number of nodes in current community $C_{t_i}^{k_i}$.

$$Activeness(C_{t_i}^{k_i}, C_{t_{i-1}}^{k_{i-1}}) = \frac{|E_{t_i}^{k_i} \cap E_{t_{i-1}}^{k_{i-1}}|}{|V_{t_i}^{k_i}|}$$

The last two temporal features are computed for individual communities instead of pairs.

Lifespan Given a community $C_{t_w}^{k_w}$ which is part of dynamic community M and belongs to time frame t_w, the lifeSpan is defined as,

$$LifeSpan(C_{t_w}^{k_w}) = \frac{|\{C_{t_p}^{k_p} \in M : p < w\}|}{t_w - 1}$$

which is the ratio of the number of time frames between the current community $C_{t_w}^{k_w}$ and the very first instance of the same dynamic community (total number of ancestors of $C_{t_w}^{k_w}$), to the maximum number of ancestors $C_{t_w}^{k_w}$ could have. The maximum number of ancestors $C_{t_w}^{k_w}$ could have is equal to $t_w - 1$, where t_w is the number of the time frame where $C_{t_w}^{k_w}$ belongs to. In that case there would be an instance of dynamic community M in every time frame from the very first one until time frame $t_w - 1$.

Aging of a community $C_{t_w}^{k_w}$, which is part of the dynamic community M is the average age of the community members. The age of a member is increased by 1 every time it is found to be also a member of an ancestor community of $C_{t_w}^{k_w}$ in the corresponding dynamic community. Aging is normalized by dividing with the maximum possible age of members, which equals w.

$$Aging(C_{t_w}^{k_w}) = \frac{\sum_{v \in V_{t_w}^{k_w}} |\{C_{t_p}^{k_p} \in M : p \leq w, v \in V_{t_p}^{k_p}\}|}{(|\{C_{t_p}^{k_p} \in M : p < w\}| + 1) n(C_{t_w}^{k_w})}$$

Feature Quantization. In OLED the values of variables are discrete thus we implemented two methods to quantize variables. Let q_{value} be the number of quantized values, f_v be the set with values of feature f. In first method, for each feature we split values' total range to q_{value} intervals and the width of each is $\frac{max\{f_v\} - min\{f_v\}}{q_{value}}$. Thus the first interval is $(min\{f_v\}, min\{f_v\} + q_{value})$, the second is $(min\{f_v\} + q_{value} + 1, min\{f_v\} + 2q_{value})$ and so on. The quantized value of each feature is the index of the interval it belongs to. The second method sorts feature's values in a list and creates q_{value} sets. Taking one by one the values from sorted list, we begin to fill the q_{value} sets with consecutive values until each set has $\frac{|f_v|}{q_{value}}$ feature's values. Finally, if at least one feature or tag of community C_k is missing we delete the C_k.

4.2 Community Evolution Prediction

OLED was used to predict four evolutionary events: *growth*, *shrinkage*, *continuation*, *dissolution*. Note that OLED handles two-class problems, so it predicts if a community will sustain or will stop sustaining an evolutionary event. In Figure 1 we present the architecture of the prediction system.

The performance of an ILP system may degrade if the background knowledge provided contains large amounts of irrelevant information so experts are required to set the background knowledge they believe to be useful. Table 5 presents an example of background knowledge, where the following types of rules can be defined: Rules for community entity recognition; rules for time entity recognition; facts for features' quantized values recognition; rules for values of ground truth recognition; and rules which represent the inertia of Event Calculus. OLED can produce predicates of many forms. For example, an argument of a predicate can be considered as input or as output. *Modes* declaration is a language that limits the forms a predicate can have. Table 7 presents an example of modes.

The form of rules that OLED learns is presented in Table 6. In the head of the rule, *predicted_event* is one of labels we try to predict (*growth*, *shrinkage*, *continuation*, *dissolution*). Notice that the $community_i$ and $time_j$ indices are the same in the body and head.

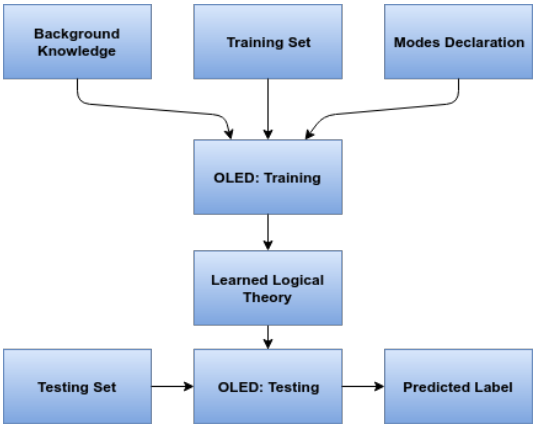


Figure 1 Learning Architecture.

Table 5 Example of A OLED's Background Knowledge File.

Background Knowledge File	
holdsAt(F,T_e) :- fluent(F), holdsAt(F,T_s), not terminatedAt(F,T_s), $T_e = T_s + 1$, time(T_s),time(T_e).	holdsAt(F,T_e) :- fluent(F), initiatedAt(F,T_s), $T_e = T_s + 1$, time(T_s),time(T_e). Inertia of Event Calculus
fluent($growth(X)$) :- community(X).	Ground truth recognition
community(X) :- happensAt(size(X,_),_). community(X) :- happensAt(density(X,_),_).	Community entity recognition
time(X) :- happensAt(size(_,_),X). time(X) :- happensAt(density(_,_),X).	Time entity recognition
value(1..5).	Features' quantized values recognition

Table 6 Rules That OLED Learns.

Rules	
initiatedAt/terminatedAt($< predicted\_event >(< community_i >),< time_j >$) :- happensAt($< feature_1 >(< community_i >,< value_1 >),< time_j >$)), ... happensAt($< feature_n >(< community_i >,< value_n >),< time_j >$)).	(1)/(2)

Table 7 Example of A OLED's Mode Declarations File.

Mode Declarations File	
modeh(initiatedAt($growth$(+community),+time)) modeh(terminatedAt($growth$(+community),+time))	The form of the rule's head
modeb(happensAt(size(+community,#value),+time)) modeb(happensAt(density(+community,#value),+time))	The form of the rule's body

Rules can be interpreted as if $feature_1$ of community $community_i$ has $value_1$ at $time_j$ and the same is true for the rest of features then the initiation of event *predicted_event* is fired. This means that the *predicted_event* will start to occur at $time_{j+1}$. The *happensAt* predicates that are required will be discovered by OLED. Likewise, if the body of rule (2) is true then the termination of event *predicted_event* is fired, thus the *predicted_event* will stop to occur at time $time_{j+1}$.

Training and Testing. As shown in Figure 1, the dataset was split into training and testing sets according to the Time Series Cross Validation, because it takes into account the temporal relationship between the training and testing sets. Each training comprises only observations that occurred prior to the observation.

Fold 1:	Training set includes low events of communities from timeframes F_1, F_2 and high events of communities in timeframe F_2. Testing set includes low events of communities from timeframe F_2 and high events of communities from timeframes F_2, F_3.
Fold 2:	Training set includes low events of communities from timeframes F_1, F_2, F_3 and high events of communities from timeframe F_2, F_3. Testing set includes low events of communities from timeframe F_3 and high events of communities from timeframes F_3, F_4.
...	...
Fold T-2:	Training set includes low events of communities from timeframes F_1, F_2, ..., F_{T-1} and high events of communities from timeframe F_2, F_3, ..., F_{T-1}. Testing set includes low events of communities from timeframe F_{T-1} and high events of communities from timeframes F_{T-1}, F_T.

T is total number of timeframes in the dataset. Note that the first timeframe has no evolutionary events (high events) since there is no previous timeframe in order to track the communities' evolution of the first timeframe. Respectively, the last timeframe has not features (low events) because there is no next timeframe to predict evolution of its communities. Also, in the training set we comprise the low level events of the timeframe that we are going to predict so that OLED extracts the time variable for high level events. Finally, notice that in the testing set we also comprise the high level events of the previous timeframe than that we are going to predict. This is required by OLED to initiate the inertia of every community's event.

OLED as an online learner splits its input into chunks. In our experiments, we choose chunks of size 2. Thus, the imported timeframes for the training procedure are split into chunks two by two. We changed the functionality of OLED so that it creates rolling chunks. It means that first chunk contains the timeframes 1,2; the second one the timeframes 2,3; the third the timeframes 3,4 and so on. This is necessary because, for example, timeframe 2 has to be in the first and second chunk. In the first chunk, we need the high level events of timeframe 2 for getting the ground truth. While in the second chunk we use the low level events of timeframe 2 as features for our supervised learning classification.

The outline of training process is the following: Initially there is an empty theory. Each time OLED receives a chunk of training examples and transforms the existing theory to satisfy as close as possible the right prediction of the current examples. When the training process is completed, a logical theory is derived as the learnt model. Its form is illustrated in Table 6. Using this theory we predict the evolutionary events of communities which are in testing set.

Table 8 Survival Experiment.

	Survival Structural	Survival Temporal
Micro/Macro Precision	0.9737/0.8125	0.9882/0.9941
Micro/Macro Recall	0.9949/0.6289	1.0000/0.6765
Micro/Macro Fscore	0.9842/0.7090	0.9941/0.8051

5 Experiments

Dataset description. The data were collected from the Mathematics Stack Exchange forum[1], which is a question and answer site for mathematics. All questions are tagged with their subject areas. The dataset comprises 376,030 posts, 261,600 answers and comments, between 28-09-2009 and 31-05-2013. Each user is represented by a node in a graph and there is an edge between two user nodes if one of them posts an answer or a comment on the other user's post. The dataset was split into 10 equally sized, with respect to the number of posts (questions, answers or comments), timeframes with 60% overlap between them.

Building the ground truth means obtaining community labels per time frame, and then obtaining the evolution of each community across time frames. We considered that a group of users belongs in the same community if they post (questions, answers or comments) about the same topic. In particular, we used tags to determine the communities and since on each post there are multiple tags, thus each user will be assigned to multiple communities. Answer and comment posts inherit the tag of the question they correspond to. Also communities with no more than 3 members were removed. The evolutionary events of each community (*Growth*, *Shrinkage*, *Continuation*, *Dissolution*) were obtained by thresholding. In particular if the size of the community in the next time frame is more (less) than 30 nodes compared to the size in the current frame then the community grows (shrinks).

There are communities which do not appear in each timeframe, although they may not have been dissolved yet. It happens because communities with few members in a timeframe are pruned from dataset. So, we are looking for the evolution of a community in every timeframe of the dataset and consider a community as dissolved only after its last appearance. The evolutionary events of dataset are imbalanced. In particular, the percentage of each class is: *Growth*: 0.5%, *Shrinkage*: 0.2%, *Continuation*: 90% and *Dissolution*: 0.3%.

The features were quantized into 5 levels, and represented as low level events. The high level events are the evolutionary events. Experiments were executed with both structural and temporal features, where the number of ancestors was set to 4. At the end, the dataset was split in training and testing sets using Time Series Cross Validation method. Because the data are highly imbalanced apart from Micro Average measures, we also used Macro Averages.

Survival Experiment. First, we conducted an experiment with an event, named *survival* that incorporated all the *growth*, *shrinkage* and *continuation* events. We used both structural and temporal features. In Table 8 we present the results. The micro measures are very high because the survival events are 97% of the total data. As OLED initialized the inertia of the survival event, it did not find enough negatives examples (dissolution events) to fail in its prediction. Thus the Macro values are much lower.

[1] https://archive.org/download/stackexchange

Table 9 All events Structural features.

	Growth	Shrinkage	Continuation	Dissolution
Micro/Macro Precision	0.2358/0.6037	0.1884/0.5832	0.9293/0.8066	0.6512/0.8125
Micro/Macro Recall	0.3027/0.6316	0.1512/0.5671	0.9760/0.6939	0.2629/0.6289
Micro/Macro Fscore	0.2651/0.6174	0.1677/0.5750	0.9521/0.7460	0.3746/0.7090

Table 10 All events Temporal features.

	Growth	Shrinkage	Continuation
Micro/Macro Precision	0.1828/0.5730	0.1882/0.5743	0.9182/0.9222
Micro/Macro Recall	0.1828/0.5730	0.1633/0.5649	0.9955/0.6932
Micro/Macro Fscore	0.1828/0.5730	0.1749/0.5696	0.9553/0.7915

Experiment with all events. The results on all events with *structural* features appear in Table 9. The macro precision is highest in the *dissolution*. The dataset with the *temporal features* contains the features of the previous 4 instances of a community, the first timeframe for this dataset is at time 5 (see Table 10). The theory which was derived for the dissolution event was empty. The predictor could not evaluate any rule with high score because there were not many available examples, since the number of timeframes (6, from F_5 to F_{10}) and the number of communities is small. Thus, the dissolution event is not included in the experiments with *temporal features*. The *growth* and *shrinkage* events with temporal features have lower performance than the best corresponding events with *structural* features. But for the *continuation* event, the reverse is true for both micro and macro values.

Experiment with long range rules. We changed the way rules are formed so that they contain features of any of a community's ancestors. This is similar to the temporal features, but we do not have their values as different features but as the same features at different time steps. In order to change the form of the derived rules we changed the modes declaration so that the new form of rules captures long range relationships (see also Section 4.2). The form of the new rules is shown in Table 11, which is the same as in Table 6 except of the *<time>* value in the head can be different from that in the body and the *geqn/3* predicate, which denotes that $time_k$ is num_1 units after $time_l$. Also because now OLED could include more than two timeframes we had to increase the chunk size. If the chunk size equals to $N + 2$, then the derived rules can contain up to N ancestors' features. However, a big chunk size entails greater CPU and memory requirements. In the experiments we selected a chunk of size 3, so we obtained rules that contained features of the first ancestor. The results are presented in Table 12.

Experiment with weighted TPs, FPs, FNs. Neither the previous method increased the performance significantly. A problem is that the learnt theories contain more termination than initiation rules, thus the initiation of some events does not happen. It means OLED predicts a negative event (i.e. event that does not occur) for a community at next timeframe but in reality it is a positive event (i.e. the event occurs). In this case the FNs frequency of OLED is increased. The numbers of initiation and termination rules are not balanced because OLED evaluates its rules based on TPs, FPs and FNs values. Using these values, it computes a score which evaluates the accuracy of a rule. To control score's value we can add weights on TPs, FPs, FNs values during the training. For example, if the FNs weight is set

Table 11 New Rules That OLED Learns With Long Range Relationships.

Rules
initiatedAt/terminatedAt(< $predicted\_event$ >(< $community_i$ >),< $time_j$ >) :- happensAt(< $feature_1$ >(< $community_i$ >,< $value_1$ >),< $time_1$ >)), ..., happensAt(< $feature_n$ >(< $community_i$ >, < $value_n$ >), < $time_n$ >)), geqn($time_k$, $time_l$, num_1), ... geqn($time_m$, $time_n$, num_1).

Table 12 Long Range Relationships Experiment.

	Growth	Shrinkage
Micro/Macro Precision	0.2446/0.6090	0.2016/0.5898
Micro/Macro Recall	0.3487/0.6527	0.1512/0.5678
Micro/Macro Fscore	0.2875/0.6300	0.1728/0.5786

to 10, it means that the FNs will be considered as ten times more than it really is, in other words the termination rules will overestimate the termination condition. Thus the score of termination rules is getting decreased. With this way we focus more in quality than quantity of termination rules. In Table 13, we present the best weights for each class in a experiment with structural and temporal features. The results are presented in Table 14 and Table 15 for the experiment with structural features and with the temporal features respectively. While we were trying various values to weights, we noticed in the results that:

- If TPs's weight is increased then TPs is increased, FPs is increased and FNs is decreased because the number of initiations rules is increased.
- If FPs's weight is increased then TPs is decreased, FPs is decreased and FNs is increased because the number of initiations rules is decreased.
- If FNs's weight is increased then TPs is increased, FPs is increased and FNs is decreased because the number of termination rules is decreased.

We tried to increase the low TPs number by setting appropriate weights, but FPs also increased. OLED overestimated the initiation condition because its initiation rules are not specialised enough to detect correctly in which communities an event will occur. This is a strong indication that with the current features OLED performance could not improve. In Appendix A, we present some of the clauses that derived by above experiments.

6 Conclusions

We tried to predict the evolution of communities in a dynamic social network. The evolution of a community is described as the occurrence of *growth*, *shrinkage*, *continuation* and *dissolution* events. We carried out the prediction using OLED, an Inductive Logic Programming system for learning logical theories from data streams. Initially, we tracked the evolution of communities over the time and obtained the ground truth of evolutionary events. As features we used structural characteristics of communities. Moreover, we also tried temporal features where a preset number of previous instances of each communities were used as well as features that capture change between consecutive instances of a community. Subsequently,

Table 13 Best weights for each class with structural/temporal features.

	TPs-weight	FPs-weight	FNs-weight
Growth	1/1	1/5	15/1
Shrinkage	20/1	1/1	15/1
Continuation	1/1	1/1	1/15
Dissolution	1	1	15

Table 14 Weights on TPs,FPs,FNs - Experiment With Structural Features.

	Growth	Shrinkage	Continuation	Dissolution
Micro/Macro Precision	0.2376/0.6055	0.1127/0.5533	0.9247/0.9623	0.8036/0.8878
Micro/Macro Recall	0.3487/0.6519	0.8023/0.8187	0.9845/0.6772	0.2113/0.6047
Micro/Macro Fscore	0.2826/0.6278	0.1977/0.6603	0.9537/0.7950	0.3346/0.7194

Table 15 Weights on TPs,FPs,FNs - Experiment With Temporal Features.

	Growth	Shrinkage	Continuation
Micro/Macro Precision	0.2184/0.5913	0.2459/0.6032	0.9182/0.9222
Micro/Macro Recall	0.2043/0.5857	0.1531/0.5654	0.9955/0.6933
Micro/Macro Fscore	0.2111/0.5885	0.1887/0.5837	0.9553/0.7915

the features were quantized. The dataset was obtained from the Mathematics Stack Exchange forum. We presented the micro and macro averages, because the classes (*Growth*, *Shrinkage*, *Continuation* and *Dissolution*) were unbalanced.

We also investigated the best pruning values for the theory in OLED, which did not improve the results. Overall, the experiments with the temporal features had a worse performance than the experiment with the structural features, probably because there were not many timeframes. Then, we execute experiments where OLED learnt rules that represent long range relationships between an evolutionary event and features.

Subsequently, weights were applied to TPs, FPs and FNs values to change rules' scores. This was the experiment with the best results. Finally, we presented the features that were the most influential for each evolutionary event.

Future work could be directed to a range of different fields. Others classifiers can be used to predict the evolution of communities (e.g. SVM, Random Forest) and compare them to our results. Additional, evolutionary events can be added such as merge or split. Other types of features (i.e. topics or context of discussions in social networks) could be studied as well as features that capture the dynamics of communities, such as the rate of change of an existing feature. Also, another quantization algorithm, which will adapt better the quantized values to the distribution of the values of the features might help. Because OLED is an online system, it needs many data to decide if a rule is trusted. However, in our dataset we had only 10 frames. Thus, datasets with more timeframes could be examined. Moreover, a different segmentation of the data stream could be tried, to study its effect on the prediction. Finally, it would be very interesting to test the learnt rules in other datasets to notice how relevant they are at the problem of community evolution prediction.

References

1. Hendrik Blockeel, Luc De Raedt, Nico Jacobs, and Bart Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93, Mar 1999. doi:10.1023/A:1009867806624.
2. P. Bródka, P. Kazienko, and B. Kołoszczyk. Predicting Group Evolution in the Social Network. *ArXiv e-prints*, 2012. arXiv:1210.5161.
3. L. De Raedt. *Logical and Relational Learning*. Cognitive Technologies. Springer Berlin Heidelberg, 2008. URL: https://books.google.gr/books?id=FFYIOXvwq7MC.
4. Georgios Diakidis, Despoina Karna, Dimitris Fasarakis-Hilliard, Dimitrios Vogiatzis, and George Paliouras. Predicting the evolution of communities in social networks. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, WIMS '15, pages 1:1–1:6, New York, NY, USA, 2015. ACM. doi:10.1145/2797115.2797119.
5. Opher Etzion and Peter Niblett. *Event Processing in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2010.
6. Santo Fortunato. Community detection in graphs. *CoRR*, abs/0906.0612, 2009. URL: http://arxiv.org/abs/0906.0612.
7. Bogdan Gliwa, Piotr Bródka, Anna Zygmunt, Stanislaw Saganowski, Przemyslaw Kazienko, and Jaroslaw Kozlak. Different approaches to community evolution prediction in blogosphere. In *Advances in Social Networks Analysis and Mining 2013, ASONAM '13, Niagara, ON, Canada - August 25 - 29, 2013*, pages 1291–1298, 2013. doi:10.1145/2492517.2500231.
8. Mark Goldberg, Malik Magdon ismail, Srinivas Nambirajan, and James Thompson. Tracking and predicting evolution of social communities. -, -.
9. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi:10.1080/01621459.1963.10500830.
10. Nagehan İlhan and Şule Gündüz Öğüdücü. Predicting community evolution based on time series modeling. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, ASONAM '15, pages 1509–1516, New York, NY, USA, 2015. ACM. doi:10.1145/2808797.2808913.
11. Sanjay Ram Kairam, Dan J. Wang, and Jure Leskovec. The life and death of online groups: Predicting group growth and longevity. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 673–682, New York, NY, USA, 2012. ACM. doi:10.1145/2124295.2124374.
12. Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. Online learning of event definitions. *Theory and Practice of Logic Programming*, 16(5-6):817–833, 2016. doi:10.1017/S1471068416000260.
13. Robert Kowalski and Marek Sergot. *A Logic-Based Calculus of Events*, pages 23–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989. doi:10.1007/978-3-642-83397-7_2.
14. Akshay Patil, Juan Liu, and Jie Gao. Predicting group stability in online social networks. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1021–1030, New York, NY, USA, 2013. ACM. doi:10.1145/2488388.2488477.
15. Maria Evangelia G. Pavlopoulou, Grigorios Tzortzis, Dimitrios Vogiatzis, and George Paliouras. Predicting the evolution of communities in social networks using structural and temporal features. In *12th International Workshop on Semantic and Social Media Adaptation and Personalization, SMAP 2017, Bratislava, Slovakia, July 9-10, 2017*, pages 40–45, 2017. doi:10.1109/SMAP.2017.8022665.
16. Mansoureh Takaffoli, Reihaneh Rabbany, and Osmar R. Zaiane. Community evolution prediction in dynamic social networks. doi:10.1109/ASONAM.2014.6921553.
17. R. Zafarani, M.A. Abbasi, and H. Liu. *Social Media Mining: An Introduction*. Cambridge University Press, 2014. URL: https://books.google.gr/books?id=H9FkAwAAQBAJ.

Table 16 Rules learnt in the best experiment: Growth and Shrinkage.

initiatedAt(*growth*(X0),T1) :- happensAt(density(X0,1),T1), happensAt(diameter(X0,2),T1).
terminatedAt(*growth*(X0),T1) :- happensAt(ratio_cut(X0,3),T1), happensAt(average_path_length(X0,3),T1), happensAt(normalized_edges_number(X0,5),T1).
terminatedAt(*growth*(X0),T1) :- happensAt(ratio_cut(X0,3),T1), happensAt(closeness_centrality(X0,3),T1), happensAt(normalized_edges_number(X0,5),T1).
terminatedAt(*growth*(X0),T1) :- happensAt(cohesion(X0,2),T1), happensAt(average_path_length(X0,3),T1), happensAt(diameter(X0,2),T1).
terminatedAt(*shrinkage*(X0),T1) :- happensAt(ratio_association(X0,3),T1).
terminatedAt(*shrinkage*(X0),T1) :- happensAt(average_path_length(X0,2),T1).
terminatedAt(*shrinkage*(X0),T1) :- happensAt(closeness_centrality(X0,2),T1), happensAt(ratio_cut(X0,1),T1).
initiatedAt(*shrinkage*(X0),T1) :- happensAt(eigenvector_centrality(X0,1),T1), happensAt(ratio_association(X0,5),T1).

A Appendix

An advantage of OLED is that the predictive model (Theory) it derives is human-readable. Thus the rules can be read, analyzed and interesting results can be derived from them. Some of the best performing rules are shown in Tables 16 and 17. Transferability to new datasets is also an interesting possibility.

Some features appeared more often in rules of specific evolutionary events than others; while some never appeared. In Tables 18 and 19 we present for each evolutionary event (*growth*, *shrinkage*, *continuation*, *dissolution*), the frequency of the structural features the bodies of rules.

In Table 18 it can be noticed that features like *diameter*, *cohesion*, *ratio_cut* and *average_path_length* affect the prediction of the *growth* event since they represent 54.14% of total features which appeared in the rules. On the contrary features like *betweenness_centrality* and *normalized_association* did not appear at all. For the *shrinkage* event the most used features are *ratio_association*, *ratio_cut*, *cohesion* and *clustering_coefficient*, while the *normalized_association* feature does not appear. In Table 19 the features of the *continuation* and the *dissolution* events are presented. The continuation event seems to be affected mostly from *ratio_cut*, *ratio_association* and the *clustering_coefficient* and not by the cohesion. While for the *dissolution* event, every feature is used in prediction, and especially the *clustering_coefficient*, *cohesion* and the *betweenness_centrality*.

Table 17 Rules learnt: Survival.

terminatedAt(*survival*($X0$),$T1$) :- happensAt(ratio_association($X0$,4),$T1$), happensAt(density($X0$,2),$T1$).
terminatedAt(*survival*($X0$),$T1$) :- happensAt(ratio_association($X0$,3),$T1$), happensAt(clustering_coefficient($X0$,4),$T1$).
terminatedAt(*survival*($X0$),$T1$) :- happensAt(diameter($X0$,3),$T1$), happensAt(ratio_cut($X0$,3),$T1$), happensAt(ratio_association($X0$,2),$T1$).
terminatedAt(*survival*($X0$),$T1$) :- happensAt(ratio_association($X0$,3),$T1$), happensAt(closeness_centrality($X0$,3),$T1$).
terminatedAt(*survival*($X0$),$T1$) :- happensAt(clustering_coefficient($X0$,1),$T1$), happensAt(closeness_centrality($X0$,5),$T1$), happensAt(cohesion($X0$,4),$T1$).
terminatedAt(*survival*($X0$),$T1$) :- happensAt(normalized_edges_number($X0$,2),$T1$), happensAt(cohesion($X0$,2),$T1$), happensAt(average_path_length($X0$,1),$T1$).
terminatedAt(*survival*($X0$),$T1$) :- happensAt(size($X0$,1),$T1$), happensAt(betweenness_centrality($X0$,1),$T1$), happensAt(cohesion($X0$,3),$T1$).
terminatedAt(*survival*($X0$),$T1$) :- happensAt(normalized_edges_number($X0$,1),$T1$), happensAt(cohesion($X0$,3),$T1$), happensAt(average_path_length($X0$,1),$T1$).

In Tables 20 and 21 we present temporal features which appear in the rules of corresponding experiments. For the *growth* event the temporal features: *ancestor4_average_path_length*, *activeness_ancestor_2_ancestor3*, *aging_ancestor0, ancestor1_diameter*, *ancestor3_closeness_centrality* and the rest that are presented in Table 20, are the equally important. *Shrinkage* event prediction uses the values of *ancestor1_event_is_shrinking*, *ancestor4_clustering_coefficient*, *cohesion*, *eigenvector_centrality*, *joinNodesRatio_currentCommunity_ancestor0*, *ratio_cut*. Many temporal features are missing from the bodies of rules for both *growth* and *shrinkage* events. For the *continuation* event only *cohesion* and *ratio_cut* were used.

Experiment with pruning. A learnt theory can be pruned to remove clauses whose score is smaller than a quality threshold S_{min}. In the previous experiments S_{min} was 0.9. Next we tried for each event the values 0.5, 0.7, 0.3 as S_{min} and choose the ones with the best performance. With the structural features, the best pruning value for the *growth* event is 0.7, for *shrinkage* 0.5, for *continuation* 0.9 and for *dissolution* 0.9 (see Table 22). In the temporal features, the best pruning value for the *growth* event is 0.7, for *shrinkage* 0.7, and for *continuation* 0.9 (see Table 23). Overall, pruning had a marginal improvement on the results.

Table 18 Structural features frequency: Growth and Shrinkage.

Growth	Percentage	Shrinkage	Percentage
diameter	17.68%	ratio_association	20%
cohesion	13.26%	ratio_cut	13.55%
ratio_cut	11.60%	cohesion	11.61%
average_path_length	11.60%	clustering_coefficient	10.97%
density	8.29%	eigenvector_centrality	9.03%
ratio_association	7.73%	density	8.39%
clustering_coefficient	7.73%	average_path_length	7.10%
size	6.63%	closeness_centrality	6.45%
closeness_centrality	6.08%	centrality	3.871%
eigenvector_centrality	3.87%	diameter	3.87%
normalized_edges_number	2.76%	betweenness_centrality	2.58%
centrality	2.76%	size	1.29%
		normalized_edges_number	1.29%

Table 19 Structural features frequency: Continuation and Dissolution.

Continuation	Percentage	Dissolution	Percentage
ratio_cut	16.07%	clustering_coefficient	25.93%
ratio_association	15%	cohesion	15.74%
clustering_coefficient	10%	betweenness_centrality	10.19%
density	9.64%	diameter	9.26%
diameter	8.21%	size	8.33%
closeness_centrality	8.21%	normalized_edges_number	6.48%
eigenvector_centrality	7.14%	ratio_association	5.56%
centrality	6.79%	closeness_centrality	3.70%
betweenness_centrality	6.43%	average_path_length	3.70%
normalized_association	4.64%	ratio_cut	2.78%
average_path_length	4.64%	normalized_association	2.78%
normalized_edges_number	2.5%	centrality	2.78%
size	0.71%	density	1.85%
		eigenvector_centrality	0.93%

Table 20 Temporal features frequency: Growth.

Growth	Percentage
ancestor4_average_path_length	12.5%
activeness_ancestor_2_ancestor3	6.25%
aging_ancestor0	6.25%
ancestor1_diameter	6.25%
ancestor3_closeness_centrality	6.25%
ancestor3_clustering_coefficient	6.25%
ancestor3_diameter	6.25%
ancestor4_centrality	6.25%
ancestor4_clustering_coefficient	6.25%
ancestor4_diameter	6.25%
jaccardCoefficient_ancestor_0_ancestor1	6.25%
jaccardCoefficient_ancestor_2_Ancestor3	6.25%
joinNodesRatio_ancestor_2_ancestor3	6.25%
joinNodesRatio_currentCommunity_ancestor0	6.25%
leftNodesRatio_ancestor_0_ancestor1	6.25%

Table 21 Temporal features frequency: Shrinkage and Continuation.

Shrinkage	Percentage
ancestor1_event_is_shrinking	16.66%
ancestor4_clustering_coefficient	16.66%
cohesion	16.66%
eigenvector_centrality	16.66%
joinNodesRatio_currentCommunity_ancestor0	16.66%
ratio_cut	16.66%

Continuation	Percentage
cohesion	50%
ratio_cut	50%

Table 22 Best Pruning Experiment with Structural Features.

	Growth	Shrinkage	Continuation	Dissolution
Micro/Macro Precision	0.2343/0.6035	0.2047/0.5913	0.9293/0.8066	0.6512/0.8125
Micro/Macro Recall	0.3295/0.6431	0.1512/0.5679	0.9760/0.6939	0.2629/0.6289
Micro/Macro Fscore	0.2739/0.6227	0.1739/0.5794	0.9521/0.7460	0.3746/0.7090

Table 23 Best Pruning Experiment with Temporal Features.

	Growth	Shrinkage	Continuation
Micro/Macro Precision	0.1828/0.5730	0.1951/0.5778	0.9182/0.9222
Micro/Macro Recall	0.1828/0.5730	0.1633/0.5656	0.9955/0.6932
Micro/Macro Fscore	0.1828/0.5730	0.1778/0.5716	0.9553/0.7915

Extending Fairness Expressibility of ECTL$^+$: A Tree-Style One-Pass Tableau Approach

Alexander Bolotov[1]
University of Westminster,
W1W 6UW, London, UK
A.Bolotov@westminster.ac.uk

Montserrat Hermo[2]
University of the Basque Country
P. Manuel de Lardizabal, 1, 20018-San Sebastián, Spain
montserrat.hermo@ehu.eus

Paqui Lucio[3]
University of the Basque Country
P. Manuel de Lardizabal, 1, 20018-San Sebastián, Spain
paqui.lucio@ehu.eus

Abstract

Temporal logic has become essential for various areas in computer science, most notably for the specification and verification of hardware and software systems. For the specification purposes rich temporal languages are required that, in particular, can express fairness constraints. For linear-time logics which deal with fairness in the linear-time setting, one-pass and two-pass tableau methods have been developed. In the repository of the CTL-type branching-time setting, the well-known logics ECTL and ECTL$^+$ were developed to explicitly deal with fairness. However, due to the syntactical restrictions, these logics can only express restricted versions of fairness. The logic CTL*, often considered as "the full branching-time logic" overcomes these restrictions on expressing fairness. However, this logic itself, is extremely challenging for the application of verification techniques, and the tableau technique, in particular. For example, there is no one-pass tableau construction for this logic, while it is known that one-pass tableau has an additional benefit enabling the formulation of dual sequent calculi that are often treated as more "natural" being more friendly for human understanding. Based on these two considerations, the following problem arises - are there logics that have richer expressiveness than ECTL$^+$ yet "simpler" than CTL* for which a one-pass tableau can be developed? In this paper we give a solution to this problem. We present a tree-style one-pass tableau for a sub-logic of CTL* that we call ECTL$^\#$, which is more expressive than ECTL$^+$ allowing the formulation of a new range of fairness constraints with "until" operator. The presentation of the tableau construction is accompanied by an algorithm for constructing a systematic tableau, for any given input of admissible branching-time formulae. We prove the termination, soundness and completeness of the method. As tree-shaped one-pass tableaux are well suited for the automation and are amenable for the implementation and for the formulation of sequent calculi, our results also open a prospect of relevant developments of the automation and implementation of the tableau method for ECTL$^\#$, and of a dual sequent calculi.

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

[1] The author would like to thank the University of Westminster for supporting the sabbatical in 2017.
[2] This author has been partially supported by Spanish Projects TIN2013-46181-C2-2-R and TIN2017-86727-C2-2-R, and by the University of the Basque Country under Project LoRea GIU15/30.
[3] This author has been partially supported by Spanish Projects TIN2013-46181-C2-2-R and TIN2017-86727-C2-2-R, and by the University of the Basque Country under Project LoRea GIU15/30.

25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek; Article No. 5; pp. 5:1–5:22

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

Keywords and phrases Temporal logic, fairness, tableau, branching-time, one-pass tableau

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.5

Acknowledgements The authors are grateful to Jose Gaintzarain for his contribution.

1 Introduction

Temporal logic has become essential for the specification and verification of hardware and software systems. For the specification of the reactive and distributed systems, or, most recently, autonomous systems, the modelling of the possibilities "branching" into the future is essential. Branching-time logics (BTL) give us an appropriate framework. Among important properties of these systems, so called fairness properties are important. In the standard formalisation of fairness, operators $\Diamond$ (eventually) and $\Box$ (always) have been used: $\mathsf{A}\Diamond\Box p$ – "p" is true along all computation paths except possibly their finite initial interval, where "A" is "for all paths" quantifier, and $\mathsf{E}\Box\Diamond p$ – "p" is true along a computation path at infinitely many states, where "E" is "there exists a path" quantifier.

For the branching-time setting, the most used class of formalisms are "CTL" (Computation Tree Logic) type logics. CTL itself requires every temporal operator to be preceded by a path quantifier, thus, cannot express fairness. ECTL (Extended CTL) [5] enables simple fairness constraints but not their Boolean combinations. ECTL$^+$ [6] further extends the expressiveness of ECTL allowing Boolean combinations of temporal operators and ECTL fairness constraints (but not permitting their nesting). Both ECTL and ECTL$^+$ extend the expressiveness of CTL in tackling fairness, instead of changing the semantics of the logic, as Fair CTL did [3]. The logic CTL*, often considered as "the full branching-time logic" overcomes these restrictions on expressing fairness. However, this logic is extremely challenging for the application of any known technique of automated reasoning. From another perspective, the literature on fairness constraints, even in the linear-time setting, lacks the analysis of their formulation with the $\mathcal{U}$ ("until") operator. To the best of our knowledge, there are only a few research papers that raise or discuss the problem. Among them are [10], which introduces the logic LCTL, providing an extension of liveness constraints by the "until" operator. However, LCTL belongs to 'Fair CTL-type' logics [7]. "Generalised liveness assumptions, which allow to express that the conclusion $f_2 \mathcal{U} f_3$ of a liveness assumption $\Box(f_1 \Rightarrow (f_2 \mathcal{U} f_3))$ has to be satisfied" are addressed in [1]. The $\mathcal{U}$ operator in the formulation of the fairness can also be found in [15] which considers the sequential composition of processes, providing the following example - the composition of processes P_1 and P_2 "behaves as $P1$ until its termination and then behaves as $P2$". Finally, [11] utilises restricted linear-time fairness constraints with $\mathcal{U}$ in the linear-time setting. We are not aware of any other analysis of fairness constraints in branching-time setting using the $\mathcal{U}$ operator and without restricting the underlying logic to be interpreted over the "fair" paths. We bridge this gap, presenting the logic ECTL$^\#$ (we use $\#$ to indicate some restrictions on concatenations of the modalities and their Boolean combinations). It is weaker than CTL* but extends ECTL$^+$ by allowing the combinations $\Box(A \mathcal{U} B)$ or $A \mathcal{U} \Box B$, referred to as modalities $\Box\mathcal{U}$ and $\mathcal{U}\Box$. This enables the formulation of stronger fairness constraints in the branching-time setting. The fairness constraint $\mathsf{A}(p \mathcal{U} \Box q)$ reads as "q is true along all paths of the computation except possibly their finite initial interval, where p is true", for example,"if a user of a system cannot repeat any old password and needs to change passwords from time to time, then the following property should hold: $\mathsf{A}((password = pw) \mathcal{U} \Box(password \neq pw))$ provided that pw is the current password". Table 1 places our logic in the hierarchy of BTL representing their

Table 1 Classification of CTL-type logics and their expressiveness.

$\mathcal{B}(\mathcal{U},\circ)$ (CTL) extensions	$\mathsf{E}(\Box\Diamond q)$	$\mathsf{E}(\Box\Diamond q \wedge \Box\Diamond r)$	$\mathsf{A}((p\,\mathcal{U}\,\Box q) \vee (s\,\mathcal{U}\,\Box\neg r))$	$\mathsf{A}\Diamond(\circ p \wedge \mathsf{E}\circ\neg p)$
$\mathcal{B}(\mathcal{U},\circ,\Box\Diamond)$ (ECTL)	$\surd$	X	X	X
$\mathcal{B}^{+}(\mathcal{U},\circ,\Box\Diamond)$ (ECTL^{+})	$\surd$	$\surd$	X	X
$\mathcal{B}^{+}(\mathcal{U},\circ,\mathcal{U}\Box)$ ($\mathsf{ECTL}^{\#}$)	$\surd$	$\surd$	$\surd$	X
$\mathcal{B}^{\star}(\mathcal{U},\circ)$ ($\mathsf{CTL}^{\star}$)	$\surd$	$\surd$	$\surd$	$\surd$

expressiveness: logics are classified by 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 $\mathcal{B}^{\star}$ reflects "no restrictions" in either concatenations of the modalities or Boolean combinations between them.[4] Thus, $\mathcal{B}(\mathcal{U},\circ)$ denotes the logic CTL. In this hierarchy $\mathsf{ECTL}^{\#}$ is $\mathcal{B}^{+}(\mathcal{U},\circ,\mathcal{U}\Box)$.

We present a tree-style one-pass tableau for this logic continuing the analogous developments in linear-time case [2, 8] and for CTL [2]. An indicative feature of this approach is context-based tableau technique. To the best of our knowledge, the context-based tableau has not been extended to more expressive BTL, though for them different other kinds of tableaux exist. In particular, [13] presents a tableau based decision procedure for $\mathsf{CTL}^{\star}$, which would definitely cover $\mathsf{ECTL}^{\#}$ as a sublogic of $\mathsf{CTL}^{\star}$. However, this tableau method is (unavoidably) complicated. For example, it utilises "global conditions on infinite branches" to be checked by the automata-theoretic approach. Though such complications may be well justified by the complexity of $\mathsf{CTL}^{\star}$, aiming at a weaker logic, we would benefit by reducing the complications to the minimum. Also, a distinctive feature of the tableau method in [13] is the control of loops, specifically, of so called "bad loops". While it looks necessary for this technique, we would like to avoid similar complications for a simpler logic. Moreover, due to the essential use of the notion of "context" (see §3) our tableau rules only produce "good loops". Tree-style one-pass tableaux (without additional procedures for checking meta-logical properties) have dual (cut-free) sequent calculi, see [8], enabling the construction of human-understandable proofs. In addition, these tableaux are well suited for the automation and are amenable for the implementation.[5] Our tableau is effectively an AND-OR tree where nodes are labelled by sets of state (see the definitions in §2) formulae. The difficult cases of $\mathsf{ECTL}^{\#}$ formulae appear due to the enriched syntax: disjunctions of formulae in the scope of the A quantifier and conjunctions of formulae in the scope of the E quantifier. To tackle these cases, in addition to $\alpha - \beta$ rules, that are standard to the tableaux, we use novel β^{+}-rules which use the context to force the eventualities to be fulfilled as soon as possible.

Outline of the paper. In §2 we describe the syntax and semantics of $\mathsf{ECTL}^{\#}$. The formulation of the tableau method is given in §3, where we define and explain tableau rules. A *systematic* tableau construction and relevant examples are introduced in §4. The correctness of our tableau method, its soundness, refutational completeness, and termination, are sketched in §5. In §6 we draw the conclusions and prospects of future work that the presented results open. Finally, the Appendix in §A collects the proofs of the main Propositions, Lemmas and Theorems.

[4] This notation goes back to [4], here we use its nice tuning by Nicolas Markey in [12].
[5] An excellent survey of the seminal tableau techniques for temporal logics can be found in [9].

2 Syntax and Semantics of ECTL$^\#$

In the language of ECTL$^\#$ we utilise classical connectives $(\neg, \wedge, \vee)$ [6], classically defined constants $\mathbf{F}$ ("false") and $\mathbf{T}$ ("true"), linear-time temporal operators $\Box$ (always), $\circ$ (next time), and $\mathcal{U}$ (until), and path quantifiers - A (on all future paths) and E (on some future path). Similarly to other BTL, we distinguish *state* (σ) and *path* (π) formulae, such that well formed formulae are state formulae.

▶ **Definition 1** (Syntax of ECTL$^\#$)**.** Let Prop be a fixed set of propositions, and let Lit be the set of literals, $Lit ::= \mathbf{F} \mid \mathbf{T} \mid \rho \mid \neg\rho$, where $\rho \in \mathsf{Prop}$. We inductively define the set of ECTL$^\#$-formulae, $\mathcal{F}_{\mathsf{Prop}}$, over Prop as follows:
$\sigma ::= Lit \mid \sigma_1 \wedge \sigma_2 \mid \sigma_1 \vee \sigma_2 \mid \mathsf{A}\pi \mid \mathsf{E}\pi$
$\pi ::= \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \mid \circ\sigma \mid \Box(\sigma \vee \Box\sigma) \mid \sigma\,\mathcal{U}\,(\sigma \wedge \Diamond\sigma) \mid \sigma\,\mathcal{U}\,(\Box\sigma) \mid \Box(\sigma\,\mathcal{U}\,\sigma)$
where σ means a state formula, π means a path formula and $\Diamond\sigma$ abbreviates $\mathbf{T}\,\mathcal{U}\,\sigma$.

Note that $\sigma_1\,\mathcal{U}\,\sigma_2$ and $\Box\sigma$ abbreviate $\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\mathbf{T})$ and $\Box(\sigma \vee \Box\mathbf{F})$, respectively. Also, instead of using a minimal set of temporal operators, we use a richer syntax above to make the presentation of the tableau technique more transparent.

▶ **Definition 2** (Consistent Set of Formulae)**.** A set Σ of state formulae σ is *syntactically consistent* (we will write "consistent" further in the paper) abbreviated as $\Sigma_\top$ if $\mathbf{F} \notin \Sigma$ and $\{\sigma, \neg\sigma\} \not\subseteq \Sigma$ for any σ; otherwise, Σ is *inconsistent* abbreviated as $\Sigma_\bot$.

Formulae of ECTL$^\#$ are interpreted over labelled *Kripke structures*.

▶ **Definition 3** (Kripke structure $\mathcal{K} = (S, R, L)$)**.** A Kripke structure $\mathcal{K}$ is a triple of the form (S, R, L) such that S is a non-empty set of *states*, $R \subseteq S \times S$ is a total binary relation, called *the transition relation*, and $L : S \to 2^{\mathsf{Prop}}$ is a *labelling function*.

A *fullpath* x through a Kripke structure $\mathcal{K}$ is an infinite sequence of states $s_0, s_1, \ldots$ such that $(s_i, s_{i+1}) \in R$, for every $i \geq 0$. Given a path $x = s_0, s_1, \ldots, s_i, \ldots \quad (i \geq 0)$, we denote its state $s_i, 0 \leq i$, by $x(i)$ and its finite prefix by the sequence $x^{\leq i} = s_0, s_1, \ldots, s_i$. When path x is given, instead of $x(i)$ we will often write i, referring to i as "a state index of x". If x is a fullpath and y is a path such that $y(0) = x(i)$, for some $i > 0$, then the juxtaposition $x^{\leq i}y$ is a fullpath. 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 [4].

▶ **Definition 4** (Evaluation relation $\models$ for Kripke structures)**.** The relation $\mathcal{K}, x, i \models \varphi$ evaluates ECTL$^\#$ formula φ at the state index i of the given path x in the given structure $\mathcal{K} = (S, R, L)$ and is inductively defined as follows (we omit cases for Booleans, $\mathbf{F}$ and $\mathbf{T}$).

$\mathcal{K}, x, i \models \mathsf{A}\varphi$	iff	$\mathcal{K}, y, 0 \models \varphi$ holds for every path y such that $y(0) = x(i)$.
$\mathcal{K}, x, i \models \mathsf{E}\varphi$	iff	there exists a path y such that $y(0) = x(i)$ and $\mathcal{K}, y, 0 \models \varphi$.
$\mathcal{K}, x, i \models \circ\varphi$	iff	$\mathcal{K}, x, i+1 \models \varphi$.
$\mathcal{K}, x, i \models \Box\varphi$	iff	$\mathcal{K}, x, k \models \varphi$ holds for all $k \geq i$.
$\mathcal{K}, x, i \models \varphi_1\,\mathcal{U}\,\varphi_2$	iff	there exists $k \geq i$ such that $\mathcal{K}, x, k \models \varphi_2$ and $\mathcal{K}, x, j \models \varphi_1$ for all $j \in \{i, \ldots, k-1\}$.

[6] Since inputs to our tableau are supposed to be transformed into the negation normal form, see below.

Table 2 Difficult cases of temporal operators in the scope of path quantifiers.

Type of a difficult case	A-disjunctive formula	E-conjunctive formula
Example	$A(\circ q \vee \Box r)$	$E(\circ r \wedge q\,\mathcal{U}\,\Box\neg p)$
Our representation	$A(\circ q, \Box r)$	$E(\circ r, q\,\mathcal{U}\,\Box\neg p)$

We extend, in the standard way, the relation $\models$ to sets of formulae: given a set, Φ, of formulae, $\mathcal{K}, x, i \models \Phi$ iff $\mathcal{K}, x, i \models \varphi$, for all $\varphi \in \Phi$.

▶ **Definition 5** (Satisfiable Set of Formulae)**.** Given a set of formulae Φ, the set of its models, $\mathsf{Mod}(\Phi)$, is formed by all triples $(\mathcal{K}, x, i)$ such that $\mathcal{K}, x, i \models \Phi$. Then Φ is *satisfiable* ($\mathsf{Sat}(\Phi)$) if $\mathsf{Mod}(\Phi) \neq \emptyset$, otherwise Φ is *unsatisfiable* ($\mathsf{UnSat}(\Phi)$).

The sets Φ and Ψ are *equi-satisfiable* if the following holds: Φ is satisfiable iff Ψ is satisfiable. If $\mathsf{Mod}(\Phi) = \mathsf{Mod}(\Psi)$ then Φ and Ψ are equivalent denoted $\Phi \equiv \Psi$.

▶ **Definition 6** (Validity)**.** $\mathsf{ECTL}^{\#}$ formula σ is valid iff $\sigma \equiv \mathbf{T}$.

For a set of state formulae Σ, we sometimes write $\mathcal{K} \models \Sigma$ instead of $\mathcal{K}, x, 0 \models \Sigma$. For any $\mathcal{K}$, any $x \in \mathsf{fullpaths}(\mathcal{K})$ and any natural number i, the notation $\mathcal{K}\restriction x(i)$ denotes a Kripke structure with the set of states of $\mathcal{K}$ restricted to those that are R-reachable from $x(i)$. $\mathsf{CTL}^{\star}$, hence its sublogic $\mathsf{ECTL}^{\#}$ [14], has the *small model property*. Thus, we can consider *cyclic* $\mathsf{ECTL}^{\#}$-structures with fullpaths cyclic. Relevant concepts are defined below.

▶ **Definition 7** (Cyclic Sequence, Cyclic Path)**.** A finite sequence of states $z = s_0, s_1, \ldots, s_j$ is *cyclic* iff there exists s_i, $0 \leq i \leq j$ such that $(s_j, s_i) \in R$. The sequence $s_i, \ldots, s_j$ is a loop with a *cycling element* s_i. A *path* over z called *cyclic* is an infinite sequence where the loop $s_i, s_{i+1}, \ldots, s_j$ is repeated infinitely: $\mathsf{path}(z) = s_0, s_1, \ldots, s_{i-1}\langle s_i, s_{i+1}, \ldots, s_j\rangle^{\omega}$.

Cyclic paths do not have states after the "period", they are also called ultimately periodic.

▶ **Definition 8** (Cyclic Kripke structure)**.** A Kripke structure $\mathcal{K}$ is *cyclic* if every fullpath of $\mathcal{K}$ is a path over a cyclic sequence of states.

For $\mathsf{ECTL}^{\#}$, we identify the following difficult cases of the nesting and Boolean combinations of temporal operators in the scope of path quantifiers: A-disjunctive formula – disjunctions of temporal operators in the scope of A and E-conjunctive formula – conjunctions of temporal operators in the scope of E. For convenience, we will, respectively, write $\mathsf{A}(\pi_1, \ldots, \pi_n)$ and $\mathsf{E}(\pi_1, \ldots, \pi_n)$, where $n \geq 1$, and "," in the scope of A means $\vee$ and while in the scope of E it means $\wedge$. Examples given in Table 2 will be used to illustrate tableau, in Figure 2. Note that any A-formula (E-formula) σ can be transformed into an equivalent boolean combination of A-disjunctive formulae $\mathsf{A}(\pi_1, \ldots, \pi_n)$ (E-conjunctive formulae $\mathsf{E}(\pi_1, \ldots, \pi_n)$), such that every π_i $(1 \leq i \leq n)$ is of one of the following: $\circ\sigma$, $\sigma\,\mathcal{U}\,(\sigma \wedge \Diamond\sigma)$, $\sigma\,\mathcal{U}\,\Box\sigma$, $\Box(\sigma \vee \Box\sigma)$, and $\Box(\sigma\,\mathcal{U}\,\sigma)$, and σ stands for a state formula. For example, the formula $\mathsf{A}(((\circ q) \wedge (\Box\mathsf{E}\circ r)) \vee \circ p)$ is equivalent to $\mathsf{A}(\circ q, \circ p) \wedge \mathsf{A}(\Box\mathsf{E}\circ r, \circ p)$; and $\mathsf{E}(((\circ\mathsf{A}\circ r) \vee (q\,\mathcal{U}\,\Box\mathsf{E}\neg p)) \wedge \circ q)\}$ is equivalent to $\mathsf{E}(\circ\mathsf{A}\circ r) \vee \mathsf{E}(q\,\mathcal{U}\,\Box\mathsf{E}\neg p, \circ q)$. In what follows, Q abbreviates either of the path quantifiers. For a set of path formulae $\Pi = \{\pi_1, \ldots, \pi_n\}$, we write $\mathsf{Q}\Pi$ to denote the formula $\mathsf{Q}(\pi_1, \ldots, \pi_n)$, and $\mathsf{Q}\circ\Pi$ denotes $\mathsf{Q}(\circ\pi_1, \ldots, \circ\pi_n)$. An empty set of formulae Φ means $\mathbf{T}$ when Φ occurs in a conjunctive expression, while an empty Φ means $\mathbf{F}$ in a disjunctive expression. In particular, for empty Π, $\mathsf{A}\Pi = \mathbf{F}$ and $\mathsf{E}\Pi = \mathbf{T}$. We write Σ, σ to represent the set $\Sigma \cup \{\sigma\}$.

We assume that each $\sigma \in \mathcal{F}_{\mathsf{Prop}}$ is in its "negation normal form" called $\mathsf{nnf}(\sigma)$. The set of ECTL$^\#$-formulae is obviously closed under nnf: for any $\varphi \in \mathcal{F}_{\mathsf{Prop}}$, $\mathsf{nnf}(\neg\varphi) \in \mathcal{F}_{\mathsf{Prop}}$. Also, the negation of a state (path) formula is a state (path) formula. For example, $\mathsf{nnf}(\neg\mathsf{A}(p\,\mathcal{U}\,\Box q)) = \mathsf{E}((\Box\Diamond\neg q) \vee (\Diamond(\neg p \wedge \Diamond\neg q)))$. For simplicity, we will write $\neg\varphi$ instead of $\mathsf{nnf}(\neg\varphi)$, and for a finite set $\Delta = \{\varphi_1, \ldots, \varphi_n\}$, $\neg\Delta$ denotes the negation normal form of $\neg\bigwedge_{i=1}^{n}\varphi_i$.

3 The Tableau Method

3.1 Preliminaries

To make the subsequent sections more transparent we informally overview here the construction of the tableau. Recall that the initial set in a tableau is exclusively formed by state formulae (i.e. boolean combinations of literals and formulae of the form $\mathsf{Q}\Pi$).

▶ **Definition 9** (Tableau, Consistent Node, Closed branch)**.** A *tableau* for a set of state formulae Σ is a labelled tree T, where nodes are τ-labeled with sets of state formulae, such that the following two conditions hold:

(a) The root is labelled by the set Σ.

(b) Any other node m is labelled with sets of state formulae as the result of the application of one of the rules in Table 3, Table 4, Figure 1 and Table 5 to its parent node n. Given the applied rule is R, we term m an R-successor of n.

A node n of T is consistent, abbreviated as $n_\top$, if $\tau(n)$ is a consistent set of formulae (see Def. 2), else n is *inconsistent*, abbreviated as $n_\bot$. If for a branch b of T, there exists $n_\bot \in b$, then b is *closed* else b *open*.

A node of the tableau is labelled by a set of state formulae. To extend a node we apply one of the following three rules: α, β or β^+ rules. The first two types of rules are standard to the tableau, and are essentially based on the fixpoint characterisation of $\mathsf{Q}\Box$ and $\mathsf{Q}\,\mathcal{U}$ modalities, while β^+ rules are characteristic (and crucial!) for our construction. They tackle difficult cases of formulae in ECTL$^\#$, and are related to our dedicated account of the eventualities. Namely, we treat an eventuality as occurring in some *context*, which, in turn, is a collection of all state formulae (we will call this an outer context) or path formulae (we will call this an inner context). Subsequently, β^+ rules use the context to force eventualities to be fulfilled as soon as possible. $\alpha - \beta - \beta^+$ rules are applied to expand a node, generating its children labelled by the sets of state formulae. They apply repeatedly unless they produce an inconsistent node $n_\bot$, or we reach a node with the labels that already occurred within the path under consideration. In the former case the expansion of the given branch terminates with $n_\bot$ as a leaf. In the latter case, a repetitive node in a branch suggests that the formula under consideration is satisfied forever, so we change to select another eventuality (if any). Obviously, $n_\bot$ has an unsatifiable $\tau(n)$ and is a "deadlock" in the construction of a model. However, open branches do not necessarily give us a model. In particular, an open branch could be a prefix of a closed one. Later we introduce the notion of an expanded branch that enables the model construction. Once no more expansion rules are applicable to the given branch with the last node $n_\top$, the $\alpha - \beta$ rules ensure that $\tau(n) = \Sigma, \mathsf{A}\circ\Phi_1, \ldots, \mathsf{A}\circ\Phi_n, \mathsf{E}\circ\Psi_1, \ldots, \mathsf{E}\circ\Psi_m$ where Σ is a set of literals. As $\tau(n)$ only contains literals or formulae with the outer $\mathsf{Q}\circ$, this labelling is similar to a "pre-state" in the standard temporal tableau. Then the "next-state" rule applies to generate successors with the labels that are arguments of all $\mathsf{A}\circ$ modalities and the whole cycle of applying $\alpha - \beta - \beta^+$ and "next-state" rules is repeated until the tableau construction terminates. The nature of our rules ensures that the terminated tableau represents a model for the tableau input if

Table 3 ALPHA RULES. (σ_i, σ_j are state formulae, Π is a (possibly empty) set of path formulae.)

	α	S_α
$(\wedge)$	$\sigma_1 \wedge \sigma_2$	$\{\sigma_1, \sigma_2\}$
$(\mathsf{E}\sigma)$	$\mathsf{E}(\sigma_1, \ldots, \sigma_n, \Pi)$	$\{\sigma_1, \ldots, \sigma_n, \mathsf{E}\Pi\}$
$(\mathsf{E}\Box\mathcal{U})$	$\mathsf{E}(\Box(\sigma_1 \,\mathcal{U}\, \sigma_2), \Pi)$	$\{\mathsf{E}(\sigma_1 \,\mathcal{U}\, \sigma_2, \circ\Box(\sigma_1 \,\mathcal{U}\, \sigma_2), \Pi)\}$
$(\mathsf{A}\Box\mathcal{U})$	$\mathsf{A}(\Box(\sigma_1 \,\mathcal{U}\, \sigma_2), \Pi)$	$\{\mathsf{A}(\sigma_1 \,\mathcal{U}\, \sigma_2, \Pi), \mathsf{A}(\circ\Box(\sigma_1 \,\mathcal{U}\, \sigma_2), \Pi)\}$

Table 4 BETA RULES. (σ, σ_i are state formulae, Σ is a (possibly empty) set of state formulae, π_i is a path formula, Π is a (possibly empty) set of path formulae.)

β_Rule	β	k	$S_{\beta_i}(1 \leq i \leq k)$
$(\vee)$	$\sigma_1 \vee \sigma_2$	2	$S_{\beta_1} = \{\sigma_1\}$ $S_{\beta_2} = \{\sigma_2\}$
$(\mathsf{A}\sigma)$	$\mathsf{A}(\sigma_1, \ldots, \sigma_n, \Pi)$	$n+1$	$S_{\beta_1} = \{\sigma_1\}$ $\vdots$ $S_{\beta_n} = \{\sigma_n\}$ $S_{\beta_{n+1}} = \{\mathsf{A}\Pi\}$
$(\mathsf{E}\Box\sigma)$	$\mathsf{E}(\Box(\sigma_1 \vee \Box\sigma_2), \Pi)$	2	$S_{\beta_1} = \{\sigma_1, \mathsf{E}(\circ\Box(\sigma_1 \vee \Box\sigma_2), \Pi)\}$ $S_{\beta_2} = \{\neg\sigma_1, \sigma_2, \mathsf{E}(\circ\Box\sigma_2, \Pi)\}$
$(\mathsf{E}\mathcal{U}\,\sigma)$	$\mathsf{E}(\sigma_1 \,\mathcal{U}\, (\sigma_2 \wedge \Diamond\sigma_3), \Pi)$	2	$S_{\beta_1} = \{\sigma_2, \mathsf{E}(\Diamond\sigma_3, \Pi\}$ $S_{\beta_2} = \{\sigma_1, \mathsf{E}(\circ(\sigma_1 \,\mathcal{U}\, (\sigma_2 \wedge \Diamond\sigma_3)), \Pi)\}$
$(\mathsf{E}\mathcal{U}\,\Box)$	$\mathsf{E}(\sigma_1 \,\mathcal{U}\, \Box\sigma_2, \Pi)$	2	$S_{\beta_1} = \{\mathsf{E}(\Box\sigma_2, \Pi)\}$ $S_{\beta_2} = \{\sigma_1, \mathsf{E}(\circ(\sigma_1 \,\mathcal{U}\, \Box\sigma_2), \Pi)\}$
$(\mathsf{A}\Box\sigma)$	$\mathsf{A}(\Box(\sigma_1 \vee \Box\sigma_2), \Pi)$	3	$S_{\beta_1} = \{\sigma_1, \mathsf{A}(\circ\Box(\sigma_1 \vee \Box\sigma_2), \Pi)\}$ $S_{\beta_2} = \{\neg\sigma_1, \sigma_2, \mathsf{A}(\circ\Box\sigma_2, \Pi)\}$ $S_{\beta_3} = \{\mathsf{A}\Pi\}$
$(\mathsf{A}\mathcal{U}\,\sigma)$	$\mathsf{A}(\sigma_1 \,\mathcal{U}\, (\sigma_2 \wedge \Diamond\sigma_3), \Pi)$	3	$S_{\beta_1} = \{\sigma_2, \mathsf{A}(\Diamond\sigma_3, \Pi)\}$ $S_{\beta_2} = \{\sigma_1, \mathsf{A}(\circ(\sigma_1 \,\mathcal{U}\, (\sigma_2 \wedge \Diamond\sigma_3)), \Pi)\}$ $S_{\beta_3} = \{\mathsf{A}\Pi\}$
$(\mathsf{A}\mathcal{U}\,\Box)$	$\mathsf{A}(\sigma_1 \,\mathcal{U}\, \Box\sigma_2, \Pi)$	2	$S_{\beta_1} = \{\mathsf{A}(\Box\sigma_2, \Pi)\}$ $S_{\beta_2} = \{\sigma_1, \sigma_2, \mathsf{A}(\circ(\sigma_1 \,\mathcal{U}\, \Box\sigma_2), \Pi)\}$

all the leaves in a collection of branches, called a bunch, are consistent and all eventualities occurring in looping branches are fulfilled, otherwise, the tableau input is unsatisfiable. In addition to the tableau rules introduced in the rest of this Section, we also use simplifications rules that are given in the Appendix (Def. 42).

$$(\mathsf{Q}\bigcirc)\ \frac{\Sigma, \mathsf{A}\bigcirc\Phi_1, \ldots, \mathsf{A}\bigcirc\Phi_n, \mathsf{E}\bigcirc\Psi_1, \ldots, \mathsf{E}\bigcirc\Psi_m,}{\mathsf{A}\Phi_1, \ldots, \mathsf{A}\Phi_n, \mathsf{E}\Psi_1\ \&\ \ldots\ \&\ \mathsf{A}\Phi_1, \ldots, \mathsf{A}\Phi_n, \mathsf{E}\Psi_m}$$

Figure 1 Next-State Rule. (Set of literals Σ is possibly empty, $\Phi_i, \Psi_i \neq \emptyset$ are sets of formulae.)

3.2 Alpha and Beta Rules

The α- and β-rules are the most elementary rules. An application of an α-rule *enlarges* a branch with a node labelled by Σ, α, by a successor node labelled by Σ, S_α, where S_α is the set of formulae associated to α in Table 3. An α-rule has the following representation $\frac{\Sigma, \alpha}{\Sigma, S_\alpha}$. β-rules have the following representation $\frac{\Sigma, \beta}{\Sigma, S_{\beta_1} \mid \cdots \mid \Sigma, S_{\beta_k}}$. An application of a β-rule splits a branch containing a node with a set Σ, β, where β is one of the formulae of Table 4, in k new nodes each labelled by the corresponding Σ, S_{β_i}, see Table 4.

3.3 The Next-State Rule

The next-state rule ($\mathsf{Q}\bigcirc$) in Figure 1 is the only rule that splits branches in a *conjunctive way*: it produces m branches rooted by a node n labelled by a set $\mathsf{A}\Phi_1, \ldots, \mathsf{A}\Phi_n, \mathsf{E}\Psi_i$, for $i \in \{1, \ldots, m\}$. The generation of AND-successors of node n is represented by "&". If both n and m are zero, then the rule yields a unique new node labelled by the empty set. We assume that whenever m is zero and $n > 0$, there is a unique descendant labelled by $\mathsf{A}\Phi_1, \ldots, \mathsf{A}\Phi_n$.

▶ **Example 10.** Let n be a node such that $\tau(n) = \{a, \neg b, \mathsf{A}\bigcirc c, \mathsf{E}\bigcirc p, \mathsf{E}\bigcirc\neg p, \mathsf{A}\bigcirc\Box((\mathsf{E}\bigcirc p) \wedge (\mathsf{E}\bigcirc\neg p))\}$. Then only the next-state rule ($\mathsf{Q}\bigcirc$) can be applied to n generating the following AND-successors of n: $\{\mathsf{A}c, p, \mathsf{A}\Box((\mathsf{E}\bigcirc p) \wedge (\mathsf{E}\bigcirc\neg p))\}$ and $\{\mathsf{A}c, \neg p, \mathsf{A}\Box((\mathsf{E}\bigcirc p) \wedge (\mathsf{E}\bigcirc\neg p))\}$. Note that the formula $\mathsf{A}c$ requires the application of the β-rule ($\mathsf{A}\sigma$) to be reduced to c.

3.4 The Uniform Tableau

Here we present tableau with specific labels for leaves – *Uniform* sets of state formulae.

▶ **Definition 11** (Elementary Set of State Formulae)**.** A set of state formulae is *elementary* if and only if it is exclusively formed by literals and formulae of the form $Q\bigcirc\Pi$.

Repeatedly applying α-β-rules to consistent leafs, we get leaves labelled by elementary sets.

▶ **Proposition 12.** *Any set of state formulae has a tableau T such that all its leaves are labelled by elementary sets of state formulae.*

▶ **Definition 13** (Basic Path/State Formula, Uniform Set of Formulae)**.** Path formulae $\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)$, $\sigma_1\,\mathcal{U}\,(\Box\sigma_2)$, $\Box(\sigma_1 \vee \Box\sigma_2)$, $\Box(\sigma_1\,\mathcal{U}\,\sigma_2)$ are *basic*. If Π is a set of basic path formulae then $Q\Pi$ is *basic*. A set of state formulae Σ is *uniform set* (US) iff Σ is only formed by literals and basic state formulae, and Σ contains at most one E-conjunctive formula.

▶ **Proposition 14.** *Any set of state formulae Σ has a tableau T s.t. all leaves are labelled by US of state formulae; all open branches contain exactly one application of* ($\mathsf{Q}\bigcirc$).

▶ **Definition 15** (Uniform Tableaux)**.** For any set Σ of state formulae, the tableau for Σ provided by Proposition 14 is denoted Uniform_Tableau(Σ).

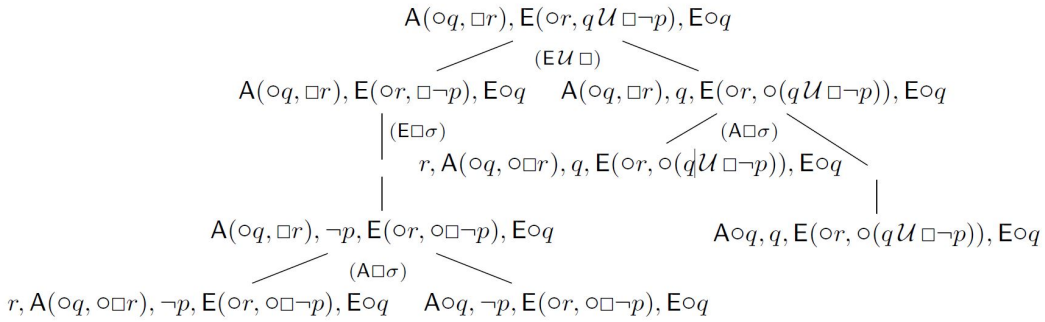


Figure 2 A tableau for $\mathsf{A}(\bigcirc q, \Box r), \mathsf{E}(\bigcirc r, q\,\mathcal{U}\,\Box\neg p)$; its leaves are elementary.

Table 5 Beta-Plus Rules. (Notation: σ, σ_i stand for state formulae, Σ is a (possibly empty) set of state formulae, Π is a (possibly empty) set of basic-path formulae.)

β^+-Rule	Σ, β	k	$S^+_{\Sigma,\beta_i}(1 \le i \le k)$
$(\mathsf{E}\,\mathcal{U}\,\sigma)^+$	$\Sigma, \mathsf{E}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$	2	$S^+_{\Sigma,\beta_1} = \{\sigma_2, \mathsf{E}(\Diamond\sigma_3, \Pi)\}$ $S^+_{\Sigma,\beta_2} = \{\sigma_1, \mathsf{E}(\bigcirc((\sigma_1 \wedge \neg\Sigma)\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)), \Pi)\}$
$(\mathsf{E}\,\mathcal{U}\,\Box)^+$	$\Sigma, \mathsf{E}(\sigma_1\,\mathcal{U}\,\Box\sigma_2, \Pi)$	2	$S^+_{\Sigma,\beta_1} = \{\mathsf{E}(\Box\sigma_2, \Pi)\}$ $S^+_{\Sigma,\beta_2} = \{\sigma_1, \mathsf{E}(\bigcirc((\sigma_1 \wedge \neg\Sigma)\,\mathcal{U}\,\Box\sigma_2), \Pi)\}$
$(\mathsf{A}\,\mathcal{U}\,\sigma)^+$	$\Sigma, \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$	3	$S^+_{\Sigma,\beta_1} = \{\sigma_2, \mathsf{A}(\Diamond\sigma_3, \Pi)\}$ $S^+_{\Sigma,\beta_2} = \{\sigma_1, \mathsf{A}(\bigcirc((\sigma_1 \wedge (\neg\Sigma \vee \varphi_\Pi))\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)), \Pi)\}$ $S^+_{\Sigma,\beta_3} = \{\mathsf{A}\Pi\}$
$(\mathsf{A}\,\mathcal{U}\,\Box)^+$	$\Sigma, \mathsf{A}(\sigma_1\,\mathcal{U}\,\Box\sigma_2, \Pi)$	2	$S^+_{\Sigma,\beta_1} = \{\mathsf{A}(\Box\sigma_2, \Pi)\}$ $S^+_{\Sigma,\beta_2} = \{\sigma_1, \sigma_2, \mathsf{A}(\bigcirc((\sigma_1 \wedge \neg\Sigma \vee \varphi_\Pi \vee \sigma_2))\,\mathcal{U}\,\Box\sigma_2), \Pi)\}$

▶ **Example 16.** We construct a uniform tableau for the set $\{\mathsf{A}(\bigcirc q, \Box r), \mathsf{E}(\bigcirc r, q\,\mathcal{U}\,\Box\neg p), \mathsf{E}\bigcirc q)\}$. The first step gives the tableau in Figure 2. Subsequently, we enlarge each of the four branches by applying the rule $(\mathsf{Q}\bigcirc)$, obtaining the following eight leaves (left to right):

1. $\mathsf{A}(q, \Box r), \mathsf{E}(r, \Box\neg p)$
2. $\mathsf{A}(q, \Box r), \mathsf{E}q$
3. $\mathsf{A}q, \mathsf{E}(r, \Box\neg p)$
4. $\mathsf{A}q, \mathsf{E}q$
5. $\mathsf{A}(q, \Box r), \mathsf{E}(r, q\,\mathcal{U}\,\Box\neg p)$
6. $\mathsf{A}(q, \Box r), \mathsf{E}q$
7. $\mathsf{A}q, \mathsf{E}(r, q\,\mathcal{U}\,\Box\neg p)$
8. $\mathsf{A}q, \mathsf{E}q$

Then by rules $(\mathsf{A}\sigma)$ and $(\mathsf{E}\sigma)$: the first branch is split into $q, r, \mathsf{E}\Box\neg p$ and $\mathsf{A}\Box r, r, \mathsf{E}\Box\neg p$; the second into q and $\mathsf{A}\Box r, q$; the third yields only a child $q, r, \mathsf{E}\Box\neg p$; the fourth and the eighth yield only q; the fifth is split into two nodes $q, r, \mathsf{E}(q\,\mathcal{U}\,\Box\neg p)$ and $\mathsf{A}\Box r, r, \mathsf{E}(q\,\mathcal{U}\,\Box\neg p)$; the sixth into q and $\mathsf{A}\Box r, q$; and the seventh yields the unique child $q, r, \mathsf{E}(q\,\mathcal{U}\,\Box\neg p)$.

3.5 The Beta-plus Rules

We extend tableau rules with so called β^+-rules (Table 5), that, similarly to β-rules, split a branch into two or three branches. They apply to a set Σ, β, where β has the form $\mathsf{Q}(\pi, \Pi)$ and Π is a set of basic-path formulae and π is an $\mathcal{U}$-formula. The β^+ rules are the only rules that make use of the so-called context for forcing the eventualities to be satisfied as soon as possible. The context is given by the sets Σ containing state formulae and Π containing path formulae. We name Σ the outer context and Π the inner context. The outer context is used by all the β^+rules. The inner context is only needed to deal with formulae $\mathsf{A}\Pi$. Note that each β^+-rule, when applied to a formula of the form $\mathsf{Q}(\sigma_1\,\mathcal{U}\,\varphi, \Pi)$ –where φ could

be $\sigma_2 \wedge \Diamond\sigma_3$ or $\Box\sigma_2$– generates one or more successors that contain a formula of the form $\mathsf{Q}(\circ((\sigma_1 \wedge \sigma)\,\mathcal{U}\,\varphi), \Pi)$ where σ depends on both the inner and the outer context, and is defined based on whether Q is E or A. We call $(\sigma_1 \wedge \sigma)\,\mathcal{U}\,\varphi$ the *next-step variant* of $\sigma\,\mathcal{U}\,\varphi$.

▶ **Definition 17** (Formula φ_Π for β^+-rules)**.** Let Π be a set of basic path formulae. We define the formula φ_Π to be the following disjunction of state formulae:[7]

$$\bigvee_{\Box(\sigma_1 \vee \Box\sigma_2) \in \Pi} (\sigma_1 \vee \sigma_2) \;\vee\; \bigvee_{\sigma_1\,\mathcal{U}\,\Box\sigma_2 \in \Pi} \sigma_2 \;\vee\; \bigvee_{\Box(\sigma_1\,\mathcal{U}\,\sigma_2) \in \Pi} \mathsf{E}(\Diamond\sigma_2).$$

The proof of Proposition 29 and the role that this proposition plays in the proof of Lemma 30 point out some hints about the relation of the formula φ_Π (in Definition 17) with a limit path, which is a model for the formula $\mathsf{E}(\neg\pi_1, \ldots, \neg\pi_n)$, but is not a model for $\mathsf{A}(\pi_1, \ldots, \pi_n)$. The following example aims to provide some intuition on the role of φ_Π from the constructive view, i.e when we construct a tableau for a formula $\mathsf{A}(\pi_1, \ldots, \pi_n)$.

▶ **Example 18.** Consider the application of $(\mathsf{A}\,\mathcal{U}\,\sigma)^+$ rule to the formula $\mathsf{A}(a\,\mathcal{U}\,b, \Pi)$, where $\Pi = \{\Box c, r\,\mathcal{U}\,\Box s, \Box(p\,\mathcal{U}\,q)\}$ and $a, b, c, p, q, r, s \in \mathsf{Prop}$. The outer context, Σ, is empty and the inner context is Π. Then $\neg\Sigma = \mathbf{F}$ and $\varphi_\Pi = c \vee s \vee \mathsf{E}\Diamond q$. Hence, the second child, namely $S^+_{\beta_2}$, raised by the application of $(\mathsf{A}\,\mathcal{U}\,\sigma)^+$ is labelled by $\{a, \mathsf{A}(\circ((a \wedge (c \vee s \vee \mathsf{E}\Diamond q))\,\mathcal{U}\,b), \Pi)\}$. Then applying the rules $(\mathsf{A}\Box\sigma)$ to $\Box c$, $(\mathsf{A}\,\mathcal{U}\,\Box)$ to $r\,\mathcal{U}\,\Box s$, and $(\mathsf{A}\Box\,\mathcal{U}\,)$ (followed by $(\mathsf{A}\,\mathcal{U}\,\sigma)$) to $\Box(p\,\mathcal{U}\,q)$ we get, in one of the branches, a node n labelled by the set:

$$\{a, c, r, s, p, \mathsf{A}(\circ((a \wedge (c \vee s \vee \mathsf{E}\Diamond q))\,\mathcal{U}\,b), \circ\Box c, \circ(r\,\mathcal{U}\,\Box s), \circ\Box(p\,\mathcal{U}\,q))\} \tag{1}$$

Then, by rule $(\mathsf{Q}\circ)$, $\tau(n_1) = \{\mathsf{A}((a \wedge (c \vee s \vee \mathsf{E}\Diamond q))\,\mathcal{U}\,b, \Box c, r\,\mathcal{U}\,\Box s, \Box(p\,\mathcal{U}\,q))\}$, where n_1 is the unique child of n. Now, repeating the previous steps, we get a node m labelled by

$$\{a \wedge (c \vee s \vee \mathsf{E}\Diamond q), c, r, s, p, \mathsf{A}(\circ((a \wedge (c \vee s \vee \mathsf{E}\Diamond q))\,\mathcal{U}\,b), \circ\Box c, \circ(r\,\mathcal{U}\,\Box s), \circ\Box(p\,\mathcal{U}\,q))\}$$

Using the rules $(\wedge)$ and $(\vee)$, we get three children of m. Two of them are labelled by the set (1). Hence, by rule $(\mathsf{Q}\circ)$, we get a cycle to node n_1. It is worth noting that this branch represents a model where the initial A-disjunctive formula $\mathsf{A}(a\,\mathcal{U}\,b, \Box c, r\,\mathcal{U}\,\Box s, \Box(p\,\mathcal{U}\,q))$ is satisfied because both $\Box c$ and $r\,\mathcal{U}\,\Box s$ are satisfied. The third node is labelled by

$$\{a, \mathsf{E}\Diamond q, c, r, s, p, \mathsf{A}(\circ((a \wedge (c \vee s \vee \mathsf{E}\Diamond q))\,\mathcal{U}\,b), \circ\Box c, \circ(r\,\mathcal{U}\,\Box s), \circ\Box(p\,\mathcal{U}\,q))\}$$

Therefore, one of its children, due to the rule $(\mathsf{E}\,\mathcal{U}\,\sigma)$, is labelled by

$$\{a, q, c, r, s, p, \mathsf{A}(\circ((a \wedge (c \vee s \vee \mathsf{E}\Diamond q))\,\mathcal{U}\,b), \circ\Box c, \circ(r\,\mathcal{U}\,\Box s), \circ\Box(p\,\mathcal{U}\,q))\}$$

and after applying $(\mathsf{Q}\circ)$ we get a node labelled as n_1 is, obtaining a cycle. This cycling branch represents a model for the initial A-disjunctive formula $\mathsf{A}(a\,\mathcal{U}\,b, \Box c, r\,\mathcal{U}\,\Box s, \Box(p\,\mathcal{U}\,q))$ because $\Box(p\,\mathcal{U}\,q)$ is satisfied. Here, for simplicity, we consider an empty outer context. However, note that $\neg\Sigma$ is also a disjunction of state formulae. For non-empty Σ, the application of the rule $(\vee)$ would generate a child for each disjunct in $\neg\Sigma$. Each of these children represents a trial to satisfy the formula $a\,\mathcal{U}\,b$ (in the initial A-disjunctive formula) as soon as possible.

[7] If the disjunction is empty, then $\varphi_\Pi = \mathbf{F}$.

Algorithm 1 Systematic Tableau Construction.

procedure SYSTEMATIC_TABLEAU(Σ_0) ▷ where Σ_0: set of state formulae
  if Σ_0 is not uniform **then** $T :=$ Uniform_Tableau(Σ_0)
  while T has at least one non-terminal leaf **do**
    ▷ Invariant: Any non-terminal leaf of T is labelled by a US
    Choose any leaf ℓ in T such that $\tau(\ell)$ is not terminal
    Let $\Sigma = \tau(\ell)$ ▷ Σ is uniform
    if there are not eventualities in ℓ **then** $T := T[\ell \leftarrow$Uniform_Tableau(Σ)$]$
    else
      Eventuality_Selection(Σ)
      Apply_β^+-rule(Σ)
      Let $k \in \{2, 3\}$ the number of new leaves
      Let $\ell_1, \ldots, \ell_k$ be the new leaves and let $\Sigma_1, \ldots, \Sigma_k$ be their respective labels
      for i = 1 .. k **do**
        if ℓ_i is non-terminal and Σ_i is not uniform **then**
          $T := T[\ell_i \leftarrow$Uniform_Tableau(Σ_i)$]$
  return T

4 Systematic Tableau Construction

Here we define an algorithm, $\mathcal{A}^{sys}$, that constructs a *systematic tableau* and illustrate its performance. Recall that due to the rule (Q○), any open tableau should have a collection of open branches including all the (Q○)-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 $\mathcal{A}^{sys}$, gives us of a model for the initial set of formulae. $\mathcal{A}^{sys}$ constructs an *expanded* tableau (see Definition 34) for the given input. $\mathcal{A}^{sys}$ applied to the input Σ_0, denoted as $\mathcal{A}^{sys}(\Sigma_0)$, returns a systematic tableau $\mathcal{A}^{sys}_{\Sigma_0}$. 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 the sake of clarity we prefer to formulate it as a breadth-first construction of a collection of subtrees. The procedure Uniform_Tableau in Algorithm 1 was introduced in Definition 15 along with the notion of a US of state formulae. The notation $T_1[\ell \leftarrow T_2]$ stands for the tableau T_1 where the leaf ℓ is substituted by the tableau T_2. In particular, $T[\ell \leftarrow$Uniform_Tableau(Σ)$]$ is the tableau T where the leaf ℓ is substituted by the Uniform_Tableau(Σ).

To introduce the procedures Eventuality_Selection and Apply_β^+-rule and related concepts of *terminal* node and *eventuality-covered* branch, let $\pi_{\mathcal{U}}$ denote a basic path formula that contains the operator $\mathcal{U}$, i.e. $\pi_{\mathcal{U}}$ is either $\sigma_1 \mathcal{U} (\sigma_2 \wedge \Diamond\sigma_3)$ or $\sigma_1 \mathcal{U} \Box\sigma_2$ or $\Box(\sigma_1 \mathcal{U} \sigma_2)$. We call these formulae *eventualities*. Consequently, the notation $\mathsf{Q}(\pi_{\mathcal{U}}, \Pi)$ stands for a formula that contains at least one eventuality.

Eventuality_Selection selects a state formula $\mathsf{Q}(\pi_{\mathcal{U}}, \Pi)$ (if there is one) and marks the eventuality $\pi_{\mathcal{U}}$ while Apply_β^+-rule(Σ) applies the corresponding rule (or pair of rules) depending on the selected eventuality:

- If $\pi_{\mathcal{U}} = \sigma_1 \mathcal{U} (\sigma_2 \wedge \Diamond\sigma_3)$ is the marked eventuality, then apply $(\mathsf{Q}\mathcal{U}\sigma)^+$
- If $\pi_{\mathcal{U}} = \sigma_1 \mathcal{U} \Box\sigma_2$ is the marked eventuality, then apply the respective rule $(\mathsf{Q}\mathcal{U}\Box)^+$
- If $\pi_{\mathcal{U}} = \Box(\sigma_1 \mathcal{U} \sigma_2)$ is the marked eventuality, then apply first the rule $(\mathsf{Q}\Box\mathcal{U})$ and then the rule $(\mathsf{Q}\mathcal{U}\sigma)^+$ with the selected eventuality $\sigma_1 \mathcal{U} \sigma_2$.

Each application of a β^+-rule introduces a next-step variant of the marked eventuality.

The call Eventuality_Selection(Σ) keeps the selection of $\mathsf{Q}\Pi \in \Sigma$ which contains a next-step variant of the previously marked eventuality, whenever the leaf $\ell, (\Sigma = \tau(\ell))$ is not a loop-node. If ℓ is a loop-node, then a new selection should be made, if possible. If the branch is already eventuality-covered and ℓ is a loop-node, ℓ is the leaf of an expanded open branch (see Definition 34). When making the selection, priorities are guided by Definition 19 - the idea is that the tableau branches represent paths in possible models. Since any path in a model is cyclic, it has a possibly empty initial sequence of states followed by a looping-sequence. The highest priority formulae are those that cannot produce a loop. The formulae that are not of highest priority can be potentially-cycling formulae or cycling. Once all the highest priority formulae have been selected, we only have cycling and potentially-cycling formulae. Now the objective is to have a sequence of loop-nodes.

▶ **Definition 19** (Priorities for Eventuality Selection)**.** The formulae of the *highest priority* for Eventuality_Selection are the formulae of the form:

- $\mathsf{A}\Pi$ where Π is exclusively formed by formulae of the form $\sigma_1 \,\mathcal{U}\, (\sigma_2 \wedge \Diamond\sigma_3)$, and
- $\mathsf{E}\Pi$ where Π contains at least one eventuality. Eventualities of the form $\sigma_1 \,\mathcal{U}\, (\sigma_2 \wedge \Diamond\sigma_3)$ and $\sigma_1 \,\mathcal{U}\, \Box\sigma_2$ have also the maximal priority to be marked in the selected $\mathsf{E}\Pi$.

▶ **Definition 20** (Cycling Formula)**.** A formula is *cycling* if it is of the form $\mathsf{Q}\Pi$ where Π only contains formulae of the form: $\Box(\sigma_1 \vee \Box\sigma_2)$ and $\Box(\sigma_1 \,\mathcal{U}\, \sigma_2)$.

For any E-conjunctive formula σ, σ is not highest priority if and only if σ is a cycling formula. For A-disjunctive formulae it is different - we also have potentially-cycling formulae.

▶ **Definition 21** (Potentially-Cycling Formula)**.** A formula is *potentially-cycling* if it is of the form $A\Pi$ where Π contains at least one formula of the form $\sigma_1 \,\mathcal{U}\, (\sigma_2 \wedge \Diamond\sigma_3)$ or $\sigma_1 \,\mathcal{U}\, \Box\sigma_2$ and also contains at least one formula of the form $\Box(\sigma_1 \vee \sigma_2)$ or $\Box(\sigma_1 \,\mathcal{U}\, \sigma_2)$.

▶ **Definition 22** (Loop-node)**.** Given b is a branch of T and $n_i \in b$ $(0 \leq i)$, n_i is a *loop-node* if there exists $n_j \in b$ $(0 \leq j < i)$ and $\tau(n_i) = \tau(n_j)$; n_j is called a *companion node* of n_i.

▶ **Definition 23** (Eventuality-covered Branch)**.** A branch $b = n_0, n_1, ..., n_i$ of T is *eventuality-covered* if n_i is a loop-node, with a companion node n_j $(0 \leq j < i)$, both labelled by a US $\Sigma = \mathsf{Q}_1\Pi_1, \ldots, \mathsf{Q}_m\Pi_m$ such that for each $h \in \{1, \ldots, m\}$:

- If $\mathsf{Q}_h = \mathsf{E}$, then every eventuality in Π_h has been marked in some node n_k for some k such that $j \leq k < i$.
- If $\mathsf{Q}_h = \mathsf{A}$ and $\mathsf{A}\Pi_h$ contains at least one eventuality, then $\mathsf{A}\Pi_h$ has been selected once in some node n_k such that $j \leq k < i$ and one of the eventualities in Π_h has been marked.

The procedure Eventuality_Selection performs in some fair way that ensures that any open branch will ever be *eventuality-covered*.

▶ **Definition 24** (Terminal Node)**.** A node n is terminal, if $\tau(n) = \Sigma_\perp$ or n is a loop-node of the branch b and b is eventuality-covered. Otherwise, we say that n is non-terminal.

Consequently, a non-terminal node is either a node that is not a loop-node or a loop-node whose branch is not eventuality-covered. A potentially-cycling formula could be selected more than once in a branch because the loop-node could change along the branch. In fact, the loop-node decreases in a well-founded order. The following example illustrates this issue.

▶ **Example 25.** Let $\sigma_1 = \mathsf{A}(\mathbf{T} \,\mathcal{U}\, (\neg c), \Box a)$, $\sigma_2 = \mathsf{A}(a \,\mathcal{U}\, \Box b)$, and $\sigma_3 = \mathsf{A}\Box c$, where $a, b, c \in \mathsf{Prop}$. Let $\Sigma_0 = \{\sigma_1, \sigma_2, \sigma_3\}$ be the initial set of state formulae. Let σ_1 be selected and $\mathbf{T} \,\mathcal{U}\, (\neg c)$ is the first marked eventuality. Then, $(\mathsf{A} \,\mathcal{U}\, \sigma)^+$ is applied to σ_1. Now in the node that

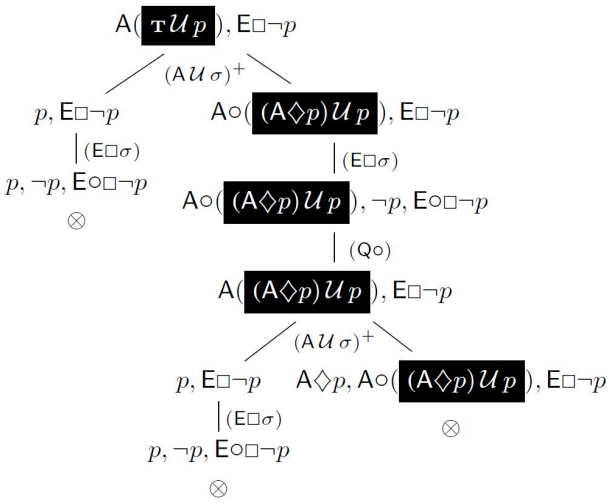


Figure 3 A closed tableau for $\mathsf{A}(\top \mathcal{U}\, p), \mathsf{E}\Box\neg p$. Marked eventualities are in black boxes, big circles represent AND-nodes or bunches. Whenever a bunch has a unique successor, we omit the big circle in the edge before the $(\mathsf{Q}\bigcirc)$-successor.

contains the next-step variant of σ_1, $(\mathsf{A}\Box\sigma)$ is applied to the next-step variant of σ_1; $(\mathsf{A}\,\mathcal{U}\,\Box)$ is applied to σ_2; and $(\mathsf{A}\Box\sigma)$ to σ_3. Then one of the open tableau branches is labelled by the elementary set: $\{a, c, \mathsf{A}(\bigcirc(((\neg\sigma_2)\vee(\neg\sigma_3)\vee a)\,\mathcal{U}\,(\neg c)), \bigcirc\Box a), \mathsf{A}\bigcirc(a\,\mathcal{U}\,\Box b), \mathsf{A}\bigcirc\Box c\}$. By rule $(\mathsf{Q}\bigcirc)$, we get a node n_1 labelled by the US $\tau(n_1) = \{\sigma_1', \sigma_2, \sigma_3\}$ where σ_1' is the selected formula: $\mathsf{A}(((\neg\sigma_2)\vee(\neg\sigma_3)\vee a)\,\mathcal{U}\,(\neg c), \Box a)$ and the eventuality $((\neg\sigma_2)\vee(\neg\sigma_3)\vee a)\,\mathcal{U}\,(\neg c)$ is kept marked. Hence, in one of the branches that enlarges n_1, the same sequence of rules produces a loop-node n_2 such that $\tau(n_2) = \tau(n_1)$. However, this branch is not eventuality-covered since σ_2 has not been selected yet. Then the selected formula in n_2 must be σ_2, and $(\mathsf{A}\,\mathcal{U}\,\Box)^+$ is applied to σ_2. Now, one of the open branches gets a node n_3 such that $\tau(n_3) = \{\sigma_1', \sigma_2', \sigma_3\}$ where $\sigma_2' = \mathsf{A}\Box b$. Then σ_1' is selected again. Note that the outer context of σ_1' has changed because σ_2 has been replaced by σ_2'. Hence, applying $(\mathsf{A}\,\mathcal{U}\,\sigma)^+$ to σ_1' and $(\mathsf{A}\Box\sigma)$ to σ_2' and σ_3, the leaf of one of the open branches is a node n_4 labelled by the US $\tau(n_4) = \{\sigma_1'', \sigma_2', \sigma_3\}$ where σ_1'' is the selected formula: $\mathsf{A}((((\neg\sigma_2)\wedge(\neg\sigma_2'))\vee(\neg\sigma_3)\vee a)\,\mathcal{U}\,(\neg c), \Box a)$. Note that $((\neg\sigma_2)\wedge(\neg\sigma_2'))\vee(\neg\sigma_3)\vee a$ is the conjunctive normal form of $((\neg\sigma_2)\vee(\neg\sigma_3)\vee a)\wedge((\neg\sigma_2')\vee(\neg\sigma_3)\vee a)$. Since, σ_1'' is kept selected, we will get a loop-node n_5 such that $\tau(n_5) = \tau(n_4)$ and now the branch is eventuality-covered. Indeed, this branch represents the following model of Σ_0: $\{a, c\}\langle\{a, b, c\}\rangle^\omega$.

▶ **Definition 26** (Bunch in a Tableaux, Closed Bunch and Tableaux)**.** A *bunch* b is a collection of branches that is maximal with respect to $(\mathsf{Q}\bigcirc)$-successor, i.e. every $(\mathsf{Q}\bigcirc)$-successor of any node in b are also in b. A bunch is closed 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.

Any open tableau has at least one open bunch, formed by one or more open branches.

▶ **Example 27.** (Figure 3) In the applications of $(\mathsf{A}\,\mathcal{U}\,\sigma)^+$ rule, the inner context is empty; the outer context is $\mathsf{E}\Box\neg p$, its negation in nnf is $\mathsf{A}\Diamond p$. Hence, the label of the rightmost leaf, $\mathsf{A}\bigcirc((\mathsf{A}\Diamond p)\,\mathcal{U}\,p)$ is the simplification of the selected formula $\mathsf{A}\bigcirc(((\mathsf{A}\Diamond p)\wedge(\mathsf{A}\Diamond p))\,\mathcal{U}\,p)$.

▶ **Example 28.** On the left of Figure 4 we depict a representative open bunch of a tableau for the set of formulae: $p, \mathsf{A}\Box(\mathsf{E}\bigcirc p\wedge\mathsf{E}\bigcirc\neg p), \mathsf{A}(\Diamond\neg p, \Box p)$. We apply at once the Uniform_Tableau

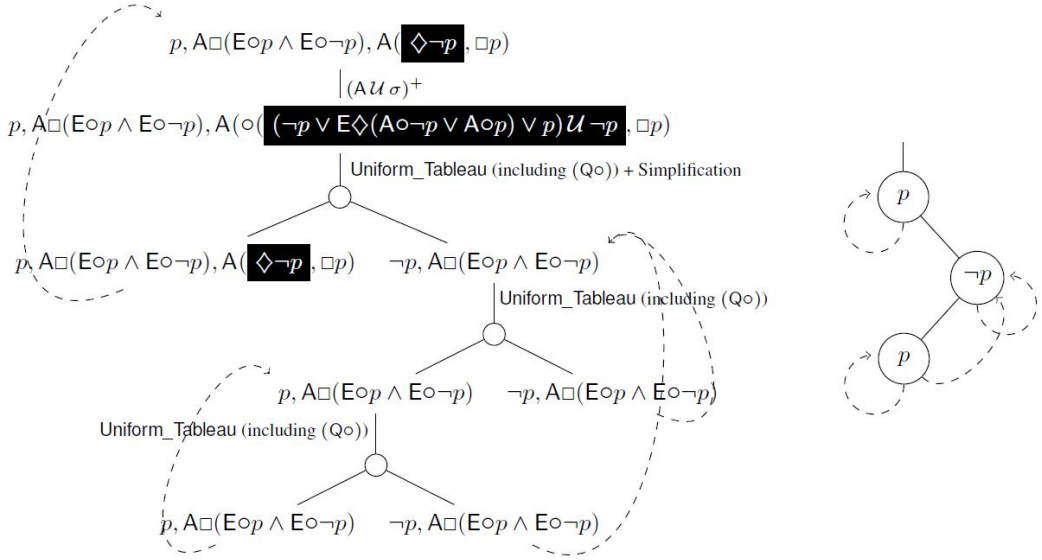


Figure 4 Open bunch in the tableau for $p, \mathsf{A}\Box(\mathsf{E}\bigcirc p \wedge \mathsf{E}\bigcirc\neg p), \mathsf{A}(\Diamond\neg p, \Box p)$ and represented model.

procedure subsequently choosing one of the leaves produced. For each node, we draw only one of the OR-children, but all the AND-children. In the marked eventuality, $\neg p \vee \mathsf{E}\Diamond(\mathsf{A}\bigcirc\neg p)$ comes from the negation of the outer context, and the disjunct p from the inner context. By "Simplification" $\neg p \vee \mathsf{E}\Diamond(\mathsf{A}\bigcirc\neg p \vee \mathsf{A}\bigcirc p) \vee p$ is reduced to $\mathbf{T}$ (in the left-hand child). In the right-hand node, $\neg p$ subsumes $\mathsf{A}((\ldots)\,\mathcal{U}\,\neg p, \Box p)$. This open bunch represents a model (of the input set of formulae) that we depict on the right of Figure 4.

5 Correctness: Soundness, Completeness and Termination

The soundness of our tableau method (Theorem 31) is proved on the basis that tableau rule preserve satisfiability (Lemma 30). For the latter it is essential to prove that the satisfaction of the negated inner context is preserved from segments of a limit path to the limit path itself (Proposition 29). The use of the formula φ_Π (Definition 17) is crucial for that.

▶ **Proposition 29** (Preservation of the Negated Inner Context). *Let Π be any set of basic path formulae and let φ_Π be as in Definition 17. Let $y = x_1^{\leq i_1} x_2^{\leq i_2} \cdots x_k^{\leq i_k} \cdots$ be a limit path in* $\mathsf{fullpaths}(\mathcal{K})$ *(of some Kripke structure $\mathcal{K}$). Then $\mathcal{K}, y \models \neg\pi$ holds for all $\pi \in \Pi$, provided that the following two conditions hold for all $n \geq 1$:*
(a) *$\mathcal{K}, x_1^{\leq i_1} x_2^{\leq i_2} \cdots x_n, j \models \neg\sigma_2$ for all $\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3) \in \Pi$ and all $j \in \{0..i_n\}$, and*
(b) *$\mathcal{K}, x_1^{\leq i_1} x_2^{\leq i_2} \cdots x_n^{\leq i_n}, i_n \models \neg\varphi_\Pi$.*

▶ **Lemma 30** (Soundness of the Tableau Rules). *For any set of state formulae Σ:*
(i) *For any α-formula α :* $\mathsf{Sat}(\Sigma, \alpha)$ *iff* $\mathsf{Sat}(\Sigma, S_\alpha)$.
(ii) *For any β-formula β of range k:* $\mathsf{Sat}(\Sigma, \beta)$ *iff* $\mathsf{Sat}(\Sigma, S_{\beta_i})$ *for some $1 \leq i \leq k$.*
(iii) *For any β^+-formula β of range k:* $\mathsf{Sat}(\Sigma, \beta)$ *iff* $\mathsf{Sat}(\Sigma, S^+_{\Sigma,\beta_i})$ *for some $1 \leq i \leq k$.*
(iv) *If Σ is a set of literals:* $\mathsf{Sat}(\Sigma, \mathsf{A}\bigcirc\Phi_1, \ldots, \mathsf{A}\bigcirc\Phi_n, \mathsf{E}\bigcirc\Psi_1, \ldots, \mathsf{E}\bigcirc\Psi_m)$ *iff for all $0 \leq i \leq m$:* $\mathsf{Sat}(\mathsf{A}\Phi_1, \ldots, \mathsf{A}\Phi_n, \mathsf{E}\Psi_i)$.

▶ **Theorem 31** (Soundness of the Tableau Method). *Given any set of state formulae Σ, if there exists a closed tableau for Σ then* $\mathsf{UnSat}(\Sigma)$.

In the rest of this section, we sketch the main stages of the proof of completeness of the presented tableaux method. Detailed proofs can be found in the technical report at `http://www.sc.ehu.es/jiwlucap/TechReport18.pdf` while relevant proofs of the most important Propositions, Lemmas and Theorems are given in the Appendix.

▶ **Definition 32** (Stage of a Tableaux). Given a branch, b, of a tableau T, a *stage* in T is every maximal subsequence of successive nodes $n_i, n_{i+1}, \ldots, n_j$ in b such that $\tau(n_k)$ is a not a (Q○)-child of $\tau(n_{k-1})$, for all k such that $i < k \leq j$. We denote by $\mathsf{stages}(b)$ the *sequence of all stages* of b.

▶ **Definition 33** ($\alpha\beta^+$-saturated Stage). A stage s in $\mathcal{A}^{sys}_{\Sigma}$ is $\alpha\beta^+$-*saturated* iff for all $\sigma \in \tau(s)$:

1. If σ is an α-formula then $S_\sigma \subseteq \tau(s)$
2. If σ is a β-formula of range k, but is not a β^+-formula, then $S_{\beta_i} \subseteq \tau(s)$ for some $1 \leq i \leq k$.
3. If σ is a β-formula and also a β^+-formula of range k then either $S_{\beta_i} \subseteq \tau(s)$ or $S^+_{\Sigma,\beta_i} \subseteq \tau(s)$ for some $1 \leq i \leq k$ and $\Sigma = \tau(n) \setminus \{\sigma\}$ for some $n \in s$.

▶ **Definition 34** (Expanded Bunch and Tableau). An open branch b is *expanded* if each stage $s \in \mathsf{stages}(b)$ 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.

▶ **Proposition 35.** *(Trivial by construction) Given any set of state formulae Σ, the systematic tableau $\mathcal{A}^{sys}_{\Sigma}$ is expanded.*

▶ **Definition 36** (Open Bunch Model Construction). For any expanded bunch H of $\mathcal{A}^{sys}_{\Sigma}$, let $\mathcal{K}_H = (S, R, L)$ be a Kripke structure such that $S = \bigcup_{b \in H} \mathsf{stages}(b)$, R is the relation over $\mathsf{stages}(b)$ for any $b \in H$, and for any $s \in S$: $L(s) = \{p \mid p \in \tau(n) \cap \mathsf{Prop}$ for node $n \in s\}$.

▶ **Lemma 37** (Model Existence). *Let $\mathcal{A}^{sys}_{\Sigma}$ have an expanded bunch H and $\mathcal{K}_H = (S, R, L)$ be as in Definition 36. For every state $s \in S$, if $\sigma \in L(s)$ then $\mathcal{K}_H, s, 0 \models \sigma$. Therefore, for any expanded bunch H of $\mathcal{A}^{sys}_{\Sigma}$, $\mathcal{K}_H \models \Sigma$.*

▶ **Theorem 38** (Refutational Completeness). *Given any set of state formulae Σ, if $\mathsf{UnSat}(\Sigma)$ then there exists a closed tableau for Σ.*

▶ **Theorem 39** (Termination). *Given any set of state formulae Σ , the construction of the expanded tableau $\mathcal{A}^{sys}_{\Sigma}$ terminates.*

Finally Theorems 38 and 39 give us the desired completeness result stated in Theorem 40.

▶ **Theorem 40** (Completeness). *Given any set of state formulae Σ, if Σ is satisfiable then there exists a (finite) open expanded tableau for Σ.*

6 Conclusion

We introduced a new logic, ECTL$^{\#}$, in the family of BTL, which can represent a richer class of fairness constraints with the $\mathcal{U}$ operator. The tree-style one pass tableau method for ECTL$^{\#}$ handles inputs in an "analytic" way, due to the new, crucial for branching structures, concept of "inner context", in which eventualities are to be fulfilled. The tableau rules that invoke the inner context, are essential to handle A-disjunctive formulae. Our analysis of A-disjunctive and E-conjunctive formulae and of the prioritisation of eventualities, based on their structure and the context for their fulfillment, are important from the methodological point of view.

Our tableau technique is not directly extensible to CTL*. Without any significant modifications, β^+-rules become unsound for inputs that are beyond ECTL$^\#$ syntax due to nested path subformulae as in $\mathsf{A}\Diamond(\circ p \wedge \mathsf{E}\circ\neg p)$. To show the correctness of β^+-rules, we developed the technique to identify relevant state-formulae inside the specific path-modalities. This technique will be useful in studying more expressive logics (e.g. CTL*), as it allows to identify those subformulae that do not affect the "context", thus enabling the simplification of the structures.

The size of the systematic tableau for the input of size m is bounded by $2^{2^{O(m^2)}}$ (see technical report `http://www.sc.ehu.es/jiwlucap/TechReport18.pdf`). However, the method aims at the "shortest" way to fulfil the eventualities and, for many examples, finds the first open bunch, giving us a model for the tableau input. This significantly reduces the complexity. Finally, the presented technique is amenable for implementation – and this will be another stream of our future work. In the refinement and implementation of the algorithm we will be able to rely on similar techniques used in the implementation of its linear-time analogue.

References

1 Therese Berg and Harald Raffelt. Model checking. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems*, pages 557–603. Springer-Verlag, Berlin Heidelberg, 2005. `doi:10.1007/b137241`.

2 Kai Brünnler and Martin Lange. Cut-free sequent systems for temporal logic. *Journal of Logic and Algebraic Programming*, 76(2):216–225, 2008. `doi:10.1016/j.jlap.2008.02.004`.

3 Edmund M. Clarke, E. Allen Emerson, and Aravinda P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986. `doi:10.1145/5397.5399`.

4 E. Allen 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. URL: `http://dl.acm.org/citation.cfm?id=114891.114907`.

5 E. Allen Emerson and Joseph 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`.

6 E. Allen Emerson and Joseph Y. Halpern. Sometimes and not never revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986. `doi:10.1145/4904.4999`.

7 E. Allen Emerson and Chin-Laung Lei. Temporal reasoning under generalized fairness constraints. In Monien B. and Vidal-Naquet G., editors, *STACS 1986, Lecture Notes in Computer Science, vol 210*, pages 21–37. Springer-Verlag Berlin Heidelberg, Karlsruhe, Federal Republic of Germany, 1986. `doi:10.1007/3-540-16078-7_62`.

8 Jose Gaintzarain, Montserrat Hermo, Paqui Lucio, Marisa Navarro, and Fernando 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`.

9 Rajeev Gore. 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. `doi:10.1007/978-94-017-1754-0_6`.

10 Bernhard Josko. Model checking of ctl formulae under liveness assumptions. In Thomas Ottmann, editor, *Automata, Languages and Programming, 14th International Colloquium*,

pages 5–24. Springer-Verlag Berlin Heidelberg, Karlsruhe, Federal Republic of Germany, 1987. doi:10.1007/3-540-18088-5.

11 Jan Kretinsky and Ruslan Ledesma Garza. Rabinizer 2: Small deterministic automata for LTL\GU. In *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013*, pages 446–450, Heidelberg Dordrecht London New York, 2013. Springer. doi:10.1007/978-3-319-02444-8_32.

12 Nicolas 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.

13 Mark Reynolds. A tableau for CTL*. In Ana Cavalcanti and Dennis Dams, editors, *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2009. doi:10.1007/978-3-642-05089-3_26.

14 Robert S. Streett and E. Allen Emerson. The propositional mu-calculus is elementary. In Jan Paredaens, editor, *Automata, Languages and Programming, 11th Colloquium, Antwerp, Belgium, July 16-20, 1984, Proceedings*, volume 172 of *Lecture Notes in Computer Science*, pages 465–472. Springer, 1984. doi:10.1007/3-540-13345-3_43.

15 Jun Sun, Yang Liu, Jin Song Dong, and Hai H. Wang. Specifying and verifying event-based fairness enhanced systems. In Shaoying Liu, Tom Maibaum, and Keijiro Araki, editors, *Formal Methods and Software Engineering: 10th International Conference on Formal Engineering Methods ICFEM 2008*, pages 5–24. Springer Science and Business Media, Kitakyushu-City, Japan, 2008. doi:10.1007/978-3-540-88194-0.

A Appendix

Proof of Proposition 14. Use Proposition 12 to construct a tableau with all its leaves labelled by elementary sets of formulae. Then apply the rule (Q○) to any leaf. Finally, apply (to every leaf) the rules (Eσ), (Aσ), ($\wedge$), and ($\vee$), as long as they are applicable. ◀

Proof of Proposition 29 (Preservation of the Negated Inner Context). We check the four cases of a basic path formula $\pi \in \Pi$. If π is of the form $\sigma_1 \mathcal{U} (\sigma_2 \wedge \Diamond\sigma_3)$, then property (a) ensures that every state in y satisfies $\neg\sigma_2$. Therefore, $\neg(\sigma_1 \mathcal{U} (\sigma_2 \wedge \Diamond\sigma_3))$ is satisfied in the limit path y. The remaining three cases are proved on the basis of (b) and Definition 17:
If $\pi = \Box(\sigma_1 \vee \Box\sigma_2)$, then $\mathcal{K}, x_1^{\leq i_1} x_2^{\leq i_2} \cdots x_n^{\leq i_n}, i_n \models \neg\sigma_1 \wedge \neg\sigma_2$ for all n. Therefore, it holds that $\mathcal{K}, y \models \neg\Box(\sigma_1 \vee \Box\sigma_2)$.
If $\pi = \Box(\sigma_1 \mathcal{U} \sigma_2)$, then $\mathcal{K}, x_1^{\leq i_1} x_2^{\leq i_2} \cdots x_n^{\leq i_n}, i_n \models \neg\mathsf{E}(\Diamond\sigma_2)$ for all n. Hence, $\mathcal{K}, y \models \neg\Box(\sigma_1 \mathcal{U} \sigma_2)$.
If $\pi = \sigma_1 \mathcal{U} \Box\sigma_2$, then $\mathcal{K}, x_1^{\leq i_1} x_2^{\leq i_2} \cdots x_n^{\leq i_n}, i_n \models \neg\sigma_2$ for all n. Hence, $\mathcal{K}, y \models \neg(\sigma_1 \mathcal{U} \Box\sigma_2)$. ◀

Proof of Lemma 30 (Soundness of Tableau Rules). Noting that (i), (i) and (iv) can be easily proved by the "systematic" application of the semantic definitions of temporal operators, we prove (iii). The "only if" direction" for each of the cases of β^+-rules is trivial.
For the "if" direction of rule $(\mathsf{E}\mathcal{U}\sigma)^+$, let us suppose that $\mathcal{K} \models \Sigma, \mathsf{E}(\sigma_1 \mathcal{U} (\sigma_2 \wedge \Diamond\sigma_3), \Pi)$. There exists $x \in \mathsf{fullpaths}(\mathcal{K})$ such that $\mathcal{K}, x, 0 \models \Sigma, \Pi, \sigma_1 \mathcal{U} (\sigma_2 \wedge \Diamond\sigma_3)$. We are going to prove that there exists $\mathcal{K}'$ such that one of the following two properties holds:
(a) $\mathcal{K}' \models \Sigma, \sigma_2, \mathsf{E}(\Diamond\sigma_3, \Pi)$
(b) $\mathcal{K}' \models \Sigma, \sigma_1, \mathsf{E}(\circ((\sigma \wedge \neg\Sigma) \mathcal{U} \sigma_2), \Pi)$.
If $\mathcal{K}, x, 0 \models \sigma_2 \wedge \Diamond\sigma_3$, then (a) is trivially satisfied for $\mathcal{K}' = \mathcal{K}$. Otherwise, for some $i > 0$, it holds that $\mathcal{K}, x, i \models \sigma_2 \wedge \Diamond\sigma_3$ and for all $j < i$: $\mathcal{K}, x, j \models \sigma_1$. Let j be the least number greater than 0 such that $\mathcal{K}, x, j \models \sigma_2 \wedge \Diamond\sigma_3$. Consider k to be the greatest index in $\{0, \ldots, j\}$ such that $\mathcal{K}, x, k \models \Sigma, \Pi$. If $k = j$, then (a) is satisfied for $\mathcal{K}' = \mathcal{K} \restriction x(k)$. Otherwise, if $k < j$

then $\mathcal{K}, x, k \models \Sigma, \sigma_1, \circ((\sigma_1 \wedge \neg\Sigma)\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)), \Pi$. Hence, item (b) holds for $\mathcal{K}' = \mathcal{K} \restriction x(k)$. Rewriting in the above proof $\sigma_2 \wedge \Diamond\sigma_3$ by $\Box\sigma_2$, we obtain the proof for rule $(\mathsf{E}\,\mathcal{U}\,\Box)^+$. Indeed, both proofs for $(\mathsf{E}\,\mathcal{U}\,\sigma)^+$ and $(\mathsf{E}\,\mathcal{U}\,\Box)^+$ are very similar to the context-based rule in linear-time case of PLTL (see Lemma 5.1 in [8]).

For the "if" direction of rule $(\mathsf{A}\,\mathcal{U}\,\sigma)^+$, let us suppose that the three sets $\Sigma \cup S_{\Sigma,\beta_1}$, $\Sigma \cup S_{\Sigma,\beta_2}$, and $\Sigma \cup S_{\Sigma,\beta_3}$ of the rule $(\mathsf{A}\,\mathcal{U}\,\sigma)^+$ are unsatisfiable. We will show that the set $\Sigma, \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$ must be also unsatisfiable. By the hypothesis, we know that any model of Σ is not a model of S_{Σ,β_i} for all $i \in \{1,2,3\}$. In other words, for any $\mathcal{K}$ such that $\mathcal{K} \models \Sigma$, the followings three facts holds:

(a) $\mathcal{K} \not\models \sigma_2 \wedge \mathsf{A}(\Diamond\sigma_3, \Pi)$

(b) $\mathcal{K} \not\models \sigma_1 \wedge \mathsf{A}(\circ((\sigma_1 \wedge (\neg\Sigma \vee \varphi_\Pi))\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)), \Pi)$

(c) $\mathcal{K} \not\models \mathsf{A}\Pi$

To show that $\Sigma, \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$ is unsatisfiable, we consider any $\mathcal{K}$ such that $\mathcal{K} \models \Sigma$ and prove that $\mathcal{K} \not\models \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$. Since $\mathcal{K} \models \Sigma$, then (a), (b) and (c) hold. According to (b), there are two possible cases:

(Case 1): If $\mathcal{K} \not\models \sigma_1$ then, by (a), either $\mathcal{K} \models \neg\sigma_1 \wedge \neg\sigma_2$ or $\mathcal{K} \models \neg\sigma_1 \wedge \mathsf{E}(\Box\neg\sigma_3, \neg\Pi)$. In both cases, it is easy to see that $\mathcal{K} \not\models \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$.

(Case 2): Otherwise, if $\mathcal{K} \not\models \mathsf{A}(\circ((\sigma_1 \wedge (\neg\Sigma \vee \varphi_\Pi))\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)), \Pi)$, then we have that there exists $x_1 \in \mathsf{fullpaths}(\mathcal{K})$ such that $\mathcal{K}, x_1 \models \circ\neg((\sigma_1 \wedge (\neg\Sigma \vee \varphi_\Pi))\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)) \wedge \neg\Pi$. This yields two possible cases:

(Case 2.1): If $\mathcal{K}, x_1 \models \circ\Box(\neg\sigma_2 \vee \Box\neg\sigma_3) \wedge \neg\Pi$, then it is trivial that $\mathcal{K} \not\models \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$.

(Case 2.2): Otherwise, there should exist $i_1 > 0$ that satisfies the following three properties:

(i) $\mathcal{K}, x_1, j \models (\neg\sigma_2) \vee \Box\neg\sigma_3$ for all j such that $0 \leq j \leq i_1$, and

(ii) $\mathcal{K}, x_1, i_1 \models \neg\sigma_1 \vee (\Sigma \wedge \neg\varphi_\Pi)$, and

(iii) $\mathcal{K}, x_1, 0 \models \neg\Pi$

If (i) is satisfied because $\mathcal{K}, x_1, j \models \Box\neg\sigma_3$ for some j such that $0 \leq j \leq i_1$, then trivially $\mathcal{K}, x_1, 0 \not\models \sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)$. This, along with the fact (iii), ensures that $\mathcal{K} \not\models \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$. Moreover, this also applies to any other formula $\sigma_1'\,\mathcal{U}\,(\sigma_2' \wedge \Diamond\sigma_3')$ in Π. Henceforth, in what follows, we can suppose that for all j such that $0 \leq j \leq i_1$: $\mathcal{K}, x_1, j \models \neg\sigma_2$ and also that $\mathcal{K}, x_1, j \models \neg\sigma_2'$ for all $\sigma_1'\,\mathcal{U}\,(\sigma_2' \wedge \Diamond\sigma_3') \in \Pi$.

If (ii) is satisfied because $\mathcal{K}, x_1, i_1 \models \neg\sigma_1$ then it is clear that $\mathcal{K}, x_1, 0 \not\models \sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)$. Therefore, by (i) and (iii), $\mathcal{K} \not\models \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$.

Otherwise, if (ii) is satisfied because $\mathcal{K}, x_1, i_1 \models \Sigma \wedge \neg\varphi_\Pi$, then, since $\mathcal{K}, x_1, i_1 \models \Sigma$, then again (a), (b) and (c) hold for $\mathcal{K} \restriction x_1(i_1)$ (instead of $\mathcal{K}$). Hence, reasoning for $\mathcal{K} \restriction x_1(i_1)$ as we do above for $\mathcal{K}$, there should exist a path $x_2 \in \mathsf{fullpaths}(\mathcal{K} \restriction x_1(i_1))$ such that one of the following two facts holds:

(Case 2.2.1): $\mathcal{K} \restriction x_1(i_1), x_2 \models \Box\neg(\sigma_2 \wedge \Diamond\sigma_3) \wedge \neg\Pi$, and therefore $\mathcal{K} \not\models \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$.

(Case 2.2.2): there should exist $i_2 > 0$ such that $\mathcal{K} \restriction x_1(i_1), x_2, i_2 \models \Sigma \wedge \neg\varphi_\Pi$ and for all $j \in \{0..i_2\}$:

- $\mathcal{K} \restriction x_1(i_1), x_2, j \models \neg\sigma_2$, and
- $\mathcal{K} \restriction x_1(i_1), x_2, j \models \neg\sigma_2'$ for all $\sigma_1'\,\mathcal{U}\,(\sigma_2' \wedge \Diamond\sigma_3') \in \Pi$j

Now, (a), (b) and (c) apply to $\mathcal{K} \restriction x_2(i_2)$. Hence, the infinite iteration of the second case yields a path $y = x_1^{\leq i_1} x_2^{\leq i_2} \cdots x_k^{\leq i_k} \cdots$ (that exists by the limit closure property) for which the Proposition 29 ensures that $\mathcal{K}, y \not\models \mathsf{A}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$.

The proof for rule $(\mathsf{A}\,\mathcal{U}\,\Box)^+$ follows the same scheme. It is worth noting that, according to rule $(\mathsf{A}\,\mathcal{U}\,\Box)^+$, for all n: $\mathcal{K}, x_1^{\leq i_1} x_2^{\leq i_2} \cdots x_n^{\leq i_n}, i_{n+1} \models \neg\varphi_\Pi \wedge \neg\sigma_2$ because this ensures that the limit path y satisfies $\neg\Box\sigma_2$. ◀

Proof of Theorem 31 (Soundness of Tableau Method). Let T_Σ be a closed tableau for Σ. The set of formulas labelling at least one leaf in each bunch is inconsistent and therefore unsatisfiable. Then, by Lemma 30, the root Σ is unsatisfiable. ◀

To prove refutational completeness and termination, we first define a partial order relation on the set of basic state formulae. This order is the basis for inductively proving that $\mathcal{K}_H$ is a model of the tableau input (Lemma 37). The termination of Algorithm 1 (Theorem 39) is based on the extension that order to the set of finite sets of basic state formulae.

▶ **Definition 41** (Order on Basic state formulae)**.** The order $\preceq$ is defined as the reflexive-transitive closure of the smallest binary relation $\prec \subset \mathcal{F}_{\mathsf{Prop}} \times \mathcal{F}_{\mathsf{Prop}}$ that satisfies the following conditions:

1. $\mathsf{Q}\Pi' \prec \mathsf{Q}\Pi$ if $\Pi' \subset \Pi$.
2. $\sigma \prec \mathsf{Q}\Pi$ if σ is a proper state-subformula of $\mathsf{Q}\Pi$.
3. $\mathsf{Q}((\sigma_1 \wedge \delta)\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi) \prec \mathsf{Q}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$. In particular, $\mathsf{Q}((\sigma_1 \wedge \delta)\,\mathcal{U}\,\sigma_2, \Pi) \prec \mathsf{Q}(\sigma_1\,\mathcal{U}\,\sigma_2, \Pi)$.
4. $\mathsf{Q}(\Diamond\sigma_3, \Pi) \prec \mathsf{Q}(\sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3), \Pi)$.
5. $\mathsf{Q}((\sigma_1 \wedge \delta)\,\mathcal{U}\,\Box\sigma_2, \Pi) \prec \mathsf{Q}(\sigma_1\,\mathcal{U}\,\Box\sigma_2, \Pi)$.
6. $\mathsf{Q}(\Box\sigma_2, \Pi) \prec \mathsf{Q}(\sigma_1\,\mathcal{U}\,\Box\sigma_2, \Pi)$.

where $\mathsf{Q} \in \{\mathsf{A}, \mathsf{E}\}$, Π, Π' are sets of basic path formulae, $\sigma, \sigma_1, \sigma_2$ are basic state formulae, and δ is a state formula different of the constant $\mathbf{T}$.
The extension $\prec^*$ is the partial order relation on finite sets of basic state formulae defined by $\Sigma_1 \preceq^* \Sigma_2$ if and only if for every $\sigma_1 \in \Sigma_1$ either $\sigma_1 \in \Sigma_2$ or there exists $\sigma_2 \in \Sigma_2$ such that $\sigma_1 \prec \sigma_2$. Therefore $\Sigma_1 \prec^* \Sigma_2$ if and only $\Sigma_1 \preceq^* \Sigma_2$ and $\Sigma_1 \neq \Sigma_2$.

Proof of Lemma 37 (Model Existence) (Sketch). Let $\mathcal{A}^{sys}_\Sigma$ have an expanded bunch H and $\mathcal{K}_H = (S, R, L)$ be as in Definition 36. We prove that, for every state $s \in S$, if $\sigma \in L(s)$ then $\mathcal{K}_H, s, 0 \models \sigma$. This proof is made by structural induction on the formula σ. That induction requires many auxiliary properties about how $\mathsf{ECTL}^{\#}$-formulae evolve along the branches in the systematic construction of the tableau $\mathcal{A}^{sys}_\Sigma$. These properties can be found in the technical report `http://www.sc.ehu.es/jiwlucap/TechReport18.pdf`). For space reasons, we only give here the following sketch of the proof. The base case, $\sigma = p \in \mathsf{Prop}$, is ensured by Definition 36. The bunch H ensures that, whenever a tableau node (at stage s) is labelled by an elementary set $\{\Sigma, \mathsf{A}\circ\Phi_1, \ldots, \mathsf{A}\circ\Phi_n, \mathsf{E}\circ\Psi_1, \ldots, \mathsf{E}\circ\Psi_m\} \subseteq L(s)$ then, by rule ($\mathsf{Q}\circ$), H contains one successor stage s_i (for each $i \in \{1, \ldots, m\}$) that, in turn, contains $\{A\Phi_1, \ldots, \mathsf{A}\Phi_n, \mathsf{E}\Psi_i\}$. By inductive hypothesis: $\mathcal{K}_H, s_i, 0 \models A\Phi_1, \ldots, \mathsf{A}\Phi_n, \mathsf{E}\Psi_i$, for each $i \in \{1, \ldots, m\}$. Therefore, $\mathcal{K}_H, s, 0 \models \{\Sigma, \mathsf{A}\circ\Phi_1, \ldots, \mathsf{A}\circ\Phi_n, \mathsf{E}\circ\Psi_1, \ldots, \mathsf{E}\circ\Psi_m\}$. Every branch $b \in H$ is open, so it is a cyclic branch such that $\mathsf{path}(b) = s_0, s_1, \ldots, s_{i-1}\langle s_i, s_{i+1}, \ldots, s_j\rangle^\omega$. For any $s_k \in \mathsf{path}(b)$ and any formula $\mathsf{Q}\Pi$ in $\tau(s_k)$, we prove that $\mathcal{K}_H, s_k, 0 \models \mathsf{Q}\Pi$, by induction in $\mathsf{Q}\Pi$. Formulae of the highest priority have to be selected at the stage previous to s_i. Hence, the first node of stage s_i is a loop-node that must be labelled by a set exclusively formed by potentially-cycling and cycling formulas (in particular, the empty set) ◀

Proof of Theorem 38 (Refutational Completeness). Suppose that for an input Σ there is no closed tableau. Then the systematic tableau $\mathcal{A}^{sys}_\Sigma$ would be open and there would be at least one expanded bunch H in $\mathcal{A}^{sys}_\Sigma$. By Lemma 37, $\mathcal{K}_H \models \Sigma$. Consequently, Σ would be satisfiable. ◀

Simplification rules are very useful to reduce the tableau construction but, more importantly, some simplification rules are essential for termination.

▶ **Definition 42** (Simplification Rules). First, to stop the growth of the subformula σ in the successive next-step variants $(\sigma_1 \wedge \sigma)\,\mathcal{U}\,\varphi$, we use trivial simplification rules such as $\varphi \wedge \varphi \longrightarrow \varphi$ and $\varphi \vee \varphi \longrightarrow \varphi$, as well as classical subsumptions rules. Second, to simplify the detection of equal node labels (for looping in tableau branches) we use the following rules:
$(\sqsubset \mathsf{E}\Box\,\mathcal{U}\,)$ $\mathsf{E}(\sigma_1\,\mathcal{U}\,\sigma_2, \Box(\sigma_1\,\mathcal{U}\,\sigma_2), \Pi) \longrightarrow \mathsf{E}(\Box(\sigma_1\,\mathcal{U}\,\sigma_2), \Pi)$.
$(\sqsubset \mathsf{A}\Box\,\mathcal{U}\,)$ If $\Pi' \subseteq \Pi$ then $\mathsf{A}(\sigma_1\,\mathcal{U}\,\sigma_2, \Pi) \wedge \mathsf{A}(\Box(\sigma_1\,\mathcal{U}\,\sigma_2), \Pi') \longrightarrow \mathsf{A}(\Box(\sigma_1\,\mathcal{U}\,\sigma_2), \Pi')$.
Finally, to prevent the duplications of the original eventuality $\sigma_1\,\mathcal{U}\,\sigma_2$ and its successive next-step variants by rules $(\mathsf{Q}\Box\,\mathcal{U}\,)$ and $(\mathsf{Q}\,\mathcal{U}\,\sigma)^+$, and to ensure termination, we use the following rules:
$(\sqsubset \mathsf{A}\sigma\,\mathcal{U}\,)$ $\sigma_2 \wedge \mathsf{A}(\sigma_1\,\mathcal{U}\,\sigma_2, \Pi) \longrightarrow \sigma_2$.
$(\sqsubset \mathsf{E}\,\mathcal{U}\,\sigma)$ $\mathsf{E}((\sigma_1 \wedge \sigma)\,\mathcal{U}\,\varphi, \sigma_1\,\mathcal{U}\,\varphi, \Pi) \longrightarrow \mathsf{E}((\sigma_1 \wedge \sigma)\,\mathcal{U}\,\varphi, \Pi)$
$(\sqsubset \mathsf{A}\,\mathcal{U}\,\sigma)$ If $\Pi' \subseteq \Pi$ then $\mathsf{A}((\sigma_1 \wedge \sigma)\,\mathcal{U}\,\varphi, \Pi') \wedge \mathsf{A}(\sigma_1\,\mathcal{U}\,\varphi, \Pi) \longrightarrow \mathsf{A}((\sigma_1 \wedge \sigma)\,\mathcal{U}\,\varphi, \Pi')$.
In order to prove termination, the following Propositions 43 and 44 ensure that any open branch of $\mathcal{A}^{sys}_{\Sigma}$ cannot be labelled by an infinitely $\preceq^*$-decreasing succession of set of formulae in the sense of items 3 and 5 in Definition 41. In addition, Proposition 45 shows that any open branch of $\mathcal{A}^{sys}_{\Sigma}$ is eventuality-covered.

▶ **Proposition 43.** *Let b be an open branch of $\mathcal{A}^{sys}_{\Sigma}$, let $s_i \in \mathsf{stages}(b)$ and let $\Sigma \cup \{\mathsf{E}\Pi\}$ be the US labelling the first node of s_i where $\mathsf{E}\Pi$ has been selected and one of the eventualities $\pi_{\mathcal{U}} \in \Pi$ has been marked. Then there exists some $k \geq i$ such that:*
(a) *If $\pi_{\mathcal{U}} = \sigma_1\,\mathcal{U}\,(\sigma_2 \wedge \Diamond\sigma_3)$ then $\{\sigma_2, \mathsf{E}(\Diamond\sigma_3, \Pi')\} \subseteq \tau(s_k)$ for some Π' such that $\mathsf{E}(\Diamond\sigma_3, \Pi') \prec \mathsf{E}\Pi$.*
(b) *If $\pi_{\mathcal{U}} = \sigma_1\,\mathcal{U}\,\Box\sigma_2$ then $\mathsf{E}(\Box\sigma_2, \Pi') \in \tau(s_k)$ for some Π' such that $\mathsf{E}(\Box\sigma_2, \Pi') \prec \mathsf{E}\Pi$.*
(c) *If $\pi_{\mathcal{U}} = \Box(\sigma_1\,\mathcal{U}\,\sigma_2)$ then $\sigma_2 \in \tau(s_k)$.*

Proof. Suppose that the rule $(\mathsf{E}\,\mathcal{U}\,\sigma)^+$ (or $(\mathsf{E}\,\mathcal{U}\,\Box)^+$) has been successively applied, keeping the selection, between stages s_i and s_k. In the case (c), $(\mathsf{E}\,\mathcal{U}\,\sigma)^+$ is applied immediately after $(\mathsf{E}\Box\,\mathcal{U}\,)$. Then, in any of the three cases, the US labelling the first node of stage s_k has the form $\Sigma_{s_k}, \mathsf{E}((\sigma_1 \wedge (\neg\Sigma_{s_i} \wedge \cdots \wedge \neg\Sigma_{s_{k-1}}))\,\mathcal{U}\,\varphi, \Pi')$ where φ is either $(\sigma_2 \wedge \Diamond\sigma_3)$ or $\Box\sigma_2$ or σ_2; each Σ_{s_j} $(i \leq j \leq k)$ is the context of the selected formula at the first node of each stage s_j (in particular $\Sigma_{s_i} = \Sigma$); and Π' is such that : $\mathsf{E}((\sigma_1 \wedge (\neg\Sigma_{s_i} \wedge \cdots \wedge \neg\Sigma_{s_{k-1}}))\,\mathcal{U}\,\varphi, \Pi') \prec \mathsf{E}\Pi$. Since no other β^+-rule is applied between stage s_i and s_k, each Σ_{s_j} is a subset of the finite set formed by all state formulae that are subformulae of some formula in $\Sigma_{s_i} \cup (\Pi \setminus \{\pi\})$ and their negations.[8] Hence, there is a finite number of different Σ_{s_j}. Therefore, after a finite number applications of the β^+-rule, the label containing the set $\{\sigma_1 \wedge (\neg\Sigma_{s_i} \wedge \cdots \wedge \neg\Sigma_{s_{k-1}}), \Sigma_{s_k}\}$ must be inconsistent. Henceforth, the open branch b must satisfy (a) or (b) or (c), depending on the case of $\pi_{\mathcal{U}}$, and according to the rules $(\mathsf{E}\,\mathcal{U}\,\sigma)^+$ and $(\mathsf{E}\,\mathcal{U}\,\Box)^+$. ◀

For A-disjunctive formulae, not only the outer context, but also the inner context plays an important role. The proof of the next proposition is based on the use of both kinds of context.

▶ **Proposition 44.** *Let b be an open branch of $\mathcal{A}^{sys}_{\Sigma}$, let $s_i \in \mathsf{stages}(b)$ and let $\Sigma \cup \{\mathsf{A}\Pi\}$ be the US labelling the first node of s_i where $\mathsf{Q}\Pi$ has been selected and one of the eventualities $\pi_{\mathcal{U}} \in \Pi$ has been marked. Then there exists some stage $s_k \in \mathsf{stages}(b)$ (for some $k \geq i$) such that one of the following two facts holds:*

[8] This finite set can be seen as a "local closure".

(a) *if* $\pi_{\mathcal{U}} = \sigma_1 \mathcal{U} (\sigma_2 \wedge \Diamond\sigma_3)$ *then* $\{\sigma_2, \mathsf{A}(\Diamond\sigma_3, \Pi')\} \in \tau(s_k)$ *for some* Π' *such that* $\mathsf{E}(\Diamond\sigma_3, \Pi') \prec \mathsf{E}\Pi$*; if* $\pi = \sigma_1 \mathcal{U} \Box\sigma_2$*, then* $\mathsf{A}(\sigma_2, \Pi') \in \tau(s_k)$ *for some* Π' *such that* $\mathsf{A}(\Box\sigma_2, \Pi') \prec \mathsf{A}\Pi$*; and if* $\pi_{\mathcal{U}} = \Box(\sigma_1 \mathcal{U} \sigma_2)$ *then* $\sigma_2 \in \tau(s_k)$*, or*

(b) *the first node* n *of stage* s_k *is a loop-node that contains* $\{\mathsf{A}((\sigma_1 \wedge \delta) \mathcal{U} \varphi, \Pi')$ *for some* δ*, some* φ *that depends on* $\pi_{\mathcal{U}}$*, and some* Π' *such that* $\{\mathsf{A}((\sigma_1 \wedge \delta) \mathcal{U} \varphi, \Pi') \prec \mathsf{A}\Pi$*.*

Proof. Suppose that the rule $(\mathsf{A}\mathcal{U}\sigma)^+$ (or $(\mathsf{A}\mathcal{U}\Box)^+$) has been successively applied, keeping the selection, between stages s_i and s_k. Then the US labelling the first node of stage s_k has the form:

$$\Sigma_{s_k}, \mathsf{A}((\sigma_1 \wedge (\neg\Sigma_{s_i} \vee \varphi_{\Pi_{s_i}}) \wedge \cdots \wedge (\neg\Sigma_{s_{k-1}} \vee \varphi_{\Pi_{s_{k-1}}})) \mathcal{U} \varphi, \Pi_{s_k})$$

where φ is $\sigma_2 \wedge \Diamond\sigma_3$ or $\Box\sigma_2$ or σ_2 (depending on the case of $\pi_{\mathcal{U}}$); and each Σ_{s_j} and each Π_{s_j} $(i \leq j \leq k)$ are respectively the outer context and the inner context at the first node of each stage s_j. In particular, $\Sigma_{s_i} = \Sigma$ and $\Pi_{s_i} = \Pi$. Since no other β^+-rule is applied between s_i and s_k, then[9]

- each Σ_{s_j} is a subset of the finite set formed by all state formulae that are subformulae of some formula in $\Sigma \cup (\Pi \setminus \{\pi_{\mathcal{U}}\})$, their negations, and a formula $\mathsf{E}(\mathbf{T}\mathcal{U}\sigma_2)$ for each subformula $\Box(\sigma_1 \mathcal{U} \sigma_2)$ in $\Sigma \cup (\Pi \setminus \{\pi_{\mathcal{U}}\})$ (see Definition 17), and
- each Π_{s_j} is a subset of the finite set of all path formulae that are subformulae of some formula in Π.

Therefore, since we simplify $\varphi \wedge \varphi$ by φ and $\varphi \vee \varphi$ by φ, after a finite number β^+-rule applications (if b is not closed) some previous node label should be repeated. ◀

▶ **Proposition 45.** *Any open branch* b *of* $\mathcal{A}^{sys}_{\Sigma}$ *is eventuality-covered.*

Proof Sketch. This sketch is based on several properties of the systematic tableau construction which, for space reasons, can only be found in the technical report `http://www.sc.ehu.es/jiwlucap/TechReport18.pdf`. Whenever we say *by construction* we refer to some of these properties. Let b be any open branch of $\mathcal{A}^{sys}_{\Sigma}$. By construction, there exists some stage $s_i \in \mathsf{stages}(b)$ such that the label of the first node n in s_i is either empty or a non-empty US Σ that consists exclusively of potentially-cycling and cycling formulae (and possibly a set of literals). If Σ is empty or a set of literals, b is finished by a cycle consisting of two empty labels, which is trivially eventuality-covered. Otherwise, Σ contains at most one formula $\mathsf{E}\Pi$ along with a set $\mathsf{A}\Pi_1, \ldots, \mathsf{A}\Pi_k, \mathsf{A}\Pi'_1, \ldots, \mathsf{A}\Pi'_m$ such that for all $1 \leq i \leq k$: $\mathsf{A}\Pi_i$ is a potentially-cycling formula, and for all $1 \leq i \leq m$: $\mathsf{A}\Pi'_i$ is a cycling formula. Note that if there is one $\mathsf{E}\Pi$ in Σ, then $\mathsf{E}\Pi$ is a cycling formula. If $k = 0$ and $\Pi'_1, \ldots, \Pi'_m, \Pi$ does not contain any eventuality (as formula or subformula). By construction, the first-node of the next-stage s_{i+1} is a loop-node also labelled by Σ and b is eventuality-covered. Otherwise, one of the formulae in Σ (the label of node n) is selected, namely σ, and one of its eventualities, namely π, is marked. By construction, the node n should be followed (in b) by some node n', labelled by Σ', such that either π is fulfilled in Σ' or n' is a loop-node. In both cases $\Sigma' \prec^* \Sigma$. If Σ' does not contain more eventualities (apart from π), b is already eventuality-covered. Otherwise, the selection of eventualities is applied in the first state of the new stage. All the formulae of highest-priority are "solved" firstly (if Σ' have some), until the label is again a set of potentially-cycling and cycling formulae. Then we change to select a formula (if any) that has been not already selected in the branch (if it is not a loop-node) or in the loop (if it is a loop-node). This process gives –as node labels– sequences of sets of

[9] Finite sets Σ_{s_j} and Π_{s_j} can be seen as "local closures".

cycling and potentially-cycling formulae that are strictly decreasing with respect to $\preceq^*$. Henceforth, while b remains open, loop-nodes strictly decrease w.r.t. $\preceq^*$. Since the number k of potentially-cycling formulae is finite, after a finite number of stages, we get a minimal loop-node labelled by some Σ''. If Σ'' does not contain any eventuality, the Uniform_Tableau produces a new node labelled by Σ''. Otherwise, any selection made on this Σ'' leads to a stage whose first node is also labelled by Σ''. Hence, after selecting all the selectable formulae in Σ'' the branch b is eventuality-covered. ◀

Proof of Theorem 39 (Termination). Tableau rules produce a finite branching, hence König's Lemma applies. The subsumption-based simplification rules (Definition 42) do prevent the generation of formulae containing the original eventuality when a next-step variant of the eventuality has been generated. By Propositions 43 and 44, once one eventuality is marked, a kind of "local closure" allows us to ensure the finiteness of the application of a β^+-rule to the selected eventuality. Finally, Proposition 45 ensures that any open branch is eventually-covered. ◀

Results on Alternating-Time Temporal Logics with Linear Past

Laura Bozzelli
University of Napoli "Federico II", Italy

Aniello Murano
University of Napoli "Federico II", Italy

Loredana Sorrentino
University of Napoli "Federico II", Italy

Abstract

We investigate the succinctness gap between two known equally-expressive and different linear-past extensions of standard CTL* (resp., ATL*). We establish by formal non-trivial arguments that the "memoryful" linear-past extension (the history leading to the current state is taken into account) can be exponentially more succinct than the standard "local" linear-past extension (the history leading to the current state is forgotten). As a second contribution, we consider the ATL-like fragment, denoted ATL$_{lp}$, of the known "memoryful" linear-past extension of ATL*. We show that ATL$_{lp}$ is strictly more expressive than ATL, and interestingly, it can be exponentially more succinct than the more expressive logic ATL*. Moreover, we prove that both satisfiability and model-checking for the logic ATL$_{lp}$ are EXPTIME-complete.

2012 ACM Subject Classification Theory of computation → Logic → Verification by model checking

Keywords and phrases Alternating-time temporal logics, Linear Past, Model Checking

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.6

1 Introduction

Temporal logics provide a fundamental framework for the description of the dynamic behavior of reactive systems. Additionally, they support the successful model-checking approach [3] that allow complex (finite-state) systems, usually modeled by propositional Kripke structures, to be verified automatically. The model-checking methodology considers three types of temporal logics which differ in the underlying nature of time: linear, branching, and alternating. In linear-time temporal logics such as standard LTL [27], formulas are interpreted over linear sequences (corresponding to single paths of the Kripke structure), and temporal operators are provided for describing the ordering of events along a single computation path. Branching-time temporal logics such as CTL [10] and CTL* [11], on the other hand, allow to reason about several possible futures: formulas are interpreted over states of a Kripke structure, hence referring to all the possible system computations. Such logics are in general more expressive than linear-time temporal logics since they provide both temporal operators for describing properties of a single path, and path quantifiers for describing the branching structure in computation trees (resulting from unwinding a Kripke structure from the initial state). Finally, alternating-time temporal logic such as ATL* and ATL [2], generalize the branching-time paradigm (useful for the verification of closed systems) to a strategic-reasoning paradigm suitable for the verification of open and multi-agent systems [20, 2, 1, 6, 9, 25, 26, 15, 5]. In this setting, different processes or components (the agents) can interact in an adversarial or cooperative manner in order to achieve given temporal goals. The interaction among agents

is usually modeled by *concurrent game structures* [2] (CGS, for short), a generalization of Kripke structures, where each transition results from a set of decisions, one for each agent. In particular, the logic ATL*, which is interpreted over CGS, is an extension of CTL* obtained by replacing path quantifiers with more general quantifiers $\langle\langle A \rangle\rangle$ parameterized by a set A of agents which express selective quantification over those paths (from the current state) that are obtained as outcomes of the infinite game between the coalition A and the complement.

Linear past in temporal logics. Standard temporal logics such as LTL, CTL*, and ATL* do not have explicit mechanisms to refer to the past of the current time. On the other hand, it is well-known that temporal logics which combine both past and future temporal modalities make specifications easier to write and more natural. In particular, the past extension LTL$_p$ of standard LTL does not increase the complexity of model-checking and satisfiability-checking [32], and at the same time, turns out to be exponentially more succinct than LTL [22]. For branching-time temporal logics, the adding of past-time constructs has been investigated in many papers [30, 14, 17, 23, 33, 24, 4, 18]. Usually, the past is assumed to be finite (since program computations have a definite starting time) and cumulative (i.e., the history of the current situation increases with time and is never forgotten). Moreover, one can adopt either a branching-past approach (past and future are handled uniformly) or a linear-past approach. Here, we focus on known linear-past extensions of CTL* and its alternating-time counterpart ATL*, which are syntactically obtained by adding the past versions of the standard LTL temporal modalities. The simplest linear-past extension of CTL* is the logic PCTL* [14], where the semantics of path quantification is the same as for CTL*: path quantification ranges over paths that starts in the current node of the computation tree. Hence, the history (computation) from the root-node (starting time) to the current node is forgotten (*local linear-past view*). A similar local linear-past extension, denoted PATL*, can be considered for the logic ATL*, where the linear-time temporal goals, arguments of the strategy quantifiers, are evaluated along the outcomes of the selected strategy starting from the current node. A more interesting and meaningful linear-past extension of CTL* is the logic CTL$^*_{lp}$ [17, 18], where path quantification is 'memoryful', i.e., it ranges over paths that start at the root and visit the current node (*memoryful linear-past view*). The ATL* counterpart of the logic CTL$^*_{lp}$, here denoted by ATL$^*_{lp}$, has been investigated in [26] (see also [7]), where the temporal goals are evaluated at the current time of the paths obtained by prefixing the outcomes of the selected strategy with the history leading to the current node. The usefulness of memoryful linear-past has been illustrated in [18, 26]: for example, one can require that a condition is satisfied only if a precondition holds along the whole past computation. In strategic reasoning, as argued in [26], memoryful linear-past enables *relentful* reasoning (agents can relent and change their goals depending on the history) which can be applied to relevant scenarios such as strong cyclic planning and bounded verification. Satisfiability and model-checking of the mentioned linear-past extensions of CTL* (resp., ATL*) have the same complexity as CTL* (resp., ATL*) with the exception of model-checking against CTL$^*_{lp}$ which is EXPSPACE-complete [18], hence, exponentially harder than CTL* model-checking (the latter being PSPACE-complete [11]).

Our contribution. It is known that PCTL* and CTL$^*_{lp}$ have the same expressiveness as CTL* [18] and there are translations from PCTL* and CTL$^*_{lp}$ into CTL* of non-elementary complexity [18] based on the separation theorem for LTL$_p$ [12]. On the other hand, the ability to refer to the past makes both PCTL* and CTL$^*_{lp}$ exponentially more succinct than CTL*. Analogous results hold for the logics PATL* and ATL$^*_{lp}$ when compared to ATL* [26]. On the

other hand, no succinctness gap is known between the "memoryful" past view in CTL^*_{lp} (resp., ATL^*_{lp}) and the 'local' past view in PCTL^* (resp., PATL^*). Our first contribution addresses this issue: we establish by formal non-trivial arguments that CTL^*_{lp} (resp., ATL^*_{lp}) can be exponentially more succinct than PCTL^* (PATL^*).

The logics CTL [11] and ATL [2] are well-known syntactical fragments of CTL^* and ATL^* which have received a lot of attention due to the existence of polynomial-time algorithms which solve the associated model-checking problem. As a second contribution, we investigate the ATL-like fragment ATL_{lp} of ATL^*_{lp} which, to the best of our knowledge, has not been considered so far in the literature. In fact, a past perfect recall extension of ATL with a semantics equivalent to that of ATL_{lp} has been studied in [13] under an *imperfect information setting*, and in particular, for such a setting, a model-checking algorithm of non-elementary complexity is provided [13]. We show that ATL_{lp} is strictly more expressive than ATL, and interestingly, and perhaps surprisingly, it can be exponentially more succinct than the more expressive logic ATL^*. Moreover, we establish that both satisfiability and model-checking for the logic ATL_{lp} are EXPTIME-complete. Hence, while ATL_{lp} satisfiability has the same complexity as ATL satisfiability, model-checking against ATL_{lp} is exponentially harder than ATL model-checking. The upper bounds are obtained by an automata-theoretic framework based on the use of a subclass of Büchi alternating automata for CGS (Büchi ACG) [29], called *ACG with satellites* and introduced in [26].

2 Preliminaries

We fix the following notations. Let AP be a finite non-empty set of atomic propositions, Ag be a finite non-empty set of agents, and Ac be a finite non-empty set of actions that can be taken by agents. Given a set $A \subseteq Ag$ of agents, an A-decision d_A is an element in Ac^A assigning to each agent $ag \in A$ an action $d_A(ag)$. For $A, A' \subseteq Ag$ such that $A \cap A' = \emptyset$, an A-decision d_A and A'-decision $d_{A'}$, $d_A \cup d_{A'}$ denotes the $(A \cup A')$-decision defined in the obvious way. Let $Dc = Ac^{Ag}$ be the set of *full decisions* of all the agents in Ag.

Let $\mathbb{N}$ be the set of natural numbers. For all $i, j \in \mathbb{N}$, with $i \leq j$, $[i, j]$ denotes the set of natural numbers h such that $i \leq h \leq j$. Let w be a finite or infinite word over some alphabet Σ. By $|w|$ we denote the length of w (we write $|w| = \infty$ if w is infinite). For all $i, j \in \mathbb{N}$, with $i \leq j < |w|$, $w(i)$ denotes the i^{th} letter of w, and $w[i, j]$ the infix of w between positions i and j, i.e., the finite word $w(i)w(i+1)\dots w(j)$.

Given a set Υ of directions, an (*infinite*) Υ*-tree* T is a prefix closed subset of Υ^* such that for all $x \in T$, $x \cdot \gamma \in T$ for some $\gamma \in \Upsilon$. Elements of T are called nodes and ε is the root of T. For $x \in T$, the set of children of x in T is the set of nodes of the form $x \cdot \gamma$ for some $\gamma \in \Upsilon$. A path of T is a non-empty finite or infinite sequence π of nodes such that $\pi(i)$ is a child in T of $\pi(i-1)$ for all $0 < i < |\pi|$. A path π of T is *initial* if it starts at the root, i.e. $\pi(0) = \varepsilon$.

For an alphabet Σ, a Σ-labeled Υ-tree is a pair $\langle T, Lab \rangle$ consisting of a Υ-tree and a labelling $Lab : T \mapsto \Sigma$ assigning to each node in T a symbol in Σ. We extend the labeling Lab to paths in the obvious way, i.e. $Lab(\pi)$ denotes the word over Σ of length $|\pi|$ given by $Lab(\pi(0))Lab(\pi(1))\dots$. W.l.o.g. we focus on labeled trees with a finite branching degree.

2.1 Concurrent Game Structures

Concurrent game structures (CGS, for short) [2] generalize Kripke structures to a setting with multiple agents (or players). They can be viewed as multi-player arenas in which players perform concurrent actions, chosen strategically as a function of the history of the game.

▶ **Definition 1** (CGS). A CGS (over AP, Ag, and Ac) is a tuple $\mathcal{G} = \langle S, s_0, Lab, \tau\rangle$, where S is a set of states, $s_0 \in S$ is the initial state, $Lab : S \mapsto 2^{AP}$ maps each state to a set of atomic propositions, and $\tau : S \times Dc \mapsto S$ is a transition function that maps a state and a full decision to a target state. The CGS $\mathcal{G}$ is finite if S is finite. A *state s is controlled by an agent ag* if for all $\{ag\}$-decisions d and counter-decisions $d_1, d_2 \in Ac^{Ag\setminus\{ag\}}$, $\tau(s, d \cup d_1) = \tau(s, d \cup d_2)$.

We now recall the notion of *strategy* and *counter strategy* in a CGS $\mathcal{G} = \langle S, s_0, Lab, \tau\rangle$. For a state s, the set of successors of s is the set of states s' such that $s' = \tau(s, d)$ for some full decision d. A *play* is an infinite sequence of states $s_1 s_2 \dots$ such that s_{i+1} is a successor of s_i for all $i \geq 1$. An *history* (or *track*) ν is a non-empty prefix of some play. We denote by $lst(\nu)$ the last state of ν. Let Trk be the set of tracks in $\mathcal{G}$. Given a set $A \subseteq Ag$ of agents, a *strategy for A* is a mapping $f_A : Trk \mapsto Ac^A$ assigning to each track ν an A-decision. For a track ν, the set $out(\nu, f_A)$ of *plays consistent with f_A and ν* (also called *outcomes of f_A from ν*) is the non-empty set of plays of the form $\pi = \nu' \cdot s_1 s_2 \dots$ such that $\nu' \cdot s_1 = \nu$ and for all $i \geq 1$, there is a decision $d \in Ac^{Ag\setminus A}$ of agents in $Ag \setminus A$ such that $s_{i+1} = \tau(s_i, f_A(\nu' \cdot s_1 \dots s_i) \cup d)$.

Thus, the *outcome* function $out(\nu, f_A)$ returns the set of all the plays having ν as prefix that can occur when agents A execute strategy f_A from the history ν on.

A *counter strategy* is a mapping $f_A^c : Trk \times Ac^A \mapsto Ac^{Ag\setminus A}$ from tracks and decisions of the agents in A to decisions of the agents in $Ag \setminus A$. For a track ν, the set $out(\nu, f_A^c)$ of *plays consistent with the counter strategy f_A^c and ν* is the non-empty set of plays of the form $\pi = \nu' \cdot s_1 s_2 \dots$ such that $\nu' \cdot s_1 = \nu$ and for all $i \geq 1$, there is a decision $d \in Ac^A$ of agents in A so that $s_{i+1} = \tau(s_i, f_A^c(\nu' \cdot s_1 \dots s_i, d) \cup d)$. For a state s, an *s-play* is a play starting from state s. We are interested in the computation trees induced by CGS.

▶ **Definition 2.** For a set Υ of directions, a *Concurrent Game Υ-Tree* (Υ-CGT) is a CGS $\langle T, \varepsilon, Lab, \tau\rangle$, where $\langle T, Lab\rangle$ is a 2^{AP}-labeled Υ-tree, and for each node $x \in T$, the set of successors of x corresponds to the set of children of x in T. Every CGS $\mathcal{G} = \langle S, s_0, Lab, \tau\rangle$ induces a S-CGT, called *computation tree of $\mathcal{G}$* and denoted by $Unw(\mathcal{G})$, obtained by unwinding $\mathcal{G}$ from the initial state in the usual way. Formally, $Unw(\mathcal{G}) = \langle T, \varepsilon, Lab', \tau'\rangle$, where T is the set of elements ν in S^* such that $s_0 \cdot \nu$ is a track of $\mathcal{G}$, and for all $\nu \in T$ and $d \in Dc$, $Lab'(\nu) = Lab(lst(\nu))$ and $\tau'(\nu, d) = \nu \cdot \tau(lst(\nu), d)$, where $lst(\varepsilon) = s_0$. A *Concurrent Game Tree* (CGT, for short) is a Υ-CGT for some set Υ of directions.

2.2 Alternating-time temporal logic with linear past

In this section, we recall the "memoryful" linear-past extension of alternating-time temporal logic ATL*, introduced in [26] and, here, denoted by ATL$^*_{lp}$. For the given sets AP and Ag of atomic propositions and agents, the set of ATL$^*_{lp}$ *path* formulas φ are defined as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \mathsf{X}^-\varphi \mid \varphi\,\mathsf{U}\,\varphi \mid \varphi\,\mathsf{U}^-\,\varphi \mid \langle\langle A\rangle\rangle\varphi$$

where $p \in AP$, $A \subseteq Ag$, X and U are the standard "next" and "until" linear temporal modalities, X^- ("previous") and U^- ("since") are their past counterparts, respectively, and $\langle\langle A\rangle\rangle$ is the "existential strategic quantifier" parameterized by a set of agents. Formula $\langle\langle A\rangle\rangle\varphi$ expresses that the group of agents A has a collective strategy to enforce the temporal property φ. We also use some shorthands:

- the *eventually* modality $\mathsf{F}\varphi := \top\,\mathsf{U}\,\varphi$ and its past counterpart $\mathsf{F}^-\varphi := \top\,\mathsf{U}^-\,\varphi$,
- the *always* modality $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$ and its past counterpart $\mathsf{G}^-\varphi := \neg\mathsf{F}^-\neg\varphi$,
- the *release* modality $\varphi_1\mathsf{R}\varphi_2 := \neg(\neg\varphi_1\mathsf{U}\neg\varphi_2)$.

A *state formula* is a formula where each temporal modality is in the scope of a strategic quantifier. A *basic formula* is a state formula of the form $\langle\langle A\rangle\rangle\varphi$. The language ATL^*_{lp} is the set of state formulas. The size $|\varphi|$ of a formula φ is the number of distinct subformulas of φ.

The logic ATL^*_{lp} is interpreted over concurrent game trees (CGT) $\mathcal{T} = \langle T, \varepsilon, Lab, \tau\rangle$. For a path formula φ, an initial infinite path π of T (i.e., an ε-play of $\mathcal{T}$) and a position $i \geq 0$, the satisfaction relation $(\mathcal{T}, \pi, i) \models \varphi$, indicating that φ holds at position i along π, is inductively defined as follows (we omit the clauses for the Boolean connectives which are standard):

$$
\begin{array}{ll}
(\mathcal{T},\pi,i) \models p & \Leftrightarrow\ p \in Lab(\pi(i)),\\
(\mathcal{T},\pi,i) \models \mathsf{X}\varphi & \Leftrightarrow\ (\mathcal{T},\pi,i+1) \models \varphi\\
(\mathcal{T},,\pi,i) \models \mathsf{X}^{-}\varphi & \Leftrightarrow\ i>0 \text{ and } (\mathcal{T},\pi,i-1) \models \varphi\\
(\mathcal{T},\pi,i) \models \varphi_1 \mathsf{U}\, \varphi_2 & \Leftrightarrow\ \text{there is } j \geq i : (\mathcal{T},\pi,j) \models \varphi_2 \text{ and } (\mathcal{T},\pi,k) \models \varphi_1 \text{ for all } i \leq k < j\\
(\mathcal{T},\pi,i) \models \varphi_1 \mathsf{U}^{-}\, \varphi_2 & \Leftrightarrow\ \text{there is } j \leq i : (\mathcal{T},\pi,j) \models \varphi_2 \text{ and } (\mathcal{T},\pi,k) \models \varphi_1 \text{ for all } j < k \leq i\\
(\mathcal{T},\pi,i) \models \langle\langle A\rangle\rangle\varphi & \Leftrightarrow\ \text{for some } A\text{-strategy } f_A : (\mathcal{T},\pi',i) \models \varphi \text{ for all } \pi' \in out(\pi[0,i], f_A)
\end{array}
$$

Note that for each node $x \in T$ and ε-play π visiting x, π visits x exactly at position $|x|$ (the distance of node x from the root). For a node x of T, we write $(\mathcal{T}, x) \models \varphi$ to denote that there is an ε-play π visiting x such that $(\mathcal{T}, \pi, |x|) \models \varphi$. Note that if φ is a state formula, then for all ε-plays π and π' which visit node x, it holds that $(\mathcal{T}, \pi, |x|) \models \varphi$ iff $(\mathcal{T}, \pi', |x|) \models \varphi$. A CGT $\mathcal{T}$ satisfies a formula φ (we also say that $\mathcal{T}$ is a model of φ) if $(\mathcal{T}, \varepsilon) \models \varphi$. Two formulas φ and φ' are equivalent if they admit the same models.

It is worth noting that in the valuation of a strategy quantifier $\langle\langle A\rangle\rangle$, while in standard ATL^*, play quantification ranges over the plays consistent with the selected strategy which start at the current node (*local semantics*), in ATL^*_{lp} play quantification ranges over the plays obtained by prefixing the outcomes of the selected strategy from the current node with the history that starts at the root and leads to the current node (*memoryful semantics*). Evidently, for formulas which do not contain past temporal modalities, the standard local ATL^*-semantics is equivalent to the memoryful semantics.[1] Moreover, the known memoryful linear-past extension CTL^*_{lp} [18] of CTL^* corresponds to the fragment of ATL^*_{lp} where only the strategic modalities $\langle\langle Ag\rangle\rangle$ (equivalent to the existential path quantifier E) and $\langle\langle\emptyset\rangle\rangle$ (equivalent to the universal path quantifier A) are exploited. Note that CTL^*_{lp} is interpreted over 2^{AP}-labeled trees which correspond to one-agent CGT. We denote by PATL^* (resp., PCTL^*) the logic having the same syntax as ATL^*_{lp} (resp., CTL^*_{lp}) but interpreted under the local ATL^*-semantics (resp., the local CTL^*-semantics). Intuitively, in PATL^* and PCTL^* the past cannot go beyond the present.

In the following, we also consider the past extension LTL_p of standard (future) LTL [27] which syntactically corresponds to the fragment of ATL^*_{lp} where strategy quantifiers are disallowed. LTL_p formulas ψ over AP are interpreted over infinite words over 2^{AP}. For such words w and positions i, we write $(w, i) \models_{\mathsf{LTL}} \psi$ to denote that ψ holds at position i along w according to the standard LTL (LTL_p) semantics. A *pure past* LTL_p formula is an LTL_p formula which does not contain occurrences of future temporal modalities.

2.3 The logic ATL_{lp}

The logics CTL [10] and ATL [2] are well-known syntactical fragments of CTL^* and ATL^* which have received a lot of attention due to the existence of polynomial-time algorithms which solve the associated (finite) model-checking problem. In particular, CTL (resp., ATL) is obtained

[1] ATL^* formulas φ are usually interpreted over CGS $\mathcal{G}$. However, ATL^* is insensitive to unwinding: $\mathcal{G}$ is a model of φ iff the CGT $Unw(\mathcal{G})$ is a model of φ.

from the more expressive logic CTL* (resp., ATL*) by requiring that each temporal modality is immediately preceded by a path quantifier (resp., strategy quantifier). In this section, we introduce the ATL-like fragment of ATL$^*_{lp}$ which corresponds to the alternating-time version of the linear-past extension CTL$_{lp}$ of CTL introduced in [18].

Since past is linear, play quantification of past-time modalities is redundant. Thus, in ATL$_{lp}$, we require the future temporal modalities X, U, and R to be preceded by a strategic quantifier, but we impose no equivalent restriction on the past temporal modalities. Formally, the set of ATL$_{lp}$ formulas ψ is the fragment of ATL$^*_{lp}$ defined by the following grammar:

$$\psi ::= \top \mid p \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}^-\psi \mid \psi\,\mathsf{U}^-\,\psi \mid \langle\langle A\rangle\rangle \mathsf{X}\psi \mid \langle\langle A\rangle\rangle(\psi \mathsf{U} \psi) \mid \langle\langle A\rangle\rangle(\psi \mathsf{R} \psi)$$

Note that for an ATL$_{lp}$ formula ψ and for all ε-plays π and π' which visit a node x of a CGT $\mathcal{T}$, $(\mathcal{T}, \pi, |x|) \models \psi$ iff $(\mathcal{T}, \pi', |x|) \models \psi$. As an example, let us consider the formula $\psi := \neg\langle\langle A\rangle\rangle\neg\mathsf{G}\langle\langle B\rangle\rangle\mathsf{G}[(\mathit{grant} \rightarrow \mathsf{F}^-\mathit{req}) \wedge \mathsf{G}^-(\mathit{grant} \rightarrow \mathsf{F}^-\mathit{req})]$ which specifies that in each node x reached by a strategy of agents in A, the agents in B can enforce that along a full computation (starting from the root) and visiting x, every grant is preceded by some request.

While ATL$^*_{lp}$ is known to have the same expressiveness as ATL* [26], ATL$_{lp}$ turns out to be strictly more expressive than ATL. In particular, the following holds.

▶ **Proposition 1** (Expressiveness of ATL$_{lp}$). *ATL$_{lp}$ is strictly more expressive than ATL and strictly less expressive than ATL*.*

Proof. We exploit known expressiveness results about the logic CTL$_{lp}$. In particular, in [18], it is shown that the CTL$_{lp}$ formula $\mathsf{AXAF}(p \wedge \mathsf{X}^-p)$ has no equivalent CTL formula, and the CTL* formula $\mathsf{EGF}p$ has no equivalent CTL$_{lp}$ formula. Note that over 1-agent CGT (corresponding to labeled trees), a CTL$_{lp}$ formula ψ is equivalent to the ATL$_{lp}$ formula obtained from ψ by replacing the path quantifiers E and A with the strategy quantifiers $\langle\langle Ag\rangle\rangle$ and $\langle\langle\emptyset\rangle\rangle$. It follows that the ATL$_{lp}$ formula $\langle\langle\emptyset\rangle\rangle\mathsf{X}\langle\langle\emptyset\rangle\rangle\mathsf{F}(p \wedge \mathsf{X}^-p)$ cannot expressed in ATL, and the ATL* formula $\langle\langle Ag\rangle\rangle\mathsf{GF}p$ cannot be expressed in ATL$_{lp}$. Thus, since ATL$_{lp}$ is a fragment of ATL$^*_{lp}$, and ATL$^*_{lp}$ and ATL* have the same expressiveness, the result follows. ◀

We consider the following decision problems:

- *Satisfiability*: has a given ATL$_{lp}$ state formula a model?
- *(Finite) Model Checking*: given a finite CGS $\mathcal{G}$ and an ATL$_{lp}$ state formula ψ, is $\mathit{Unw}(\mathcal{G})$ a model of ψ?

3 Succinctness gap between memoryful past and local past

It is known that PCTL* and CTL$^*_{lp}$ have the same expressiveness as CTL* [18] and there are translations from PCTL* and CTL$^*_{lp}$ into CTL* of non-elementary complexity [18] based on the separation theorem for LTL$_p$ [12]. On the other hand, the ability to refer to the past makes both PCTL* and CTL$^*_{lp}$ exponentially more succinct than CTL*. Analogous results hold for the logics PATL* and ATL$^*_{lp}$ when compared to ATL* [26]. Note that as observed in [26], the succinctness gap between the past extensions of CTL* (resp., ATL*) and CTL* (resp., ATL*) are a consequence of the fact that LTL$_p$ is exponentially more succinct than LTL [22]. Interestingly, and perhaps surprisingly, we can establish a similar result for the logic CTL$_{lp}$ (resp., ATL$_{lp}$) when compared to the more expressive logic CTL* (resp., ATL*).

▶ **Theorem 3.** *CTL$_{lp}$ (resp., ATL$_{lp}$) can be exponentially more succinct than CTL* (resp., ATL*).*

Proof. We focus on ATL_{lp} and ATL^* (a similar result holds for the logics CTL_{lp} and CTL^*). For each $n \geq 1$, let $p_0, p_1, \ldots, p_n$ be $n+1$ atomic propositions, and ψ_n be the LTL_p formula $\mathsf{G}\big(\bigwedge_{i=1}^{i=n}(p_i \leftrightarrow \mathsf{F}^-(\neg \mathsf{X}^- \top \wedge p_i)) \longrightarrow (p_0 \leftrightarrow \mathsf{F}^-(\neg \mathsf{X}^- \top \wedge p_0)\big)$. Note that $|\psi_n| = O(n)$. On the other hand, it is shown in [22] that every LTL formula which is equivalent to ψ_n has size at least $2^{\Omega(n)}$. Now, we observe that $\langle\langle\emptyset\rangle\rangle\psi_n$ is an ATL_{lp} formula. Let φ be an ATL^* formula equivalent to $\langle\langle\emptyset\rangle\rangle\psi_n$, and φ_{LTL} be the LTL formula obtained from φ by removing the occurrences of strategy quantifiers $\langle\langle A\rangle\rangle$. Let Π be the set of CGT over 2^{AP} having exactly one ε-play. Evidently, we have that for each $\mathcal{T} \in \Pi$, $\mathcal{T}$ is a model of $\langle\langle\emptyset\rangle\rangle\psi_n$ (resp., φ) iff $Lab(\pi_{\mathcal{T}})$ is a model of the LTL_p formula ψ_n (resp., LTL formula φ_{LTL}), where $\pi_{\mathcal{T}}$ is the unique ε-play of $\mathcal{T}$. Thus, since φ and $\langle\langle\emptyset\rangle\rangle\psi_n$ are equivalent, and there is a bijection between Π and the set of infinite words over 2^{AP}, we obtain that φ_{LTL} is equivalent to ψ_n, hence, φ_{LTL} has size at least $2^{\Omega(n)}$. This entails that $|\varphi|$ is at least $2^{\Omega(n)}$, and we are done. ◀

To the best of our knowledge, no succinctness gap is known between the "memoryful" past view in CTL^*_{lp} (resp., ATL^*_{lp}) and the 'local' past view in PCTL^* (resp., PATL^*). In this section, we address this issue by establishing the following.

▶ **Theorem 4.** *CTL^*_{lp} (resp., ATL^*_{lp}) can be exponentially more succinct than $PCTL^*$ (resp., $PATL^*$).*

For the part of Theorem 4 concerning CTL^*_{lp} and PCTL^*, we prove the following.

▶ **Theorem 5.** *For each $n \geq 1$, there exists a CTL^*_{lp} formula φ_n of size $O(n)$ such that every equivalent $PCTL^*$ formula has size at least $2^{\Omega(n)}$.*

As a corollary of Theorem 5, we deduce the part of Theorem 4 for ATL^*_{lp} and PATL^*.

▶ **Corollary 6.** *Let Ag be a set of agents. For each $n \geq 1$, there exists an ATL^*_{lp} formula φ'_n over Ag of size $O(n)$ such that every equivalent $PATL^*$ formula has size at least $2^{\Omega(n)}$.*

Proof. Fix a set Ag of agents and an agent $ag \in Ag$. Let $n \geq 1$ and φ_n be the CTL^*_{lp} formula of size $O(n)$ satisfying the statement of Theorem 5, and AP_n the set of propositions occurring in φ_n. We denote by φ'_n the ATL^*_{lp} version over Ag of φ_n, i.e., the formula obtained from φ_n, by replacing the path quantifier E (resp., A) with the strategy quantifier $\langle\langle Ag\rangle\rangle$ (resp., $\langle\langle\emptyset\rangle\rangle$). Note that $|\varphi'_n| = |\varphi_n|$. We can associate with each 2^{AP_n}-labeled tree $\mathcal{T} = \langle T, Lab\rangle$ a CGT of the form $f(\mathcal{T}) = \langle T, \varepsilon, Lab, \tau\rangle$ over Ag such that each node in $f(\mathcal{T})$ is controlled by agent ag.

Now, let ψ'_n be a PATL^* formula over Ag and AP_n equivalent to φ'_n. We show that the size of ψ'_n is at least $2^{\Omega(n)}$, hence, the result follows. We denote by ψ_n the PCTL^* formula obtained from ψ'_n by replacing each strategy quantifier $\langle\langle A\rangle\rangle$ with E if $ag \in A$, and with A otherwise. By construction, for each 2^{AP_n}-labeled tree $\mathcal{T} = \langle T, Lab\rangle$, $\mathcal{T}$ is a model of ψ_n (resp., φ_n) iff $f(\mathcal{T})$ is a model of ψ'_n (resp., φ'_n). Thus, since ψ'_n and φ'_n are equivalent, we obtain that the PCTL^* formula ψ_n is equivalent to φ_n. By Theorem 5, the size of ψ_n is at least $2^{\Omega(n)}$, hence, being $|\psi'_n| \geq |\psi_n|$, the result follows. ◀

In the rest of this section, we provide a proof of Theorem 5. Fix $n \geq 1$. Let $AP_n := \{p_0, p_1, \ldots, p_n\}$ be a set consisting of $n+1$ distinct atomic propositions, $AP'_n := AP_n \setminus \{p_0\}$ be the set obtained from AP_n by removing proposition p_0, and $\# \notin AP_n$ be a special atomic proposition. Fix an ordering $a_1, \ldots, a_{2^n}$ of the 2^n distinct symbols of the alphabet $2^{AP'_n}$. An *n-word* is a word of the form $w = b_1 \ldots b_{2^n} \cdot \{\#\}$ over $2^{AP_n \cup \{\#\}}$ of length $2^n + 1$ such that there is a set $K \subseteq [1, 2^n]$ so that for all $j \in [1, 2^n]$, $b_j = a_j$ if $j \notin K$ and $b_j = a_j \cup \{p_0\}$ otherwise. Intuitively an n-word is obtained from $\nu = a_1, \ldots, a_{2^n} \cdot \{\#\}$ by augmenting some

non-last symbols in ν with proposition p_0. Note that there are exactly 2^{2^n} distinct n-words. Next, we define the notions of n-block and n-configuration. Let $check \notin AP_n \cup \{\#\}$ be an additional atomic proposition.

▶ **Definition 7** (n-blocks). An n-block is a $2^{AP_n \cup \{check,\#\}}$-labeled tree $\langle T, Lab\rangle$ such that:

- there is a finite path π from the root so that $Lab(\pi)$ is an n-word and each node x of π has exactly one child c_x which is not visited by π. Moreover, there is exactly one infinite path starting from c_x, and this path has label $\{check\}^\omega$.

Intuitively, an n-block is a labeled tree consisting of an n-word augmented with *check*-branches starting from the n-word nodes.

▶ **Definition 8** (n-configurations). An n-configuration is a $2^{AP_n \cup \{check,\#\}}$-labeled tree $\mathcal{T} = \langle T, Lab\rangle$ such that for some $k \geq 1$, there exist $k+1$ n-blocks $\mathcal{T}_0, \ldots, \mathcal{T}_k$ so that $\mathcal{T}$ is obtained by connecting the $k+1$ n-blocks $\mathcal{T}_0, \ldots, \mathcal{T}_k$ as follows:

- the $\{\#\}$-node of $\mathcal{T}_0$ has as children the roots of $\mathcal{T}_1 \ldots \mathcal{T}_k$.

We say that $\mathcal{T}_0$ is the *master* n-block of $\mathcal{T}$, and $\mathcal{T}_1 \ldots, \mathcal{T}_k$ are the *slave* n-blocks of $\mathcal{T}$. The n-configuration $\mathcal{T}$ is *well-formed* if the n-word associated with the master coincides with the n-word of some slave.

We now show that one can construct a CTL^*_{lp} formula φ_n of size $O(n)$ which distinguishes the n-configurations which are well-formed from the non-well-formed ones.

▶ **Lemma 9.** *For each $n \geq 1$, there exists a CTL^*_{lp} formula φ_n over $2^{AP_n \cup \{check,\#\}}$ having size $O(n)$ such that for each n-configuration $\langle T, Lab\rangle$, $\langle T, Lab\rangle$ is a model of φ_n if and only if $\langle T, Lab\rangle$ is well-formed.*

Proof. Given $n \geq 1$, we construct a CTL^*_{lp} formula φ_n which is satisfied by an n-configuration $\mathcal{T}$ iff there exists an infinite path π of $\mathcal{T}$ from the root such that:
(1) π visits the n-word of the master and the n-word of some slave, and
(2) the two n-words visited by π coincide.

The crucial property which allow us to define a CTL^*_{lp} formula of size $O(n)$ satisfying Requirement (2) is that for each node x of π associated with the n-word of the slave, there is a child y of x whose subtree reduces to a path labelled by the word $(\{check\})^\omega$. Thus, the "memoryful" semantics of the path quantifier E allow us to "select" the prefix of π until node x, by using the *check*-node y as marker, and to require that such a prefix satisfies the following

- (*) for each node z along the n-word of the master, if the labels of z and x agree on propositions $p_1, \ldots, p_n$, then they also agree on proposition p_0.

Thus, the CTL^*_{lp} formula φ_n is $\mathsf{EF}\big[\# \wedge (\mathsf{XF}\#) \wedge \mathsf{G}\big((\neg\# \wedge \neg check) \rightarrow \mathsf{E}(\mathsf{X}check \wedge \psi_n)\big)\big]$ where ψ_n is an LTL_p formula ensuring Requirement (*) for the selected 2^{AP_n}-labeled node of the selected slave: $\psi_n := \mathsf{F}^-\big[\# \wedge \mathsf{X}^-\mathsf{G}^-\big(\bigwedge_{i=1}^{i=n} \bigwedge_{q_i \in \{p_i, \neg p_i\}} (q_i \rightarrow \mathsf{F}(\neg check \wedge \mathsf{X}check \wedge q_i)) \longrightarrow \bigwedge_{q_0 \in \{p_0, \neg p_0\}} (q_0 \rightarrow \mathsf{F}(\neg check \wedge \mathsf{X}check \wedge q_0))\big)\big]$. ◀

Next, in order to complete the proof of Theorem 5, we show that for all $n \geq 1$, every PCTL^* formula equivalent to the CTL^*_{lp} formula φ_n of Lemma 9 has size at least $2^{\Omega(n)}$. For this, we exploit the well-known result concerning the translation of CTL^* (and PCTL^* as well) formulas into equivalent parity symmetric alternating tree-automata (parity SATA). SATA [16] are a variation of classical (asymmetric) alternating automata in which it is not

necessary to specify the direction (i.e., the choice of the children) of the tree on which a copy is sent. In fact, through existential and universal moves, it is possible to send a copy of the automaton, starting from a node of the input tree, to one or all its successors. We now recall syntax and semantics of parity SATA. For a set X, $\mathbb{B}^+(X)$ denotes the set of *positive* Boolean formulas over X, i.e. Boolean formulas built from elements in X using $\vee$ and $\wedge$ (we also allow the formulas `true` and `false`).

A parity SATA over a finite alphabet Σ is a tuple $\mathcal{A} = \langle \Sigma, q_0, Q, \delta, \Omega \rangle$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times 2^{AP} \to \mathbb{B}^+(Q \times \{\Box, \Diamond\})$ is the transition function, and $\Omega : Q \mapsto \mathbb{N}$ is a parity acceptance condition over Q assigning to each state an integer (*color*). Intuitively, a target of a move of $\mathcal{A}$ is encoded by an element in $Q \times \{\Box, \Diamond\}$. An atom $(q, \Diamond)$ means that from the current node x of the Σ-labeled input tree, $\mathcal{A}$ moves to some child of x and the state is updated to q. On the other hand, an atom $(q, \Box)$ means that from the current node x, the automaton splits in multiple copies and, for each child x' of x in the input tree, one of such copies moves to node x' and the state is updated to q.

Formally, for a Σ-labeled tree $\mathcal{T} = \langle T, Lab \rangle$, a run of $\mathcal{A}$ over $\mathcal{T}$ is a $(Q \times T)$-labeled $\mathbb{N}$-tree $r = \langle T_r, Lab_r \rangle$, where each node of T_r labelled by (q, x) describes a copy of the automaton that is in state q and reads the node x of T. Moreover, we require that $r(\varepsilon) = (q_0, \varepsilon)$ (initially, the automaton is in state q_0 reading the root node), and for each $y \in T_r$ with $r(y) = (q, x)$, there is a (possibly empty) *minimal* set $H \subseteq Q \times \{\Box, \Diamond\}$ satisfying $\delta(q, Lab(x))$ such that the set $L(y)$ of labels of children of y in T is the smallest set satisfying the following conditions: for all atoms $at \in H$,

- if $at = (q', \Diamond)$, then for some child x' of x in T, $(q', x') \in L(y)$;
- if $at = (q', \Box)$, then for each child x' of x in T, $(q', x') \in L(y)$.

The run r is accepting if for all infinite paths π starting from the root, the smallest color of the states in Q that occur infinitely often along $Lab_r(\pi)$ is *even*. A Σ-labeled tree $\mathcal{T} = \langle T, Lab \rangle$ is accepted by $\mathcal{A}$ if there is an accepting run over $\mathcal{T}$. The following is a well-known result [21].

▶ **Proposition 2** ([21]). *Given a PCTL* formula ψ, one can construct a parity SATA with $2^{O(|\psi|)}$ states accepting the set of models of ψ.*

Theorem 5 directly follows from Lemma 9, Proposition 2, and the following result.

▶ **Lemma 10.** *Let $n \geq 1$ and $\mathcal{A}_n$ be a parity SATA over $2^{AP_n \cup \{check, \#\}}$ accepting the set of models of the CTL$^*_{lp}$ formula φ_n in Lemma 9. Then, $\mathcal{A}_n$ has at least 2^{2^n} states.*

Proof. Let $n \geq 1$ and $\mathcal{A}_n = \langle 2^{AP_n \cup \{check, \#\}}, q_0, Q, \delta, \Omega \rangle$ as in the statement of the lemma. For each state $q \in Q$, we denote by $\mathcal{A}_n^q$ the parity SATA $\langle 2^{AP_n \cup \{check, \#\}}, q, Q, \delta, \Omega \rangle$, i.e., $\mathcal{A}_n^q$ is obtained from $\mathcal{A}_n$ by considering q as initial state instead of q_0. We show the following.

Claim. For each n-word w, there is a state q_w of $\mathcal{A}_n$ such that:

- there is an accepting run of $\mathcal{A}_n^{q_w}$ over the n-block associated with w;
- for each n-word w' distinct from w, there is no accepting run of $\mathcal{A}_n^{q_w}$ over the n-block related to w'.

By the claim above, it follows that there is a bijection between the set of n-words and a subset of the set Q of $\mathcal{A}_n$-states. Thus, since the number of distinct n-words is 2^{2^n}, the result follows. It remains to prove the claim. An n-configuration $\mathcal{T}$ is *complete* if $\mathcal{T}$ has 2^{2^n} slaves and for each n-word w, there is a slave of $\mathcal{T}$ associated with w. Fix an n-word w and let $\mathcal{T}_w$ be the complete n-configuration whose master matches the n-word w. Since $\mathcal{T}_w$ is well-formed, by hypothesis and Lemma 9, there is an accepting run r_w of $\mathcal{A}_n$ over $\mathcal{T}_w$. Let

$x_\#$ be the $\{\#\}$-labeled node of the $\mathcal{T}_w$-master, and $x_1, \ldots, x_{2^{2^n}}$ be the 2^{2^n} children of $x_\#$ which correspond to the roots of the $\mathcal{T}_w$-slaves. Without loss of generality, assume that x_1 is the root of the slave associated with the n-word w. Moreover, let $\mathcal{T}_w^{\setminus w}$ be the n-configuration obtained from $\mathcal{T}_w$ by removing the subtree rooted at node x_1 (i.e., the slave associated with w), and $Q_w^\Diamond$ be the set of states associated with the copies of the run r_w reading node x_1 and obtained by *existential* moves. Since $\mathcal{T}_w^{\setminus w}$ is not well-formed, by hypothesis and Lemma 9, there is no accepting run of $\mathcal{A}_n$ over $\mathcal{T}_w^{\setminus w}$.

Since the parity acceptance condition is prefix independent, by construction, for each $q \in Q_w^\Diamond$, there exists an accepting run of $\mathcal{A}_n^q$ over the n-block associated with w (in particular, the subtree rooted at the node of the run r_w associated with the copy of $\mathcal{A}_n$ reading node x_1 in state q is such a run). Thus, since w is an arbitrary n-word, in order to prove the claim, it suffices to show that there exists a state $q_w \in Q_w^\Diamond$ such that for each n-word w' distinct from w, there is no accepting run of $\mathcal{A}_n^{q_w}$ over the n-block related to w'. We assume the contrary and derive a contradiction. Let $Q_w^\Diamond = \{q_1, \ldots, q_k\}$. By hypothesis, for all $i \in [1, k]$, there exists an n-word $w_i \neq w$ and an accepting run r_i of $\mathcal{A}_n^{q_i}$ over the n-block related to w_i. Let r' be the labeled tree obtained from the accepting run r_w by replacing for all $i \in [1, n]$ and node z_i of r_w associated with the copy of $\mathcal{A}_n$ reading node x_1 in state q_i, the subtree rooted at node z_i with a copy of the run r_i. Since for each $i \in [1, k]$, the n-configuration $\mathcal{T}_w^{\setminus w}$ has a slave associated with the n-word w_i, by construction, we obtain that r' is an accepting run of $\mathcal{A}_n$ over $\mathcal{T}_w^{\setminus w}$, which is a contradiction, and the result follows. ◀

4 Decision procedures for ATL$_{lp}$

In this section, we establish that both satisfiability and model-checking for the logic ATL$_{lp}$ are EXPTIME-complete. The upper bounds are obtained by an automata-theoretic framework based on the use of a subclass of Büchi alternating automata for CGS (Büchi ACG) [29], called *ACG with satellites* and introduced in [26]. By translating ATL$_{lp}$ formulas into equivalent Büchi ACG with satellites, we reduce model-checking (resp., satisfiability) of ATL$_{lp}$ to the membership (resp., non-emptiness) problem of such a class of automata. Notice that the symbolic algorithm, based on reachability analysis, used for solving ATL model-checking [2] cannot applied to ATL$_{lp}$. This is because, differently from ATL, the valuation of an ATL$_{lp}$ formula φ at a node x of the unwinding $Unw(\mathcal{G})$ of a finite CGS does not depend only on the state of $\mathcal{G}$ associated with node x, but also depends on the history from the root to node x.

The rest of the section is organized as follows. In Subsection 4.1, we establish a preliminary result concerning the dualization of basic ATL$_{lp}$ formulas which generalizes to linear past a similar result holding for ATL*. In Subsection 4.2, we recall the framework of ACG with satellites, and in Subsection 4.3, we solve satisfiability and model-checking of ATL$_{lp}$ by providing a translation of ATL$_{lp}$ formulas into equivalent Büchi ACG with satellites involving a single exponential blowup.

4.1 Dualization of basic ATL$_{lp}$ formulas

In order to solve the considered decision problems for the logic ATL$_{lp}$, we exploit the following preliminary result (which is known to hold for ATL* [28]).

▶ **Proposition 3.** *Given a basic ATL$_{lp}$ formula $\langle\langle A\rangle\rangle\psi$, a CGT $\mathcal{T}$, and a node x, the following holds, where ν_x is the history from the root to node x: $(\mathcal{T}, x) \models \neg\langle\langle A\rangle\rangle\psi$ if and only if there is a* counter *strategy f_A^c for A such that for all ε-plays $\pi \in out(\nu_x, f_A^c)$, $(\mathcal{T}, \pi, |x|) \models \neg\psi$.*

We now provide a proof of Proposition 3. Given two path formulas φ and φ' of ATL^*_{lp} (note that we consider the more expressive logic ATL^*_{lp}), φ and φ' are *congruent* if for every CGT $\mathcal{T}$, ε-play π of $\mathcal{T}$ and position $i \geq 0$, $(\mathcal{T}, \pi, i) \models \varphi$ if and only if $(\mathcal{T}, \pi, i) \models \varphi'$ (note that congruence is a *stronger* requirement than equivalence). We first show that given an ATL^*_{lp} path formula, it is possible to suitably separate past and future temporal modalities.

▶ **Lemma 11.** *Let φ be an ATL^*_{lp} path formula. Then, φ is congruent to a Boolean combination of ATL^* formulas and ATL^*_{lp} formulas that correspond to* pure past *LTL_p formulas over the set of propositions $AP \cup \mathcal{H}$, where $\mathcal{H}$ consists of basic ATL^* formulas.*

The proof of Lemma 11, which is provided in Appendix A, is based on the well-known separation theorem for LTL_p over infinite words [12], which states that any LTL_p formula can be effectively converted into an equivalent Boolean combination of LTL formulas and pure past LTL_p formulas. We now prove Proposition 3. Let $\langle\langle A\rangle\rangle\psi$, $\mathcal{T}$, x, and ν_x be as in Proposition 3. First, assume that there is a counter strategy f^c_A for A such that for all ε-plays $\pi \in out(\nu_x, f^c_A)$, $(\mathcal{T}, \pi, |x|) \models \neg\psi$. Since for each strategy f_A for A, $out(\nu_x, f^c_A) \cap out(\nu_x, f_A) \neq \emptyset$, we deduce that $(\mathcal{T}, x) \models \neg\langle\langle A\rangle\rangle\psi$.

For the converse direction, assume that $(\mathcal{T}, x) \models \neg\langle\langle A\rangle\rangle\psi$. By Lemma 11, the ATL_{lp} formula ψ is congruent to an ATL^*_{lp} path formula θ of the form

$$\theta := \bigvee_{\ell \in I} (\theta_{p,\ell} \wedge \theta_{f,\ell}) \tag{1}$$

such that $I \neq \emptyset$, for all $\ell \in I$, $\theta_{f,\ell}$ is a path ATL^* formula and $\theta_{p,\ell}$ corresponds to a pure past LTL_p formula over the set of propositions $AP \cup \mathcal{H}$ where $\mathcal{H}$ consists of basic ATL^* formulas. Let J_x be the set of elements $\ell \in I$ such that $(\mathcal{T}, x) \models \theta_{p,\ell}$. If $J_x = \emptyset$, then by Point 1, for each ε-play π visiting node x, we have that $(\mathcal{T}, \pi, |x|) \models \neg\theta$. Thus, since θ is congruent to ψ, the result follows. Now, assume that $J_x \neq \emptyset$. In the proof of Lemma 11 (see Appendix A), we exploit the fact that $\langle\langle A\rangle\rangle \bigvee_{\ell \in I} (\theta_{p,\ell} \wedge \theta_{f,\ell})$ is congruent to $\bigvee_{J \subseteq I \mid J \neq \emptyset} ((\bigwedge_{\ell \in J} \theta_{p,\ell}) \wedge \langle\langle A\rangle\rangle (\bigvee_{\ell \in J} \theta_{f,\ell}))$. By hypothesis, $(\mathcal{T}, x) \models \neg\langle\langle A\rangle\rangle\psi$. Hence, by definition of J_x, it follows that $(\mathcal{T}, x) \models \neg\langle\langle A\rangle\rangle(\bigvee_{\ell \in J_x} \theta_{f,\ell})$

Since $\langle\langle A\rangle\rangle(\bigvee_{\ell \in J_x} \theta_{f,\ell})$ is a basic ATL^* formula, by [28], there exists a counter strategy f^c_A for agents in A such that for all ε-plays $\pi \in out(\nu_x, f^c_A)$ and $\ell \in J_x$, $(\mathcal{T}, \pi, |x|) \models \neg\theta_{f,\ell}$.[2] Moreover, since $\theta_{p,\ell}$ is a pure past LTL_p formula over $AP \cup \mathcal{H}$, where $\mathcal{H}$ consists of basic ATL^* formulas, by definition of J_x, for all $\ell \in I \setminus J_x$ and ε-plays π visiting x, $(\mathcal{T}, \pi, |x|) \models \neg\theta_{p,\ell}$. Thus, by Point 1 and since θ is congruent to ψ, we obtain that for all ε-plays $\pi \in out(\nu_x, f^c_A)$, $(\mathcal{T}, \pi, |x|) \models \neg\psi$, which concludes the proof of Proposition 3.

4.2 Automata for ATL_{lp}

In this section, we recall the class of ACG [29] and the subclass of *ACG with satellites* [26]. ACG generalize the class of symmetric alternating automata (recalled in Section 3) by branching universally or existentially over all successors that result from the agents' decisions. Formally, a Büchi ACG over 2^{AP} and Ag is a tuple $\mathcal{A} = \langle 2^{AP}, q_0, Q, \delta, F\rangle$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times 2^{AP} \to \mathbb{B}^+(Q \times \{\Box, \Diamond\} \times 2^{Ag})$ is the transition function, and $F \subseteq Q$ is a Büchi acceptance condition on Q. The transition function δ maps

[2] The result in [28] is based on the well-known determinacy of two-players turn-based games with LTL objectives.

a state and an input letter to a positive Boolean combination of universal atoms $(q, \Box, A)$ which refer to *all* successors states for some A-decision, and existential atoms $(q, \Diamond, A)$ which refer to *some* successor state for all A-decisions.

We interpret the Büchi ACG $\mathcal{A}$ over CGT $\mathcal{T} = \langle T, \varepsilon, Lab, \tau \rangle$ on AP and Ag. A run of $\mathcal{A}$ over $\mathcal{T}$ is a $(Q \times T)$-labeled $\mathbb{N}$-tree $r = \langle T_r, Lab_r \rangle$, where each node of T_r labelled by (q, x) describes a copy of the automaton that is in the state q and reads the node x of T. Moreover, we require that $Lab_r(\varepsilon) = (q_0, \varepsilon)$ (initially, the automaton is in state q_0 reading the root node), and for each $y \in T_r$ with $Lab_r(y) = (q, x)$, there is a set $H \subseteq Q \times \{\Box, \Diamond\} \times 2^{Ag}$ such that H is model of $\delta(q, Lab(x))$ and the set L of labels associated with the children of y in T_r minimally satisfies the following conditions:

- for all universal atoms $(q', \Box, A) \in H$, there is some A-decision d_A such that for all the children x' of x in $\mathcal{T}$ which are consistent with d_A, $(q', x') \in L$;
- for all existential atoms $(q', \Diamond, A) \in H$ and for all A-decisions d_A, there is some child x' of x in $\mathcal{T}$ which is consistent with d_A such that $(q', x') \in L$.

The run r is accepting if for all infinite paths π starting from the root, $Lab_r(\pi)$ visits infinitely often elements in F. The language $\mathcal{L}(\mathcal{A})$ accepted by $\mathcal{A}$ consists of the CGT $\mathcal{T}$ over AP and Ag such that there is an accepting run of $\mathcal{A}$ over $\mathcal{T}$.

In the following, we also consider standard nondeterministic and deterministic word automata (NWA and DWA) with *no* acceptance condition (safety NWA and safety DWA) running on words over a finite alphabet Σ. Recall that a safety NWA is a tuple $\mathcal{A} = \langle \Sigma, q_0, Q, \delta \rangle$, where q_0 and Q are as for ACG, and $\delta : Q \times \Sigma \mapsto 2^Q$ is the transition function. The automaton is deterministic if for each state q and input symbol σ, $\delta(q, \sigma)$ is a singleton. A run π over an input $w \in \Sigma^\omega$ is an infinite sequence of states $\pi = q_0, q_1, \ldots$ (starting from the initial state) such that $q_{i+1} \in \delta(q_i, w(i))$ for all $i \geq 0$.

In the translation of ATL_{lp} formulas into equivalent ACG, the main technical obstacle is the handling of (memoryful) linear past which enables a reference to histories since the root of a CGT. This requires an automaton to remember the past. A simple solution would consist of using a two-way extension of ACG, but for such a class of automata, the membership and the non-emptiness problem would result harder than the ones for (one-way) ACG. Instead, we adopt the approach for CTL^*_{lp} [18] and ATL^*_{lp} [26] based on the use of alternating automata augmented with *satellites*. Specifically, ACG with satellites [26] represent a subclass of ACG in which the state space can be partitioned into two components, one of which (the *satellite*) is independent of the other, has no influence on the acceptance, and runs on all the branches of the input CGT by maintaining information about the past.

Formally, a Büchi ACG equipped with a satellite is a pair $(\mathcal{A}, \mathcal{U})$, where $\mathcal{U} = \langle 2^{AP}, s_0, S, \delta_S \rangle$, the satellite, is a safety NWA, while $\mathcal{A}$ (the main automaton) is a Büchi ACG $\mathcal{A} = \langle 2^{AP}, q_0, Q, \delta, F \rangle$ whose transition function is of the form $\delta : Q \times 2^{AP} \times S \to \mathbb{B}^+(Q \times \{\Box, \Diamond\} \times 2^{Ag})$. Intuitively, when a copy of $\mathcal{A}$ reads a node x of the input with label σ, a possible move is performed in two phases. In the first phase, the associated copy of $\mathcal{U}$ reads the letter σ and updates its state. In the second phase, $\mathcal{A}$ reads σ and the updated state of $\mathcal{U}$, and updates its state. Formally, $(\mathcal{A}, \mathcal{U})$ is equivalent to the ordinary Büchi ACG $\mathcal{A}' = \langle 2^{AP}, (q_0, s_0), Q \times S, \delta', F \times S \rangle$, where for all $(q, s) \in Q \times S$ and $\sigma \in 2^{AP}$, $\delta'((q, s), \sigma) = \bigvee_{s' \in \delta_S(s, \sigma)} f(q, \sigma, s')$ with $f(q, \sigma, s')$ being obtained from $\delta(q, \sigma, s')$ by replacing each atom $(q', \Box, A)$ (resp., $(q', \Diamond, A)$) with $((q', s'), \Box, A)$ (resp., $((q', s'), \Diamond, A)$).

The separation of the satellite $\mathcal{U}$ from the main automaton $\mathcal{A}$ allows a tighter analysis of the complexity of the nonemptiness problem. If $\mathcal{U}$ is *deterministic*, then the resulting exponential blow-up in the alternation removal (used for checking non-emptiness) only concerns the states of the main automaton. In particular, by [18, 26], the following holds.

▶ **Proposition 4** ([18, 26]). *The non-emptiness problem of a Büchi ACG $\mathcal{A}$ with n states and a* deterministic *satellite with n_S states can be solved in time $2^{O(n \log n_S + n^2 \log n)}$.*

In Subsection 4.3, we provide a translation of ATL_{lp} formulas φ into equivalent Büchi ACG accepting the set of models of φ. In such a way, model-checking against ATL_{lp} is reduced to checking for a finite CGS $\mathcal{G}$ and a Büchi ACG $\mathcal{A}$, whether $Unw(\mathcal{G})$ is accepted by $\mathcal{A}$. By standard arguments (see e.g. [21]), checking whether $Unw(\mathcal{G})$ is accepted by $\mathcal{A}$ can be reduced in polynomial-time to check non-emptiness of a Büchi alternating word automaton over a 1-letter input alphabet having $n \cdot m$ states, where n (resp., m) is the number of states in $\mathcal{G}$ (resp., $\mathcal{A}$). By [19], the latter problem can be solved in time $O((n \cdot m)^2)$.

▶ **Proposition 5.** *Given a Büchi ACG $\mathcal{A}$ with m states and a finite CGS $\mathcal{G}$ with n states, checking whether $Unw(\mathcal{G})$ is accepted by $\mathcal{A}$ can be done in time $O((n \cdot m)^2)$.*

4.3 From ATL_{lp} to Büchi ACG with satellites

In this section, in order to solve ATL_{lp} satisfiability, we show how to translate a given ATL_{lp} formula Φ into a Büchi ACG $\mathcal{A}_\Phi$ with $O(|\Phi|)$ main states equipped with a *deterministic* satellite $\mathcal{U}_\Phi$ having $2^{O(|\Phi|)}$ states accepting extended versions of the models of Φ (*extended basic models*). For the model-checking problem, we convert the pair $(\mathcal{A}_\Phi, \mathcal{U}_\Phi)$ into a ACG $\mathcal{A}'_\Phi$ with $O(|\Phi|)$ main states equipped with a nondeterministic satellite $\mathcal{U}'_\Phi$ having $2^{O(|\Phi|)}$ states accepting the set of models of Φ. We now proceed with the technical details.

Fix an ATL_{lp} *state* formula Φ over AP, and let B_Φ be the set of basic subformulas of Φ, and $BFL(\Phi)$ be the set of *first-level basic subformulas* of Φ, i.e. the basic subformulas of Φ for which there is an occurrence in Φ which is not in the scope of strategy quantifiers. Note that an ATL_{lp} formula ψ can be seen as a *pure past* LTL_p formula, denoted $[\psi]_{\mathsf{LTL}_p}$, over the set AP augmented with the set $BFL(\psi) \subseteq B_\psi$ of first-level basic subformulas of ψ. In particular, if ψ is a state formula, then $[\psi]_{\mathsf{LTL}_p}$ is a propositional formula over $AP \cup BFL(\psi)$.

For a finite set B disjoint from AP and a CGT $\mathcal{T} = \langle T, \varepsilon, Lab, \tau \rangle$ over AP, a B-labeling extension of $\mathcal{T}$ is a CGT over $AP \cup B$ of the form $\langle T, \varepsilon, Lab', \tau \rangle$, where $Lab'(x) \cap AP = Lab(x)$ for all $x \in T$. A CGT $\mathcal{T}_\Phi$ over $AP \cup B_\Phi$ is called an *extended basic model of* Φ iff the following holds, where Lab is the labeling of $\mathcal{T}_\Phi$:

- for each $\langle\langle A \rangle\rangle \psi \in B_\Phi$ and node x of $\mathcal{T}_\Phi$, $(\mathcal{T}_\Phi, x) \models \langle\langle A \rangle\rangle \psi$ if and only if $\langle\langle A \rangle\rangle \psi \in Lab(x)$;
- $Lab(\varepsilon)$ is a model of the propositional formula $[\Phi]_{\mathsf{LTL}_p}$.

Evidently, by the semantics of ATL_{lp}, the following holds.

▶ Remark. Let $\mathcal{T}$ be a CGT over AP and Φ be an ATL_{lp} state formula over AP. Then:

- $\mathcal{T}$ is a model of Φ *iff* there exists a B_Φ-labeling extension of $\mathcal{T}$ which is an extended basic model of Φ;
- there is at most one B_Φ-labeling extension of $\mathcal{T}$ which is an extended basic model of Φ.

Next, we provide a characterization of the set of extended basic models of the given state formula Φ based on a set of conditions which can be easily checked by Büchi ACG equipped with deterministic satellites. Let ψ be a pure past LTL_p formula over the set of propositions $AP \cup B_\Phi$. The closure $cl(\psi)$ of ψ is the set of pure past LTL_p formulas consisting of the subformulas of ψ and their negations (we identify $\neg\neg\psi$ with ψ). Note that $\psi, \neg\psi \in cl(\psi)$ and $|cl(\psi)| = O(|\psi|)$. It is worth noting that in the previous definition, elements in B_Φ are considered as atomic propositions. By an adaptation of the well-known translation of LTL formulas into equivalent generalized Büchi NWA [31], we obtain the following result (for details, see Appendix B).

▶ **Proposition 6.** *Let ψ be a pure past LTL_p formula over $AP \cup B_\Phi$. Then, one can construct a safety DWA $\mathcal{D}_\psi = \langle 2^{AP \cup B_\Phi}, q_0, Q \cup \{q_0\}, \delta \rangle$ such that $Q \subseteq 2^{cl(\psi)}$, there is no transition leading to q_0, and the following holds for each* infinite *word w over $2^{AP \cup B_\Phi}$:*

- *let $q_0\, C_0\, C_1 \ldots$ be the unique run of $\mathcal{D}_\psi$ over w. Then, for all $i \geq 0$ and $\theta \in cl(\psi)$, $\theta \in C_i$ if and only if $(w, i) \models \theta$.*

We are now ready to provide a characterization of the extended basic models of the ATL_{lp} state formula Φ. For each ATL_{lp} subformula ψ of Φ, let $\mathcal{D}_\psi$ be the safety DWA over $2^{AP \cup B_\Phi}$ of Proposition 6 for the pure past LTL_p formula $[\psi]_{\mathsf{LTL}_p}$. For each finite word w over $2^{AP \cup B_\Phi}$, we denote by $\mathcal{D}_\psi(w)$ the state reached by $\mathcal{D}_\psi$ on reading w.

▶ **Definition 12** (Well-formedness)**.** Let $\mathcal{T}$ be a CGT over $AP \cup B_\Phi$ with labeling Lab and $\langle\langle A \rangle\rangle \psi \in B_\Phi$ be a basic subformula of Φ. For each node x of $\mathcal{T}$, let ν_x be the track of $\mathcal{T}$ from the root to node x.

- $\mathcal{T}$ is *positively well-formed with respect to* $\langle\langle A \rangle\rangle \psi$ if for all nodes x such that $\langle\langle A \rangle\rangle \psi \in Lab(x)$, there is a strategy f_A for A such that for all the outcomes $\pi \in out(\nu_x, f_A)$:
  - Case $\psi = \mathsf{X}\psi_1$: $\psi_1 \in \mathcal{D}_{\psi_1}(Lab(\pi[0, |x| + 1]))$.
  - Case $\psi = \psi_1 \mathsf{U} \psi_2$: there is $j \geq |x|$ such that $\psi_2 \in \mathcal{D}_{\psi_2}(Lab(\pi[0, j]))$ and $\psi_1 \in \mathcal{D}_{\psi_1}(Lab(\pi[0, k]))$ for all $k \in [|x|, j - 1]$.
  - Case $\psi = \psi_1 \mathsf{R} \psi_2$: for all $j \geq |x|$ either $\psi_2 \in \mathcal{D}_{\psi_2}(Lab(\pi[0, j]))$ or $\psi_1 \in \mathcal{D}_{\psi_1}(Lab(\pi[0, k]))$ for some $k \in [|x|, j - 1]$.
- $\mathcal{T}$ is *negatively well-formed with respect to* $\langle\langle A \rangle\rangle \psi$ if for all nodes x such that $\langle\langle A \rangle\rangle \psi \notin Lab(x)$, there is a counter strategy f_A^c for A such that for all the outcomes $\pi \in out(\nu_x, f_A^c)$:
  - Case $\psi = \mathsf{X}\psi_1$: $\neg\psi_1 \in \mathcal{D}_{\psi_1}(Lab(\pi[0, |x| + 1]))$.
  - Case $\psi = \psi_1 \mathsf{U} \psi_2$: for all $j \geq |x|$, *either* $\neg\psi_2 \in \mathcal{D}_{\psi_2}(Lab(\pi[0, j]))$, *or* $\neg\psi_1 \in \mathcal{D}_{\psi_1}(Lab(\pi[0, k]))$ for some $k \in [|x|, j - 1]$.
  - Case $\psi = \psi_1 \mathsf{R} \psi_2$: there is $j \geq |x|$ such that $\neg\psi_2 \in \mathcal{D}_{\psi_2}(Lab(\pi[0, j]))$ and $\neg\psi_1 \in \mathcal{D}_{\psi_1}(Lab(\pi[0, k]))$ for all $k \in [|x|, j - 1]$.

▶ **Lemma 13.** *[Characterization of extended basic models of Φ] Let $\mathcal{T}$ be a CGT with labeling Lab over $AP \cup B_\Phi$ such that $Lab(\varepsilon)$ is a model of the propositional formula $[\Phi]_{\mathsf{LTL}_p}$. Then, $\mathcal{T}$ is an extended basic model of Φ* if and only if *for each basic subformula $\langle\langle A \rangle\rangle \psi \in B_\Phi$, $\mathcal{T}$ is both positively and negatively well-formed with respect to $\langle\langle A \rangle\rangle \psi$.*

Lemma 13 easily follows from the dualization result (Proposition 3) and Proposition 6 (see Appendix C). Based on the characterization Lemma, we obtain the following result.

▶ **Theorem 14.** *Given an ATL_{lp} state formula Φ, one can build in single exponential time a Büchi ACG $\mathcal{A}_\Phi$, equipped with a deterministic satellite $\mathcal{U}_\Phi$, that accepts the set of extended basic models of Φ. Moreover, $\mathcal{A}_\Phi$ has $O(|\Phi|)$ states and $\mathcal{U}_\Phi$ has $2^{O(|\Phi|)}$ states.*

Proof. Let $\mathcal{F}$ be the set of subformulas ψ of Φ such that there exists a basic subformula of Φ having one of the following forms: $\langle\langle A \rangle\rangle \mathsf{X}\psi$ or $\langle\langle A \rangle\rangle (\theta \mathcal{O} \psi)$ or $\langle\langle A \rangle\rangle (\psi \mathcal{O} \theta)$ where $\mathcal{O} \in \{\mathsf{U}, \mathsf{R}\}$. Then, the deterministic satellite $\mathcal{U}_\Phi$ is the synchronous product of the safety DWA of Proposition 6 associated with the formulas $[\psi]_{\mathsf{LTL}_p}$ where $\psi \in \mathcal{F}$. For a state s of $\mathcal{U}_\Phi$ and $\psi \in \mathcal{F}$, let $s[\psi]$ be the component of the state s associated with the automaton $\mathcal{D}_\psi$. If s is not initial (hence, $s[\psi] \subseteq cl([\psi]_{\mathsf{LTL}_p})$), we write $P_+(s, \psi)$ to denote `true` if $\psi \in s[\psi]$, and `false` otherwise. Dually, we write $P_-(s, \psi)$ to denote `true` if $\neg\psi \in s[\psi]$, and `false` otherwise.

Next, we define the Büchi ACG $\mathcal{A}_\Phi$. We first build for each basic subformula b of Φ, two Büchi ACG $\mathcal{A}_b$ and $\mathcal{A}_{\neg b}$ such that $\mathcal{A}_b$ (resp., $\mathcal{A}_{\neg b}$), equipped with the satellite $\mathcal{U}_\Phi$, accepts

the set of CGT over $AP \cup B_\Phi$ which are positively (resp., negatively) well-formed with respect to b. Then $\mathcal{A}_\Phi$ is obtained by "composing" the automata $\mathcal{A}_b$ and $\mathcal{A}_{\neg b}$ for all $b \in B_\Phi$.

Fix $b \in B_\Phi$. Then, $\mathcal{A}_b = \langle 2^{AP \cup B_\Phi}, q_b, \{q_b, b\}, \delta_b, F_b \rangle$ consists of two states b and q_b. State q_b is always accepting, i.e. $q_b \in F_b$, while state b is in F_b iff b is of the form $\langle\langle A \rangle\rangle(\psi_1 \mathsf{R} \psi_2)$. The transition function δ_b is defined as follows, where s is a state of $\mathcal{U}_\Phi$:

$$\delta_b(q_b, \sigma, s) = \begin{cases} (q_b, \Box, \emptyset) & \text{if } b \notin \sigma \\ (q_b, \Box, \emptyset) \wedge (b, \Box, A) & \text{if } b \in \sigma \text{ and } b \text{ is of the form } \langle\langle A \rangle\rangle \mathsf{X} \psi_1 \\ (q_b, \Box, \emptyset) \ \wedge \ \delta_b(b, \sigma, s) & \text{otherwise} \end{cases}$$

$$\delta_b(b, \sigma, s) = \begin{cases} P_+(s, \psi_1) & \text{if } b = \langle\langle A \rangle\rangle \mathsf{X} \psi_1 \\ P_+(s, \psi_2) \vee (P_+(s, \psi_1) \wedge (b, \Box, A)) & \text{if } b = \langle\langle A \rangle\rangle (\psi_1 \mathsf{U} \psi_2) \\ P_+(s, \psi_2) \wedge (P_+(s, \psi_1) \vee (b, \Box, A)) & \text{if } b = \langle\langle A \rangle\rangle (\psi_1 \mathsf{R} \psi_2) \end{cases}$$

Intuitively, the automaton $\mathcal{A}_b$ uses the part $(q_b, \Box, \emptyset)$ of the transition function to traverse every node in an input CGT $\mathcal{T}$. Additionally, whenever the basic subformula $b = \langle\langle A \rangle\rangle \psi$ is in the label of the current input node x and $\mathcal{A}_b$ is in its initial state q_b, $\mathcal{A}_b$ guesses a strategy f_A of $\mathcal{T}$ for the coalition A and check that for all the plays $\pi \in out(\nu_x, f_A)$, $Lab(\pi)$ satisfies the conditions of Definition 12 (which depend on the form of ψ). This check is done by using the part $(b, \Box, A)$ of the transition function which allows to send copies of $\mathcal{A}_b$, all of them in state b, to all and only the children of the current node which are consistent with the guessed strategy f_A, and by consulting the current state s of the satellite (i.e., the state reached by $\mathcal{U}_\Phi$ on reading the labeling of the track from the root to the current node). The construction of the ACG $\mathcal{A}_{\neg b} = \langle 2^{AP \cup B_\Phi}, q_{\neg b}, \{q_{\neg b}, \neg b\}, \delta_{\neg b}, F_{\neg b} \rangle$ is similar but we now use $P_-(s, \psi_i)$ instead of $P_+(s, \psi_i)$, and we use atoms of the form $(q, \Diamond, A)$ for selecting plays of counter strategies for A in the input CGT. The ACG $\mathcal{A}_\Phi$ is then defined as follows, where we assume that the various automata $\mathcal{A}_b$ and $\mathcal{A}_{\neg b}$ with $b \in B_\Phi$ have no state in common: $\mathcal{A}_\phi = \langle 2^{AP \cup B_\Phi}, q_0, Q \cup q_0, \delta, F \rangle$, where Q (resp., F) consists of the states (resp., accepting states) of $\mathcal{A}_b$ and $\mathcal{A}_{\neg b}$ for all $b \in B_\Phi$, q_0 is a fresh initial state, and the transition function for the states $q \neq q_0$ is inherited from the respective ACG. For the initial state q_0, $\delta(q_0, \sigma, s)$, where s is a satellite state, is defined as follows: $\delta(q_0, \sigma, s) = \texttt{false}$ if σ is not a model of the propositional formula $[\Phi]_{\mathsf{LTL}}$, and $\delta(q_0, \sigma, s) = \bigwedge_{b \in B_\Phi \cap \sigma} \delta_b(q_b, \sigma, s) \wedge \bigwedge_{b \in B_\Phi \setminus \sigma} \delta_{\neg b}(q_{\neg b}, \sigma, s)$ otherwise. By construction and Lemma 13, for every CGT $\mathcal{T}$ over $AP \cup B_\Phi$, $\mathcal{A}_\Phi$ accepts $\mathcal{T}$ iff $\mathcal{T}$ is a basic extended model of Φ. Moreover, by Proposition 6, $\mathcal{A}_\Phi$ has $O(|\Phi|)$ states and $\mathcal{U}_\Phi$ has $2^{O(|\Phi|)}$ states, which concludes the proof of Theorem 14. ◀

For solving satisfiability for Φ, we can check the pair $(\mathcal{A}_\Phi, \mathcal{U}_\Phi)$ of Theorem 14 for non-emptiness. However, for solving the model-checking problem against ATL$_{lp}$, we need automata running on CGT over 2^{AP}. By slightly adapting the construction of Theorem 14, we obtain the following result (for a proof, see Appendix D).

▶ **Theorem 15.** *Given an ATL$_{lp}$ state formula Φ, one can build in single exponential time a Büchi ACG $\mathcal{A}'_\Phi$ over 2^{AP}, equipped with a* nondeterministic *satellite $\mathcal{U}'_\Phi$, that accepts the set of models of Φ. Moreover, $\mathcal{A}'_\Phi$ has $O(|\Phi|)$ states and $\mathcal{U}'_\Phi$ has $2^{O(|\Phi|)}$ states.*

We can show that model-checking against ATL$_{lp}$ is EXPTIME-hard by a polynomial-time reduction from the acceptance problem for linearly-bounded alternating Turing Machines (see Appendix E). Since ATL$_{lp}$ subsumes ATL and satisfiability of ATL is EXPTIME-complete [2], by Propositions 4-5 and Theorems 14-15, we obtain the main result of this section.

▶ **Corollary 16.** *Satisfiability and model-checking against ATL$_{lp}$ are both* EXPTIME*-complete.*

5 Conclusions

While it is well-known that linear past exponentially increases the succinctness of temporal logic formulas, very few is known about the succinctness gap between different linear-past semantics. In this paper, we have partially addressed this issue. Moreover, we have investigated a memoryful linear-past extension of ATL, which expressively strictly lies between ATL and ATL*, and for which satisfiability and model-checking problems are exponentially less expansive than the ones for ATL*. As future work, we aim to investigate memoryful linear-past extensions of meaningful and elementary fragments (such as One-Goal Strategy Logic) of Strategy Logic [9, 25], a well-known powerful framework for reasoning explicitly about strategic behaviors.

References

1 T. Ågotnes, V. Goranko, and W. Jamroga. Alternating-time temporal logics with irrevocable strategies. In *TARK'07*, pages 15–24, 2007.

2 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

3 C. Baier and J.P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

4 L. Bozzelli. The complexity of CTL* + linear past. In *FOSSACS'08*, LNCS 4962, pages 186–200. Springer, 2008.

5 L. Bozzelli and A. Murano. On the complexity of ATL and ATL* module checking. In *GandALF'17*, EPTCS 256, pages 268–282, 2017.

6 T. Brihaye, A. Da Costa Lopes, F. Laroussinie, and N. Markey. ATL with strategy contexts and bounded memory. In *LFCS'09*, LNCS 5407, pages 92–106. Springer, 2009.

7 N. Bulling, W. Jamroga, and M. Popovici. Agents with truly perfect recall in alternating-time temporal logic. In *AAMAS'14*, pages 1561–1562. IFAAMAS/ACM, 2014.

8 A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

9 K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010.

10 E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of Workshop on Logic of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.

11 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.

12 D. M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification*, LNCS 398, pages 409–448. Springer, 1987.

13 D.P. Guelev, C. Dima, and C. Enea. An alternating-time temporal logic with knowledge, perfect recall and past: axiomatisation and model-checking. *Journal of Applied Non-Classical Logics*, 21(1):93–131, 2011.

14 Th. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *Proc. 14th ICALP*, LNCS 267, pages 269–279. Springer, 1987.

15 W. Jamroga and A. Murano. Module checking of strategic ability. In *AAMAS'15*, pages 227–235. ACM, 2015.

16 D. Janin and I. Walukiewicz. Automata for the modal mu-calculus and related results. In *MFCS'95*, LNCS 969, pages 552–562. Springer, 1995.

17 O. Kupferman and A. Pnueli. Once and For All. In *LICS'95*, pages 25–35. IEEE Comp. Soc. Press, 1995.

18 O. Kupferman, A. Pnueli, and M. Y. Vardi. Once and for all. *J. Comput. Syst. Sci.*, 78(3):981–996, 2012.

19 O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *STOC'98*, pages 224–233. ACM, 1998.

20 O. Kupferman and M.Y. Vardi. Module checking. In *CAV'96*, LNCS 1102, pages 75–86. Springer, 1996.

21 O. Kupferman, M.Y. Vardi, and P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, 2000.

22 F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *LICS'02*, pages 383–392. IEEE Computer Society, 2002.

23 F. Laroussinie and P. Schnoebelen. A hierarchy of temporal logics with past. *Theor. Comput. Sci.*, 148(2):303–324, 1995.

24 F. Laroussinie and P. Schnoebelen. Specification in CTL+past for verification in CTL. *Inf. Comput.*, 156(1-2):236–263, 2000.

25 F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Trans. Comput. Log.*, 15(4):34:1–34:47, 2014.

26 F. Mogavero, A. Murano, and M.Y. Vardi. Relentful strategic reasoning in alternating-time temporal logic. *J. Log. Comput.*, 26(5):1663–1695, 2016.

27 A. Pnueli. The temporal logic of programs. In *FOCS'77*, pages 46–57. IEEE, 1977.

28 S. Schewe. ATL* Satisfiability is 2EXPTIME-complete. In *ICALP'08*, LNCS 5126, pages 373–385. Springer, 2008.

29 S. Schewe and B. Finkbeiner. Satisfiability and finite model property for the alternating-time *mu*-calculus. In *CSL'06*, LNCS 4207, pages 591–605. Springer, 2006.

30 R. S. Streett. Propositional dynamic logic of looping and converse. In *STOC'81*, pages 375–383. ACM, 1981.

31 M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994.

32 M.Y. Vardi. A temporal fixpoint calculus. In *POPL'88*, pages 250–259. ACM, 1988.

33 M.Y. Vardi. Reasoning about the past with two-way automata. In *ICALP'98*, LNCS 1443, pages 628–641. Springer, 1998.

A Proof of Lemma 11

For an ATL^*_{lp} path formula φ, let $BFL(\varphi)$ be the set of *first-level basic subformulas* of φ, i.e. the basic subformulas of φ for which there is an occurrence in φ which is not in the scope of strategy quantifiers. We first show that, under the assumption that the first-level basic subformulas are in ATL^*, it is possible to separate past and future temporal modalities.

▶ **Lemma 17.** *Let $\langle\langle A\rangle\rangle\varphi$ be a basic ATL^*_{lp} formula s.t. $BFL(\varphi)$ consists of ATL^* formulas. Then, $\langle\langle A\rangle\rangle\varphi$ and φ are congruent to Boolean combinations of ATL^* formulas and ATL^*_{lp} formulas that correspond to* pure past *LTL_p formulas over the set of propositions $AP \cup BFL(\varphi)$.*

Proof. Let $\overline{AP} = AP \cup BFL(\varphi)$. By hypothesis, $BFL(\varphi)$ is a set of ATL^* formulas.

Given a CGT $\mathcal{T}$ over AP with propositional labeling Lab and an ε-play π of $\mathcal{T}$, we denote by $\pi_{\overline{AP}}$ the infinite word over $2^{\overline{AP}}$ defined as follows for every position $i \geq 0$:

- $\pi_{\overline{AP}}(i) \cap AP = Lab(\pi(i))$;
- $\pi_{\overline{AP}}(i) \cap BFL(\varphi) = \{\psi \in BFL(\varphi) \mid (\mathcal{T}, \pi, i) \models \psi\}$.

Now, the path formula φ over AP can be seen as an LTL_p formula over $\overline{AP}$. By definitions of the infinite words $\pi_{\overline{AP}}$ and the LTL_p and ATL^*_{lp} semantics, one can easily show by structural induction that for all CGT $\mathcal{T}$ over AP, ε-plays π of $\mathcal{T}$, and positions $i \geq 0$:

$$(\mathcal{T}, \pi, i) \models \varphi \text{ if and only if } (\pi_{\overline{AP}}, i) \models_{\mathsf{LTL}} \varphi \tag{2}$$

By applying the separation theorem for LTL_p [12], starting from the LTL_p formula φ, one can construct an LTL_p formula θ over $\overline{AP}$ of the form

$$\theta := \bigvee_{\ell \in I} (\theta_{p,\ell} \wedge \theta_{f,\ell}) \tag{3}$$

such that $I \neq \emptyset$, for all $\ell \in I$, $\theta_{p,\ell}$ is a pure past LTL_p formula, $\theta_{f,\ell}$ is an LTL formula, and for all infinite words w over $2^{\overline{AP}}$ and $i \geq 0$, it holds that $(w, i) \models_{\mathsf{LTL}} \varphi$ iff $(w, i) \models_{\mathsf{LTL}} \theta$.

Since θ corresponds to an ATL^*_{lp} formula over AP, by replacing formula φ with θ in Point 2, we obtain that φ and θ are congruent and the given basic ATL^*_{lp} formula $\langle\langle A \rangle\rangle \varphi$ over AP is congruent to the basic ATL^*_{lp} formula $\langle\langle A \rangle\rangle \bigvee_{\ell \in I} (\theta_{p,\ell} \wedge \theta_{f,\ell})$. Thus, the statement of Lemma 11 directly follows from the following claim.

Claim. $\psi := \langle\langle A \rangle\rangle \bigvee_{\ell \in I} (\theta_{p,\ell} \wedge \theta_{f,\ell})$ is congruent to $\psi_B := \bigvee_{J \subseteq I \mid J \neq \emptyset} \big((\bigwedge_{\ell \in J} \theta_{p,\ell}) \wedge \langle\langle A \rangle\rangle (\bigvee_{\ell \in J} \theta_{f,\ell}) \big)$.

Proof of the Claim. Fix a CGT $\mathcal{T}$ over AP, an ε-play π of $\mathcal{T}$, and a position $i \geq 0$.

First, assume that $(\mathcal{T}, \pi, i) \models \psi$. Hence, there is a strategy f_A for A such that the following holds, where Π is the non-empty set of ε-plays $\pi' \in \mathit{out}(\pi[0,i], f_A)$: for all $\pi' \in \Pi$, $(\mathcal{T}, \pi', i) \models \bigvee_{\ell \in I} (\theta_{p,\ell} \wedge \theta_{f,\ell})$. Let J be the non-empty subset of I consisting of the elements $\ell \in I$ such that $(\mathcal{T}, \pi', i) \models \theta_{p,\ell} \wedge \theta_{f,\ell}$ for some $\pi' \in \Pi$. Since $\theta_{p,\ell}$ corresponds to a pure past LTL_p formula over $\overline{AP}$ and $\pi'[0,i] = \pi[0,i]$ for all $\pi' \in \Pi$, it follows that for all $\ell \in J$, $(\mathcal{T}, \pi, i) \models \theta_{p,\ell}$. Thus, we obtain that $(\mathcal{T}, \pi, i) \models (\bigwedge_{\ell \in J} \theta_{p,\ell}) \wedge \langle\langle A \rangle\rangle (\bigvee_{\ell \in J} \theta_{f,\ell})$ which entails that $(\mathcal{T}, \pi, i) \models \psi_B$.

For the converse direction, assume that $(\mathcal{T}, \pi, i) \models \psi_B$. Hence, there are a non-empty subset $J \subseteq I$ and a strategy f_A for A such that the following holds, where Π is the non-empty set of ε-plays $\pi' \in \mathit{out}(\pi[0,i], f_A)$:

- for all $\ell \in J$, $(\mathcal{T}, \pi, i) \models \theta_{p,\ell}$;
- for all $\pi' \in \Pi$, there is $\ell \in J$ such that $(\mathcal{T}, \pi', i) \models \theta_{f,\ell}$.

Again since $\theta_{\ell,p}$ corresponds to a pure past LTL_p formula over $\overline{AP}$ and $\pi'[0,i] = \pi[0,i]$ for all $\pi' \in \Pi$, it follows that for all $\ell \in J$ and $\pi' \in \Pi$, $(\mathcal{T}, \pi', i) \models \theta_{p,\ell}$. It follows that $(\mathcal{T}, \pi, i) \models \langle\langle A \rangle\rangle \bigvee_{\ell \in J} (\theta_{p,\ell} \wedge \theta_{f,\ell})$ which entails that $(\mathcal{T}, \pi, i) \models \psi$. ◀

We now prove Lemma 11.

▶ **Lemma 11.** *Let φ be an ATL^*_{lp} path formula. Then, φ is congruent to a Boolean combination of ATL^* formulas and ATL^*_{lp} formulas that correspond to* pure past *LTL_p formulas over the set of propositions $AP \cup \mathcal{H}$, where $\mathcal{H}$ consists of basic ATL^* formulas.*

Proof. The proof is by induction on the nesting depth of the strategy quantifiers in φ.

Base case: in this case $\mathit{BFL}(\varphi) = \emptyset$, and the result directly follows from Lemma 17.

Inductive step: let $\langle\langle A' \rangle\rangle \psi \in \mathit{BFL}(\varphi)$. By the inductive hypothesis, the thesis holds for formula ψ. Hence, ψ is a congruent to an ATL^*_{lp} formula ψ' such that $\mathit{BFL}(\psi')$ consists of basic ATL^* formulas. By applying Lemma 17, $\langle\langle A' \rangle\rangle \psi$ is congruent to an ATL^*_{lp} formula, say

ξ, such that $BFL(\xi)$ consists of basic ATL* formulas. By replacing each occurrence of $\langle\langle A' \rangle\rangle \psi$ in φ with ξ, and repeating the procedure for all the formulas in $BFL(\varphi)$, we obtain an ATL$^*_{lp}$ path formula θ which is congruent to φ (note that the congruence relation is closed under substitution) and such that $BFL(\theta)$ consists of basic ATL* formulas. At this point we can apply Lemma 17 to formula θ proving the assertion. ◀

B Proof of Proposition 6

▶ **Proposition 6.** *Let ψ be a pure past LTL$_p$ formula over $AP \cup B_\Phi$. Then, one can construct a safety DWA $\mathcal{D}_\psi = \langle 2^{AP \cup B_\Phi}, q_0, Q \cup \{q_0\}, \delta \rangle$ such that $Q \subseteq 2^{cl(\psi)}$, there is no transition leading to q_0, and the following holds for each* infinite *word w over $2^{AP \cup B_\Phi}$:*

- *let $q_0\, C_0\, C_1 \ldots$ be the unique run of $\mathcal{D}_\psi$ over w. Then, for all $i \geq 0$ and $\theta \in cl(\psi)$, $\theta \in C_i$ if and only if $(w, i) \models \theta$.*

Proof. The safety DWA $\mathcal{D}_\psi = \langle 2^{AP \cup B_\Phi}, q_0, Q \cup \{q_0\}, \delta \rangle$ such that $Q \subseteq 2^{cl(\psi)}$ is defined as follows. Q is the set of *atoms* of ψ consisting of the maximal propositionally consistent subsets C of $cl(\psi)$. Formally, an atom C of ψ is a subset of $cl(\psi)$ satisfying the following:

- for each $\theta \in cl(\psi)$, $\theta \in C$ iff $\neg\theta \notin C$;
- for each $\theta_1 \vee \theta_2 \in cl(\psi)$, $\theta_1 \vee \theta_2 \in C$ iff $\{\theta_1, \theta_2\} \cap C \neq \emptyset$.

An atom is *initial* if

- for each $\mathsf{X}^-\theta \in cl(\psi)$, $\mathsf{X}^-\theta \notin C$;
- for each $\theta_1 \mathsf{U}^- \theta_2 \in cl(\psi)$, $\theta_1 \mathsf{U}^- \theta_2 \in C$ iff $\theta_2 \in C$.

The transition relation δ captures the semantics of the previous modalities, and the local fixpoint characterization of the since modalities. Formally, for each $\sigma \in 2^{AP \cup B_\Phi}$, $\delta(q_0, \sigma)$ consists of the *uniquely determined initial* atom C such that C *is consistent with the input symbol* σ, i.e., $C \cap (AP \cup B_\Phi) = \sigma \cap (AP \cup B_\Phi)$. Moreover, for each atom C of ψ and $\sigma \in 2^{AP \cup B_\Phi}$, $\delta(C, \sigma)$ consists of the *uniquely determined* atom C' of ψ such that C' is consistent with the input symbol σ, and:

- for each $\mathsf{X}^-\theta \in cl(\psi)$, $\mathsf{X}^-\theta \in C'$ iff $\theta \in C$;
- for each $\theta_1 \mathsf{U}^- \theta_2 \in cl(\psi)$, $\theta_1 \mathsf{U}^- \theta_2 \in C'$ iff either $\theta_2 \in C'$, or $\theta_1 \in C'$ and $\theta_1 \mathsf{U}^- \theta_2 \in C$.

By standard arguments (see [31]), given an infinite word w over $2^{AP \cup B_\Phi}$, for the unique run $\pi = q_0\, C_0\, C_1 \ldots C_\ell$ of $\mathcal{D}_\psi$ over w, the following holds: for all $i \geq 0$ and $\theta \in cl(\psi)$, $\theta \in C_i$ if and only if $(w, i) \models \theta$. Hence, the result follows. ◀

C Proof of Lemma 13

▶ **Lemma 13** (Characterization of extended basic models of Φ). *Let $\mathcal{T}$ be a CGT with labeling Lab over $AP \cup B_\Phi$ such that $Lab(\varepsilon)$ is a model of the propositional formula $[\Phi]_{LTL_p}$. Then, $\mathcal{T}$ is an extended basic model of Φ* if and only if *for each basic subformula $\langle\langle A \rangle\rangle \psi \in B_\Phi$, $\mathcal{T}$ is both positively and negatively well-formed with respect to $\langle\langle A \rangle\rangle \psi$.*

Proof. Let $\mathcal{T}$ as in the statement of the lemma, and for a node x of $\mathcal{T}$, let ν_x be the track of $\mathcal{T}$ from the root to node x.

First assume that for each basic subformula $\langle\langle A \rangle\rangle \psi \in B_\Phi$, $\mathcal{T}$ is both positively and negatively well-formed with respect to $\langle\langle A \rangle\rangle \psi$. Fix a node x of $\mathcal{T}$ and $\langle\langle A \rangle\rangle \psi \in B_\Phi$. We need to show that $\langle\langle A \rangle\rangle \psi \in Lab(x)$ *iff* $(\mathcal{T}, x) \models \langle\langle A \rangle\rangle \psi$. The proof is by structural induction on the nesting depth of strategy quantifiers in ψ. We focus on the implication $\langle\langle A \rangle\rangle \psi \notin Lab(x) \Rightarrow (\mathcal{T}, x) \models \neg\langle\langle A \rangle\rangle \psi$ (the converse implication $\langle\langle A \rangle\rangle \psi \in Lab(x) \Rightarrow (\mathcal{T}, x) \models \langle\langle A \rangle\rangle \psi$ is similar) and assume that $\psi = \psi_1 \mathsf{U} \psi_2$ (the other cases being similar). Thus, let $\langle\langle A \rangle\rangle \psi \notin Lab(x)$.

Since $\mathcal{T}$ is negatively well-formed with respect to $\langle\langle A\rangle\rangle\psi$, $\langle\langle A\rangle\rangle\psi \notin Lab(x)$, and $\psi = \psi_1 \mathsf{U} \psi_2$, by Definition 12 there exists a counter strategy f_A^c of $\mathcal{T}$ for A such that for all the outcomes $\pi \in out(\nu_x, f_A^c)$, the following holds, where $\mathcal{D}_{\psi_1}$ and $\mathcal{D}_{\psi_2}$ are the safety DWA of Proposition 6 associated with the pure past LTL_p formulas $[\psi_1]_{\mathsf{LTL}_p}$ and $[\psi_2]_{\mathsf{LTL}_p}$:

- (*) for all $j \geq |x|$ either $\neg\psi_2 \in \mathcal{D}_{\psi_2}(Lab(\pi[0,j]))$ or $\neg\psi_1 \in \mathcal{D}_{\psi_1}(Lab(\pi[0,k]))$ for some $k \in [|x|, j-1]$.

By applying Proposition 6 to the unique run of the DWA $\mathcal{D}_{\psi_1}$ (resp., $\mathcal{D}_{\psi_2}$) over the infinite word $Lab(\pi)$, by Condition (*) we deduce that

- (**) for all $j \geq |x|$ either $(Lab(\pi), j) \models_{\mathsf{LTL}} [\neg\psi_2]_{\mathsf{LTL}_p}$ or $(Lab(\pi), k) \models_{\mathsf{LTL}} [\neg\psi_1]_{\mathsf{LTL}_p}$ for some $k \in [|x|, j-1]$.

It follows that $(Lab(\pi), |x|) \models_{\mathsf{LTL}} [\neg\psi]_{\mathsf{LTL}_p}$. By the induction hypothesis, for each $\langle\langle A'\rangle\rangle\theta \in BFL(\psi)$ and position $j \geq 0$, we have that $\langle\langle A'\rangle\rangle\theta \in Lab(\pi(j))$ iff $(\mathcal{T}, \pi(j)) \models \langle\langle A'\rangle\rangle\theta$ (note that in the base step, $BFL(\psi) = \emptyset$, hence ψ in a pure past LTL_p formula over AP). Thus, by the semantics of ATL_{lp}, it follows that $(\mathcal{T}, \pi, |x|) \models \neg\psi$. Since π in an arbitrary outcome of the counter strategy f_A^c from the history ν_x, by Proposition 3, we conclude that $(\mathcal{T}, x) \models \neg\langle\langle A\rangle\rangle\psi$, and the result follows.

For the converse implication, assume that for each node x of $\mathcal{T}$ and basic subformula $\langle\langle A\rangle\rangle\psi \in B_\Phi$, $\langle\langle A\rangle\rangle\psi \in Lab(x)$ *iff* $(\mathcal{T}, x) \models \langle\langle A\rangle\rangle\psi$. Fix $\langle\langle A\rangle\rangle\psi \in B_\Phi$. We show that $\mathcal{T}$ is both positively and negatively well-formed with respect to $\langle\langle A\rangle\rangle\psi$. We focus on the negatively well-formedness condition and assume that $\psi = \psi_1 \mathsf{U} \psi_2$ (the other cases being similar). Thus, let x be a node of $\mathcal{T}$ such that $\langle\langle A\rangle\rangle\psi \notin Lab(x)$. By hypothesis, $(\mathcal{T}, x) \models \neg\langle\langle A\rangle\rangle\psi$. By Proposition 3, there exists a counter strategy f_A^c of $\mathcal{T}$ for A such that for all the outcomes $\pi \in out(\nu_x, f_A^c)$, it holds that $(\mathcal{T}, \pi, |x|) \models \neg\psi$, hence, $(Lab(\pi), |x|) \models_{\mathsf{LTL}} [\neg\psi]_{\mathsf{LTL}_p}$. Fix $\pi \in out(\nu_x, f_A^c)$. Let $q_0^1\, C_0^1\, C_1^1 \ldots$ (resp., $q_0^2\, C_0^2\, C_1^2 \ldots$) be the unique run of the safety DWA $\mathcal{D}_{\psi_1}$ (resp., $\mathcal{D}_{\psi_2}$). Since $(Lab(\pi), |x|) \models_{\mathsf{LTL}} [\neg\psi]_{\mathsf{LTL}_p}$ and $\psi = \psi_1 \mathsf{U} \psi_2$, by Proposition 6, it follows that for all $j \geq |x|$ either $\neg\psi_2 \in C_j^2$ or $\neg\psi_1 \in C_k^1$ for some $k \in [|x|, j-1]$.

Since, π and x are arbitrary, by Definition 12, we conclude that $\mathcal{T}$ is negatively well-formed with respect to $\langle\langle A\rangle\rangle\psi$. ◀

D Proof of Theorem 15

▶ **Theorem 15.** *Given an ATL_{lp} state formula Φ, one can build in single exponential time a Büchi ACG $\mathcal{A}'_\Phi$ over 2^{AP}, equipped with a* nondeterministic *satellite $\mathcal{U}'_\Phi$, that accepts the set of models of Φ. Moreover, $\mathcal{A}'_\Phi$ has $O(|\Phi|)$ states and $\mathcal{U}'_\Phi$ has $2^{O(|\Phi|)}$ states.*

Proof. Let $\mathcal{A}_\Phi$ and $\mathcal{U}_\Phi$ be the Büchi ACG and the deterministic satellite of Theorem 14 running on CGT with labeling over $2^{AP \cup B_\Phi}$. Starting from $\mathcal{A}_\Phi$ and $\mathcal{U}_\Phi$, we construct a new Büchi ACG $\mathcal{A}'_\Phi$ equipped with a nondeterministic satellite $\mathcal{U}'_\Phi$ over 2^{AP} accepting the set of models Φ. Intuitively, given an input CGT $\mathcal{T}$, $\mathcal{A}'_\Phi$ and $\mathcal{U}'_\Phi$ guess a B_Φ-labeling extension of the input $\mathcal{T}$ and check that such an extension is an extended basic model of Φ. In particular, we let the satellite $\mathcal{U}'_\Phi$ to guess the B_Φ-labeling extension and let the ACG $\mathcal{A}'_\Phi$ check the guess. Thus, $\mathcal{U}'_\Phi$ is obtained from $\mathcal{U}_\Phi$ by adding on top of the deterministic transition relation a guess of the subset of B_Φ to be read in the current node of the input CGT. Note that since an ACG is an alternating automaton, it is not to possible, in general, for an ACG to guess a labelling of the input, since different copies of the automaton which read the same input node may take different guesses. However, here, we exploit the crucial fact that for a CGT $\mathcal{T}$ over 2^{AP}, there is at most one B_Φ-labeling extension of $\mathcal{T}$ which is an extended basic model of Φ (Remark 4.3). Formally, let $\mathcal{A}_\Phi = \langle 2^{AP \cup B_\Phi}, q_0, Q, \delta, F\rangle$ and $\mathcal{U}_\Phi = \langle 2^{AP \cup B_\Phi}, s_0, S \cup \{s_0\}, \delta_S\rangle$.

Then, $\mathcal{A}'_\Phi = \langle 2^{AP}, q_0, Q, \delta', F\rangle$ and $\mathcal{U}'_\Phi = \langle 2^{AP}, s_0, S \times 2^{B_\Phi} \cup \{s_0\}, \delta'_S\rangle$, where δ' and δ'_S are defined as follows:

- $\delta'_S(s_0, \sigma) = \bigcup_{\sigma' \in 2^{B_\Phi}} \delta_S(s_0, \sigma \cup \sigma') \times \{\sigma'\}$.
- $\delta'_S((s, \sigma'), \sigma) = \bigcup_{\sigma'' \in 2^{B_\Phi}} \delta_S(s_0, \sigma \cup \sigma'') \times \{\sigma''\}$.
- $\delta'(q, \sigma, (s, \sigma')) = \delta(q, \sigma \cup \sigma', s)$.

We now prove that the construction is correct. Evidently, by construction and Theorem 14, it suffices to show that for each input $\mathcal{T}$, if $\mathcal{T}$ is accepted by $\mathcal{A}'_\Phi$, then $\mathcal{T}$ is a model of Φ. Let r be an accepting run of $\mathcal{A}'_\Phi$ over an input $\mathcal{T}$. Note that for each subformula ψ of Φ, since $[\psi]_{\mathsf{LTL}_p}$ is an LTL_p formula over $AP \cup BFL(\psi)$, the transition function $\delta_\psi(q, \sigma)$ of the safety DWA $\mathcal{D}_\psi$ of Proposition 6 is independent of the $B_\Phi \setminus BFL(\psi)$-part of σ. Thus, by a straightforward double induction on the nesting depth of strategy quantifiers in θ for a basic formula $\langle\langle A\rangle\rangle\theta \in B_\phi$ and the distance $|x|$ from the root of a node x of the input $\mathcal{T}$, and by construction and the proof of Theorem 14, the following holds:

- for all nodes x and copies of the nondeterministic satellite $\mathcal{U}'_\Phi$ in states (s', σ') and (s'', σ'') in the run r resulting on reading the node x, it holds that: $(s', \sigma') = (s'', \sigma'')$ and for all $\langle\langle A\rangle\rangle\theta \in B_\phi$, $\langle\langle A\rangle\rangle\theta \in \sigma'$ iff $(\mathcal{T}, x) \models \langle\langle A\rangle\rangle\theta$.

Hence, the result follows. ◀

E EXPTIME-hardness of ATL_{lp} model-checking

In this section, we establish the following result, where a two-player turn-based CGS is a CGS with two agents where each state is controlled by an agent.

▶ **Theorem 18.** *Model checking against ATL_{lp} is* EXPTIME*–hard even for finite two-player turn-based CGS of fixed size.*

Theorem 18 is proved by a polynomial-time reduction from the acceptance problem for *linearly-bounded alternating* Turing Machines (TM) with a binary branching degree and with a fixed size, which is EXPTIME-complete [8]. In the rest of this section, we fix such a TM machine $\mathcal{M} = \langle \Sigma, Q, Q_\forall, Q_\exists, q_0, \delta, F\rangle$, where Σ is the input alphabet, Q is the finite set of states which is partitioned into $Q = Q_\forall \cup Q_\exists$, $Q_\exists$ (resp., $Q_\forall$) is the set of existential (resp., universal) states, q_0 is the initial state, $F \subseteq Q$ is the set of accepting states, and the transition function δ is a mapping $\delta : Q \times \Sigma \to (Q \times \Sigma \times \{L, R\})^2$. Moreover, we fix an input $\alpha \in \Sigma^*$ such that $|\alpha| \geq 1$ and consider the parameter $n = |\alpha|$.

Since $\mathcal{M}$ is linearly bounded, we can assume that $\mathcal{M}$ uses exactly n tape cells when started on the input α. Hence, a TM configuration (of $\mathcal{M}$ over α) is a word $C = \eta \cdot (q, \sigma) \cdot \eta' \in \Sigma^* \cdot (Q \times \Sigma) \cdot \Sigma^*$ of length exactly n denoting that the tape content is $\eta \cdot \sigma \cdot \eta'$, the current state is q, and the tape head is at position $|\eta| + 1$. From configuration C, the machine $\mathcal{M}$ nondeterministically chooses a triple (q', σ', dir) in $\delta(q, \sigma) = \langle (q_l, \sigma_l, dir_l), (q_r, \sigma_r, dir_r)\rangle$, and then moves to state q', writes σ' in the current tape cell, and its tape head moves one cell to the left or to the right, according to dir. We denote by $succ_l(C)$ and $succ_r(C)$ the successors of C obtained by choosing respectively the left and the right triple in $\langle (q_l, \sigma_l, dir_l), (q_r, \sigma_r, dir_r)\rangle$. The configuration C is accepting (resp., universal, resp., existential) if the associated state q is in F (resp., in $Q_\forall$, resp., in $Q_\exists$). A (finite) computation tree of $\mathcal{M}$ over α is a finite tree in which each node is labeled by a configuration. The root of the tree corresponds to the initial configuration C_α given by $(q_0, \alpha(0))\alpha(1) \ldots \alpha(n-1)$. An *internal* node that is labeled by a universal configuration C has two children, corresponding to $succ_l(C)$ and $succ_r(C)$, while an internal node labeled by an existential configuration C has a single child, corresponding to either $succ_l(C)$ or $succ_r(C)$. The tree is accepting iff each its leaf is labeled by an accepting

configuration. The input α is *accepted* by $\mathcal{M}$ iff there is an accepting computation tree of $\mathcal{M}$ over α. We prove the following result from which Theorem 18 directly follows.

▶ **Theorem 19.** *One can construct, in time polynomial in n and the size of $\mathcal{M}$, a finite turn-based CGS $\mathcal{G}$ and an ATL$_{lp}$ state formula φ over the set of agents $Ag = \{ag_\exists, ag_\forall\}$ such that $\mathcal{M}$ accepts α iff $Unw(\mathcal{G}) \models \varphi$. Moreover, the size of $\mathcal{G}$ depends only on the size of $\mathcal{M}$.*

In order to prove Theorem 19, we first define a suitable encoding of the TM computations of $\mathcal{M}$ over α. Formally, we exploit the set AP of atomic propositions given by $AP := \Sigma \cup Q \times \Sigma \times \{\forall, \exists, l, r, acc\}$. A TM configuration $C = u_1 u_2 \dots u_n$ is encoded by words w_C over 2^{AP} of the form $w_C = \{tag_1\}\{u_1\} \dots \{u_k\}\{tag_2\}$, where $tag_1 \in \{l, r\}$, $tag_2 = acc$ if C is an accepting configuration, $tag_2 = \exists$ if C is a non-accepting existential configuration, and $tag_2 = \forall$ otherwise. The symbols l and r are used to mark a left and a right TM successor, respectively. We also use the symbol l to mark the initial configuration C_α. A sequence $w_{C_1} \cdot \dots \cdot w_{C_p}$ of TM configuration codes is *good* if the following holds:

- C_1 is the initial configuration and w_{C_1} is marked by symbol l;
- C_p is accepting;
- for each $1 \leq i < p$, either $w_{C_{i+1}}$ is marked by symbol l and $C_{i+1} = succ_l(C_i)$, or $w_{C_{i+1}}$ is marked by symbol r and $C_{i+1} = succ_r(C_i)$.

Now, we prove Theorem 19. The finite turn-based CGS $\mathcal{G}$ over the set of agents $Ag = \{ag_\exists, ag_\forall\}$ in Theorem 19 is defined as follows:

- the set of states of $\mathcal{G}$ is AP and the initial state is l;
- the label of each state $p \in AP$ is $\{p\}$;
- state $\forall$ is controlled by agent $ag_\forall$, and each other state is controlled by agent $ag_\exists$;
- state $\forall$ has as successors the states l and r, state acc has as successor itself, while each other state has as successors the whole set of states.

Note that the CGS $\mathcal{G}$ is independent of n. By construction, the following hold.

Claim. $\mathcal{M}$ accepts α iff there exists a strategy $f_\exists$ of agent $ag_\exists$ in $Unw(\mathcal{G})$ such that for all ε-plays π consistent with $f_\exists$, the label of π is an infinite word over 2^{AP} of the form $w \cdot \{acc\}^\omega$ such that w is a *good* sequence of TM configuration codes.

Thus, the ATL$_{lp}$ formula φ satisfying the statement of Theorem 19 is given by $\varphi := \langle\langle ag_\exists \rangle\rangle \mathsf{F}(acc \wedge \psi_{good})$, where ψ_{good} is a pure past LTL$_p$ formula of size $O(n^2)$ capturing the good sequences of TM configuration codes. The construction of ψ_{good} is standard and, therefore, we omit further details.

Extracting Interval Temporal Logic Rules: A First Approach

Davide Bresolin
Department of Mathematics
University of Padova, Italy
davide.bresolin@unipd.it
https://orcid.org/0000-0003-2253-9878

Enrico Cominato
Department of Mathematics, Computer Science and Physics
University of Udine, Italy
enrico.cominato@gmail.com

Simone Gnani
Department of Mathematics and Computer Science
University of Ferrara, Italy
simone.gnani@student.unife.it

Emilio Muñoz-Velasco
Department of Applied Mathematics
University of Malaga, Spain
ejmunoz@uma.es
https://orcid.org/0000-0002-0117-4219

Guido Sciavicco
Department of Mathematics and Computer Science
University of Ferrara, Italy
guido.sciavicco@unife.it
https://orcid.org/0000-0002-9221-879X

Abstract

Discovering association rules is a classical data mining task with a wide range of applications that include the medical, the financial, and the planning domains, among others. Modern rule extraction algorithms focus on *static* rules, typically expressed in the language of Horn propositional logic, as opposed to *temporal* ones, which have received less attention in the literature. Since in many application domains temporal information is stored in form of intervals, extracting interval-based temporal rules seems the natural choice. In this paper we extend the well-known algorithm APRIORI for rule extraction to discover interval temporal rules written in the Horn fragment of Halpern and Shoham's interval temporal logic.

2012 ACM Subject Classification Information systems → Clustering and classification

Keywords and phrases Interval temporal logic, Horn fragment, Rule extraction

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.7

Acknowledgements D. Bresolin and G. Sciavicco acknowledge partial support by the Italian INDAM GNCS project *Formal methods for verification and synthesis of discrete and hybrid systems* (2018).

25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek; Article No. 7; pp. 7:1–7:15

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

1 Introduction

Rule-based methods are a popular class of techniques in machine learning and data mining [16]. They share the goal of finding regularities in data that can be expressed in the form of *if-then* rules. Depending on the type of rules, we can discriminate between *descriptive rules* discovery, which aims at describing significant patterns in the given data set in terms of rules, and *predictive rules* discovery, which is focused on learning a collection of the rules that collectively cover the instance space and can make a prediction for every possible instance. In this paper, we are interested in descriptive (or *association*) rules, for which there are three popular approaches. The first one is based on *inductive logic programming*, which uses logic programming as a uniform representation for examples, background knowledge and hypotheses, and aims at deriving a hypothesised logic program (that is, a set of rules) which entails all the positive and none of the negative examples (see, e.g., [29, 32]). A second typical approach is based on *rule induction via metaheuristics*, typically driven by evolutionary algorithms (see [24] for an example); in this case, not only static propositional rules may be derived, but also, and even more often, *fuzzy* rules (see [21], and references within). The third classical rule extraction approach is based on APRIORI [2] and its subsequent developments. These approaches have been extensively compared in the literature (see, e.g., [17] and references therein); apparently, although APRIORI is probably the first technology for rule extraction that gained some acknowledgment in the community, its main ideas are still widely used, since no negative examples are needed (in contrast to inductive logic programming), and since it is considered reliable and fast (in contrast to metaheuristic approaches, which are computationally expensive).

If-then rules are extracted from an abstract data base, in which every transaction (or instance), is characterized by a set of items or features that represents the attributes of the instance. If-then rules are usually represented as logical rules, even though they are not interpreted as implications in strict logical terms. On the one hand, rules represent positive information only: instances where the implication is trivially satisfied by the absence of the antecedent are not relevant in this setting. On the other hand, rules express a likelihood information, such as *if these items are present, it is very likely that this other item will be present, too*, rather than a deterministic Boolean value. Classical static rules (such as rules extracted from frequent item sets, as in APRIORI) are *propositional* Horn logic rules:

$$\rho : p_1 \wedge p_2 \wedge \ldots \wedge p_k \Rightarrow p$$

where $p_1, \ldots, p_k, p$ are propositional letters associated to the items of the instances in the data set. Other approaches make use of more complex languages. In [14, 15] first-order point-based *temporal* logics are used to describe temporal classification rules. In [3] a methodology to mine temporal sequential patterns is proposed. In this work patterns are based on instantaneous events and are purely existential; moreover, the algorithm can detect only one temporal relation, that is, *sometime in the future* (F, in the point-based temporal logic notation). Since in many application domains temporal information is stored in form of intervals, extracting interval-based temporal rules seems the natural step forward. The most representative interval temporal language is probably Halpern and Shoham's Modal Logic of Time Intervals [19], often referred to as HS, which is a modal propositional language that features precisely one existential modal operator and one universal modal operator for each basic relation between two intervals. Motivated by the search of computationally affordable versions of HS, several fragments have been explored, that include fragments with restricted sets of modal operators [28, 10, 1], fragments with softer (reflexive) semantics [27], fragments

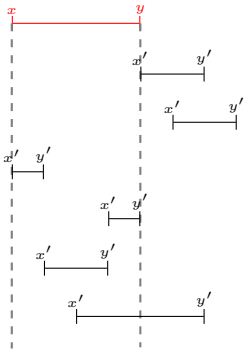
HS	Allen's relations	Graphical representation
$\langle A \rangle$	$[x,y]R_A[x',y'] \Leftrightarrow y = x'$	
$\langle L \rangle$	$[x,y]R_L[x',y'] \Leftrightarrow y < x'$	
$\langle B \rangle$	$[x,y]R_B[x',y'] \Leftrightarrow x = x', y' < y$	
$\langle E \rangle$	$[x,y]R_E[x',y'] \Leftrightarrow y = y', x < x'$	
$\langle D \rangle$	$[x,y]R_D[x',y'] \Leftrightarrow x < x', y' < y$	
$\langle O \rangle$	$[x,y]R_O[x',y'] \Leftrightarrow x < x' < y < y'$	

Figure 1 Allen's interval relations and HS modalities.

in which the nesting of modal operators is limited [8], and sub-propositional (Horn-like) fragments [9]. In particular, the Horn fragment of HS presents some advantages that are relevant to this study:

(i) it naturally generalizes the classical propositional Horn logic, and
(ii) some of its interesting sub-fragments are decidable and tractable [9, 11].

In this paper, we start from a set of finite *timelines* (that is, a *temporal data set*) $\mathcal{T} = \{T_1, \ldots, T_n\}$, where each timeline is a model for HS and can be interpreted as a temporal *history*. For example, in the medical domain, a temporal history T_i may be the set of all relevant events for a certain patient i (e.g., patient i has undergone a certain therapy, or has shown a certain combination of symptoms, in the interval $[x, y]$). As another example, in text mining, a temporal history T_i may represent the sequence of contexts of the conversation i. We propose a temporal extension of the algorithm APRIORI that extracts meaningful association rules from a temporal data set, written in the language of Horn HS [9, 11]. We use classical and well-recognized metrics for static rule evaluation (that is, support and confidence) in association with suitable new metrics for temporal rule evaluation, and we test our algorithm against a synthetic temporal data set.

This paper is organized as follows. In the next section we give the necessary preliminaries, while in Section 3 we briefly discuss some possible examples of applications. In Section 4 we discuss the problem of evaluating a rule and describe our algorithm. Then, in Section 5 we consider a synthetic temporal data set and show how our algorithm extracts useful rules from it, before concluding.

2 Preliminaries

Let $(D, \leq)$ be a linearly ordered set. A (strict) *interval* over D is an ordered pair $[x, y]$, where $x, y \in D$ and $x < y$. We denote by $I(D)$ the set of all (strict) intervals over D. If we exclude the identity relation, there are 12 different relations between two intervals in a linearly ordered set, often called *Allen's relations* [4]: 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 Fig. 1, and their inverses, that is, $R_{\overline{X}} = (R_X)^{-1}$, for each $X \in \mathcal{A} = \{A, L, B, E, D, O\}$.

Halpern and Shoham's Modal Logic of Time Intervals [19] (HS) is a modal logic that features a universal modality $[X]$ and an existential modalitiy $\langle X \rangle$ for each Allen relation R_X. For each $X \in \mathcal{A}$, the *transposes* of the modalities $[X]$ and $\langle X \rangle$ are respectively the

modalities $[\overline{X}]$ and $\langle\overline{X}\rangle$, corresponding to the inverse relation $R_{\overline{X}}$ of R_X, and vice versa. Motivated by the search of computationally well-behaved versions and fragments of HS, several sub-propositional fragments of HS have been introduced and studied (see [9], and references therein). In this paper we focus our attention on the *Horn fragment* of HS. The basic blocks of the language are (*positive temporal*) *literals* λ, defined by the following grammar:

$$\lambda ::= \top \mid \bot \mid p \mid \langle X\rangle\lambda \mid [X]\lambda, \tag{1}$$

where R_X is one of the interval relations and p is a *propositional letter* from a finite, non-empty set $\mathcal{AP}$. Literals of the type $\langle X\rangle p$ (resp., $[X]p$) are called *existential* (resp., *universal*) literals. Formulas of the Horn fragment are in *clausal* form:

$$\varphi ::= \lambda \mid [G](\lambda_1 \wedge \ldots \wedge \lambda_k \rightarrow \lambda) \mid \varphi_1 \wedge \varphi_2, \tag{2}$$

where $[G]$ is the *global operator* (which imposes that something is true everywhere in the model, and can be defined within the language HS), and each λ is obtained from (1). The conjuncts of the form λ are called the *initial conditions* of φ, and those of the form $[G](\lambda_1 \wedge \ldots \wedge \lambda_k \rightarrow \lambda)$ the *clauses* or *global rules* of φ. The formula $\lambda_1 \wedge \ldots \wedge \lambda_k \rightarrow \lambda$, under the scope of a global operator, is referred to as *body* of the rule. In this paper we will use $\rightarrow$ to denote the classical logical implication, and $\Rightarrow$ to denote implicative rules, to emphasise the fact that rules are *not* logical implications in the classical sense. The semantics of Horn HS formulas is given in terms of *interval models* (or *timelines*) of the type $T = \langle D, V\rangle$, where $(D, \leq)$ is a linearly ordered set and $V : \mathcal{AP} \rightarrow 2^{I(D)}$ 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 φ on a given interval $[x, y]$ in a timeline T is defined by structural induction on formulas as follows:

- $T, [x, y] \Vdash \top$ and $T, [x, y] \nVdash \bot$ for every $[x, y] \in I(D)$;
- $T, [x, y] \Vdash p$ if $[x, y] \in V(p)$;
- $T, [x, y] \Vdash \langle X\rangle\psi$ if there exists $[w, z]$ such that $[x, y]R_X[w, z]$ and $T, [w, z] \Vdash \psi$;
- $T, [x, y] \Vdash [X]\psi$ if, for all $[w, z]$ such that $[x, y]R_X[w, z]$, we have that $T, [w, z] \Vdash \psi$;
- $T, [x, y] \Vdash [G](\lambda_1 \wedge \ldots \wedge \lambda_k \rightarrow \lambda)$ if, for all $[w, z]$ such that $T, [w, z] \Vdash \lambda_1 \wedge \cdots \wedge \lambda_k$ we have that $T, [w, z] \Vdash \lambda$;
- $T, [x, y] \Vdash \psi_1 \wedge \psi_2$ if $T, [x, y] \Vdash \psi_1$ and $T, [x, y] \Vdash \psi_2$.

In this work, we are interested in finite domains only, so, from now on, $D = \{0, 1, \ldots, N-1\}$. The set of propositional letters that are true on a given interval is also called *label* of the interval.

The chosen semantics for HS formulas is strict, irreflexive, and non-homogeneous; that is, point-intervals are excluded, the range of the Allen's relations R_X does not include the current interval, and there is no relationship between the truth value of a proposition over an interval and the truth value of the same proposition over its sub-intervals (in the homogeneous semantics, a letter p is true over an interval if and only if it is true in each of its sub-interval). Other choices are possible that may have an impact, sometimes dramatic, in the computational properties of the resulting logic (see, e.g., [7, 25] for homogeneous HS, and [9] for reflexive HS). However, in this paper we are concerned about rule extraction, not rule satisfiability, which means that in our case the semantical choices have little effect.

3 Applying Temporal Rules

There are several application domains in which temporal rules may have a relevant role. Here, we limit ourselves to some illuminating examples.

Rules in the medical domain. In the medical context a timeline may represent the *medical history* of a patient, that is, the collection of all relevant pieces of information about tests, results, symptoms, and hospitalizations of the patient that occurred during the entire observation period [13, 12]. As a concrete example, having a fever can be represented by the propositional letter Low – meaning lower than 40 degrees – or $High$ – meaning higher than or equal to 40 degrees; similarly, the proposition letter $Headache$ can be used to indicate the presence of a headache. Consider, now, a certain $Therapy$ that may be administered to a group of patient under consideration, and consider the problem of establishing the possible counter-effects that such medication or combination of medications may have, in particular, with the insurgence of headaches. Assume, now, that a relevant number of patients under observation show some relationship between the therapy and the insurgence of headache, and, in particular, a headache seems to start during the administration, but only if the patient is running high fever. Such a situation may be described by a Horn HS rule, as follows:

$$\rho : [G](Therapy \wedge \langle \overline{D} \rangle Fever \Rightarrow \langle O \rangle Headache).$$

Rules in natural language processing. In the context of natural language processing a timeline may represent a *conversation* between two individuals. As a matter of fact, it is sometimes interesting to label each interval of time of a conversation with one or more *contexts*, that is, a particular topic that is being discussed [31, 5, 6], in order to discover the existence of unexpected or interesting temporal relationships among them. Suppose, for example, that a certain company wants to analyze conversations between *agents* and *clients*. The agents contact the clients with the aim of selling a certain product, and it is known that certain contexts, such as the price of the product ($Price$), its known advantages ($Advantages$) over other products, and its possible minor defects ($Disadvantages$) are interesting. By analyzing a sufficiently high number of conversations, we may discover that if some known disadvantage is mentioned while discussing the price, the client typically shows some kind of negative reaction ($Negative$), which can be described by the rule:

$$\rho : [G](Price \wedge \langle D \rangle Disadvantages \Rightarrow \langle L \rangle Negative).$$

4 Extracting Interval Temporal Rules

In this section we describe our algorithm for temporal rule extraction from a temporal data set. We start by recalling some well-known concepts of static rule extraction, and by formalizing the problem of evaluating a static rule; then, we generalize our approach to the temporal case, and, finally, we describe a temporal version of the APRIORI algorithm for rule extraction.

Evaluating static rules. Extracting static rules from a non-temporal data base is founded on two simple concepts: support and confidence. Consider an abstract data base $\mathcal{T} = \{T_1, \ldots, T_n\}$, where each T_i is a *transaction* (that is, an instance) and it is characterized by a set of *items* $p_1, \ldots, p_m$ (each item is completely described by a propositional letter). A

transaction T_i can be seen as a propositional model, and, therefore, the notion of T_i *satisfying* a set $P = \{p_1, p_2, \ldots\}$ of items ($T_i \Vdash P$) can be defined naturally. Such a notion can be immediately generalized to a set $\mathcal{T}' \subseteq \mathcal{T}$ of transactions, so that we can say that $\mathcal{T}'$ *satisfies* a set P of items ($\mathcal{T}' \Vdash P$) if every transaction in it does. Now, given a rule:

$$\rho : p_1 \wedge p_2 \wedge \ldots \wedge p_k \Rightarrow p,$$

Agrawal, Imieliński and Swami [2] implicitly define a notion of ρ *holding* on $\mathcal{T}$, which we denote $\mathcal{T} \Vdash^{\dagger} \rho$; given two real numbers s (*support*) and c (*confidence*), where $s, c \in (0, 1]$, we say that:

$$\mathcal{T} \Vdash^{\dagger} \rho \quad \text{if} \quad \exists \mathcal{T}' \subseteq \mathcal{T} \text{ s.t.} \quad \begin{array}{l} \frac{|\mathcal{T}'|}{|\mathcal{T}|} \geq s, \mathcal{T}' \Vdash \{p_1, \ldots, p_k, p\}, \text{ and} \\ \forall \mathcal{T}'' \supseteq \mathcal{T}' (\mathcal{T}'' \Vdash \{p_1, \ldots, p_k\} \rightarrow \frac{|\mathcal{T}'|}{|\mathcal{T}''|} \geq c), \end{array} \tag{3}$$

that is, we say that ρ holds on $\mathcal{T}$ if the fraction of transactions that show all items is sufficiently high (guaranteeing that the rule has enough support), and the fraction of transactions that only show the antecedent but not the consequent is sufficiently low (ensuring that the rule is – statistically – confident); we say, therefore, that a rule is *meaningful* if it has enough support and confidence. In the context of the definition of $\Vdash^{\dagger}$, the set $\{p_1, p_2, \ldots, p_k, p\}$ of propositions that occur in a rule that holds on $\mathcal{T}$ is said to be a *frequent set of items* (or a *frequent set of propositions*), because its support is high.

Evaluating temporal rules. In our case, transactions are generalized to ordered sets of transactions, that is, timelines, and a temporal data set $\mathcal{T} = \{T_1, \ldots, T_n\}$ is a collection of timelines. In addition to being meaningful, we want our rules to be *temporally meaningful*. To better understand this concept, consider, again, the medical domain, and assume that *NoSym* denotes the absence of a particular symptom. Under the standard logical semantics, the bodies of rules as the following ones:

$$\rho_1 : [G](Therapy \Rightarrow [A]NoSym),$$
$$\rho_2 : [G](Therapy \Rightarrow [L]NoSym),$$

expressing the fact that *after the therapy the symptom disappears*, are true on every interval that ends at the last point of the model. Since there are no intervals that start at the last point or later the literals $[A]NoSym$ and $[L]NoSym$ are true, and this makes the implication trivially true. However, in this situation the rules are not temporally meaningful: we do not have enough points to check whether the rule holds or not (in the same sense as the relation $\Vdash^{\dagger}$ given above for static rules). Similarly, when establishing if a certain universal literal holds on an interval, for example, $[B]p$ on $[x, y]$, we should consider the literal as holding on $[x, y]$ not only when $[B]p$ is logically true on $[x, y]$, but also also when p is falsified by a small fraction of the intervals of the type $[x, z]$ with $z < y$. Finally, if the body of a rule holds over almost every interval of a timeline, it seems natural to conclude that the rule holds on that timeline, even if there are some intervals that falsify the rules.

To avoid degenerate situations, in addition to support (s) and confidence (c) we introduce several new parameters that help us to define a notion of literal (or temporal item) holding at an interval on a timeline, as well as a notion of global rule holding on a timeline. For each relation R_X we introduce a real parameter called *universal confidence*, given by a pair of real numbers (e_X, u_X), which we use to determine on which intervals it does not make sense to ask whether a certain universal literal $[X]\lambda$ holds (parameter e_X), and on which fraction of the set of intervals captured by $[X]$ should a certain literal λ be true for $[X]\lambda$ to

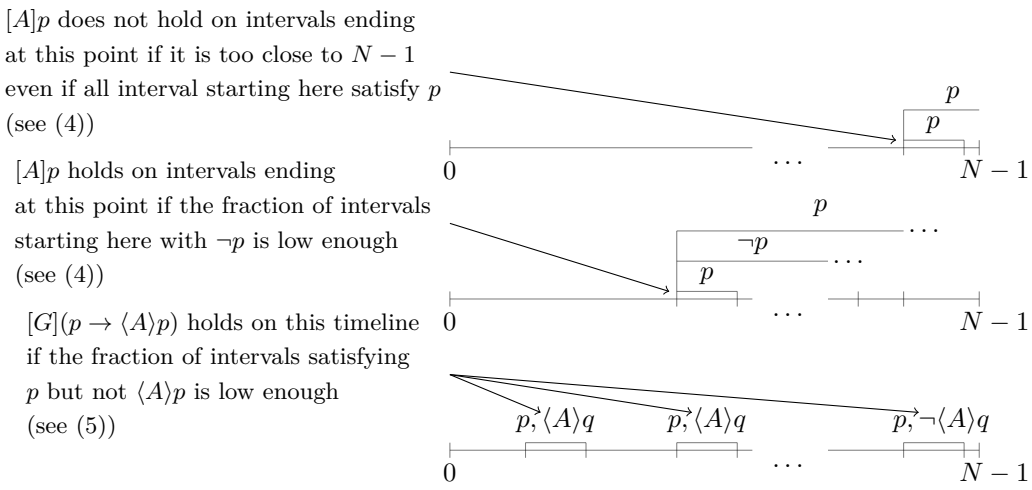


Figure 2 A pictorial (intuitive) representation of the concept of universal confidence (top and middle), and of that of global confidence (bottom).

be considered as holding on a given interval (parameter u_X). Recall that $T = \langle D, V\rangle$ and that D is finite, that is, $D = \{0, 1, \ldots, N-1\}$; for a given interval $[x, y]$, we define the notion of *literal holding at an interval* $(T, [x, y] \Vdash^\dagger \lambda)$ using the classical notion of $T, [x, y] \Vdash \lambda$ if λ is a propositional letter or an existential literal, and the following definition for universal literals:

$$\begin{array}{lll} T, [x,y] \Vdash^\dagger [A]\lambda & \text{if} & \frac{y+1}{N} \leq e_A \text{ and } \forall D' \subseteq D(\forall z \in D'((z > y) \wedge T, [y, z] \not\Vdash^\dagger \lambda) \to \frac{|D'|}{|N-y-1|} \leq u_A); \\ T, [x,y] \Vdash^\dagger [B]\lambda & \text{if} & \frac{y-x}{N} \geq e_B \text{ and } \forall D' \subseteq D(\forall z \in D'((x < z < y) \wedge T, [x, z] \not\Vdash^\dagger \lambda) \to \frac{|D'|}{|y-x|} \leq u_B); \\ T, [x,y] \Vdash^\dagger [E]\lambda & \text{if} & \frac{y-x}{N} \geq e_E \text{ and } \forall D' \subseteq D(\forall z \in D'((x < z < y) \wedge T, [z, y] \not\Vdash^\dagger \lambda) \to \frac{|D'|}{|y-x|} \leq u_E). \end{array} \quad (4)$$

The remaining modalities can be dealt with in a similar way. Notice that by setting $e_A = 1, e_B = e_E = 0$ and $u_A = u_B = u_E = 0$, one obtains the original, irreflexive semantics of HS. Moreover, simple modifications to the above semantics allow one to define reflexive and non-strict version of each operator.

Given a literal λ we say that λ *holds on a timeline* $T = (D, V)$, and we denote it by $T \Vdash^\dagger \lambda$, if there exists an interval $[x, y]$ such that $T, [x, y] \Vdash^\dagger \lambda$, and given a set $\Lambda = \{\lambda_1, \lambda_2, \ldots\}$, we say that Λ *holds on a timeline* T *at an interval* $[x, y]$, denoted by $T, [x, y] \Vdash^\dagger \Lambda$, if $T, [x, y] \Vdash^\dagger \lambda_j$ for each $\lambda_j \in \Lambda$. We also say that Λ *holds on a timeline* T, and write $T \Vdash^\dagger \Lambda$, to indicate that there exists an interval $[x, y]$ such that $T, [x, y] \Vdash^\dagger \Lambda$, and we write $T, I \Vdash^\dagger \Lambda$, where $I \subseteq I(D)$ if for each interval $[x, y] \in I$, we have $T, [x, y] \Vdash^\dagger \Lambda$. As in the static case, we indicate by $\mathcal{T}' \Vdash^\dagger \Lambda$ the fact that each timeline T in a set of timelines $\mathcal{T}' \subseteq \mathcal{T}$ is such that $T \Vdash^\dagger \Lambda$. Now, we introduce the real parameter $0 < g \leq 1$ (*global confidence*) to modulate on which fraction of all intervals of a timeline should the body of a rule hold to be considered a global rule. More precisely, given:

$$\rho : [G](\lambda_1 \wedge \ldots \wedge \lambda_k \Rightarrow \lambda),$$

we define the notion of ρ *holding on a timeline* T $(T \Vdash^\dagger \rho)$ by imposing:

$$\begin{array}{lll} T \Vdash^\dagger \rho & \text{if} & T \Vdash^\dagger \{\lambda_1, \ldots, \lambda_k, \lambda\} \text{ and} \\ & & \forall I(I \subseteq I(D) \wedge T, I \Vdash^\dagger \{\lambda_1, \ldots, \lambda_k, \neg\lambda\} \to \frac{|I|}{|I(D)|} \leq g). \end{array} \quad (5)$$

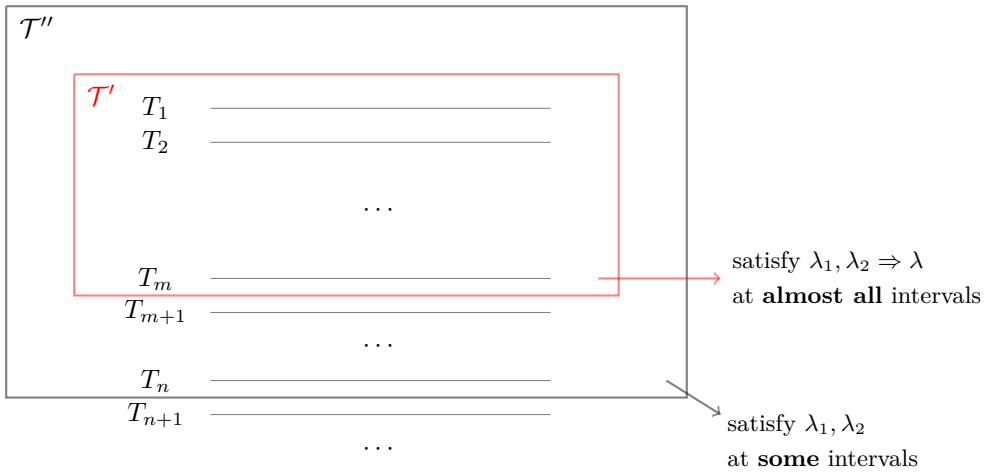


Figure 3 A pictorial (intuitive) representation of a rule $\lambda_1 \wedge \lambda_2 \Rightarrow \lambda$ holding on a temporal data set $\mathcal{T}$.

Setting $g = 0$ is equivalent to interpret a rule as a logical implication w.r.t. a single model (timeline). Fig. 2 gives an intuitive explanation of the use of universal and global confidence, which, together, we call *temporal confidence*.

Finally, we can generalize static evaluation to temporal evaluation, by using support and confidence, and define when ρ *holds in the temporal data set* $\mathcal{T}$ as follows:

$$\mathcal{T} \Vdash^{\dagger} \rho \quad \text{if} \quad \exists \mathcal{T}' \subseteq \mathcal{T} \quad \text{such that } \frac{|\mathcal{T}'|}{|\mathcal{T}|} \geq s \text{ and } \forall T \in \mathcal{T}', T \Vdash^{\dagger} \rho \text{ and } \forall \mathcal{T}'' \supseteq \mathcal{T}' (\mathcal{T}'' \Vdash^{\dagger} \{\lambda_1, \ldots, \lambda_k\} \rightarrow \frac{|\mathcal{T}'|}{|\mathcal{T}''|} \geq c). \tag{6}$$

In the context of temporal rule extraction, therefore, we say that ρ holds on $\mathcal{T}$ if it has enough support (the fraction of timelines in which the rule holds as defined in (5) is high) and it is confident (the fraction of timelines in which at least one interval satisfies $\{\lambda_1, \ldots, \lambda_k\}$ but no interval satisfy $\{\lambda_1, \ldots, \lambda_k, \lambda\}$ is low). A pictorial example of a simple rule holding on a temporal data set is given in Fig. 3.

To conclude, observe that while in static rules items are predetermined by the abstract data base of transactions, temporal items are not: for a given set $\mathcal{AP}$ of propositions, there are infinitely many possible literals, because there is no natural bound to the modal depth of the literals in the rules (i.e., the formulas) that are being mined. The most general solution to this problem is to introduce a parameter $m \in \mathbb{N}$, and limit ourselves to build rules in which literals have modal depth less or equal to m (the set of all such literals will be denoted by $\mathcal{L}_m(\mathcal{AP})$). Setting $m = 0$ is equivalent to extracting static rules that hold on (almost) every interval of the timelines in the support.

Temporal APRIORI. Our temporal extension of APRIORI is described in Algorithm 1. The procedure takes as input a temporal data set $\mathcal{T}$, the values for support, confidence, temporal confidence, and modal depth, and returns all and only global rules (limited to the specified modal depth) that hold on $\mathcal{T}$. In the following we analyse each step individually assuming, for the sake of simplicity, that $\mathcal{T}$ has n timelines of N points each, and that each timeline in $\mathcal{T}$ is *non-sparse* [26], that is, we assume that the number of intervals with non-empty propositional labels is at least linear in N (in this way, if timelines are represented as in Fig. 4, the complexity of the representation is linear in N).

Algorithm 1 Temporal APRIORI.

Require: $\mathcal{T}, s, c, g, (e_X, u_X), m$

1: $\Gamma, \mathcal{S} = \emptyset$;
2: compute the set $\mathcal{AP}$ of propositional letters occurring in $\mathcal{T}$;
3: compute the set $\mathcal{L}_m(\mathcal{AP})$;
4: label each $T_j \in \mathcal{T}$ and each interval $[x, y] \in I(D)$ with the maximal $\Lambda \subseteq \mathcal{L}_m(\mathcal{AP})$ such that $T_j, [x, y] \Vdash^\dagger \Lambda$;
5: **for** $\Lambda \subseteq \mathcal{L}_m(\mathcal{AP})$ **do**
6: **if** $\exists \mathcal{T}' \subseteq \mathcal{T}$ such that $\frac{|\mathcal{T}'|}{|\mathcal{T}|} \geq s$ and $\mathcal{T}' \Vdash^\dagger \Lambda$ **then**
7: $\mathcal{S} = \mathcal{S} \cup \{\Lambda\}$;
8: **end if**
9: **end for**
10: **for** $\Lambda \in \mathcal{S}$ **do**
11: **for** $\lambda \in \Lambda$ **do**
12: put $\rho = [G](\lambda_1 \wedge \ldots \wedge \lambda_k \Rightarrow \lambda)$, where $\{\lambda_1, \ldots, \lambda_k, \lambda\} = \Lambda$;
13: **if** $\mathcal{T} \Vdash^\dagger \rho$ **then**
14: $\Gamma = \Gamma \cup \{\rho\}$
15: **end if**
16: **end for**
17: **end for**
18: **for** $\rho \in \Gamma$ **do**
19: **if** ρ is redundant **then**
20: $\Gamma = \Gamma \setminus \{\rho\}$;
21: **end if**
22: **end for**
23: **return** Γ

- Computing temporal items for each timeline (lines 1–4). In our representation, scanning the data base to compute $\mathcal{AP}$ requires $O(|\mathcal{AP}| \cdot n)$ steps. Then, labeling each $T_i \in \mathcal{T}$ with the maximal set Λ of literals that holds at some interval of T_i while building, at the same time, the set $\mathcal{L}_m(\mathcal{AP})$ of all and only temporal items of depth up to m, requires executing (a simplified version of) the finite model checking algorithm for HS described in [26]. Under the assumption of non-sparseness, the algorithm requires $O(N^2 \cdot n \cdot |\mathcal{L}_m(\mathcal{AP})|)$ steps, using a symbolic representation, and $O(N^3 \cdot n \cdot |\mathcal{L}_m(\mathcal{AP})|)$ steps using the standard explicit representation.
- Computing frequent sets of literals (lines 5–9). The problem of finding frequent sets of temporal items can be reduced to the classical problem of finding frequent sets of items by adapting classical efficient solutions, such as FP-GROWTH [20], to the interval setting. FP-GROWTH builds a compact *frequent patterns tree* (also called *fp-tree*) that contains all necessary information for frequent sets generation. In analogy with the classical solution, our version of FP-GROWTH queries every timeline T in the temporal data base $\mathcal{T}$ a constant number of times to build the fp-tree [23]. In contrast to the classical solution, every query may need to examine every interval of the timeline to compute the frequency of some set of temporal items and and decide its position on the tree, and thus the complexity of a single query increases from $O(1)$ to $O(N^2)$. In terms of the overall complexity of the construction, this adds a factor $O(N^2)$ to the original solution, yet leaving the complexity linear in terms of the dimension of the temporal data base and polynomial in the number of frequent items. Since the number of frequent

N
$p : [0,1], [0,2], [0,3], [1,3], [2,3], \ldots$
$q : [0,1], [0,3], \ldots$
$\ldots$

Figure 4 A succinct representation of a timeline: the first line specifies the size of the frame (number of points in the model); the next lines encode the valuation function V.

items is bounded by the number of temporal items in $\mathcal{L}_m(\mathcal{AP})$, the worst-case complexity of FP-GROWTH in our setting is $O(n \cdot |\mathcal{L}_m(\mathcal{AP})| + |\mathcal{L}_m(\mathcal{AP})| \log |\mathcal{L}_m(\mathcal{AP})|)$, where $n \cdot |\mathcal{L}_m(\mathcal{AP})|$ is the cost of building the fp-tree and $|\mathcal{L}_m(\mathcal{AP})| \log |\mathcal{L}_m(\mathcal{AP})|$ the cost of sorting the set of frequent items by decreasing support (as required by the algortithm). Enumerating the frequent sets can be performed by suitably querying the fp-tree built by FP-GROWTH, which is, at this point, essentially indistinguishable from a non-temporal fp-tree. We can assume that the structure containing every frequent set, that is, $\mathcal{S}$, can be designed in such a way that the cost of querying the frequency of any $\Lambda \in \mathcal{S}$ is constant.

- Generating and testing rules (lines 10–17). For a single frequent set Λ, we generate $|\Lambda|$ different rules. Testing each one of the rules against (5) requires:
  - **(i)** two lookup operations (each with constant cost) on the data structure that contains $\mathcal{S}$, to compare the support of the antecedent with the support of the entire set, and establish if the rule is confident, plus
  - **(ii)** $O(n \cdot N^2)$ operations to establish if the rule is also global.

  Therefore this step costs $O(|\mathcal{S}| \cdot |\mathcal{L}_m(\mathcal{AP})| \cdot n \cdot N^2)$. While the size of the frequent patterns tree is polynomial in the dimension of the data set, the number of possible frequent sets that can be extracted from it (that is, $|\mathcal{S}|$) may be exponential, and its actual size depends on the particular data set as well as the setting of each parameter.
- Checking redundancy. From the set Γ of all meaningful rules that hold on $\mathcal{T}$, we eliminate all redundant ones. A rule $[G](\lambda_1 \wedge \ldots \wedge \lambda_k \Rightarrow \lambda)$ is redundant if Γ includes also a rule with λ as a consequence and with an antecedent that is a subset of $\{\lambda_1, \ldots, \lambda_k\}$. This step can be easily completed within $O(|\Gamma|^2 \cdot |\mathcal{L}_m(\mathcal{AP})|^2)$ operations (this limit can be improved with a suitable symbolic representation of the rules).

By summing up all the above steps we obtain the following overall worst-case complexity for Temporal APRIORI:

$$O(|\mathcal{S}| \cdot |\mathcal{L}_m(\mathcal{AP})| \cdot n \cdot N^2 + |\mathcal{L}_m(\mathcal{AP})| \log |\mathcal{L}_m(\mathcal{AP})| + |\Gamma|^2 \cdot |\mathcal{L}_m(\mathcal{AP})|^2).$$

The set $\mathcal{L}_m(\mathcal{AP})$ contains all positive temporal literals that can be built over the set of propositional letters $\mathcal{AP}$ with maximum modal depth m. Since the syntax of HS includes 24 different temporal modalities (one diamond and one box operator for each of the 12 Allen's relations), we have that $|\mathcal{L}_m(\mathcal{AP})| = O(24^m \cdot |\mathcal{AP}|)$. Moreover, in the worst-case scenario the set $\mathcal{S}$ includes all subsets of $\mathcal{L}_m(\mathcal{AP})$ while the set of meaningful rules includes all temporal rules that can be built from $\mathcal{L}_m(\mathcal{AP})$. These observations lead to a worst-case upper bound on the complexity of Temporal APRIORI that is exponential in m and $|\mathcal{AP}|$, but polynomial in n and N. As a general observation, for enumerative algorithms the exponential number of solutions makes the usual analysis of running time, that considers only the input size and ignore the output size, scarcely informative. In the literature, notions of efficiency have been developed for enumerative algorithms [22, 18], but a precise assessment of the complexity of Temporal APRIORI following such notions is outside the scope of this paper; here, we limit ourselves to some empirical considerations. First of all, performance studies show

that the size of a fp-tree is usually substantially smaller than the original data base [20] and well beyond the worst-case upper bound of a complete tree. Similarly, providing a tighter upper bound to the number of frequent sets (and therefore to the number of rules) that can be extracted from a data set is not really possible as the number of frequent sets depends on the application domain and on the actual data. As we have observed, efficient implementation solutions for APRIORI can be adapted to our temporal generalization, and further optimizations should be focused on the temporal steps only. Experimental studies show that the elapsed time for efficient implementations of APRIORI grows linearly with the dimension of the data base. For comparison, assume that N and m are constants, and that the set $\mathcal{AP}$ (and therefore the set $\mathcal{L}_m(\mathcal{AP})$) is fixed. Then, the complexity of Temporal APRIORI is $O(n)$ plus the complexity of checking redundancy in rules. Since this latter aspect is not directly discussed in APRIORI, and the available experimental data do not include the redundancy check, in our simplified circumstances, Temporal APRIORI has the same complexity as APRIORI.

While considering a situation in which the set of interesting items grows is not really interesting, there are certain scenarios in which the length of the timelines may grow. Indeed, this may occur in both examples discussed in Section 3. In the medical scenario the length of timelines may grow as the duration of recorded medical histories increases with time. In the natural language knowledge extraction, timelines may grow as the average duration of a conversation increases. If N is not constant, the complexity of Temporal APRIORI (besides redundancy checking) is $O(N^2 \cdot n)$, and the bottleneck is temporal items labeling and rule globality checking.

5 Experimental Results

Testing the ability of Temporal APRIORI to extract meaningful rules from a temporal data set presents two main difficulties. First, there are very few publicly available data sets with a relevant temporal component. Such a data set, moreover, should be preprocessed in order to highlight the interesting propositions, and to hide irrelevant aspects (this is a standard procedure in knowledge extraction), which requires a profound knowledge of the applicative domain. In this way, the experiment would be influenced by such a preparation, making it more difficult for the performances of the algorithm to emerge. Second, the rules of a real-life data set are unknown beforehand; although there are indirect techniques that allow one to evaluate a set of rules, these evaluations are not always adequate to assess the capabilities of a new learning algorithm.

We designed a simple technique to generate an artificial data set $\mathcal{T}$ in such a way that we can control the rules to be extracted. Given a set Γ of input rules written in the language of Horn HS over a set $\mathcal{AP}$ of propositional letters, a natural number $N \geq 2$, fixed the parameter (e_X, u_X) for every relation R_X, and fixed the support s_ρ for every rule $\rho \in \Gamma$, our generator produces a temporal data set $\mathcal{T}$, where every timeline T has domain D_T of length N. To this end, for each rule $\rho \in \Gamma$, where $\rho : (\lambda_1 \wedge \ldots \wedge \lambda_k \Rightarrow \lambda)$, we first choose randomly the set $\mathcal{T}_\rho$ of timelines that will satisfy the rule ρ. Then, for each timeline $T \in \mathcal{T}_\rho$, we choose, randomly, an interval $[x, y]$, and we fix the propositional labeling of the intervals in T so that $T, [x, y] \Vdash^\dagger \{\lambda_1, \ldots, \lambda_k, \lambda\}$, which entails setting the labeling of a certain subset $S_{T,\rho}$ of $I(D_T)$ (e.g., forcing $\langle A \rangle p$ to be true on $[x, y]$ entails forcing p to be true on some interval $[y, z]$). The propositional labeling of every interval in the set $I(D_T) \setminus S_{T,\rho}$ is chosen randomly. Then, in order to ensure that ρ is global on T, every interval in $I(D)$ is considered (again) and its labeling fixed, if necessary. Notice that no contradictions may arise, since only positive

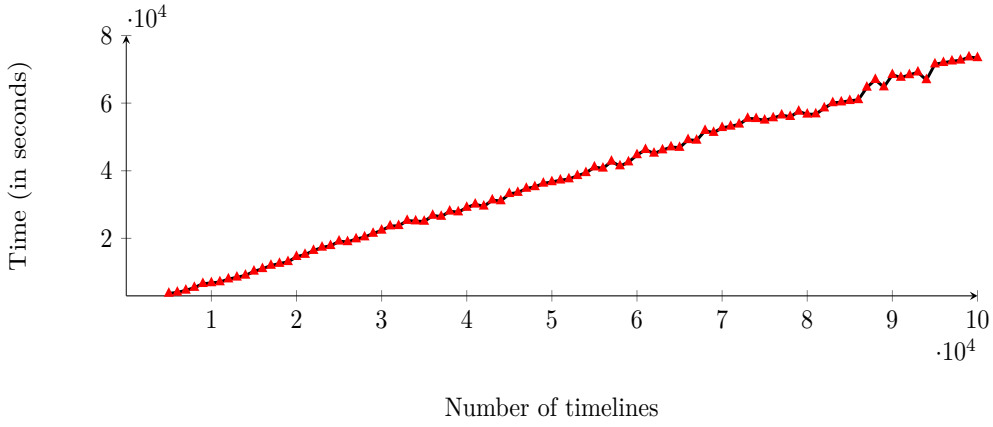


Figure 5 Elapsed time for rule extraction.

propositional literals are added at each step in each labeling: obviously, we generate only satisfiable examples of rules without $\bot$. In this way, the confidence of each rule ρ is 1, as well as its global confidence, and its support is equal to, or grater, than s_ρ. The aim of testing a rule extraction algorithm on an artificial data set is not that of finding an optimal set of parameters, which makes sense on real data sets only: in a controlled environment, every meaningful rule can be eventually discovered. Therefore, besides the proof-of-concept, our test is focused on performances only. We generated 96 temporal data sets, where timelines have a domain of cardinality less or equal to 100 points. The size of temporal data sets ranges from a minimum of 500 to a maximum of 10000 timelines. We fixed the following three rules:

$$\rho_1 : [G](p_1 \wedge [A]p_2 \wedge [B]p_3 \Rightarrow [A]p_4)$$
$$\rho_2 : [G]([A]p_1 \wedge [A]p_2 \Rightarrow [A]p_3)$$
$$\rho_3 : [G]([A]p_1 \wedge [E]p_2 \Rightarrow [B]p_3),$$

and we generated each data set in such a way that (at least) 60% of timelines satisfy ρ_1, the 80% of which satisfy also ρ_2, and the 60% of the latter satisfy also ρ_3.

As we have observed in the previous section, under certain conditions the complexity of Temporal APRIORI is linear in the number of timelines in $\mathcal{T}$. In our experiment, run on an Intel Core i7 with 4 cores at 2.80GHz, such conditions are met: N and m are constant (respectively 100 points an modal depth 1), and the set $\mathcal{AP} = \{p_1, p_2, p_3, p_4\}$ is fixed. Fig. 5 shows the expected linear behaviour.

6 Conclusions

Rule-based methods are algorithms that extract regularities in data that can be expressed in the form of *if-then* association rules, and are very popular in machine learning and data mining. The first and most recognized algorithm for extracting static association rules is known as APRIORI. The idea underlying APRIORI is that from a set of instances, referred to as transactions, each one characterized by a set of items (propositional letters), one can extract, first, frequent sets of items (that is, sets of propositions that occur often together), and, from them, rules. The latter are evaluated by using their support, that is, the fraction of transactions that satisfy all items in the rule, and their confidence, that is, the set of transactions that satisfy only the set of items corresponding to the antecedent of the rule.

This work is inspired by the fact that static rules, as those extracted by APRIORI, can be seen as implications written in the Horn fragment of propositional logic, though they are not interpreted as such. With the aim of extracting temporal rules from a temporal data set using similar principles, we first considered a suitable language that extends Horn propositional logic with temporal capabilities, that is, the Horn fragment of the temporal logic HS. Such a language has been recently studied, along with its computational properties, and the fact that it directly generalizes Horn propositional logic, and that some of its fragments have interesting computational characteristics, make it the ideal candidate for our purposes. Second, we generalized APRIORI in such a way that it is able to extract rules written in Horn HS. To this end, we introduced the notion of temporal confidence, which, paired with the classical notions of support and confidence, allowed us to define a relaxed version of the truth relation, and, ultimately, to devise the algorithm Temporal APRIORI, which we tested on an artificial data set.

While HS is a very general and expressively powerful language, the ability of Temporal APRIORI to extract meaningful rules decreases as the modal depth of the rules to be captured (that is, the cardinality of $\mathcal{L}_m(\mathcal{AP})$) increases. For example, in the language of HS, interpreted over finite linearly ordered sets, one can express the rule: *if p holds on some interval with length more than K, then q will hold on some of its subintervals*. However, this requires forcing an interval to have a minimal length, which in HS can be expressed as:

$$len_{>K} = \underbrace{\langle B\rangle\langle B\rangle \dots \langle B\rangle}_{K} \top,$$

In turn, extracting a rule with $len_{>K}$ among its antecedents entails setting $m \geq K$, which, ultimately, leads the algorithm to generate and test a very high number of rules, and, in many cases, to return many rules that are not significant from the domain point of view. A similar situation arises with *coarser* Allen's relations in HS [30], which are relations that allow one to express an incomplete temporal knowledge (e.g.: *the symptom disappears after or immediately after the therapy*) and are expressible in the language of HS, but with formulas that do not immediately translate to Horn HS:

$$[G](Therapy \Rightarrow ([A]NoSym \vee [L]NoSym)).$$

As future work, we plan to improve Temporal APRIORI to extract rules with complex temporal literals and with an enriched set of basic relations in a direct way (that is, enriching the language in which the rules are written), effectively circumventing the above problems. Moreover, we plan to test Temporal APRIORI on natural data sets coming from relevant application domains like the medical domain and the natural language processing domain. This will allow us to evaluate the impact of parameters setting, as well as the expected improvement on the quality of extracted rules, with respect to real case studies.

References

1 L. Aceto, D. Della Monica, V. Goranko, A. Ingólfsdóttir, A. Montanari, and G. Sciavicco. A complete classification of the expressiveness of interval logics of Allen's relations: the general and the dense cases. *Acta Informatica*, 53(3):207–246, 2016.

2 R. Agrawal, T. Imieliński, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

3 R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the 11th International Conference on Data Engineering*, pages 3–14, 1995.

4 J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

5 R. Alluhaibi. Simple interval temporal logic for natural language assertion descriptions. In *Proc. of the 11th International Conference on Computational Semantics*, pages 283–293, 2015.

6 R.A. Baeza-Yates. Challenges in the interaction of information retrieval and natural language processing. In *Proc. of the 5th International on Computational Linguistics and Intelligent Text Processing (CICLing)*, pages 445–456, 2004.

7 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Satisfiability and model checking for the logic of sub-intervals under the homogeneity assumption. In *Proc. of the 44th International Colloquium on Automata, Languages, and Programming*, pages 120:1–120:14, 2017.

8 D. Bresolin, D. Della Monica, A. Montanari, and G. Sciavicco. The light side of interval temporal logic: the Bernays-Schönfinkel fragment of CDT. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):11–39, 2014.

9 D. Bresolin, A. Kurucz, E. Muñoz-Velasco, V. Ryzhikov, G. Sciavicco, and M. Zakharyaschev. Horn fragments of the Halpern-Shoham interval temporal logic. *ACM Transactions on Computational Logic*, 18(3):22:1–22:39, 2017.

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

11 D. Bresolin, E. Muñoz-Velasco, and G. Sciavicco. Fast(er) reasoning in interval temporal logic. In *Proc. of the 26th Annual Conference on Computer Science Logic*, volume 82 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

12 C. Combi, M. Mantovani, and P. Sala. Discovering quantitative temporal functional dependencies on clinical data. In *Proc. of the 2017 IEEE International Conference on Healthcare Informatics*, pages 248–257, 2017.

13 C. Combi and P. Sala. Mining approximate interval-based temporal dependencies. *Acta Informatica*, 53(6-8):547–585, 2016.

14 P. Cotofrei and K. Stoffel. Classification rules + time = temporal rules. In *Proc. of the 2002 International Conference on Computational Science*, pages 572–581, 2002.

15 S. de Amo, D. A. Furtado, A. Giacometti, and D. Laurent. An apriori-based approach for first-order temporal pattern. *Journal of Information and Data Management*, 1(1):57–70, 2010.

16 J. Fürnkranz, D. Gamberger, and N. Lavrac. *Foundations of Rule Learning*. Cognitive Technologies. Springer, 2012.

17 J. Fürnkranz and T. Kliegr. A brief overview of rule learning. In *Proc. of the 9th International Symposium Rule Technologies: Foundations, Tools, and Applications*, pages 54–69, 2015.

18 Leslie Ann Goldberg. *Efficient Algorithms for Listing Combinatorial Structures*. Distinguished Dissertations in Computer Science. Cambridge University Press, New York, NY, USA, 1993.

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

20 J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

21 F. Jiménez, G. Sánchez, and J.M. Juárez. Multi-objective evolutionary algorithms for fuzzy classification in survival prediction. *Artificial Intelligence in Medicine*, 60(3):197–219, 2014.

22 David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

23 W.A. Kosters, W. Pijls, and V. Popova. Complexity analysis of depth first and fp-growth implementations of APRIORI. In *Proc. of the 3rd International Conference on Machine Learning and Data Mining in Pattern Recognition*, pages 284–292, 2003.

24 Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Artificial Intelligence. Springer, 1992.

25 A. Molinari, A. Montanari, A. Murano, G. Perelli, and A. Peron. Checking interval properties of computations. *Acta Informatica*, 53(6-8):587–619, 2016.

26 D. Della Monica, D. de Frutos-Escrig, A. Montanari, A. Murano, and G. Sciavicco. Evaluation of temporal datasets via interval temporal logic model checking. In *Proc. of the 24th International Symposium on Temporal Representation and Reasoning*, volume 90 of *LIPIcs*, pages 11:1–11:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

27 A. Montanari, I. Pratt-Hartmann, and P. Sala. Decidability of the logics of the reflexive sub-interval and super-interval relations over finite linear orders. In *Proc. of the 17th International Symposium on Temporal Representation and Reasoning*, pages 27–34. IEEE Computer Society, 2010.

28 A. Montanari, G. Puppis, and P. Sala. Maximal decidable fragments of Halpern and Shoham's modal logic of intervals. In *Proc. of the 37th International Colloquium on Automata, Languages and Programming*, volume 6199 of *LNCS*, pages 345–356. Springer, 2010.

29 S. Muggleton. Inductive logic programming: Issues, results and the challenge of learning language in logic. *Artificial Intelligence*, 114(1-2):283–296, 1999.

30 E. Muñoz-Velasco, M. Pelegrín-García, P. Sala, and G. Sciavicco. On coarser interval temporal logics and their satisfiability problem. In *Proc. of the 16th Conference of the Spanish Association for Artificial Intelligence*, volume 9422 of *LNCS*, pages 105–115. Springer, 2015.

31 I. Pratt-Hartmann. Temporal prepositions and their logic. *Artificial Intelligence*, 166(1–2):1–36, 2005.

32 L. De Raedt. *Logical and relational learning*. Cognitive Technologies. Springer, 2008.

Faster Dynamic Controllability Checking for Simple Temporal Networks with Uncertainty

Massimo Cairo
Department of Mathematics, University of Trento, Italy
massimo.cairo@unitn.it

Luke Hunsberger
Computer Science Department, Vassar College, Poughkeepsie, NY USA
hunsberger@vassar.edu

Romeo Rizzi
Department of Computer Science, University of Verona, Italy
romeo.rizzi@univr.it

Abstract

Simple Temporal Networks (STNs) are a well-studied model for representing and reasoning about time. An STN comprises a set of real-valued variables called time-points, together with a set of binary constraints, each of the form $Y \leq X + w$. The problem of finding a feasible schedule (i.e., an assignment of real numbers to time-points such that all of the constraints are satisfied) is equivalent to the Single Source Shortest Path problem (SSSP) in the STN graph.

Simple Temporal Networks with Uncertainty (STNUs) augment STNs to include *contingent links* that can be used, for example, to represent actions with uncertain durations. The duration of a contingent link is not controlled by the planner, but is instead controlled by a (possibly adversarial) environment. Each contingent link has the form, $\langle A, \ell, u, C \rangle$, where $0 < \ell \leq u < \infty$. Once the planner executes the activation time-point A, the environment must execute the contingent time-point C at some time $A + \Delta$, where $\Delta \in [\ell, u]$. Crucially, the planner does not know the value of Δ in advance, but only discovers it when C executes. An STNU is dynamically controllable (DC) if there is a strategy that the planner can use to execute all of the non-contingent time-points, such that all of the constraints are guaranteed to be satisfied no matter which durations the environment chooses for the contingent links. The strategy can be dynamic in that it can react in real time to the contingent durations it observes. Recently, an upper bound of $O(N^3)$ was given for the DC-checking problem for STNUs, where N is the number of time-points.

This paper introduces a new algorithm, called the RUL^- algorithm, for solving the DC-checking problem for STNUs that improves on the $O(N^3)$ bound. The worst-case complexity of the RUL^- algorithm is $O(MN + K^2N + KN \log N)$, where N is the number of time-points, M is the number of constraints, and K is the number of contingent time-points. If M is $O(N^2)$, then the complexity reduces to $O(N^3)$; however, in sparse graphs the complexity can be much less. For example, if M is $O(N \log N)$, and K is $O(\sqrt{N})$, then the complexity of the RUL^- algorithm reduces to $O(N^2 \log N)$.

The RUL^- algorithm begins by using the Bellman-Ford algorithm to compute a potential function. It then performs at most $2K$ rounds of computations, interleaving novel applications of Dijkstra's algorithm to (1) generate new edges and (2) update the potential function in response to those new edges. The constraint-propagation/edge-generation rules used by the RUL^- algorithm are distinguished from related work in two ways. First, they only generate *unlabeled* edges. Second, their applicability conditions are more restrictive. As a result, the RUL^- algorithm requires only $O(K)$ rounds of Dijkstra's algorithm, instead of the $O(N)$ rounds required by other approaches. The paper proves that the RUL^- algorithm is sound and complete for the DC-checking problem for STNUs.

2012 ACM Subject Classification Networks → Network algorithms

25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek; Article No. 8; pp. 8:1–8:16

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

Keywords and phrases Simple Temporal Networks with Uncertainty, Dynamic Controllability, Temporal Planning under Uncertainty

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.8

1 Introduction

Simple Temporal Networks (STNs) are a well-studied model for representing temporal constraints [5]. An STN comprises a set of time-points $\{P, Q, R, \dots\}$, together with a set of constraints on those time-points, where each constraint has the form $Q \leq P + w$ with $w \in \mathbb{R}$. The goal for a planning agent is to schedule the execution of the time-points (i.e., to assign a real value to each variable $P, Q, R, \dots$ representing its execution time) so that all of the constraints in the network are satisfied. The STN model forms the base of many other models for temporal reasoning and planning.

It is possible to check whether an STN is consistent (i.e., whether it admits a schedule that satisfies all of the constraints, and find such a scheduling if one exists) in polynomial time. Specifically, this problem can be reduced to the Single Source Shortest Path problem for the STN graph [5], which contains a node for each time-point and a weighted, directed edge for each constraint. In turn, this problem can be solved with the Bellman-Ford algorithm [4], whose running time is $O(MN)$, where N is the number of time-points, and M is the number of edges in the network.

Simple Temporal Networks with Uncertainty (STNUs) extend STNs to include *contingent links,* which can be used, for example, to represent actions with uncertain durations [14]. Each contingent link has the form $\langle A, \ell, u, C \rangle$, where A is called the *activation* time-point, C is called the *contingent* time-point, and $0 < \ell \leq u < \infty$. The contingent link is activated when the time-point A is executed. Once that happens, the execution of C is determined not by the planning agent, but by the (possibly adversarial) environment. In particular, the environment must execute C at some time Δ after the execution of A, where $\Delta \in [\ell, u]$. The value Δ is called the duration of the contingent link, it is under the control of the environment, and is unknown to the planner until C is actually executed.

An STNU is said to be dynamically controllable (DC) if there exists a strategy for the planning agent to execute all of the *non-contingent* (a.k.a., *executable*) time-points such that all of the constraints in the network are guaranteed to be satisfied no matter how the durations of the contingent links are chosen by the environment. The strategy must be *dynamic* in the sense that the execution time it chooses for each executable time-point X can only depend on the durations of contingent links that have already completed. In other words, the strategy's execution decisions must depend only on past execution events.

The DC-checking problem for STNUs, hereinafter called the STNU-DC problem, is that of determining whether any given STNU is DC. Recently, Morris [12] presented an $O(N^3)$-time algorithm for the STNU-DC problem.

Being such a simple and flexible model, STNUs have been extended in several ways in the literature [15, 10, 3, 6, 2]. For this reason, it is crucial to study and optimize algorithms for the STNU-DC problem.

2 Preliminaries and notation

Following Morris et al. [14], an STNU is a tuple $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ where:

- $\mathcal{T}$ is a finite set of real-valued variables called *time-points,* denoted by capital letters $P, Q, R, \dots$;

- $\mathcal{C}$ is a finite set of *constraints,* each of the form $Q \leq P + w$, for some $P, Q \in \mathcal{T}$ and $w \in \mathbb{R}$; and
- $\mathcal{L}$ is a finite set of *contingent links,* each of the form $\langle A, \ell, u, C\rangle$, for some $A, C \in \mathcal{T}$ and $\ell, u \in \mathbb{R}$, where $0 < \ell \leq u < \infty$.

In a contingent link $\langle A, \ell, u, C\rangle$, A is called the *activation* time-point, and C is called the *contingent* time-point. Distinct contingent links must have distinct contingent time-points. Given the contingent time-point C of a contingent link $\langle A, \ell, u, C\rangle \in \mathcal{L}$ we define $A^C = A$, $u^C = u$, $l^C = l$; however, when context allows, the superscripts will be omitted. The set of contingent time-points is $\mathcal{T}_{\mathrm{C}} := \{C \mid (A, l, u, C) \in \mathcal{L}\}$; and the set of *executable* (or *non-contingent*) time-points is $\mathcal{T}_{\mathrm{X}} = \mathcal{T} \setminus \mathcal{T}_{\mathrm{C}}$. As in prior work, and without loss of generality, we assume that each activation time-point is an executable time-point. In the following, we assume that an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is given.

2.1 Dynamic controllability

Following Morris et al. [14], a *situation* is a function $\sigma\colon \mathcal{T}_{\mathrm{C}} \to \mathbb{R}$ that assigns a *duration* $\Delta^C_\omega := \omega(C) \in [\ell, u]$ to each contingent link $\langle A, \ell, u, C\rangle \in \mathcal{L}$. The set of all possible situations is denoted by $\Omega := \prod_{C \in \mathcal{T}_{\mathrm{C}}} [\ell^C, u^C]$. An *execution strategy* is a function $\sigma\colon (\Omega, \mathcal{T}_{\mathrm{X}}) \to \mathbb{R}$ that assigns an *execution time* $[\sigma(\omega)]_X := \sigma(\omega, X)$ to each executable time-point X, in each possible situation $\omega \in \Omega$. Given a contingent link $\langle A, \ell, u, C\rangle \in \mathcal{L}$, we define the *execution time* of its contingent time-point C to be $[\sigma(\omega)]_C := [\sigma(\omega)]_A + \Delta^C_\omega$. In general, $[\sigma(\omega)]_P \in \mathbb{R}$ denotes the execution time of the (executable or contingent) time-point $P \in \mathcal{T}$ in the situation ω under the strategy σ.

An execution strategy σ is *viable* if it satisfies every constraint (i.e., for each constraint $Q \leq P + w$ in $\mathcal{C}$, and each situation $\omega \in \Omega$, $[\sigma(\omega)]_Q \leq [\sigma(\omega)]_P + w$). The strategy σ is *dynamic* if, for any two situations $\omega_1, \omega_2 \in \Omega$, and for each executable time-point $X \in \mathcal{T}_{\mathrm{X}}$, if $\{\langle C, \Delta^C_{\omega_1}\rangle \mid [\sigma(\omega_1)]_C < [\sigma(\omega_1)]_X\} = \{\langle C, \Delta^C_{\omega_2}\rangle \mid [\sigma(\omega_2)]_C < [\sigma(\omega_1)]_X\}$ then $[\sigma(\omega_2)]_X = [\sigma(\omega_1)]_X$. This definition correctly captures the intuitive notion that the execution time of X can only depend only on the durations of contingent links whose contingent time-points have executed *before* X [7]. An STNU is said to be *dynamically controllable* (DC) if it admits an execution strategy that is both dynamic and viable.

To simplify the mathematics, Morris [11] provided an alternative notion of dynamic controllability that allows a dynamic strategy to react *instantaneously* to observations of contingent executions. The only change required to the above definition of a dynamic strategy is to replace "$<$" by "$\leq$". That change allows the execution of X to depend on contingent links that have completed *at or before* X. The rest of the paper presumes DC with instantaneous reaction.

2.2 Constraint Propagation/Edge Generation in STNU Graphs

Following Morris and Muscettola [13], each STNU can be represented by a directed, weighted graph where the nodes correspond to the time-points, and the edges come in three varieties depending on whether and how they are labeled. First, each constraint $Q \leq P + w$ in $\mathcal{C}$ is represented by an *ordinary* (i.e., unlabeled) edge from P to Q of length (or weight) w, notated as (P, w, Q).[1] Next, for each contingent link $\langle A, \ell, u, C\rangle$ there is a *lower-case* edge from A to C labeled by $[c\colon \ell]$ that represents the *uncontrollable possibility* that the contingent

[1] Putting the weight w between the time-points P and Q makes notating paths easier. For example, the path consisting of the consecutive edges (P, w, Q) and (Q, v, R) can be notated as (P, w, Q, v, R).

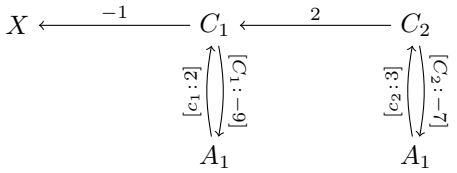


Figure 1 A sample STNU graph with two contingent links $(A_1, 2, 9, C_1)$ and $(A_2, 3, 7, C_2)$.

Table 1 Contraint-propagation rules from Morris and Muscettola [13].

Rule	Graphical Representation	Applicability Conditions
No Case	$X \xrightarrow{v} Y \xrightarrow{w} W$; $X \dashrightarrow W$: $v+w$	(none)
Upper Case	$X \xrightarrow{v} Y \xrightarrow{[C:-w]} A$; $X \dashrightarrow A$: $[C:v-w]$	(none)
Lower Case	$A \xrightarrow{[c:\ell]} C \xrightarrow{w} X$; $A \dashrightarrow X$: $\ell+w$	$w < 0$
Cross Case	$A \xrightarrow{[c:\ell]} C \xrightarrow{[K:-w]} A^K$; $A \dashrightarrow A^K$: $[K:\ell-w]$	$K \not\equiv C$, $w < 0$
Label Removal	$X \xrightarrow{[C:-y]} A \xrightarrow{[c:\ell]} C$; $X \dashrightarrow A$: $-y$	$-y \geq -\ell$

duration $C - A$ might take on its lower bound ℓ; and an *upper-case* edge from C to A labeled by $[C:-u]$ that represents the *uncontrollable possibility* that the contingent duration $C - A$ might take on its upper bound u. These edges are notated as $(A, [c:\ell], C)$ and $(C, [C:-u], A)$, respectively. The sets of ordinary, lower-case and upper-case edges are denoted by $\mathcal{E}^o, \mathcal{E}^\ell$ and $\mathcal{E}^u$, respectively:

- $\mathcal{E}^o = \{(P, w, Q) \mid (Q \leq P + w) \in \mathcal{C}\}$
- $\mathcal{E}^\ell = \{(A, [c:\ell], C) \mid \langle A, \ell, u, C\rangle \in \mathcal{T}_{\mathrm{C}}\}$
- $\mathcal{E}^u = \{(C, [C:-u], A) \mid \langle A, \ell, u, C\rangle \in \mathcal{T}_{\mathrm{C}}\}$

Fig. 1 shows a sample STNU graph borrowed from Hunsberger [8].

For convenience, the graph consisting of all of the ordinary and upper-case edges (i.e., $\mathcal{E}^o \cup \mathcal{E}^u$) is called the OU-graph; and the graph consisting of all of the lower-case and ordinary edges (i.e., $\mathcal{E}^o \cup \mathcal{E}^\ell$) is called the LO-graph.

Morris et al. [14] introduced a set of *triangular reductions* that formed the basis of a *pseudo-polynomial* DC-checking algorithm for STNUs. Their reductions generated and propagated a new kind of conditional constraint called a *wait* that captures part of the dynamism of the STNU-DC problem. Each wait constraint can be glossed as "while the contingent time-point C remains unexecuted, the time-point X must wait until $A + \delta$, where A is the activation time-point for C and $\delta \in \mathbb{R}$." Morris and Muscettola [13] then recast those reductions as the five constraint-propagation rules listed in Table 1.[2] In the figure, the pre-existing edges are drawn with solid arrows, while the generated edges are drawn with dashed arrows.

[2] The rules in Table 1 are presented in the form that allows instantaneous reaction. To disallow instantaneous reaction, one need only change each occurrence of "$w < 0$" to "$w \leq 0$".

The *No Case* rule is the same as the RELAX rule from standard shortest-path algorithms [4]. The *Lower Case* rule generates new constraints that guard against the possibility that a contingent link $\langle A, \ell, u, C\rangle$ might take on its minimum duration ℓ. Similarly, the *Upper Case* rule generates new constraints that guard against the possibility that a contingent link might take on its maximum duration u. However, unlike the *Lower Case* rule, which generates ordinary edges, the *Upper Case* rule generates upper-case edges that represent conditional wait constraints. For example, the generated edge $(X, [C\!:\!v - w], A)$ represents the conditional constraint that X must wait at least $(w - v)$ after A as long as C remains unexecuted. The *Cross Case* rule generates constraints that guard against one contingent link $\langle A, \ell, u, C\rangle$ taking on its minimum duration ℓ, while another contingent link $\langle A^K, \ell^K, u^K, K\rangle$ takes on its maximum duration u^K. Like the *Upper Case* rule, it generates upper-case edges. Finally, the *Label Removal* rule stipulates that in certain cases an upper-case edge has the force of an unconditional constraint. Each of the rules in Table 1 has been proven to be sound in the sense that if a valid and dynamic execution strategy σ satisfies the pre-existing edge(s) in the rule, then it must necessarily also satisfy the edge generated by that rule.

An important property of the rules in Table 1 is that they are *length-preserving* (i.e., the length of the generated edge is the same as the length of the corresponding path/edge from which it was derived). This property is exploited by the $O(N^5), O(N^4)$ and $O(N^3)$ DC-checking algorithms due to Morris and colleagues [13, 11, 12].

Each of the rules from Table 1 can be viewed as a path-transformation rule. For example, suppose that a path P contains two consecutive edges E_1 and E_2 to which one of the first four rules can be applied to generate a new edge E. If P' is the path obtained from P by replacing the two edges E_1 and E_2 by the new edge E, then we say that P has been transformed into P'. Similarly, the *Label Removal* rule can be viewed as a path-transformation rule that involves the replacement of just one edge.

Morris [11] provided a theoretical analysis of *semi-reducible* paths in STNU graphs that underlies much of the important work on DC-checking algorithms. A semi-reducible path is any path P that can be transformed into a path P' by any sequence of applications of the rules from Table 1 such that P' contains only ordinary or upper-case edges. Morris proved that an STNU is DC if and only if its graph has no semi-reducible negative loops (SRN loops). (Note that a negative loop containing only ordinary or upper-case edges can be further transformed by the *Upper Case* rule into a negative loop that contains only upper-case edges, which represents an inherently unsatisfiable cycle of constraints.) Central to his analysis was the process of "reducing away" lower-case edges – that is, performing a sequence of transformations that eliminate lower-case edges from the path.

Morris' $O(N^4)$ DC-checking algorithm searches for SRN loops by propagating forward from each contingent time-point C, looking for opportunities to reduce away the lower-case edge $(A, [c\!:\!\ell], C)$. (The path used to reduce away a lower-case edge e is called an *extension sub-path* for e. The path consisting of e followed by its extension sub-path is transformed into an ordinary or upper-case edge using the rules from Table 1.) To enable a modified use of Dijkstra's algorithm [4] to guide these forward propagations from each contingent time-point C, Morris' algorithm uses Bellman-Ford to compute (and update) a potential function for the OU-graph (i.e., the graph consisting of all ordinary and upper-case edges). He proved that in any non-DC network there must be an SRN loop in which the extension sub-paths used to reduce away lower-case edges are nested to a maximum depth of K, where K is the number of contingent links in the network. Thus, his algorithm performs at most K rounds, in each round doing forward propagations from each contingent time-point. After each round of edge generation, Bellman-Ford is run again, effectively recomputing

Table 2 The (non-length-preserving) *General Unordered Reduction* rule from Morris et al. [14].

Graphical Representation	Applicability Condition
$X \xrightarrow[-\ell]{[C:-y]} A \xrightarrow{[c:\ell]} C$	$-y < -\ell$

the potential function to accommodate the newly generated edges. The overall complexity is thus $O(K((M + KN)N + N \log N)) = O(MNK + K^2N^2 + KN \log N)$ which, in dense graphs, reduces to $O(N^4)$. Hunsberger [9] subsequently introduced two improvements to the algorithm: (1) using a modified version of Dijkstra to compute a new potential function after each forward-propagation processing of a contingent time-point; and (2) using a heuristic to choose a "good" order in which to process the contingent time-points. These changes allowed newly generated edges to be inserted more quickly, and allowed the algorithm to terminate much earlier depending on the quality of the heuristic ordering. Empirically, the algorithm would often behave like an $O(N^3)$ algorithm, but in the worst case it was still $O(N^4)$.

More recently, Morris [12] presented a new approach to searching for lower-case reducing paths in an STNU graph: by propagating backward from negative edges. The intuition is that each negative edge could serve as the final edge (called a *moat edge*) in a path used to reduce away a lower-case edge. The advantages to this approach included that: (1) no potential function need be computed because the backward propagation could focus on propagating through only positive edges; and (2) the conflict between lower-case and upper-case edges from the same contingent link (which are not allowed in the *Cross Case* rule) effectively disappeared. As a result, at most N rounds of Dijkstra-like traversals are required, leading to a worst-case complexity of $O(N((M + N^2) + N \log N)) = O(N^3)$, which stands as the tightest upper bound on the complexity of the STNU-DC problem to date.

In contrast to all of the preceding algorithms, the algorithm presented in this paper, called the RUL$^-$ algorithm: (1) focuses on reducing away *upper-case* edges; (2) uses a *lower-bound* potential function to enable Dijkstra's algorithm to guide the traversal of edges in the LO-graph (i.e., the graph consisting of all lower-case and ordinary edges); (4) only generates *ordinary* (i.e., *unlabeled*) edges; (5) includes a rule that is *not* length-preserving; and (6) only generates edges that terminate at either contingent or activation time-points. The RUL$^-$ algorithm uses one run of Bellman-Ford to initialize a potential function. It then does at most $2K$ rounds of constraint propagation. After each round, it uses a Dijkstra-like traversal to update the potential function for the next round. Thus, its overall complexity is $O(MN + K((M + KN) + N \log N)) = O(MN + K^2N + KN \log N)$. Although in the worst case this reduces to $O(N^3)$, in sparse graphs it can be much lower. For example, if $M = O(N \log N)$ and $K = O(\sqrt{N})$, it reduces to $O(N^2 \log N)$.

Notably, the one reduction from the earlier work that is not represented in Table 1 is the *General Unordered Reduction*, shown in Table 2, which is *not* length-preserving. It stipulates that, for a given contingent link $\langle A, \ell, u, C\rangle$, if X must wait at least y after A while C remains unexecuted, where $y > \ell$, then in *every* situation, X must wait at least ℓ after A, since C cannot execute before then. This rule will play an important role in this paper.

Table 3 The edge-generation rules for the RUL$^-$ algorithm.

Rule	Graphical representation	Applicability Conditions
RELAX$^-$	$P \xrightarrow{v} Q \xrightarrow{w} R$; $P \dashrightarrow R$: $v+w$	$Q \in \mathcal{T}_X, w < u^R - \ell^R, R \in \mathcal{T}_C$
UPPER$^-$	$P \xrightarrow{v} C \xrightarrow{[C:-u]} A$; $P \dashrightarrow A$: $\max\{v-u, -\ell\}$	(none)
LOWER$^-$	$A \xrightarrow{[c:\ell]} C \xrightarrow{w} R$; $A \dashrightarrow R$: $\ell + w$	$C \not\equiv R, w < u^R - \ell^R, R \in \mathcal{T}_C$

2.3 The RUL$^-$ Edge-Generation Rules

The RUL$^-$ algorithm uses three rules: RELAX$^-$, UPPER$^-$ and LOWER$^-$.[3] As summarized in Table 3, each rule takes two consecutive edges (S, x, T) and (T, y, U) from the network and, if certain conditions are satisfied, generates a new *ordinary* edge from S to U. Although the rules have many similarities to rules from prior work [14, 13], they are distinguished in the following important ways.

- The RUL$^-$ rules only generate *ordinary* (i.e., *unlabeled*) edges; they *never* generate *lower-case* or *upper-case* edges.
- The RELAX$^-$ and LOWER$^-$ rules only generate edges terminating at contingent time-points.
- The value $u_R - \ell_R$, which represents the amount of uncertainty associated with the contingent link $\langle A_R, \ell_R, u_R, R\rangle$, plays an important role in the applicability conditions for the RELAX$^-$ and LOWER$^-$ rules.

The RELAX$^-$ rule is identical to the RELAX rule from the literature on shortest-path problems [4], except that it has very restrictive applicability conditions. The RELAX$^-$ rule takes two ordinary edges (P, v, Q) and (Q, w, R), and generates the ordinary edge $(P, v+w, R)$. It only applies if Q is non-contingent, R is contingent, and $w < u_R - \ell_R$.

The UPPER$^-$ rule is a combination of the *Upper Case* and *Label Removal* rules from Table 1, and the *General Unordered Reduction* rule from Table 2. It takes an ordinary edge (P, v, C), where C is contingent, and the original upper-case edge $(C, [C:-u], A)$ associated with C, and generates the ordinary edge (P, m, A), where $m = \max\{v-u, -\ell\}$. Note that the UPPER$^-$ rule is *not* length-preserving.

The LOWER$^-$ rule is similar to the *Lower Case* rule from Table 1; however, its applicability conditions are quite different. The LOWER$^-$ rule takes a lower-case edge $(A, [c:\ell], C)$ and an ordinary edge (C, w, R), and generates the ordinary edge $(A, \ell + w, R)$. Unlike the *Lower Case* rule, the LOWER$^-$ rule only applies if R is a contingent time-point. For this reason, it can be applied whenever $w < u^R - \ell^R$, whereas the *Lower Case* rule only applies if $w < 0$.

▶ **Proposition 1.** *The* RELAX$^-$ *rule from Table 3 is sound.*

Proof. The RELAX$^-$ rule is identical to the sound *No Case* rule from Table 1, except that it has more restrictive applicability conditions. Thus, it too must be sound. ◀

[3] The RUL$^-$ rules have the same form as the RUL$^+$ rules developed in prior work [1], but their applicability conditions are quite different. As a result, the RUL$^+$ rules do not form the basis of an efficient DC-checking algorithm, whereas the RUL$^-$ rules do. To avoid confusion, the RUL$^+$ rules are not discussed further in this paper.

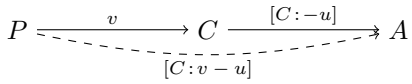


Figure 2 An upper-case edge generated during the proof of Prop. 2.

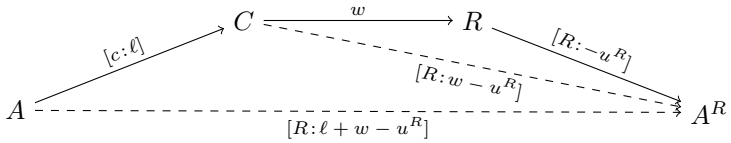


Figure 3 Upper-case edges generated during the proof of Prop. 3.

▶ **Proposition 2.** *The* Upper⁻ *rule from Table 3 is sound.*

Proof. The soundness of the Upper⁻ rule is obtained from the soundness of the *Upper Case* and *Label Removal* rules from Table 1, and the *General Unordered Reduction* rule from Table 2. First, let (P, v, C) and $(C, [C : -u], A)$ be the pre-existing edges in the Upper⁻ rule, as shown in Table 3. The *Upper Case* rule generates the *upper-case* edge $(P, [C : v - u], A)$, as illustrated in Fig. 2.

Case 1: $v - u \geq -\ell$ (i.e., $m = v - u$). In this case, the *Label Removal* rule applies, yielding the *unlabeled* edge $(P, v - u, A)$ (i.e., (P, m, A)).

Case 2: $v - u < -\ell$ (i.e., $m = -\ell$). In this case, σ satisfying the *upper-case* edge $(P, [C : v - u], A)$ implies that in each situation ω, either P waits at least $(u - v)$ after A or P executes at or after C.[4] Therefore, either $[\sigma(\omega)]_P \geq [\sigma(\omega)]_A + (u - v) > [\sigma(\omega)]_A + \ell$ or $[\sigma(\omega)]_P > [\sigma(\omega)]_C \geq [\sigma(\omega)]_A + \ell$. As a result, in each situation ω, σ necessarily satisfies $[\sigma(\omega)]_P \geq [\sigma(\omega)]_A + \ell = [\sigma(\omega)]_A - m$, represented by the edge (P, m, A). ◀

▶ **Proposition 3.** *The* Lower⁻ *rule from Table 3 is sound.*

Proof. The soundness of the Lower⁻ rule derives from the soundness of several of the rules from Table 1, as follows. Let $(A, [c : \ell], C)$ and (C, w, R) be the two pre-existing edges for the Lower⁻ rule, as shown in Table 3, where $w < u^R - \ell^R$ and R is contingent.

Case 1: $w < 0$. Here, the *Lower Case* rule applies, yielding the desired edge.

Case 2: $w \in [0, u^R - \ell^R)$. First, the *Upper Case* rule applied to the edges (C, w, R) and $(R, [R : -u^R], A^R)$ yields the *upper-case* edge $(C, [R : w - u^R], A^R)$, as illustrated in Fig. 3. And since $w - u^R < -\ell^R < 0$, the *Cross Case* rule can then be applied to yield the *upper-case* edge $(A, [R : \ell + w - u^R], A^R)$, also shown in Fig. 3. Since σ is valid, it must satisfy this generated *upper-case* edge. Therefore, in each situation ω, either A executes at least $(-\ell - w + u^R)$ after A^R; or A executes at or after R. Hence, one of the following holds:

(1) $[\sigma(\omega)]_A \geq [\sigma(\omega)]_{A^R} - \ell - w + u^R$; or

(2) $[\sigma(\omega)]_A \geq [\sigma(\omega)]_R$.

For (1), it follows that $[\sigma(\omega)]_R - [\sigma(\omega)]_A \leq [\sigma(\omega)]_R - [\sigma(\omega)]_{A^R} + \ell + w - u^R$. And, since the semantics of contingent links ensures that $[\sigma(\omega)]_R - [\sigma(\omega)]_{A^R} \leq u^R$, it follows that $[\sigma(\omega)]_R - [\sigma(\omega)]_{A^R} + \ell + w - u^R \leq u^R + \ell + w - u^R = \ell + w$. For (2), $[\sigma(\omega)]_R - [\sigma(\omega)]_A \leq 0 < \ell + w$, since $\ell > 0$ and $w \geq 0$. Thus, in either case, σ satisfies the edge $(A, \ell + w, R)$. ◀

[4] The semantics of satisfying a generated upper-case edge is detailed by Hunsberger [8].

▶ **Theorem 1** (Completeness of the RUL$^-$ rules). *Let $\mathcal{S}$ be any STNU; let $\mathcal{G}$ be the graph for $\mathcal{S}$ (prior to any application of the* RUL$^-$ *rules); and let $\mathcal{G}^*$ be the closure of $\mathcal{G}$ under the* RUL$^-$ *rules. If the LO-graph for $\mathcal{G}^*$ has no negative loops (i.e., is consistent when viewed as an STN, ignoring the alphabetic labels on its lower-case edges), then $\mathcal{S}$ must be DC.*

Proof. Let $\mathcal{S}$ be any STNU with graph $\mathcal{G}$ prior to any application of the RUL$^-$ rules. Suppose that $\mathcal{S}$ is not DC. By Morris [11], $\mathcal{G}$ must contain a semi-reducible negative loop π that is *breach-free* (i.e., if $(A, [c{:}\ell], C)$ is a lower-case edge in π, and $\mathcal{P}$ is the *extension sub-path* in π that is used to "reduce away" that lower-case edge, then $\mathcal{P}$ does not contain any occurrence of the corresponding upper-case edge $(C, [C{:}-u], A)$). By a similar argument, it can be shown that no loss of generality results from assuming that the extension sub-path of any lower-case edge $(A, [c{:}\ell], C)$ in π does not include any occurrence of the contingent time-point C.

First note that if π contains no upper-case edges, then π is in the LO-graph, contradicting that the LO-graph has no negative loops. Thus, π must have one or more upper-case edges.

In what follows, let Upper† denote the restriction of the Upper$^-$ rule to the case where $v \geq u - \ell$ (i.e., the *length-preserving* case). And let RUL† denote the set of rules {Relax$^-$, Lower$^-$, Upper† }. Note that the RUL† rules are length-preserving. The following shall focus on the use of the RUL† rules to reduce away all of the upper-case edges in (a suitably transformed) π. However, in certain exceptional cases, the non-length-preserving case of the Upper$^-$ rule will be applied.

⇒ This proof is supported by Morris' *analysis* of semi-reducible paths in which each lower-case edge is followed by an *extension sub-path* that can be used to reduce it away by applying the rules from Table 1. In contrast, the RUL$^-$ algorithm uses the rules from Table 3 to reduce away *upper-case edges.* The proof uses the properties of semi-reducible paths to confirm that applying the RUL$^-$ algorithm to the semi-reducible path π necessarily leads to a "Non-DC" conclusion.

Suppose that E is some upper-case edge $(C, [C{:}-u], A)$ in π that cannot be reduced away by the RUL† rules. Consider back-propagating from C using the Relax$^-$ and Lower$^-$ rules. If this process ever results in an edge (T, s, C), where $s \geq u - \ell$, then that edge could be used to reduce away E via the Upper† rule, since $s - u \geq -\ell$. To prevent this, one of the following must happen:

(1) The lower-case edge $(A, [c{:}\ell], C)$ for the *same* contingent link is encountered, resulting in a path of the form, $(A, [c{:}\ell], C, \ldots, C, [C{:}-u], A)$. Now, given that the original path π was breach-free, the sub-path used to reduce away the lower-case edge $(A, [c{:}\ell], C)$ must be a proper prefix of the path from C to C; and it must have negative length. Thus, the path must have the form $(A, [c{:}\ell], C, \ldots, X, w, C, [C{:}-u], A)$, where the path from C to X has some length $\delta < 0$, and where $w < u - \ell$, since back-propagation continued from C back to X and beyond. In this case, since $w - u < -\ell$, the non-length-preserving case of the Upper$^-$ rule generates the edge $(X, -\ell, A)$. But then the loop from A to A has length $\ell + \delta - \ell = \delta < 0$. Since this loop consists only of ordinary and lower-case edges, it contradicts the hypothesis that the LO-graph has no negative loops.

(2) A two-edge path (X, r, P, v, C) is encountered, where P is contingent, but (X, r, P) is an ordinary edge, and $v < u - \ell$. (This blocks the Relax$^-$ rule since P is contingent, and blocks the Lower$^-$ rule since (X, r, P) is ordinary.) In this case, insert the path $(P, [P{:}-u^P], A^p, [p{:}\ell^p], P)$. Then use the Lower$^-$ rule to generate the edge $(A^p, \ell^P + v, C)$, as shown in Fig. 4. Now, if $\ell^P + v \geq u - \ell$, then E can be reduced away. If not, then further back-propagation is blocked by the upper-case edge $(P, [P{:}-u^P], A^P)$.

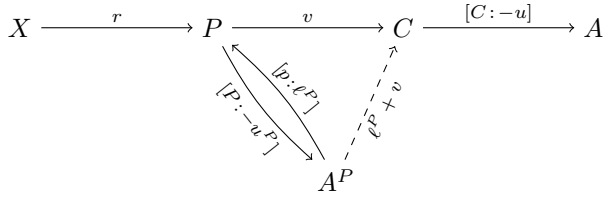


Figure 4 Inserting labeled edges into the path π in Case 2 of the proof of Theorem 1.

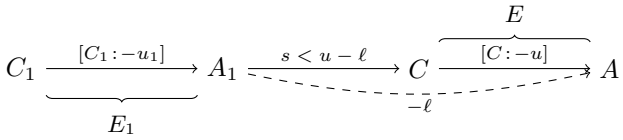


Figure 5 An upper-case edge E_1 blocking the reducing-away of another upper-case edge E.

In view of the above analysis, the only way that the back-propagation could fail to reduce away the upper-case edge E in π is if it is blocked by some upper-case edge E_1, as illustrated in Fig. 5. The upper-case edge E_1 could be one that was originally in π, or one that was added in Case (2) above, where $\ell^p + v < u - \ell$. In either case, note that the non-length-preserving case of the UPPER$^-$ rule can be used to generate a negative edge from E_1's activation time-point A_1 to E's activation time-point A, as illustrated in the figure. Thus, if the recursive processing of successive upper-case edges ever encounters a repeat, it would signal a negative loop in the LO-graph, contradicting the hypothesis.

As a result, the recursive processing of some upper-case edge E_i must *not* be blocked (i.e., E_i can be reduced away) which implies that the processing of the preceding upper-case edge E_{i-1} can resume. Continuing in this way, either E_i or one of its successor upper-case edges in π must be unblocked (i.e., can be reduced away), and so on, until all upper-case edges have been reduced away from π. Since the reducing away of upper-case edges only uses the length-preserving UPPER† rule, the transformed π (now in the LO-graph) still has negative length and, thus, contradicts the hypothesis that the LO-graph is consistent. (The non-length-preserving case of the UPPER$^-$ rule was used only to signal inconsistencies that terminated the recursion; it was not used in cases where the recursion continued.)

Finally, note that when inserting the upper-case edge $(P, [P : -u^P], A^P)$ in Case (2) above, it was assumed that the process of reducing away that upper-case edge could not affect the semi-reducibility of π. That is true, but it must be proven, as follows. First, since P was already in π, and P cannot appear in the extension sub-path $\mathcal{P}_P$ used to reduce away the lower-case edge $(A^P, [p : \ell^P], P)$, it follows that introducing the upper-case edge for P into π cannot cause a breach for the lower-case edge for P. Next, suppose that introducing the upper-case edge for P, and subsequently reducing it away, consumed the *moat* edge in the extension sub-path $\mathcal{P}_Q$ used to reduce away the lower-case edge $(A^Q, [q : \ell^Q], Q)$, as illustrated in Fig. 6. Since all proper prefixes of the extension sub-path $\mathcal{P}_Q$ must have non-negative length (otherwise they could be used to reduce away the lower-case edge), $a \geq 0$ and $a + b \geq 0$. However, the entire extension sub-path must have negative length; thus, $a + b + c < 0$, which implies that $c < -a - b \leq 0$. Now, $c < 0$ implies that $c + v < v < u^P - \ell^P$; hence the path used to reduce away the upper-case edge extends backward, beyond X, to some W. Thus, $b + c + v \geq u^P - \ell^P$ (to enable the use of the UPPER† rule). But then:

$$u^P - \ell^P \leq b + c + v < b + (-a - b) + v = -a + v \leq v < u^P - \ell^P$$

which is a contradiction. Thus, our assumption that inserting the upper-case edge involving P caused the path π to become non-semi-reducible was wrong. ◀

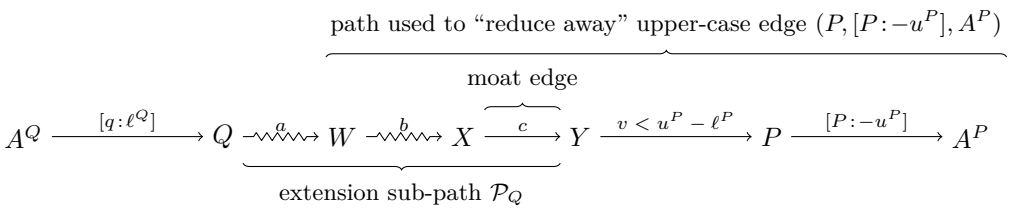
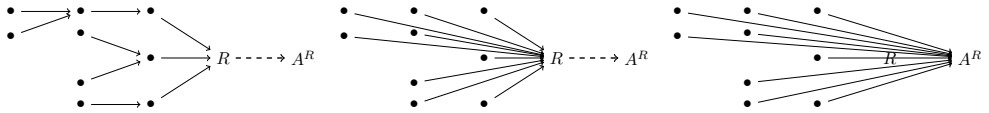


Figure 6 A path discussed in the proof of Theorem 1, where $a+b+c<0$ and $b+c+v \geq u^P - \ell^P$.



(a) Propagating back from R (b) New edges terminating at R (c) ApplyUpper

Figure 7 Reducing away an upper-case edge $(R, [R:-u^R], A^R)$ with the RUL$^-$ algorithm.

3 The RUL$^-$ DC-checking Algorithm

This section presents the main contribution of the paper: the RUL$^-$ DC-checking algorithm for STNUs. When applied to a given semi-reducible negative loop, the algorithm essentially follows the structure of the proof of Theorem 1. Of course, it is not generally given a semi-reducible negative loop in advance; therefore, it effectively carries out its back-propagation along all potential paths in parallel.

The pseudo-code for the RUL$^-$ DC-checking algorithm is given in Algorithm 1, where it is called DC-Check. The constraint-propagation functions, CloseRelaxLower, ApplyRelaxLower and ApplyUpper, that are used by DC-Check are collected in Algorithm 2. And the InitPotential, UpdatePotential and NegativeCycle functions, used to initialize, update, and verify the consistency of a potential function h for the LO-graph $\mathcal{E}^o \cup \mathcal{E}^\ell$ are collected in Algorithm 3.

The input to the DC-Check function is an STNU graph $\mathcal{G}$. The algorithm focuses on reducing away each upper-case edge $(R, [R:-u^R], A^R)$, as illustrated in Fig. 7. First, it uses the CloseRelaxLower and ApplyRelaxLower functions to propagate backward from the contingent time-point R along edges in the LO-graph $\mathcal{E}^o \cup \mathcal{E}^\ell$ (Fig. 7a) and then generate new edges terminating at R using the Relax$^-$ and Lower$^-$ rules (Fig. 7b). Then for every edge terminating at R, the ApplyUpper function uses the Upper$^-$ rule to generate new edges terminating at the activation time-point A^R (Fig. 7c), effectively providing all ways of reducing away the upper-case edge $(R, [R:-u^R], A^R)$.[5]

If the backward propagation from R is ever blocked by another upper-case edge E_1, the algorithm suspends processing of R and recursively begins to process E_1. Similarly, the processing of E_1 could subsequently be blocked by other upper-case edges, which would require suspending the processing of E_1, and so on. At each stage, the processing of an upper-case edge is only resumed when *all* of the blocking upper-case edges have been fully processed. As in the proof of Theorem 1, if the algorithm ever recursively encounters the same upper-case edge twice, it immediately terminates with a *false* (i.e., "Non-DC") answer.

[5] The algorithm could postpone using the non-length-preserving case of the Upper$^-$ rule until all other propagations were done, but it is more convenient to apply both cases of RUL$^-$ when the opportunity arises.

Algorithm 1: DC-Check, the RUL$^-$ DC-checking algorithm.

Input: $\mathcal{G}$ ▷ An STNU graph
Output: $\langle True, \mathcal{G}\rangle$, if DC; *False*, otherwise ▷ Side Effect: Modifies $\mathcal{G}$

1 $h \longleftarrow$ InitPotential $(\mathcal{G})$
2 **if** NegativeCycle $(\mathcal{G}, h)$ **then return** *False*
3 $\mathcal{R} \longleftarrow \mathcal{T}_C$
4 $S \longleftarrow$ new empty stack
5 push Z onto S
6 **while** S *is not empty* **do**
7 $R \longleftarrow$ top of S ▷ Do not pop from stack
8 $\mathcal{G} \longleftarrow$ CloseRelaxLower $(\mathcal{G}, h, R)$
9 $\mathcal{G} \longleftarrow$ ApplyUpper $(\mathcal{G}, R)$
10 $h \longleftarrow$ UpdatePotential $(\mathcal{G}, h, A^R)$
11 **if** NegativeCycle $(\mathcal{G}, h)$ **then return** *False*
12 **if** $\exists R' \in \mathcal{R}$ *such that* $w_{A^{R'}R} < u^R - \ell^R$ **then**
13 **if** $R' \in S$ **then return** *False*
14 **else** push R' onto S
15 **else**
16 $\mathcal{R} \longleftarrow \mathcal{R} \setminus \{R\}$
17 pop R from top of S
18 **if** $\mathcal{R}$ *is non-empty and* S *is empty* **then**
19 push top element of $\mathcal{R}$ onto S ▷ But keep it in $\mathcal{R}$
20 **return** $\langle True, \mathcal{G}\rangle$

But if it successfully completes the processing of all K upper-case edges without encountering a negative loop in the LO-graph, it terminates with a *true* (i.e., "DC") answer.

To efficiently perform the back-propgation processes, the algorithm uses a potential function h for the LO-graph. (The LO-graph can be viewed as an STN by ignoring any alphabetic labels on its lower-case edges.) The InitPotential function uses the Single *Sink* version of Bellman-Ford to create a potential function that specifies a lower bound for each time-point in the network. In particular, if $d(X)$ is the distance from X to some arbitrary sink node, then $h(X) = -d(X)$ specifies a lower bound for X. If the LO-graph has no negative cycles, then the potential function provides a solution to the LO-graph, as an STN. The potential function h is then used, as in Johnson's algorithm [4], to transform the weight δ of each edge (X, δ, Y) into a non-negative value, $h(X) + \delta - h(Y) = -d(X) + \delta + d(Y) \geq 0$, thereby enabling Dijkstra's algorithm to be used to guide the back-propagation from each contingent time-point R in the CloseRelaxLower function. Afterward, the ApplyUpper function is used to generate new edges terminating at the activation time-point A^R. Since these edges are derived from an upper-case edge, but the potential function is based on the LO-graph, the potential function must be updated to accommodate those new edges. The UpdatePotential function handles this chore using a modified version of Dijkstra that allows negative edges as long as they all have the same destination – in this case, they all terminate at A^R.

After updating the potential function, the main algorithm checks whether there is an upper-case edge $(C', [C' : -u'], A')$ that has not yet been processed for which there is an edge (A', s, R) for which $s < u^R - \ell^R$, which would block further back-propagation from R. (It may help to recall Fig. 5, substituting R for C, and $s < u^R - \ell^R$ for $v < u - \ell$.) If so, it recursively processes that upper-case edge. Once that process completes, it checks

Algorithm 2: Constraint-propagation functions for DC-Check algorithm.

function CloseRelaxLower ($\mathcal{G}$, *an STNU graph;* h, *a potential function;* $R \in \mathcal{T}_C$)**:**
  Pre: $h(P) \geq h(Q) - w$ for each $(P, w, Q) \in \mathcal{E}^o \cup \mathcal{E}^\ell$
  Post: All new edges (P, v, R) that can be generated by Relax$^-$ and Lower$^-$ have been added to $\mathcal{E}^o$
  $\mathcal{Q} \longleftarrow$ new priority queue
  foreach $(Q, x, R) \in \mathcal{E}^o \cup \mathcal{E}^\ell$ **do** insert Q into $\mathcal{Q}$ with priority $h(Q) + x$
  while $\mathcal{Q}$ *is not empty* **do**
    $Q \longleftarrow$ pop min priority node from $\mathcal{Q}$
    foreach $(P, v, R) \in$ ApplyRelaxLower $(\mathcal{G}, Q, R)$ **do**
      if $v < w_{PR}$ **then** insert edge (P, v, R) into $\mathcal{E}^o$
      $w_{PR} \longleftarrow \min\{w_{PR}, v\}$
      if $P \in \mathcal{Q}$ **then** decrease priority of P to $h(P) + w_{PR}$
      else insert P into $\mathcal{Q}$ with priority $h(P) + w_{PR}$
  return $\mathcal{G}$

function ApplyRelaxLower ($\mathcal{G}$, *an STNU graph;* $Q \in \mathcal{T}$*;* $R \in \mathcal{T}_C$)**:**
  Output: The set of all edges (P, w, R) obtained by applying Relax$^-$ or Lower$^-$ to the path $(P, w_{PQ}, Q, w_{QR}, R)$ in $\mathcal{E}^o \cup \mathcal{E}^\ell$
  if $w_{QR} \geq u^R - \ell^R$ **then return** $\emptyset$ ▷ Relax$^-$ and Lower$^-$ do not apply
  else if $Q \in \mathcal{T}_C$ **then return** $(A^Q, \ell^Q + w_{QR}, R)$ ▷ Apply Lower$^-$
  else return $\{(P, w_{PQ} + w_{QR}, R)\}_{(P, w_{PQ}, Q) \in \mathcal{E}^o \cup \mathcal{E}^\ell, P \in \mathcal{T} \setminus \{R\}}$ ▷ Apply Relax$^-$

function ApplyUpper ($\mathcal{G}$, *an STNU graph;* $R \in \mathcal{T}_C$)**:**
  Output: All new edges (P, w, A^R) obtained by applying Upper$^-$ to paths $(P, v, R, [R\!:\!-u^R], A^R)$ in $\mathcal{E}^o \cup \mathcal{E}^\ell$ have been added to $\mathcal{E}^o$
  foreach $(P, v, R) \in \mathcal{E}^o$ **do**
    if $v < u^R - \ell^R$ **then** $w_{PA^R} \longleftarrow \min\{w_{PA^R}, -\ell^R\}$
    else $w_{PA^R} \longleftarrow \min\{w_{PA^R}, v - u^R\}$
    insert edge (P, w_{PA^R}, A^R) into $\mathcal{E}^o$
  return $\mathcal{G}$

whether there are any other as-yet-unprocessed upper-case edges that meet that criterion. If so, it processes each one of them in turn. Once all such upper-case edges are successfully processed, the algorithm processes R again, from scratch, and then moves to process any other as-yet-unprocessed upper-case edges (even though they may not be blocking for R) until all such edges have been processed. By maintaining separate stacks, $\mathcal{R}$ and S, respectively, of contingent time-points that have not yet been processed, and those whose processing is in progress, the algorithm is guaranteed to perform at most $2K$ rounds. (Each round terminates by pushing a time-point $R' \in \mathcal{R}$ onto S, or removing a time-point R from both $\mathcal{R}$ and $\mathcal{S}$. Since no time-point is ever pushed onto S or popped off of S more than once, it follows that at most $2K$ rounds can be performed.) If updating the potential function ever fails, then the algorithm immediately returns *false* (i.e., "Non-DC"). If the algorithm ever recursively encounters the same upper-case edge twice, it immediately returns "Non-DC". If none of those things happen then, after at most $2K$ rounds, the algorithm returns *true* (i.e., "DC").

Algorithm 3: Functions to initialize/update/verify lower-bound potential function.

```
1  function InitPotential (G, an STNU graph):
       Post: h(P) ≥ h(Q) − w for each (P, w, Q) ∈ E^o ∪ E^ℓ, unless graph has neg. cycle
       ▷ N − 1 rounds of Bellman-Ford, where N = |T|
2      foreach P ∈ T do h(P) ⟵ 0
3      for i = 1 to N − 1 do
4          foreach edge (P, w, Q) ∈ E^o ∪ E^ℓ do
5              h(P) ⟵ max{h(P), h(Q) − w}        ▷ (i.e.,
                d(P) ⟵ min{d(P), d(Q) + w})
6      return h
7  function NegativeCycle (G, an STNU graph; h, a potential function for G):
       Output: True if E^o ∪ E^ℓ has a negative cycle; False otherwise
       ▷ Last round of Bellman-Ford
8      foreach edge (P, w, Q) ∈ E^o ∪ E^ℓ do
9          if h(P) < h(Q) − w then return True
10     return False
11 function UpdatePotential (G, STNU graph; h, potential function for G; A ∈ T):
       Pre: h(P) ≥ h(Q) − w for each (P, w, Q) ∈ E^o ∪ E^ℓ where Q ≠ A
       Post: h'(P) ≥ h'(Q) − w for each (P, w, Q) ∈ E^o ∪ E^ℓ unless graph has a neg.
             cycle
12     h' ⟵ h
13     Q ⟵ new priority queue
       ▷ Priority of each time-point = amount its lower-bound changes
14     insert A into Q with priority 0
15     while Q is not empty do
16         Q ⟵ extract min. priority node from Q
17         foreach (P, w, Q) ∈ E^o ∪ E^ℓ do
18             if h'(P) < h'(Q) − w (i.e., d(P) > d(Q) + w) then
19                 h'(P) ⟵ h'(Q) − w (i.e., d(P) ⟵ d(Q) + w)
20                 if P ∈ Q then decrease priority of P in Q to h(P) − h'(P)
21                 else insert P into Q with priority h(P) − h'(P)
22     return h'
```

▶ **Corollary 1.** *The* RUL$^-$ *algorithm is sound and complete for the STNU-DC problem.*

Proof. The RUL$^-$ rules are sound by Propositions 1–3. And if an STNU graph $\mathcal{G}$ has a semi-reducible negative loop, then by the proof of Theorem 1, the RUL$^-$ algorithm will return "Non-DC". ◀

4 Conclusion

This paper introduced a new algorithm, called the RUL$^-$ algorithm, for solving the DC-checking problem for STNUs. The algorithm, which is proven to be sound and complete, uses three constraint-propagation rules, Relax$^-$, Lower$^-$ and Upper$^-$, that differ from other approaches in that they generate only ordinary edges, and they focus on generating edges

that terminate either in contingent or activation time-points. As a result, the algorithm performs at most $2K$ rounds. Since each round generates at most N new edges, the algorithm generates at most $2KN$ new edges overall. The constraint propagation in each round is guided by Dijkstra's algorithm, using a lower-bound potential function (as in Johnson's algorithm). Each round of edge generation is interleaved with another run of Dijkstra to update the lower-bound potential function to accommodate the new edges. Thus, the complexity of each round is $O((M + KN) + N \log N)$; and the overall complexity of the RUL$^-$ algorithm is $O(MN + K^2N + KN \log N)$, where the $O(MN)$ term arises from the use of Bellman-Ford to initialize the potential function. For sparse networks, this upper bound on the complexity of the STNU-DC problem is tighter than the previous best-known bound of $O(N^3)$. Future work will focus on experimentally evaluating the RUL$^-$ algorithm and Morris' $O(N^3)$ algorithm to see whether further enhancements might be found.

References

1. Massimo Cairo and Romeo Rizzi. Dynamic Controllability Made Simple. In Sven Schewe, Thomas Schneider, and Jef Wijsen, editors, *24th International Symposium on Temporal Representation and Reasoning (TIME 2017)*, volume 90 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TIME.2017.8.
2. Carlo Combi, Luke Hunsberger, and Roberto Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. In *ICAART 2013*, volume 2, pages 144–156, 2013. doi:10.5220/0004256101440156.
3. Patrick R. Conrad and Brian C. Williams. Drake: An efficient executive for temporal plans with choice. *Journal of Artificial Intelligence Research (JAIR)*, 42:607–659, 2011. doi:10.1613/jair.3478.
4. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.
5. Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. doi:10.1016/0004-3702(91)90006-6.
6. Robert Effinger, Brian Williams, Gerard Kelly, and Michael Sheehy. Dynamic controllability of temporally-flexible reactive programs. In *19th International Conference on Automated Planning and Scheduling (ICAPS-2009)*, 2009.
7. Luke Hunsberger. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *TIME 2009*, pages 155–162, 2009. doi:10.1109/TIME.2009.25.
8. Luke Hunsberger. Efficient execution of dynamically controllable simple temporal networks with uncertainty. *Acta Informatica*, 53(2):89–147, 2015. doi:10.1007/s00236-015-0227-0.
9. Luke Hunsberger. New techniques for checking dynamic controllability of simple temporal networks with uncertainty. In *Lecture Notes in Computer Science*, volume 8946, pages 170–193. Springer, 2015.
10. Lina Khatib, Paul H. Morris, Robert A. Morris, and Francesca Rossi. Temporal constraint reasoning with preferences. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages 322–327, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers, Inc.
11. Paul Morris. A structural characterization of temporal dynamic controllability. In *CP 2006*, volume 4204, pages 375–389, 2006. doi:10.1007/11889205_28.

12 Paul Morris. Dynamic controllability and dispatchability relationships. In *Integration of AI and OR Techniques in Constraint Programming*, volume 8451 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2014.

13 Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *AAAI-05/IAAI-05*, pages 1193–1198, 2005.

14 Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *IJCAI 2001*, pages 494–502, 2001.

15 Francesca Rossi, Kristen Brent Venable, and Neil Yorke-Smith. Uncertainty in soft temporal constraint problems: A general framework and controllability algorithms for the fuzzy case. *Journal of Artificial Intelligence Research*, 27:617–674, 2006.

Extending Conditional Simple Temporal Networks with Partially Shrinkable Uncertainty

Carlo Combi
Department of Computer Science, University of Verona, Verona, Italy
carlo.combi@univr.it
https://orcid.org/0000-0002-4837-4701

Roberto Posenato
Department of Computer Science, University of Verona, Verona, Italy
roberto.posenato@univr.it
https://orcid.org/0000-0003-0944-0419

Abstract

The proper handling of temporal constraints is crucial in many domains. As a particular challenge, temporal constraints must be also handled when different specific situations happen (conditional constraints) and when some event occurrences can be only observed at run time (contingent constraints). In this paper we introduce *Conditional Simple Temporal Networks with Partially Shrinkable Uncertainty (CSTNPSUs)*, in which contingent constraints are made more flexible (guarded constraints) and they are also specified as conditional constraints. It turns out that guarded constraints require the ability to reason on both kinds of constraints in a seamless way. In particular, we discuss CSTNPSU features through a motivating example and, then, we introduce the concept of controllability for such networks and the related sound checking algorithm.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning, Computing methodologies → Planning under uncertainty, Computing methodologies → Planning for deterministic actions

Keywords and phrases Conditional Simple Temporal Networks with Uncertainty, Partial Shrinkable Temporal Constraint, Dynamic Controllability, Temporal Constraints

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.9

1 Introduction

Temporal constraint networks have been adopted in different research areas for reasoning on temporal requirements. Among such areas, we mention here planning and scheduling [1, 19, 24], business process [7, 14, 18] and healthcare informatics [4][5].

In such areas the proposed temporal constraint models allow the representation of both conditional and contingent constraints. In general, *conditional constraints* hold only when some specific situations happen. Thus, it is possible to specify in a compact way constraints holding in different contexts. Sometimes such contexts are related to decisions made by the executing agent, while in other cases, they are related to external events [8][23]. *Contingent constraints* allow the representation of event occurrences that are not under the control of the executing agent but are tied to happen within some specified time interval. Therefore, it is necessary to guarantee that all the other constraints are consistent for all possible event occurrences [21].

Recently, a new temporal constraint model, called *Simple Temporal Network with Partially Shrinkable Uncertainty (STNPSU)*, was proposed for the analysis of modular temporal business processes [17]. It introduces a new type of constraint, called *guarded* constraint, and allows the representation of temporal features of business sub-processes as guarded constraints. A guarded constraint is a contingent constraint that is partially under control of the executing agent (i.e., it may be shrunk to a *core*).

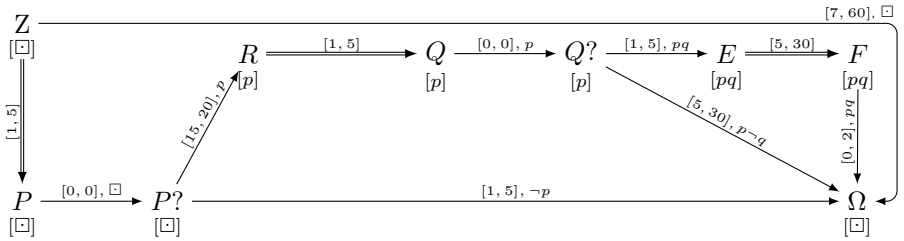
STNPSU lacks of an important feature, especially in the business process domain: the possibility of expressing conditional (possibly guarded) constraints. To the best of our knowledge, the issue of representing conditional constraints and guarded constraints in the same model has not been considered in the literature to date.

Contributions. The novelty of this paper, therefore, is that it focuses on the representation of and reasoning on both conditional constraints and guarded ones. In particular, we introduce *Conditional Simple Temporal Networks with Partially Shrinkable Uncertainty (CSTNPSUs)*, in which all constraints (guarded included) may be specified with respect to particular contexts. At first glance, conditional constraints and guarded ones seem to be orthogonal features that may be managed in an independent way. When taking a closer view on them, however, it turns out that conditional constraints in combination with the guarded ones require the ability to reason on both constraints in a seamless way, because, for example, reasoning techniques for guarded constraints are no longer applicable. Then, we focus on the *dynamic controllability* (DC) of the proposed model introducing a new DC checking algorithm. In general, DC corresponds to the capability of an executing agent to dynamically execute a network for *all* allowed occurrences of *all* guarded events, while still satisfying *all* temporal constraints in *all* possible situations (which are revealed at run time in incremental way); i.e., DC ensures that it is possible to assign a timestamp to each node of a network without any need to modify any guarded *core* to be able to satisfy anyone of the other temporal constraints in any occurring situation.

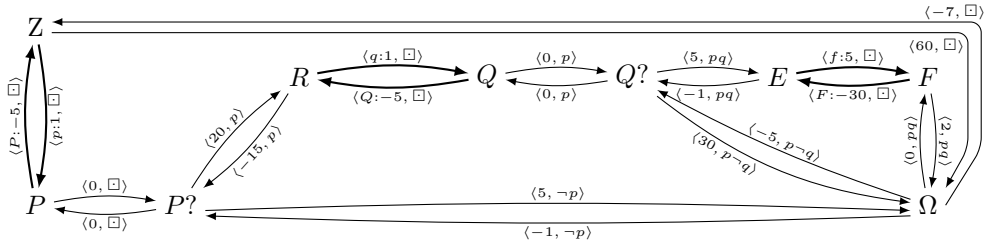
Structure of the paper. The rest of the paper is organized as follows. Section 2 provides the fundamentals of Conditional Simple Temporal Problem with Uncertainty (CSTNU) and checking related dynamic controllability. Moreover, we introduce here the motivating example we use throughout the paper. Section 3 provides the definition of the new model Conditional Simple Temporal Networks with Partially Shrinkable Uncertainty. Then, it presents the concept of dynamic controllability and how it can be checked reducing the problem to the CSTNU controllability check. Finally, we propose an improved controllability check algorithm by extending a recently proposed sound-and-complete algorithm for CSTNU to deal with the introduced guarded constraints. Section 5 gives some concluding remarks and outlines some future work.

2 Background and Related Work

Given a set $\mathcal{P}$ of propositional letters, a *label* ℓ is any conjunction of literals, where a literal is either a propositional letter $p \in \mathcal{P}$ or its negation $\neg p$. The *empty label* is denoted by $\boxdot$. The *label universe of* $\mathcal{P}$, denoted by $\mathcal{P}^*$, is the set of all labels whose literals are drawn from $\mathcal{P}$. Two labels $\ell_1, \ell_2 \in \mathcal{P}^*$ are *consistent* if and only if their conjunction $\ell_1 \wedge \ell_2$ is satisfiable and a consistent label ℓ_1 *entails* a consistent label ℓ_2 (written $\ell_1 \Rightarrow \ell_2$) if and only if all literals in ℓ_2 appear in ℓ_1 too.



(a) A CSTNU instance.



(b) Distance graph of CSTNU in Figure 1a.

Time-point	Meaning
Z	Begin First Blood Test
P	End First Blood Test
$P?$	Check Blood Test (generates truth value for p; $p = \top$ indicates positive reaction)
R	Begin Second Blood Test
Q	End Second Blood Test
$Q?$	Check again Blood Test (only applicable if $p = \top$; generates truth value for q; $q = \top$ indicates positive reaction)
E	Do Emergency Procedure (only applicable if $p = q = \top$)
F	End Emergency Procedure (only applicable if $p = q = \top$)
Ω	End

Constraints are shown as single arrows, whereas contingent links as double ones. Each single-arrow edge between X and Y is decorated by a range $[l, u]$ and label ℓ corresponding to constraint $(l \leq Y - X \leq u, \ell)$. Each double-arrow edge between A and C is decorated by a range $[x, y]$ corresponding to constraint (A, x, y, C). Each time-point has a label $\ell \in \mathcal{P}^*$ representing the scenario in which it is present.

Figure 1 Example of CSTNU.

In this section, we recall the definition of Conditional Simple Temporal Network with Uncertainty [12] and some basic properties presented in [13] that must hold also in such networks.

▶ **Definition 1** (Conditional Simple Temporal Network with Uncertainty)**.** A *Conditional Simple Temporal Network with Uncertainty* (*CSTNU*) is a tuple $(\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, O, \mathcal{C}, \mathcal{L})$, where:

- $\mathcal{T} = \{X, Y, \dots\}$ is a finite set of *time-points* (i.e., variables with continuous domain).
- $\mathcal{OT} \subseteq \mathcal{T}$ is a set of *observation time-points*.
- $\mathcal{P} = \{p, q, \dots\}$ is a finite set of propositional letters.
- $L \colon \mathcal{T} \to \mathcal{P}^*$ is a function assigning a label to each time-point $X \in \mathcal{T}$.
- $O \colon \mathcal{P} \to \mathcal{OT}$ is a bijection associating a unique observation time-point to each propositional letter.
- $\mathcal{C}$ is a set of *(labeled) constraints* each one having the form $(l \leq Y - X \leq u, \ell)$, where $X, Y \in \mathcal{T}$, $l, u \in \mathbb{R}$ with $l \leq u$ and $\ell \in \mathcal{P}^*$.
- $\mathcal{L}$ is a set of *contingent links* each having the form (A, x, y, C), where $A \in \mathcal{T}$ and $C \in \mathcal{T} \setminus \mathcal{OT}$ are different time-points (written $A \not\equiv C$), $0 < x < y < \infty$, $L(A) = L(C)$. For any pair (A_1, x_1, y_1, C_1), $(A_2, x_2, y_2, C_2) \in \mathcal{L}$ with $A_1 \not\equiv A_2$ it holds $C_1 \not\equiv C_2$.

- For each constraint $(l \leq Y - X \leq u, \ell) \in \mathcal{C}$, $\ell \Rightarrow L(Y) \wedge L(X)$ (*constraint label coherence*).
- For each literal p or $\neg p$ appearing in ℓ, $\ell \Rightarrow L(O(p))$ (*constraint label honesty*).
- For each $X \in \mathcal{T}$, if literal p or $\neg p$ appears in $L(X)$, then $L(X) \Rightarrow L(O(p))$, and $O(p)$ has to occur before X, i.e., $(\epsilon \leq X - O(p) \leq +\infty, L(X)) \in \mathcal{C}$ for some $\epsilon > 0$ (*time-point label honesty*).

For any contingent link (A, x, y, C), A is called *activation time-point* whereas C is called *contingent time-point*. Once A is executed, C is only observed to occur. However, C is guaranteed to execute such that $C - A \in [x, y]$. Moreover, a contingent link has an implicit label given by the label $\ell = L(A) = L(C)$.

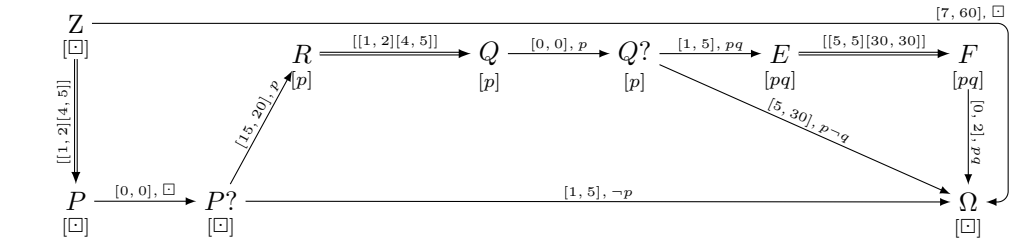
Figure 1a gives an example of CSTNU representing a temporal plan in a healthcare setting with minutes as a temporal granularity. The CSTNU is shown in its graphical form where nodes represent time-points, single-line directed edges represent constraints, and double-line directed edges represent contingent links. It starts with the execution of Z, which is *usually assumed to be the first time-point to be executed*, and ends with the execution of Ω, which is *usually assumed to be the last time-point to be executed*. For example, time-point $P?$ represents the time at which a particular blood test is checked. This test generates a truth value for the propositional letter p, where $p = \top$ indicates that the patient tested positive. In this example, if the test generates a positive result, then the test is repeated at time-point R and ends at time-point Q; then, at time-point $Q?$ the truth value for the propositional letter q is set. Since the time-points R, Q, and $Q?$ apply only in scenarios where $p = \top$, they are labeled by p. Similarly, the edge from $P?$ to R is labeled by p. It represents the constraint, $R - P? \in [15, 20]$ (i.e., the second test must be performed between 15 and 20 minutes after the first one). Finally, note that the constraints from $Q?$ to E, and from E to Ω are labeled by pq, indicating that they apply only in scenarios where both p and q are $\top$. The three contingent links represent the execution of the first blood test, the second one, and the emergency procedure, respectively. Constraints and time-points labeled by $\boxdot$ are always taken into consideration. Constraint $(7 \leq \Omega - \mathrm{Z} \leq 60, \boxdot)$ represents a constraint for the overall execution time of the network. It is worth noting that the couple P and $P?$ represents two time-points executed at the same time but in the given order. The first one, P, refers to the end of blood test, while the second one refers to observing the blood test result (the same for couple Q and $Q?$) They must be different, according to CSTNU model.

In general, by *executing* a non-contingent time-point we mean to assign a real value to it, while a contingent time-point is *executed* when the "environment" sets a real value to it.

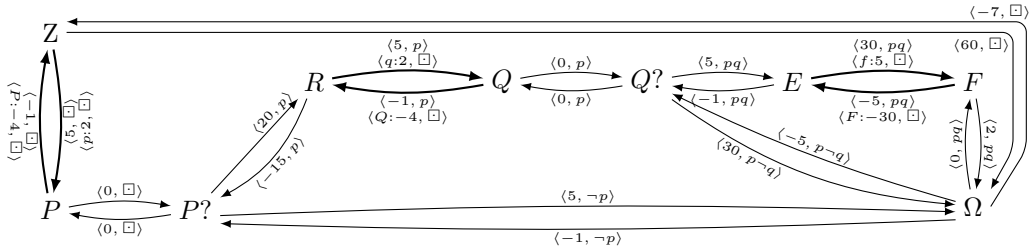
The truth values of propositions and the duration of contingent links in a CSTNU are not known in advance. Instead, they are incrementally revealed over time as the corresponding observation/contingent time-points are executed, respectively. A *dynamic execution strategy* for executing the time-points in a CSTNU is defined in a way that it can react to observations and contingent time-points as time passes (after an $\epsilon > 0$ reaction time). A *viable* and *dynamic execution strategy* for a CSTNU is a strategy that guarantees that all relevant constraints will be satisfied no matter which truth values for propositions and durations for contingent links are incrementally revealed over time. A CSTNU with such a strategy is called *dynamically controllable (DC).* The problem of checking the dynamic controllability (*DC-checking problem*) consists of verifying, at design time, whether a CSTNU is DC.

In [2], authors showed that the problem of checking the dynamic consistency of conditional simple temporal networks (CSTNs) is PSPACE-complete. It is straightforward to show that a CSTN is a special case of a CSTNU: it is a network where there is no contingent links. Therefore, the CSTNU DC-checking problem is PSPACE-hard.

Nevertheless, in literature there are some proposals to solve the DC-checking problem using different algorithm techniques [2, 3, 6] and only one of them, presented in [6], results



(a) A CSTNPSU extension of the CSTNU in Figure 1a.



(b) Distance graph of CSTNPSU in Figure 2a.

Figure 2 A CSTNPSU and its distance graph.

to be applicable to real world problems as shown in [5, 15, 19]. It determines all possible constraints from the initial ones using a suitable set of constraint propagation rules, while verifying their satisfiability. Recently, a new *sound-and-complete* algorithm for CSTNUs DC checking has been proposed in [10], where the authors extend and merge the algorithms proposed in [5] and [13] using a new technique and show its practical applicability.

3 Conditional Simple Temporal Networks with Partially Shrinkable Uncertainty

In some real-world contexts, contingent links seem to be not flexible enough to represent task durations. Indeed, it may be that in some cases task duration constraints can be slightly modified during the execution of a complex plan. For example, the duration range $[1, 5]$ of blood tests in the CSTNU in Figure 1a could be made more flexible by saying that it can be reduced to $[1, 4]$ if needed. In other contexts it could be that also the lower bound of a contingent link can be modified, e.g., range $[2, 4]$ for blood test could be less expensive because it requires different devices.

In [16] and [17] the authors addressed this issue in the context of STNUs (and related business process modeling). A STNU is a simplified version of a CSTNU without conditions [21]. In this section we extend such approach by introducing a new proposal of temporal constraint networks, named *Conditional Simple Temporal Networks with Partially Shrinkable Uncertainty (CSTNPSU)*.

Conditional Simple Temporal Networks with Partially Shrinkable Uncertainty is an extension of CSTNU where contingent links are generalized to *guarded links*. Each guarded link has the form, $(A, [x, x'], [y', y], C)$, where A is the activation time-point, and C is the contingent one, and $0 < x < y < \infty$, $x \leq x'$, $x' < y'$, $y' \leq y$. Once A is executed, C is guaranteed to execute such that $C - A \in [x, y]$. However, bounds x and y can be modified before executing A with the limitations specified by the two *guards* x' and y', i.e., x can be incremented up to x', while y can be reduced down to y'. The range $[x', y'] \subseteq [x, y]$

of a guarded link represents its unshrinkable range, named guarded *core*. Moreover, if $x = x' \wedge y = y'$ hold, the guarded link is equivalent to a contingent link of CSTNU.

▶ **Definition 2** (CSTNPSU). A *Conditional Simple Temporal Networks with Partially Shrinkable Uncertainty* (CSTNPSU) is a tuple $(\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, O, \mathcal{C}, \mathcal{G})$, where $\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, O, \mathcal{C}$ are the same as in CSTNU definition and have the properties specified in Theorem 1. $\mathcal{G}$ is a set of *guarded links* each having the form $(A, [x, x'], [y', y], C)$, where A and C are time-points, $x, y \in \mathbb{R}$, $0 < x \leq x' < y' \leq y < \infty$, $L(A) = L(C)$.

Figure 2a shows an extension of the example introduced in Section 2. Previous contingent links have been replaced by guarded links. In particular, both contingent links $(R, 1, 5, Q)$ and $(\mathrm{Z}, 1, 5, P)$ were replaced by the guarded links $(R, [1, 2], [4, 5], Q)$ and $(\mathrm{Z}, [1, 2], [4, 5], P)$, respectively, representing the fact that blood tests do not need to have $[1, 5]$ as contingent range but the less demanding $[2, 4]$ if it is required during the execution. Contingent link $(E, 5, 30, F)$ was replaced by the guarded link $(E, [5, 5], [30, 30], F)$ meaning that, in this case, the guarded link is identical to the original contingent one: such constraint cannot be partially shrunk.

3.1 Dynamic Controllability of CSTNPSU

Informally, a CSTNPSU is dynamically controllable if there exists a strategy for executing the time-points in the network such that all constraints are guaranteed to be satisfied no matter how the durations of the guarded links turn out, and no matter how the observations of the various propositions turn out, in real time – paying attention to the fact that in any given scenario, only the time-points whose labels are true in that scenario need to be executed, and only the constraints whose labels are meaningful in that scenario need to be satisfied.

In order to verify whether a CSTNPSU is dynamically controllable it is possible to consider the CSTNU corresponding to the CSTNPSU where all guarded links have been restricted to their core.

▶ **Definition 3** (Core Situations). Let $\mathcal{S} = (\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, O, \mathcal{C}, \mathcal{G})$ be a CSTNPSU. If $\mathcal{G}$ contains k guarded links, $(A_1, [x_1, x_1'], [y_1', y_1], C_1), \ldots, (A_k, [x_k, x_k'], [y_k', y_k], C_k)$, then $\Omega_{\mathcal{S}} = [x_1', y_1'] \times \ldots \times [x_k', y_k']$ is called the *space of core situations* for $\mathcal{S}$. Any $\omega = (d_1, \ldots d_k) \in \Omega_{\mathcal{S}}$ is called a *core situation*.

Given the space of core situations $\Omega_{\mathcal{S}}$ of a CSTNPSU $\mathcal{S}$, a projection of $\mathcal{S}$ onto a CSTNU can be obtained as follows: each guarded link in $\mathcal{G}$ is replaced by a contingent link with the range specified in $\Omega_{\mathcal{S}}$.

▶ **Definition 4** (Core CSTNU of a CSTNPSU). Let $\mathcal{S} = (\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, O, \mathcal{C}, \mathcal{G})$ be a CSTNPSU. The *core CSTNU* of $\mathcal{S}$ onto its space of core situations $\Omega_{\mathcal{S}}$ corresponds to a CSTNU $(\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, O, \mathcal{C}, \mathcal{L})$ where:

$$\mathcal{L} = \{(A_i, x_i', y_i', C_i) \mid 1 \leq i \leq k, [x_i', y_i'] \text{ is the } i\text{-th component of } \Omega_{\mathcal{S}}\}$$

Finally, this leads us to the dynamic controllability of an CSTNPSU. We provide a formalization of the dynamic controllability of an CSTNPSU based on the dynamic controllability of a CSTNU. We choose this approach since the formalization of dynamic controllability of CSTNU is robust and verified in literature [6][10][11][12].

▶ **Theorem 5** (Dynamic Controllability of CSTNPSU). *A CSTNPSU $\mathcal{S} = (\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, O, \mathcal{C}, \mathcal{G})$ is* dynamically controllable *(DC) if its core CSTNU is dynamically controllable.*

Proof. $\Rightarrow$ It is a matter of definitions to show that, if the core CSTNU is DC, then $\mathcal{S}$ is DC as well. Indeed, it is always possible to restrict $\mathcal{S}$ to its core situations, and for each core situation of $\mathcal{S}$, a viable dynamic execution strategy (DES) for the core CSTNU, is also a viable DES for $\mathcal{S}$.
$\Leftarrow$ If the core CSTNU is not DC, at least one core situation ω exists for which no DES exists. Hence, $\mathcal{S}$ is not DC either. ◀

To check the controllability of a CSTNPSU, we proceed in two steps:

1. we consider the sound-and-complete CSTNU DC checking algorithm proposed in [10];
2. we extend such algorithm for directly checking the controllability of CSTNPSUs obtaining also minimized constraint networks.

3.2 An improved CSTNU DC checking algorithm

Before introducing the algorithm, it is necessary to consider the CSTNU representation by *distance graphs*. In [10], the authors showed that, for the DC checking task, it is possible to consider a *streamlined* representation of distance graphs where nodes are not labeled, i.e., labels are present only on constraints[1]. A *distance graph* $\mathcal{D} = (\mathcal{T}, \mathcal{E})$ of a CSTNU $(\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, O, \mathcal{C}, \mathcal{L})$ is a graph having the same set of nodes and edges derived from the upper and lower bound constraints.

Each constraint $(l \leq Y - X \leq u, \ell) \in \mathcal{C}$ between a pair of time-points X and Y is represented in the distance graph as two *ordinary edges* in $\mathcal{E}$: $X \xrightarrow{\langle u, \ell \rangle} Y$, representing constraint $Y \leq X + u$, and $X \xleftarrow{\langle -l, \ell \rangle} Y$, for constraint $Y \geq X + l$.

Moreover, for each contingent link (A, x, y, C) between a pair of time-points A and C, $\mathcal{E}$ contains two other edges with special values, called *lower* and *upper case values* [20]. An edge with a lower case value, $A \xrightarrow{\langle c{:}x, \boxdot \rangle} C$, represents the fact that C cannot be forced to be executed at a time greater than x after A, i.e., it is not possible to add a constraint $A \xleftarrow{\langle -x', \boxdot \rangle} C, x < x'$ to the network. In turn, an edge with an upper case value, $A \xleftarrow{\langle C{:}-y, \boxdot \rangle} C$, represents the fact that C cannot be forced to be executed at a time less than y after A, i.e., it is not possible to add a constraint $A \xrightarrow{\langle y', \boxdot \rangle} C, y' < y$ to the network.

These two kinds of edges containing special values are fundamental for determining the dynamic controllability of the network as explained in the following.

Figure 1b depicts the distance graph of CSTNU in Figure 1a. If multiple edges exist between two nodes (e.g., an ordinary and an upper-case edge), for the sake of readability, we draw only one arrow between the nodes and annotate it with the values of the respective edges.

The DC checking algorithm, CSTNU-DC-Check, consists of deriving new constraints (edges) through the application of propagation rules. Such technique narrows the search space of possible partial solutions by creating equivalent yet more explicit networks. In such equivalent networks, the search for a solution is more efficient. On the other hand, solving a complete problem by inference often requires the addition of an exponential number of new constraints [9].

Propagation rules for CSTNUs fall into two main groups. The first group extends the edge-generation rules proposed in [20] for STNU DC checking algorithm to accommodate labeled edges; the second group consists of label-modification rules that address interactions involving observation nodes.

[1] The notation of constraint values we use in this paper is slightly different from the one adopted in [10].

Table 1 Edge-generation rules of CSTNU-DC-Check algorithm.

Rule	Conditions	Pre-existing and Generated Edges
rG_1	$u+v<0$, $\alpha\beta\in\mathcal{P}^*$	$\mathrm{Z} \xleftarrow{\langle \aleph{:}v,\,\beta\rangle} Y \xleftarrow{\langle u,\,\alpha\rangle} X$; $\mathrm{Z} \xleftarrow{\langle \aleph{:}u+v,\,\alpha\beta\rangle} X$
rG_2	$x+v<0$, $C\notin\aleph$, $\beta\in\mathcal{P}^*$	$\mathrm{Z} \xleftarrow{\langle \aleph{:}v,\,\beta\rangle} C \xleftarrow{\langle c{:}x,\,\boxdot\rangle} A$; $\mathrm{Z} \xleftarrow{\langle \aleph{:}x+v,\,\beta\rangle} A$
rG_3	$-y+v<0$, $\beta\in\mathcal{P}^*$	$\mathrm{Z} \xleftarrow{\langle \aleph{:}v,\,\beta\rangle} A \xleftarrow{\langle C{:}-y,\,\boxdot\rangle} C$; $\mathrm{Z} \xleftarrow{\langle C\aleph{:}-y+v,\,\beta\rangle} C$
rG_4	$m=\max\{v,w-x\}$, $C\notin\aleph\aleph_1$, $\beta,\gamma\in\mathcal{Q}^*$	$Y \xrightarrow[\langle \aleph\aleph_1{:}m,\,\beta\star\gamma\rangle]{\langle C\aleph{:}v,\,\beta\rangle} \mathrm{Z} \xleftarrow{\langle \aleph_1{:}w,\,\gamma\rangle} A \xrightarrow{\langle c{:}x,\,\boxdot\rangle} C$
rM_1	$w<0$, $\beta\in\mathcal{Q}^*$, $\tilde{p}\in\{p,\neg p,?p\}$	$\mathrm{Z} \xleftarrow[\langle \aleph{:}w,\,\beta\rangle]{\langle \aleph{:}w,\,\beta\tilde{p}\rangle} P?$
rM_2	$w<0$, $\beta,\gamma\in\mathcal{Q}^*$, $\tilde{p}\in\{p,\neg p,?p\}$	$Y \xrightarrow[\langle \aleph\aleph_1{:}\max\{v,\,w\},\,\beta\star\gamma\rangle]{\langle \aleph{:}v,\,\beta\tilde{p}\rangle} \mathrm{Z} \xleftarrow{\langle \aleph_1{:}w,\,\gamma\rangle} P?$

$\mathrm{Z}=0$; $A,C,X,Y\in\mathcal{T}$; C is contingent; $P?\in\mathcal{OT}$; each of $\aleph$ and $\aleph_1$ is a conjunction of one or more upper-case names of contingent nodes, possibly empty.

The first group consists of four rules, rG_1–rG_4, depicted in Table 1. All rules only generate ordinary or conjuncted upper-case edges (introduced below).

Each rule generates an edge whose label, e.g., $\alpha\beta$, is the conjunction of the labels of its parent edges, e.g., α and β. If the resulting label is unsatisfiable (e.g., $p\neg p$), then the new edge is not generated (or kept). By $\aleph$ we represent a, possibly empty, conjunction of one or more upper-case names of contingent nodes.

When the generated edge contains a label having a non-empty $\aleph$, then it represents an extension of *wait* constraints [21]. An edge labeled by a non-empty conjuncted upper-case value, i.e., $X \xleftarrow{\langle \aleph:w,\,\boxdot\rangle} Y$, represents the following constraint[2]: *as long any of contingent time-points in* $\aleph$ *remains unexecuted,* Y *must wait at least* w *units after the execution of* X. If any contingent time-point in $\aleph$ is executed before w units after the execution of X, then the constraint is not significant and, therefore, ignored.

Effectively, rule rG_4 does not generate a new conjuncted upper-case value, but reduces one already present removing a contingent name from it. Moreover, rG_4 (as well as rM_1 and rM_2) can handle also a new kind of labels, named *q-labels*, that indicate that a constraint need only hold as long as the value of its special literals, named *q-literals*, are unknown [10].

More formally, q-literals and q-label can be introduced as the following definition.

▶ **Definition 6** (Q-literals, q-labels)**.** If $p\in\mathcal{P}$, then $?p$ is a *q-literal*. A *q-label* is a (possibly empty) conjunction of literals and/or q-literals. $\mathcal{Q}^*$ denotes the set of all q-labels.
(For example, $p(?q)\neg r$ *and* $(?q)(?r)t\neg u$ *are both q-labels.)*

The $\star$ operator extends ordinary label conjunction to q-labels. Intuitively, if constraint C_1 is labeled by p, and constraint C_2 is labeled by $\neg p$, then *both* C_1 and C_2 must hold as long as p is unknown, which is represented by $p\star\neg p\ =?p$.

▶ **Definition 7** ($\star$)**.** The operator, $\star$: $\mathcal{Q}^*\times\mathcal{Q}^*\to\mathcal{Q}^\star$, is defined as follows. First, for any $p\in\mathcal{P}$, $p\star p=p$ and $\neg p\star\neg p=\neg p$; otherwise, for any $p_1,p_2\in\{p,\neg p,?p\}$, $p_1\star p_2=?p$. Next, for any $\ell_1,\ell_2\in\mathcal{Q}^*$, $\ell_1\star\ell_2\in\mathcal{Q}^*$ denotes the conjunction obtained by applying $\star$ in pairwise fashion to matching literals from ℓ_1 and ℓ_2, and conjoining any unmatched literals.
(For example, $(p\neg q(?r)t)\star(qr\neg s)=p(?q)(?r)\neg st$*.)*

[2] Morris et al. [21] named it also as *conditional*. Hereinafter, we call it *wait* to avoid confusion with respect to the introduced conditional constraints.
Moreover, if in $X \xleftarrow{\langle \aleph:w,\,\boxdot\rangle} Y$ $\aleph$ is empty, then the wait constraint degenerates to an ordinary one $X \xleftarrow{\langle w,\,\boxdot\rangle} Y$.

Table 2 Updated Upper-Case Removal Rule of CSTNPSU-DC-Check algorithm.

Rule	Conditions	Pre-existing and Generated Edges
rG_4^*	$m = \max\{v, w - x\}$, x is the lower bound of the guarded link $(A, [x, x'], [y', y], C)$, $C \notin \aleph\aleph_1$, $\beta, \gamma \in \mathcal{Q}^*$, and γ entails γ'.	$Y \xrightarrow[\langle \aleph\aleph_1 : m,\ \beta \star \gamma \rangle]{\langle C\aleph : v,\ \beta \rangle} \mathrm{Z} \xleftarrow{\langle \aleph_1 : w,\ \gamma \rangle} A \underset{\langle -x,\ \gamma' \rangle;\ \langle C : -y', \boxdot \rangle}{\overset{\langle y,\ \gamma' \rangle;\ \langle c : x', \boxdot \rangle}{\rightleftarrows}} C$

$\mathrm{Z} = 0$; $A, C, Y \in \mathcal{T}$; C is contingent; each of $\aleph$ and $\aleph_1$ is a conjunction of one or more upper-case names of contingent nodes, possibly empty.

The second group of propagation rules, composed by rM_1 and rM_2, consists of a variety of label-modification rules that share some resemblance to rule rG_4. The label-modification rules rM_1 and rM_2 have the same general flavor, except that they deal with the uncertainty associated with observation nodes, rather than contingent links.

For example, consider the edge, $\mathrm{Z} \xleftarrow{\langle w, \alpha p \rangle} P?$, where neither p nor $\neg p$ appears in α, and $w < 0$. This edge represents the requirement that "*in scenarios where αp is true, since $\mathrm{Z} = 0$, $P? \geq -w$ must hold.*" But that, in turn, implies that the truth value of p cannot be known at the time 0, i.e., when Z is executed. And, of course, the truth value of p cannot be known when the decision to execute $P?$ is made either. As a result, decisions about when to execute $P?$ cannot depend on the truth value of p. Thus, the label on the edge from $P?$ to Z should be modified to remove the occurrence of p, yielding the new edge, $\mathrm{Z} \xleftarrow{\langle w, \alpha \rangle} P?$, which represents the constraint that in scenarios where α holds, $\mathrm{Z} - P? \leq w$ must hold. This is the idea behind the label-modification rule, rM_1, shown in Table 1. While rM_1 simplifies labels on edges outgoing from an observation time-point to Z, rule rM_2 simplifies labels on edges going to Z with respect to the labels present on edges from observation time-points to Z.

3.3 DC-Checking for CSTNPSU considering Guarded Links

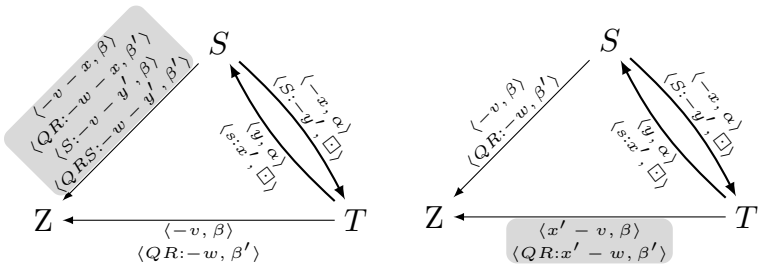
The dynamic controllability of an CSTNPSU may be checked without the need of restricting it to its core CSTNU. First, we extend the concept of distance graph. Then, we extend propagation rules in order to adapt the DC-checking algorithm for CSTNU to CSTNPSU.

▶ **Definition 8** (Distance Graph of a CSTNPSU)**.** The distance graph for a CSTNPSU $\mathcal{S}$ has the form $(\mathcal{T}, \mathcal{E})$, where each (unlabeled) time-point in $\mathcal{T}$ corresponds to a node in $\mathcal{S}$ and $\mathcal{E}$ is a set of ordinary and special value edges, defined as follow:

- Each constraint $(l \leq Y - X \leq u, \ell)$ of $\mathcal{S}$ is represented by two ordinary edges $X \xrightarrow{\langle u, \ell \rangle} Y$ and $X \xleftarrow{\langle -l, \ell \rangle} Y$.
- Each guarded link $(A, [x, x'], [y', y], C)$ having $\ell = L(A) = L(C)$ of $\mathcal{S}$ is represented by
  - two ordinary edges $A \xrightarrow{\langle y, \ell \rangle} C$ and $A \xleftarrow{\langle -x, \ell \rangle} C$, considering lower and upper bounds x, y, respectively,
  - one lower-case edge $A \xrightarrow{\langle c : x', \boxdot \rangle} C$, considering guard x', and
  - one upper-case edge $A \xleftarrow{\langle C : -y', \boxdot \rangle} C$, considering guard y'.

Figure 2b shows the distance graph of CSTPSU of Figure 2a.

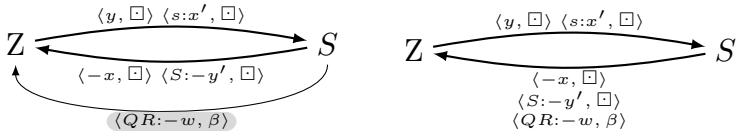
In the following we will show that rules of Table 1 may be reused in order to check dynamic controllability of a CSTNPSU, provided that rule rG_4 has to be refined as rG_4^* (cf. Table 2) where the value to use for comparison is the lower bound of a guarded link instead of the lower case value used in the original rule.



(a) A wait constraint from an activation time-point of a guarded link.

(b) A wait constraint from a contingent time-point of a guarded link.

Figure 3 Combining a Guarded Link with a Wait Constraint.



(a) Original values of Guarded Link and Wait Constraint.

(b) if $x < w \leq x'$; otherwise, it is not controllable.

Figure 4 Combining a Guarded Link with a derived Wait Constraint.

Particularly, we analyze all possible combinations of edges between three nodes of a CSTNPSU graph and show that the resulting distance graph has no negative loops if and only if the distance graph of the core CSTNU has no negative loops as well. Without loss of generality, we focus only on the more interesting cases when Z may be also an activation time point of a guarded link.

Figure 3a shows how a wait constraint on T (i.e., from T to Z having a conjuncted upper-case value) may be combined with a guarded link from T to S. Labels β and β' entail α and v and w are non negative (otherwise, rules don't apply). The shadow values denotes the ones generated by applying rG_1 and rG_3 rules for the new edge between S and Z. It can be shown that the generated edge is similar to the one that would be generated for the core CSTNU and, most important, the resulting temporal constraint is equivalent. Indeed, the generated edge for the core CSTNU would have only two labeled values that differ from the ones shown in Figure 3a: $\langle -v - x', \beta \rangle$, and $\langle QR{:}-w - x', \beta' \rangle$ instead of $\langle -v - x, \beta \rangle$, and $\langle QR{:}-w - x, \beta' \rangle$, respectively. Such two different values are not relevant for checking the DC property because the two labeled values $\langle S{:}-v - y', \beta \rangle$ and $\langle QRS{:}-w - y', \beta' \rangle$, present also in the core CSTNU, are stronger and, therefore, make the former redundant.

Figure 3b depicts a similar case where a wait constraint is present on a contingent time-point (i.e., from S to Z). In this case, applying rG_2, one wait constraint and one ordinary constraint must be added between T and Z. Note that also rG_1 can be applied, but the resulting labeled values have the same form but with weaker values than the ones generated by rG_2 rule and, therefore, they can be ignored. It is easy to verify that two new labeled values are identical to the ones that would be generated for the core CSTNU.

One of the most interesting cases is depicted in Figure 4a, where Z is the activation time-point of a guarded link. Due to possible previous constraint propagations, a wait constraint $S \xleftarrow{\langle QR:-w, \beta \rangle} \text{Z}$ has been added next to the guarded link. For the case $x < w \leq x'$ the result is depicted in Figure 4b. Note that for a CSTNU, such case is not possible. Indeed, in a CSTNU $x' = x$ and, therefore, there are two cases: either $w > x' = x$ (i.e., the CSTNU is not controllable) or $w \leq x$ (i.e., the wait constraint is redundant) holds. For a

Algorithm 1: CSTNPSU-DC-Check(G).

Input: $G = (\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, O, \mathcal{C}, \mathcal{G})$: a CSTNPSU instance
Output: the dynamic controllability status of G.
G' = distance graph of G
$h = M|\mathcal{T}|$, where M is the maximum absolute value of any negative edge
foreach $X \in \mathcal{T}$ **do**
  Add to G' the lower bound $X \xrightarrow{\langle 0, \boxdot \rangle}$ Z
  Add to G' the global upper bound Z $\xrightarrow{\langle h, \boxdot \rangle} X$
do
  $G' = rM_1(G')$ // Label Modification
  $G' = rM_2(G')$
  $G' = rG_1(G')$ // Edge Generation
  $G' = rG_2(G')$
  $G' = rG_3(G')$
  $G' = rG_4^*(G')$
  if *(any negative cycle has been found)* **then return** *not DC*
while *(rules are applied)*
return *DC*

CSTNPSU, Figure 4b can be interpreted as follows: wait $\langle QR{:}{-}w, \beta \rangle$ is not relevant for the controllability check because $\langle S{:}{-}y', \alpha \rangle$ is more restrictive. On the other hand, when executing the CSTNPSU the wait constraint must be observed in the β scenario. Particularly, if the guarded link is activated prior to the execution of time-point R and time-point Q, w must be used as lower bound when executing the guarded link. Otherwise, x is used as lower bound.

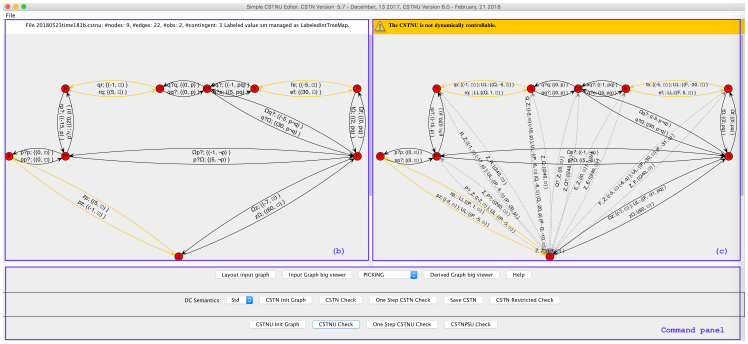
Using the above technique, we can prove that the propagation rules can be applied to all possible combinations of distance graph edges (constraints) to derive new constraints retaining the property that there are (no) negative loops if and only if the distance graph of the core CSTNU has (no) negative loops as well.

Algorithm 1 depicts the pseudo-code for dynamic controllability check, based on the propagation rules presented in Table 1–2. As shown in [10], the termination of the DC checking algorithm is guaranteed by adding a global upper upper bound and the algorithm is sound-and-complete. As regards the computational complexity of Algorithm 1, it has been shown that an upper bound is $O(M|\mathcal{T}|^4 3^{|\mathcal{P}|} 2^{|\mathcal{L}|})$, where M is the maximum absolute value of any weight in the graph [10].

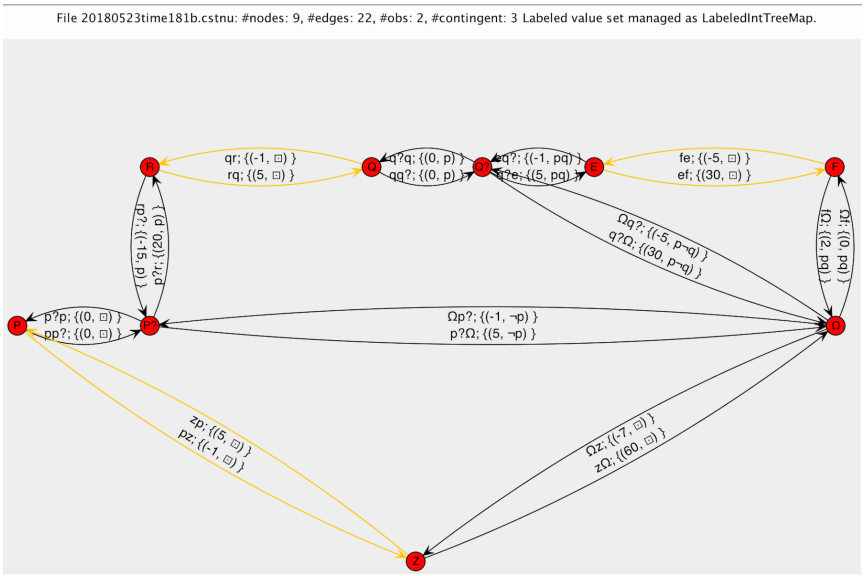
4 Proof Of Concept

The presented model and its dynamic controllability check algorithm were implemented as a proof-of-concept prototype in the CSTNU Toolset [22]. This prototype enables users to create, edit, and load/save CSTNPSU instances using a graphical editor. A loaded CSTNPSU can then be checked for dynamic controllability. Moreover, the minimal temporal constraints between any time-point and time-point Z are shown after a successful check.

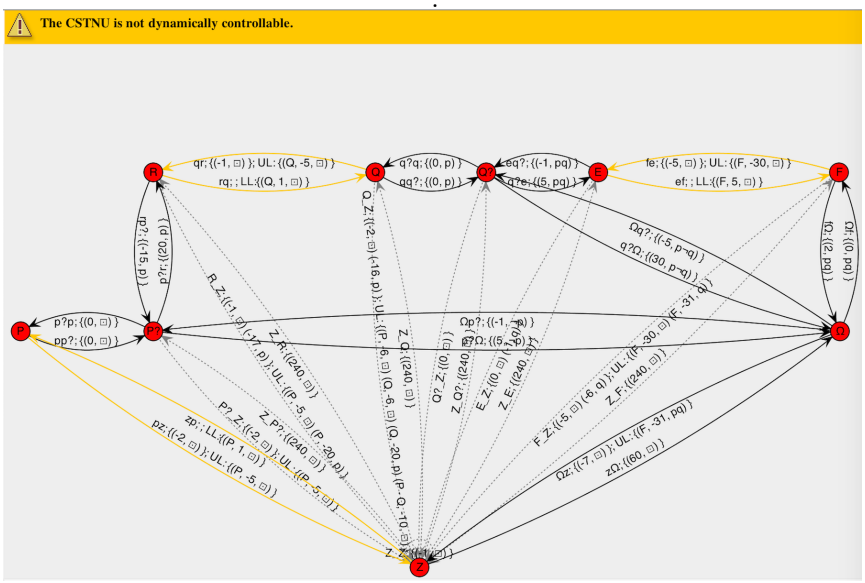
The screenshot in Figure 5 shows the CSTNU Toolset after the checking of the CSTNU depicted in Figure 1b: there are three panels, the *editor*, on the right, the *checker*, on the left, and the *command* below. Figure 5 highlights the three panels. The editor panel shows the loaded CSTNU instance while the checker panel shows the same instance after the dynamic



(a) Toolset screen after the check of CSTNU depicted in Figure 1b.

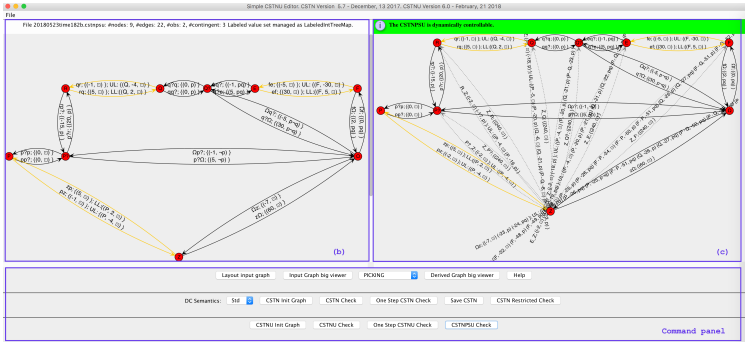


(b) Editor Detail.

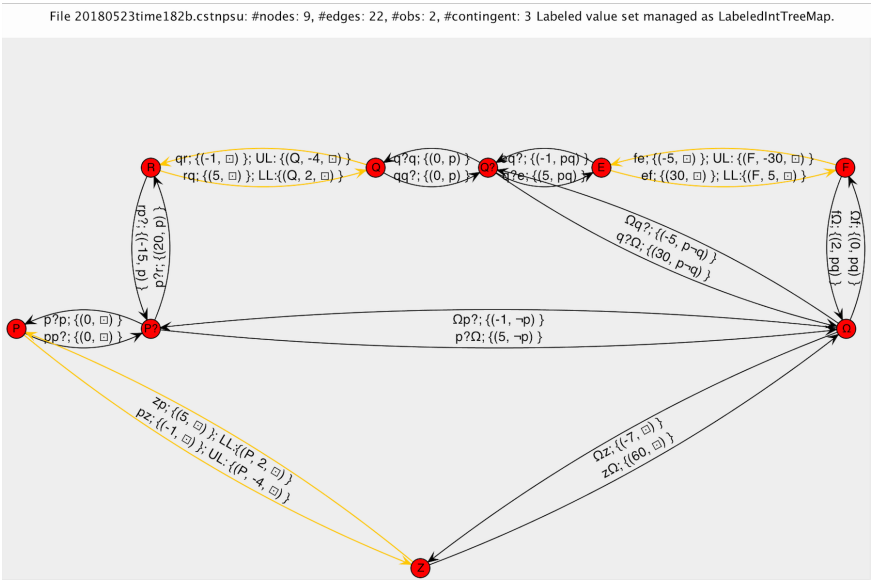


(c) Checker Detail.

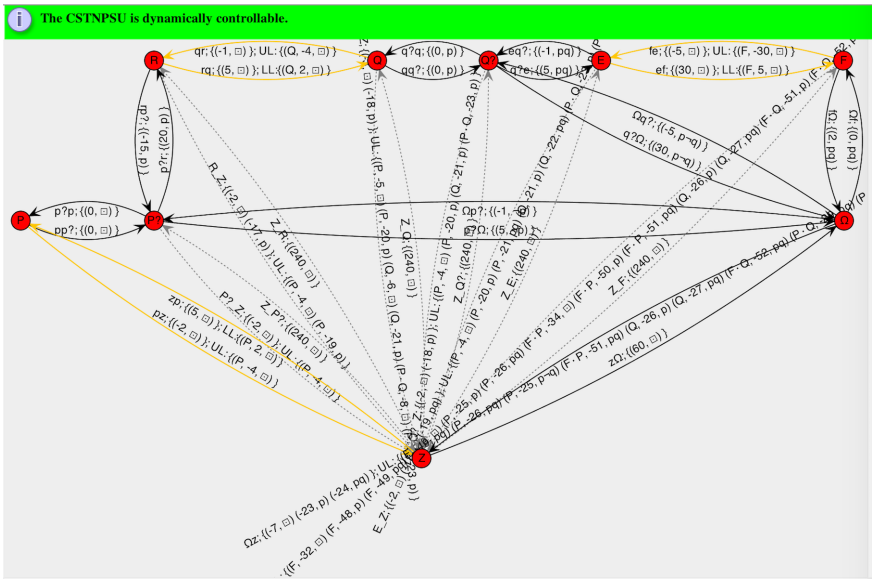
Figure 5 Determining the Dynamic Controllability of CSTNU depicted in Figure 1b.



(a) Toolset screen after the check of CSTNPSU depicted in Figure 2b.



(b) Editor Detail.



(c) Checker Detail.

Figure 6 Determining the Dynamic Controllability of CSTNPSU depicted in Figure 2b.

controllability check button has been clicked. For CSTNU depicted in Figure 1b, the check determines that it is not dynamically controllable. Therefore, the new constraints shown in the checker panel are the minimal number of constraints that determine a negative cycle in the instance.

The screenshot in Figure 6 shows the CSTNU Toolset after the checking of the CSTNPSU depicted in Figure 2b. In this case, the check determines that the instance is dynamically controllable and the new constrains shown in the checker panel represent the minimal distance between time-point Z and all the other time-points.

5 Conclusions

In this paper we presented a new temporal constraint model, CSTNPSU, that allows one to represent guarded links and conditions completing previous proposals dealing with either guarded links or conditions, separately. In particular, we discussed CSTNPSU features through a motivating example and, then, we introduced the concept of controllability for such networks and a related checking algorithm. Besides checking DC controllability of CSTNPSUs, such algorithm performs the minimization of constraints, in case the network is controllable. We showed that the algorithm is sound-and-complete. As for future work, we will focus on the proposal of an efficient run-time execution algorithm that will extends in an appropriate way the run-time execution algorithm for STNUPS presented in [16].

References

1 Arthur Bit-Monnot, Malik Ghallab, and Félix Ingrand. Which contingent events to observe for the dynamic controllability of a plan. In *Proceedings of the Twenty-Fifth Intl Joint Conf on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3038–3044, 2016. URL: `http://www.ijcai.org/Abstract/16/431`.

2 Massimo Cairo and Romeo Rizzi. Dynamic controllability of conditional simple temporal networks is PSPACE-complete. In *23rd Intl Symposium on Temporal Representation and Reasoning, TIME 2016*, pages 90–99, 2016. `doi:10.1109/TIME.2016.17`.

3 Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Dynamic controllability via Timed Game Automata. *Acta Informatica*, 53(6–8):681–722, October 2016. `doi:10.1007/s00236-016-0257-2`.

4 Carlo Combi, Mauro Gambini, Sara Migliorini, and Roberto Posenato. Modelling temporal, data-centric medical processes. In *ACM Intl Health Informatics Symposium, IHI '12, Miami, FL, USA, January 28-30, 2012*, pages 141–150, 2012. `doi:10.1145/2110363.2110382`.

5 Carlo Combi, Mauro Gambini, Sara Migliorini, and Roberto Posenato. Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways. *IEEE T. Systems, Man, and Cybernetics: Systems*, 44(9):1182–1203, September 2014. `doi:10.1109/TSMC.2014.2300055`.

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 *Agents and Artificial Intelligence - 5th Intl Conf, ICAART 2013, Revised Selected Papers*, volume 449 of *Communications in Computer and Information Science*, pages 314–331. Springer, 2014. `doi:10.1007/978-3-662-44440-5_19`.

7 Carlo Combi and Roberto Posenato. Towards temporal controllabilities for workflow schemata. In Nicolas Markey and Jef Wijsen, editors, *17th Intern. Symp. on Temporal Representation and Reasoning (TIME 2010)*, pages 129–136. IEEE Computer Society, September 2010. `doi:10.1109/TIME.2010.17`.

8 Patrick R. Conrad and Brian C. Williams. Drake: An efficient executive for temporal plans with choice. *Journal of Artificial Intelligence Research (JAIR)*, 42:607–659, 2011. doi:10.1613/jair.3478.

9 Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., 2003.

10 Luke Hunsberger and Roberto Posenato. Dynamic Controllability Checking for Conditional Simple Temporal Networks with Uncertainty: New Sound-and-Complete Algorithms based on Constraint Propagation. In Natasha Alechina, Kjetil Nørvåg, , and Wojciech Penczek, editors, *25th International Symposium on Temporal Representation and Reasoning (TIME 2018)*, volume 120 of *LIPIcs*, pages 14:1–14:17, October 2018. doi:10.4230/LIPIcs.TIME.2018.14.

11 Luke Hunsberger and Roberto Posenato. Simpler and faster algorithm for checking the dynamic consistency of conditional simple temporal networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1324–1330. International Joint Conferences on Artificial Intelligence Organization, July 2018. doi:10.24963/ijcai.2018/184.

12 Luke Hunsberger, Roberto Posenato, and Carlo Combi. The Dynamic Controllability of Conditional STNs with Uncertainty. In *Workshop on Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx) @ ICAPS-2012*, pages 1–8, Atibaia, June 2012.

13 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: 22nd Intl Symposium on Temporal Representation and Reasoning (TIME 2015)*, pages 4–18. IEEE, sep 2015. doi:10.1109/TIME.2015.26.

14 Akhil Kumar and Russell R. Barton. Controlled violation of temporal process constraints - models, algorithms and results. *Inf. Syst.*, 64:410–424, 2017. doi:10.1016/j.is.2016.06.003.

15 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 Confs - Confederated Intl Confs: CoopIS, DOA-Trusted Cloud, and ODBASE*, volume 8185 of *LNCS*, pages 39–56. Springer, September 2013. doi:10.1007/978-3-642-41030-7_4.

16 Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Simple temporal networks with partially shrinkable uncertainty. In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART 2015)*, volume 2, pages 370–381. SciTePress, 2015. doi:10.5220/0005200903700381.

17 Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Controlling Time-Awareness in Modularized Processes. In *Enterprise, Business-Process and Information Systems Modeling, 17th Intl Conf, BPMDS 2016, 21st Intl Conf, EMMSAD 2016*, volume 248 of *Lecture Notes in Business Information Processing*, pages 157–172. Springer, 2016. doi:10.1007/978-3-319-39429-9_11.

18 Andreas Lanz, Manfred Reichert, and Barbara Weber. Process time patterns: A formal foundation. *Inf. Syst.*, 57:38–68, 2016. doi:10.1016/j.is.2015.10.002.

19 Dian Liu, Hongwei Wang, Chao Qi, Peng Zhao, and Jian Wang. Hierarchical task network-based emergency task planning with incomplete information, concurrency and uncertain duration. *Knowledge-Based Systems*, 112:67–79, 2016. doi:10.1016/j.knosys.2016.08.029.

20 Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *National Conf on Artificial Intelligence (AAAI'05)*, pages 1193–1198, 2005.

21 Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *Intl Joint Conf on Artificial Intelligence (IJCAI'01)*, pages 494–502. Morgan Kaufmann, 2001.

22 Roberto Posenato. The CSTNU toolset. version 1.23. http://profs.scienze.univr.it/∼posenato/software/cstnu, 2018.

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

24 Peng Yu, Brian Charles Williams, Cheng Fang, Jing Cui, and Patrik Haslum. Resolving over-constrained temporal problems with uncertainty through conflict-directed relaxation. *J. Artif. Intell. Res.*, 60:425–490, 2017. doi:10.1613/jair.5431.

On Restricted Disjunctive Temporal Problems: Faster Algorithms and Tractability Frontier

Carlo Comin
Department of Computer Science, University of Verona, Italy
carlo.comin.86@gmail.com
https://orcid.org/0000-0001-5748-2029

Romeo Rizzi
Department of Computer Science, University of Verona, Italy
romeo.rizzi@univr.it
https://orcid.org/0000-0002-2387-0952

Abstract

In 2005 T.K.S. Kumar studied the Restricted Disjunctive Temporal Problem (RDTP), a restricted but very expressive class of Disjunctive Temporal Problems (DTPs). An RDTP comes with a finite set of temporal variables, and a finite set of temporal constraints each of which can be either one of the following three types: ($\mathtt{t}_1$) two-variable linear-difference simple constraint; ($\mathtt{t}_2$) single-variable disjunction of many interval constraints; ($\mathtt{t}_3$) two-variable disjunction of two interval constraints only. Kumar showed that RDTPs are solvable in deterministic strongly polynomial time by reducing them to the Connected Row-Convex (CRC) constraints satisfaction problem, also devising a faster randomized algorithm. Instead, the most general form of DTPs allows for multi-variable disjunctions of many interval constraints and it is NP-complete.

This work offers a deeper comprehension on the tractability of RDTPs, leading to an elementary deterministic strongly polynomial time algorithm for them, significantly improving the asymptotic running times of all the previous deterministic and randomized solutions. The result is obtained by reducing RDTPs to the Single-Source Shortest Paths (SSSP) and the 2-SAT problem (jointly), instead of reducing to CRCs. In passing, we obtain a faster (quadratic time) algorithm for RDTPs having only $\{\mathtt{t}_1, \mathtt{t}_2\}$-constraints and no $\mathtt{t}_3$-constraint. As a second main contribution, we study the tractability frontier of solving RDTPs blended with Hyper Temporal Networks (HyTNs), a disjunctive strict generalization of Simple Temporal Networks (STNs) based on hypergraphs: we prove that solving temporal problems having only $\mathtt{t}_2$-constraints and either only multi-tail or only multi-head hyperarc-constraints lies in NP $\cap$ co-NP and admits deterministic pseudo-polynomial time algorithms; on the other hand, problems having only $\mathtt{t}_3$-constraints and either only multi-tail or only multi-head hyperarc-constraints turns out strongly NP-complete.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Computing methodologies → Temporal reasoning, Computing methodologies → Planning and scheduling

Keywords and phrases Restricted Disjuctive Temporal Problems, Simple Temporal Networks, Hyper Temporal Networks, Consistency Checking, Single-Source Shortest-Paths, 2-SAT

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.10

1 Introduction

Expressive and efficient temporal reasoning is essential to a number of areas in Artificial Intelligence (AI) [8, 14, 15]. Over the past few years, many constraint-based formalisms have been developed to represent and reason about time in automated planning and temporal scheduling [4, 12]. We begin by recalling the Disjunctive Temporal Problem (DTP) [13, 16, 17]. The general form of a DTP being, given a finite set $\mathcal{T} = \{X_0, X_1, \ldots, X_N\}$ of temporal

variables (i.e., time-points), to schedule them on the real line in such a way as to satisfy a prescribed finite set $\mathcal{C}$ of temporal constraints over $\mathcal{T}$. Every constraint $c_i \in \mathcal{C}$ is a disjunction of the form $s_{(i,1)} \vee s_{(i,2)} \vee \cdots \vee s_{(i,T_i)}$, where every $s_{i,j}$ is a simple temporal constraint of the form $(l_{i,j} \leq X_{\beta_{i,j}} - X_{\alpha_{i,j}} \leq u_{i,j})$ for some integers $0 \leq \alpha_{i,j}, \beta_{i,j} \leq N$ and reals $l_{i,j}, u_{i,j}$.

Although DTPs are expressive enough to capture many tasks in automated planning and temporal scheduling, they are NP-complete [16]. The principal direct approach taken to solve DTPs has been to convert the original problem into one of selecting a disjunct from each constraint [16, 17], then to check whether the set of selected disjuncts forms a consistent Simple Temporal Problem (STP) [5]. This can be done in strongly polynomial time by computing single-source shortest paths (e.g., with the Bellman-Ford's algorithm [2]). Under this prospect, of course the prohibitive complexity of solving DTPs comes from the fact that there are exponentially many disjunct combinations possible.

In [9, 10], T.K.S. Kumar studied the Restricted Disjunctive Temporal Problem (RDTP), a tractable subclass of DTPs strictly including the classical and well established STPs [5]. In RDTPs, each constraint can be either one of the following three types: $(\mathtt{t}_1)$ $(Y - X \leq w)$, for w real (a simple temporal difference-constraint); $(\mathtt{t}_2)$ $(l_1 \leq X \leq u_1) \vee \cdots \vee (l_k \leq X \leq u_k)$, for l_i, u_i reals (a single-variable disjunction of many interval-constraints); $(\mathtt{t}_3)$ $(l_1 \leq X \leq u_1) \vee (l_2 \leq Y \leq u_2)$, for l_i, u_i reals (a two-variable disjunction of two interval-constraints).

It was shown in [10] that RDTPs are solvable in deterministic strongly polynomial time by reducing them to the Connected Row-Convex (CRC) [6] constraint satisfaction problem, faster randomized algorithms were also proposed. CRC constraints generalize many other known tractable classes of constraints like 2-SAT, implicational, and binary integer-weighted linear constraints [6]. Particularly, Kumar's deterministic algorithm for solving RDTPs works by reducing them into binary Constraint Satisfiability Problems (CSPs) over meta-variables representing $\mathtt{t}_2$ or $\mathtt{t}_3$ constraints, meanwhile showing that such binary constraints are indeed CRC constraints, finally exploiting the algorithmic tractability of CRC constraints.

An instantiation of a consistency checking algorithm (e.g., [6]) that further exploits the structure of CRC constraints leads to a time complexity of $O\big((|\mathcal{C}_{\mathtt{t}_2}| + |\mathcal{C}_{\mathtt{t}_3}|)^3 \cdot d_{\max}^2 + |\mathcal{T}| \cdot |\mathcal{C}_{\mathtt{t}_1}| \cdot (|\mathcal{C}_{\mathtt{t}_2}| + |\mathcal{C}_{\mathtt{t}_3}|)^2\big)$, where $\mathcal{C}_{\mathtt{t}_1,\mathtt{t}_2,\mathtt{t}_3}$ is the set of $\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3$ constraints (respectively), and $d_{\max}$ is the maximum number of disjuncts possible per single constraint [10]. Randomization reduces the running time to $O\big((|\mathcal{C}_{\mathtt{t}_2}| + |\mathcal{C}_{\mathtt{t}_3}|)^2 \cdot d_{\max}^2 \cdot \delta + |\mathcal{T}| \cdot |\mathcal{C}_{\mathtt{t}_1}| \cdot (|\mathcal{C}_{\mathtt{t}_2}| + |\mathcal{C}_{\mathtt{t}_3}|)^2\big)$, where δ is the degree of the CRC network (i.e., the maximum number of constraints any variable participates into) [10].

Notable applications of RDTPs include solving STPs with Taboo Regions, cfr. [11].

Contributions. This work offers a deeper comprehension on the tractability of RDTPs, leading to elementary deterministic strongly polynomial time algorithms, significantly improving the asymptotic running times of both the Kumar's deterministic and randomized solutions. Our time complexity is $O\big(|\mathcal{T}| \cdot |\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{C}_{\mathtt{t}_2}| \cdot (|\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{T}| \cdot \log|\mathcal{T}|) + |\mathcal{T}| \cdot d_{\mathcal{C}_{\mathtt{t}_2}} \cdot |\mathcal{C}_{\mathtt{t}_3}| + |\mathcal{C}_{\mathtt{t}_3}|^2\big)$, where $d_{\mathcal{C}_{\mathtt{t}_2}}$ is the total number of disjuncts counting over all $\mathtt{t}_2$-constraints. Since $d_{\mathcal{C}_{\mathtt{t}_2}} \leq d_{\max} \cdot |\mathcal{C}_{\mathtt{t}_2}|$, this improves over all of the previous solutions. The result is obtained by reducing RDTPs to the Single-Source Shortest Paths (SSSP) and the 2-SAT problem (jointly), instead of reducing to CRCs. So the full expressive power of CRCs is not needed, binary linear and 2-SAT constraints are enough. In passing, we obtain a faster (quadratic time) deterministic algorithm for solving temporal problems having only $\{\mathtt{t}_1, \mathtt{t}_2\}$-constraints and no $\mathtt{t}_3$-constraint.

As a second main contribution, we study the tractability frontier of RDTPs widened with another kind of restricted disjunctive constraints, i.e., Hyper Temporal Networks (HyTNs) [3], a strict generalization of STNs grounded on directed hypergraphs and introduced to overcome

Table 1 Summary of main results.

Problem	*Complexity*	*Improved Time Bound*	*Cfr.*
$\mathtt{t}_2$DTPs	P	$O\big(\lvert\mathcal{T}\rvert \cdot \lvert\mathcal{C}_{\mathtt{t}_1}\rvert + \lvert\mathcal{C}_{\mathtt{t}_2}\rvert \cdot (\lvert\mathcal{C}_{\mathtt{t}_1}\rvert + \lvert\mathcal{T}\rvert \cdot \log\lvert\mathcal{T}\rvert) + \lvert\mathcal{T}\rvert \cdot d_{\mathcal{C}_{\mathtt{t}_2}}\big)$	Sect. 3
RDTPs	P	$O\big(\lvert\mathcal{T}\rvert \cdot \lvert\mathcal{C}_{\mathtt{t}_1}\rvert + \lvert\mathcal{C}_{\mathtt{t}_2}\rvert \cdot (\lvert\mathcal{C}_{\mathtt{t}_1}\rvert + \lvert\mathcal{T}\rvert \cdot \log\lvert\mathcal{T}\rvert) + + \lvert\mathcal{T}\rvert \cdot d_{\mathcal{C}_{\mathtt{t}_2}} \cdot \lvert\mathcal{C}_{\mathtt{t}_3}\rvert + \lvert\mathcal{C}_{\mathtt{t}_3}\rvert^2\big)$	Sect. 4
$\mathtt{t}_2$HyTPs	NP $\cap$ co-NP	$O\big((\lvert\mathcal{T}\rvert + \lvert\mathcal{A}\rvert) \cdot m_{\mathcal{A}} \cdot W_{\mathcal{A},\mathcal{C}_{\mathtt{t}_2}}\big)$	Sect. 6
$\mathtt{t}_3$HyTPs	NP-complete	*n.a. (exponential time)*	Sect. 5

the limitation of considering only conjunctions of constraints but maintaining a practical efficiency in the consistency check of the instances. In a HyTN a single temporal multi-tail (or multi-head) hyperarc-constraint is defined as a set of two or more maximum delay (minimum anticipation, respectively) constraints which is satisfied when at least one of these delay constraints is so. We prove that solving temporal problems having only $\mathtt{t}_2$-constraints and either only multi-tail or only multi-head hyperarc-constraints lies in NP $\cap$ co-NP and admits deterministic pseudo-polynomial time algorithms; on the other hand, solving temporal problems having only $\mathtt{t}_3$-constraints and either only multi-tail or only multi-head hyperarc-constraints turns out strongly NP-complete. See Table 1 for a summary.

2 Background

This section offers the basic background notions that are assumed in the rest of the paper, let's start with Simple Temporal Networks (STNs) and related problems (STPs), cfr. [4, 5].

▶ **Definition 1** (STNs, STPs [4, 5])**.** A *Simple Temporal Network* (STN) is a pair $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a set of real-valued variables called *time-points,* and $\mathcal{C}$ is a set of linear real-weighted binary constraints over $\mathcal{T}$ called *simple (or* $\mathtt{t}_1$*)* temporal constraints, each having the form:

$$(Y - X \leq w_{X,Y}), \text{ where } X, Y \in \mathcal{T} \text{ and } w_{X,Y} \in \mathbb{R}.$$

An STN is *consistent* if it admits a *feasible schedule*, i.e., some $s : \mathcal{T} \mapsto \mathbb{R}$ such that $s(Y) \leq s(X) + w_{X,Y}$ for all $(Y - X \leq w_{X,Y}) \in \mathcal{C}$. So the *Simple Temporal Problem* (STP) is that of determining whether a given STN is consistent or not.

Any STN $\mathcal{N} = (\mathcal{T}, \mathcal{C})$ can be seen as a directed weighted graph with vertices $\mathcal{T}$ and arc set $A_{\mathcal{C}} \triangleq \{(X, Y, w_{X,Y}) \mid (Y - X \leq w_{X,Y}) \in \mathcal{C}\}$. So, a *path* p in $\mathcal{N}$ is any finite sequence of vertices $p = (v_0, v_1, \ldots, v_k)$ (some $k \geq 1$) such that $(v_i, v_{i+1}) \in A_{\mathcal{C}}$ for every $i \in [0, k) \cap \mathbb{Z}$; the total weight of p is then $w_p \triangleq \sum_{i=0}^{k-1} w_{v_i,v_{i+1}}$. A *cycle* C in $\mathcal{N}$ is any set of arcs $C \subseteq A_{\mathcal{C}}$ cyclically sequenced as $a_0, a_1, \ldots a_{\ell-1}$ where an head equals a tail, i.e., $h(a_i) = t(a_j)$, iff $j = i + 1 \mod \ell$; it is called a *negative cycle* if $w(C) \leq 0$, where $w(C)$ stands for $\sum_{a \in C} w_a$. A graph is called *conservative* when it contains no negative cycle. A *schedule* is any function $f : \mathcal{T} \mapsto \mathbb{R}$. So the *reduced weight* of an arc $a = (t, h, w_a)$ with respect to a schedule f is defined as $w_a^f \triangleq w_a - f(h) + f(t)$. A schedule f is *feasible* iff $w_a^f \geq 0$ for every $a \in A_{\mathcal{C}}$.

It is also worth noticing that, given two feasible schedules s_1, s_2 of any STN, the pointwise-minimum schedule, $s(u) \triangleq \min(s_1(u), s_2(u))\ \forall u \in \mathcal{T}$, is also feasible. Indeed, among all of the possible feasible schedules of a given consistent STN, it is natural to consider the *least* feasible one; i.e., $\hat{s} : \mathcal{T} \to \mathbb{R}_{\geq 0}$ is the *least feasible schedule* of STN $\mathcal{N}$ if $\hat{s}$ is feasible for $\mathcal{N}$ and, for any other non-negative feasible schedule $s' \geq 0$ of $\mathcal{N}$, it holds $\hat{s}(u) \leq s'(u)\ \forall u \in \mathcal{T}$.

Remarkably, finding the least feasible schedule of an STN takes polynomial time [4].

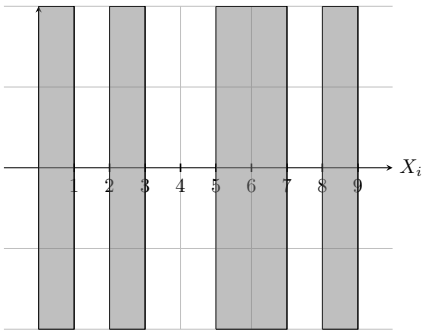


Figure 1 An example of a $\mathtt{t}_2$-constraint: $(0 \leq X_i \leq 1) \vee (2 \leq X_i \leq 3) \vee (5 \leq X_i \leq 7) \vee (8 \leq X_i \leq 9)$.

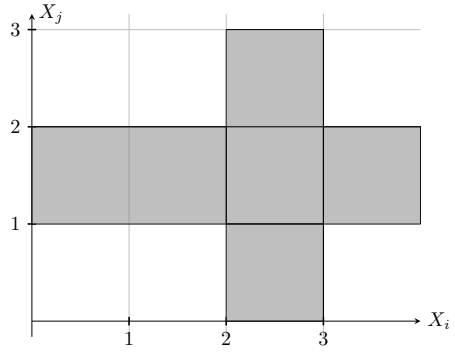


Figure 2 An example of a $\mathtt{t}_3$-constraint: $(2 \leq X_i \leq 3) \vee (1 \leq X_j \leq 2)$.

▶ **Theorem 2** ([4]). *Let $\mathcal{N} = (\mathcal{T}, \mathcal{C})$ be an STN. The* Bellman-Ford *(BF) algorithm (cfr. [2]) produces in $O(|\mathcal{T}| \cdot |\mathcal{C}|)$ time: either the least feasible schedule $\hat{s} : \mathcal{T} \to \mathbb{R}_{\geq 0}$, in case $\mathcal{N}$ is consistent; or a certificate that $\mathcal{N}$ is inconsistent in the form of a negative cycle. Moreover, if the weights of the arcs are all integers, then the scheduling values of $\hat{s}$ are all integers too.*

Concerning the BF algorithm itself, it's worth considering an improved variant of it that we call the *Bellman-Ford Value-Iteration* (BF-VI). The basic idea of BF-VI is the same as the original BF algorithm in that each vertex is used as a candidate to relax its adjacent vertices. The improvement is that instead of trying all vertices blindly, BF-VI maintains a queue Q of candidate vertices and adds a vertex to Q only if that vertex is relaxed. A candidate vertex v is extracted from Q according to a fixed policy (e.g., LIFO), and then the adjacent vertices of v are possibly relaxed as usual and added to Q (if they are not already in there, no repetitions are allowed). This process repeats until no more vertex can be relaxed.

BF-VI serves us as a basic model, to be leveraged to design faster algorithms for RDTPs.

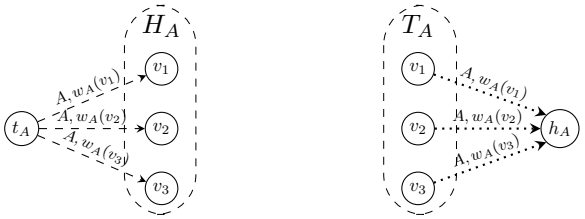
2.1 Restricted Disjunctive Temporal Problems

Let us proceed by formally defining RDTNs and RDTPs. Fig. 1 and Fig. 2 (above) illustrate an example of a $\mathtt{t}_2$-constraint and $\mathtt{t}_3$-constraint (respectively).

▶ **Definition 3** (RDTNs, RDTPs [9, 10]). A *Restricted Disjunctive Temporal Network* (RDTN) $\mathcal{N}$ is a pair $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a set of time-points and $\mathcal{C} = \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2} \cup \mathcal{C}_{\mathtt{t}_3}$ is a set of *restricted disjunctive temporal constraints* over $\mathcal{T}$, each being either one of the following three types:
$(\mathtt{t}_1)$: $(Y - X \leq w_{X,Y})$, where $X, Y \in \mathcal{T}$ and $w_{X,Y} \in \mathbb{R}$;
$(\mathtt{t}_2)$: $\bigvee_{i=1}^{k}(l_i \leq X \leq u_i)$, where $X \in \mathcal{T}$ and $l_i, u_i \in \mathbb{R}$ for every $i = 1, \ldots, k$;
$(\mathtt{t}_3)$: $(l_1 \leq X \leq u_1) \vee (l_2 \leq Y \leq u_2)$, where $X, Y \in \mathcal{T}$ and $l_i, u_i \in \mathbb{R}$ for $i = 1, 2$.
An RDTN is *consistent* if it admits a *feasible schedule*, i.e., some $s : \mathcal{T} \mapsto \mathbb{R}$ satisfying all of the disjunctive temporal constraints in $\mathcal{C}$. The *Restricted Disjunctive Temporal Problem* (RDTP) is that of determining whether a given RDTN is consistent or not.

Notice that $\mathtt{t}_1$-constraints do coincide with simple temporal constraints of STNs.

We assume w.l.o.g. that the disjuncts of any $\mathtt{t}_2$-constraint are arranged in ascending order of the end points of their corresponding intervals, i.e., $l_i < l_{i+1} \wedge u_i < u_{i+1}\ \forall i$, whenever $\bigvee_{i=1}^{k}(l_i \leq X \leq u_i) \in \mathcal{C}_{\mathtt{t}_2}$; these natural orderings on the interval domains of the time-points will be referred to as as their *nominal ordering*. For any $\tau \in \{1, 2, 3\}$, $|\mathcal{C}_{\mathtt{t}_\tau}|$ denotes the number



(a) Multi-Head Hyperarc **(b)** Multi-Tail Hyperarc

Figure 3 Hyperarcs in Hyper Temporal Networks.

of $\mathtt{t}_\tau$-constraints (i.e., the cardinality of $\mathcal{C}_{\mathtt{t}_\tau}$, not the encoding length). Also, for any $c_X \in \mathcal{C}_{\mathtt{t}_2}$, $|c_X|$ denotes the number of disjuncts of c_X, and $d_{\mathcal{C}_{\mathtt{t}_2}} \triangleq \sum_{c_X \in \mathcal{C}_{\mathtt{t}_2}} |c_X|$. Finally, let us fix a total ordering on the time-points, i.e., $\mathcal{T} = \{T_1, \ldots, T_k\}$, this induces an ordering on the pair of disjuncts in any $\mathtt{t}_3$-constraint; so, provided, $c = (l_1 \leq X_i \leq u_1) \vee (l_2 \leq X_j \leq u_2) \in \mathcal{C}_{\mathtt{t}_3}$, for some $i < j$, then, $d' \triangleq (l_1 \leq X_i \leq u_1)$ and $d'' \triangleq (l_2 \leq X_j \leq u_2)$ are called the *first* and the *second* disjunct of c (respectively).

As mentioned in the introduction, Kumar showed in [10, 9] that RDTPs are solvable in deterministic strongly polynomial time by reducing them to CRCs [6].

2.2 Hyper Temporal Networks

In order to study the tractability frontier of RDTPs, we shall consider the HyTN model which is grounded on directed hypergraphs as defined next.

▶ **Definition 4** ([3]). A *directed hypergraph* $\mathcal{H}$ is a pair $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is the set of nodes, and $\mathcal{A}$ is the set of *hyperarcs*. Each hyperarc $A \in \mathcal{A}$ is either *multi-head* or *multi-tail*:

- A *multi-head* hyperarc $A = (t_A, H_A, w_A)$ has a distinguished node t_A, called the *tail* of A, and a non-empty set $H_A \subseteq V \setminus \{t_A\}$ containing the *heads* of A; to each head $v \in H_A$, it is associated a *weight* $w_A(v) \in \mathbb{R}$, which is a real number (unless otherwise specified). Fig. 3a depicts a possible representation of a multi-head hyperarc: the tail is connected to each head by a dashed arc labeled by the name of the hyperarc and the weight associated to the considered head.
- A *multi-tail* hyperarc $A = (T_A, h_A, w_A)$ has a distinguished node h_A, called the *head* of A, and a non-empty set $T_A \subseteq V \setminus \{h_A\}$ containing the *tails* of A; to each tail $v \in T_A$, it is associated a *weight* $w_A(v) \in \mathbb{R}$, which is a real number (unless otherwise specified). Fig. 3b depicts a possible representation of a multi-tail hyperarc: the head is connected to each tail by a dotted arc labeled by the name of the hyperarc and weights.

The *cardinality* of a hyperarc $A \in \mathcal{A}$ is $|A| \triangleq |H_A \cup \{t_A\}|$ if A is multi-head, and $|A| \triangleq |T_A \cup \{h_A\}|$ if A is multi-tail; when $|A| = 2$, then $A = (u, v, w)$ is a standard arc. The *order* and *size* of a directed hypergraph $(\mathcal{T}, \mathcal{A})$ are $|\mathcal{T}|$ and $m_{\mathcal{A}} \triangleq \sum_{A \in \mathcal{A}} |A|$ (respectively).

▶ **Definition 5** (General-HyTN [3]). A *general-HyTN* is a directed hypergraph $\mathcal{H} = (\mathcal{T}, \mathcal{A})$ where each node $X \in \mathcal{T}$ represents a time-point, and each multi-head/multi-tail hyperarc stands for a set of temporal distance constraints between the tail/head and the heads/tails.

In general-HyTNs, an hyperarc is *satisfied* when at least one of its distance constraints is satisfied. Then, a HyTN is *consistent* when it is possible to assign a value to each time-point so that all of its hyperarcs are satisfied. More formally, in the HyTN model the consistency-checking problem is the following decision problem.

▶ **Definition 6** (General-HyTP [3]). Given a general-HyTN $\mathcal{H} = (\mathcal{T}, \mathcal{A})$, the *General Hyper Temporal Problem* (General-HyTP) is that of deciding whether or not there exists a schedule $s : \mathcal{T} \to \mathbb{R}$ such that, for every hyperarc $A \in \mathcal{A}$, the following hold:

- if $A = (t, h, w)$ is a standard arc, then: $s(h) - s(t) \leq w$;
- if $A = (t_A, H_A, w_A)$ is a multi-head hyperarc, then: $s(t_A) \geq \min_{v \in H_A}\{s(v) - w_A(v)\}$;
- if $A = (T_A, h_A, w_A)$ is a multi-tail hyperarc, then: $s(h_A) \leq \max_{v \in T_A}\{s(v) + w_A(v)\}$.

Any such schedule s is called *feasible*. A HyTN that admits at least one feasible schedule is called *consistent*.

Comparing the consistency of HyTNs with the consistency of STNs, the most important aspect of novelty is that, while in a distance graph of STNs each arc represents a distance constraint and all such constraints have to be satisfied by any feasible schedule, in a HyTN each hyperarc represents a disjunction of one or more distance constraints and a feasible schedule has to satisfy at least one of such distance constraints for each hyperarc.

Let us survey some interesting properties about the consistency-checking problem above. The first one is that any integer-weighted HyTN admits an integer-valued feasible schedule when it is consistent, as stated in the following proposition.

▶ **Proposition 7** ([3]). *Let $\mathcal{H} = (\mathcal{T}, \mathcal{A})$ be an integer-weighted* and consistent general-HyTN. Then $\mathcal{H}$ admits an integer feasible schedule $s : \mathcal{T} \to \{-T, -T+1, \ldots, T-1, T\}$, where $T = \sum_{A \in \mathcal{A}, v \in \mathcal{T}} |w_A(v)|$.*

The following theorem states that General-HyTP is NP-complete, in a strong sense.

▶ **Theorem 8** ([3]). General-HyTP *is an NP-complete problem even if the input instances $\mathcal{H} = (V, \mathcal{A})$ are restricted to satisfy $w_A(\cdot) \in \{-1, 0, 1\}$ and $|H_A|, |T_A| \leq 2$ for every $A \in \mathcal{A}$.*

As observed in [3], Theorem 8 motivates the study of consistency problems on HyTNs having either only multi-head or only multi-tail hyperarcs. In the former case, the consistency-checking problem is called head-HyTP, while in the latter it is Tail-HyTP; as stated in Theorem 10, the complexity of checking these two problems turns out to be lower than that for DTPs, i.e., both head-HyTP, Tail-HyTP $\in$ NP $\cap$ co-NP, instead of being NP-complete.

So it's worth considering the following specialized notion of consistency for HyTNs.

▶ **Definition 9** (head-HyTP). Given a multi-head HyTN $\mathcal{H} = (\mathcal{T}, \mathcal{A})$, the head-HyTP problem is that of deciding whether or not there exists a schedule $s : \mathcal{T} \to \mathbb{R}$ such that:

$$s(t_A) \geq \min_{v \in H_A}\{s(v) - w_A(v)\}, \quad \forall A \in \mathcal{A}.$$

The tightest currently known worst-case time complexity upper-bound for solving (integer-weighted) head-HyTPs was established in [3] and it is expressed in the following theorem.

▶ **Theorem 10** ([3]). *The following proposition holds on (integer-weighted, multi-head) HyTNs. There exists an $O\big((|\mathcal{T}| + |\mathcal{A}|) \cdot m_{\mathcal{A}} \cdot W\big)$ pseudo-polynomial time algorithm for checking* head-HyTP*; given any HyTN $\mathcal{H} = (\mathcal{T}, \mathcal{A})$, if $\mathcal{H}$ is consistent the same algorithm also returns an integer-valued feasible schedule $s : \mathcal{T} \to \mathbb{Z}$ of $\mathcal{H}$; otherwise, it returns a negative certificate in the form of a negative* hypercycle *(cfr. Appendix A for more details on that).*

Above, $W \triangleq \max_{A \in \mathcal{A}, v \in H_A} |w_A(v)|$ is the maximum absolute value among the weights.

* Integer-weighted *HyTN* means that $w_A(v) \in \mathbb{Z}$ for every $A \in \mathcal{A}$ and $v \in \mathcal{T}$ for which $w_A(v)$ is defined.

Concluding this section we recall that the two problems HEAD-HyTP and TAIL-HyTP are actually inter-reducible, i.e., one can check any one of the two models in $f(m, n, W)$-time whenever there's an $f(m, n, W)$-time procedure for checking the consistency of the other one.

▶ **Theorem 11** ([3]). HEAD-HyTP *and* TAIL-HyTP *are inter-reducible by means of* log-*space, linear-time, local-replacement reductions.*

Thus, Theorem 10 extends to multi-tail HyTNs (i.e., they're checkable in pseudo-poly time).

3 Faster Deterministic Algorithm for $\mathtt{t}_2$DTPs

This section offers a deterministic quadratic time algorithm for solving temporal problems having only $\{\mathtt{t}_1, \mathtt{t}_2\}$-constraints, as defined below.

The same algorithm will be leveraged to solve RDTPs fastly, later on in Section 4.

▶ **Definition 12.** Any RDTN $\mathcal{N} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2} \cup \mathcal{C}_{\mathtt{t}_3})$ having $\mathcal{C}_{\mathtt{t}_3} = \emptyset$ is called $\mathtt{t}_2$*DTN*.

So, $\mathtt{t}_2$DTNs are denoted simply as $(\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2})$. The corresponding temporal problem, i.e., $\mathtt{t}_2$*DTP*, is that of determining whether a given $\mathtt{t}_2$DTN is consistent or not.

One possible solution to $\mathtt{t}_2$DTPs is Kumar's reduction from RDTPs to CRCs [10]. Our solution, named `t2DTP()`, employs kind of a value-iteration approach in which all are initially set to zero and then updated monotonically upwards by necessary arc relaxations – this is somehow reminiscent of the BF-VI algorithm for STPs mentioned in Section 2. Indeed, given a $\mathtt{t}_2$DTN $\mathcal{N}_{\mathtt{t}_2} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2})$, we firstly solve the STP $\mathcal{N}_{\mathtt{t}_1} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1})$ (e.g., with BF-VI). If $\mathcal{N}_{\mathtt{t}_1}$ is consistent, the returned least feasible schedule $\hat{\varphi}_{\mathcal{N}}$ provides an initial candidate, the next step in mind being that of satisfying all the $\mathtt{t}_2$-constraints. For this, recall that the disjuncts of any $c_{\mathtt{t}_2} \in \mathcal{C}_{\mathtt{t}_2}$ are arranged according to their nominal ordering, so that we can try to satisfy any given $c_{\mathtt{t}_2}$ by iteratively picking the next (i.e., in ascending order) unsatisfied disjunct of $c_{\mathtt{t}_2}$ and by enforcing its lower-bound constraint in an auxiliary STN as if it were a $\mathtt{t}_1$-constraint. While there's an unsatisfied $\mathtt{t}_2$-constraint $c_{\mathtt{t}_2}$, the current candidate schedule is thus increased by the least necessary amount satisfying both $c_{\mathtt{t}_2}$ and the whole $\mathcal{C}_{\mathtt{t}_1}$. It turns out that this can be done efficiently by performing $|\mathcal{C}_{\mathtt{t}_2}|$ calls to the Dijkstra shortest paths algorithm [7]. In order to show this, let us point out two key facts (i.e., Lemma 13 and 14).

▶ **Lemma 13.** *Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}_{t_1})$ be any STN, and let φ, φ' be any pair of schedules of $\mathcal{N}$.*

Let $\mathcal{N}^{\varphi}$ be the STN reweighted *according to the reduced-costs weight transformation w^{φ} (i.e., each weight $w_{X,Y}$ in $\mathcal{N}$ is simply replaced by $w^{\varphi}_{X,Y}$), let $\mathcal{N}^{\varphi'}$ be the same w.r.t. $w^{\varphi'}$. Let $\delta^{\varphi}_X : \mathcal{T} \to \mathbb{Z} \cup \{+\infty\}$ be the length of the shortest path in $\mathcal{N}^{\varphi}$ from any $T \in \mathcal{T}$ to $X \in \mathcal{T}$, and let $\delta^{\varphi'}_X$ be the same w.r.t. $\mathcal{N}^{\varphi'}$.*

Then for every $T, X \in \mathcal{T}$, either $\delta^{\varphi'}_X(T)$ and $\delta^{\varphi}_X(T)$ are both $+\infty$, or the following holds:

$$\delta^{\varphi'}_X(T) - \delta^{\varphi}_X(T) = \big(\varphi'(T) - \varphi(T)\big) - \big(\varphi'(X) - \varphi(X)\big).$$

Proof. Let $T, X \in \mathcal{T}$ (arbitrarily), w.l.o.g. X is reachable from T in $\mathcal{N}$ (otherwise, $\delta^{\varphi'}_X(T)$ and $\delta^{\varphi}_X(T)$ are both $+\infty$). Consider any path $p_{T,X}$ from T to X in $\mathcal{N}^{\varphi}$, i.e., for some $k \geq 0$:

$$p_{T,X} \triangleq (T = T_0, T_1, T_2, \ldots, T_k = X), \text{ having total weight } w^{\varphi}_{p_{T,X}} \triangleq \sum_{i=0}^{k-1} w^{\varphi}_{T_i,T_{i+1}} \text{ in } \mathcal{N}^{\varphi}.$$

Then, the following holds by telescoping:

$$\begin{aligned} w^{\varphi}_{p_{T,X}} &= \big(w_{T_0,T_1} - \varphi(T_1) + \varphi(T_0)\big) + \big(w_{T_1,T_2} - \varphi(T_2) + \varphi(T_1)\big) + \dots \\ &\qquad\qquad \dots + \big(w_{T_{k-1},T_k} - \varphi(T_k) + \varphi(T_{k-1})\big) \\ &= \varphi(T_0) - \varphi(T_k) + \sum_{i=0}^{k-1} w_{T_i,T_{i+1}} = \varphi(T) - \varphi(X) + w_{p_{T,X}}. \end{aligned} \tag{1}$$

Thus, provided $\delta_X(T)$ is the shortest path distance from T to X in the original network $\mathcal{N}$, we have:

$$\begin{aligned} \delta^{\varphi}_X(T) &= \min\big\{w^{\varphi}_{p_{T,X}} \mid p_{T,X} \text{ is a path from } T \text{ to } X \text{ in } \mathcal{N}\big\} && \text{(by def. of } \delta^{\varphi}_X) \\ &= \min\big\{\varphi(T) - \varphi(X) + w_{p_{T,X}} \mid p_{T,X} \text{ is any path from } T \text{ to } X \text{ in } \mathcal{N}\big\} && \text{(by (1))} \\ &= \varphi(T) - \varphi(X) + \min\big\{w_{p_{T,X}} \mid p_{T,X} \text{ is any path from } T \text{ to } X \text{ in } \mathcal{N}\big\} && (\varphi \text{ is constant here}) \\ &= \varphi(T) - \varphi(X) + \delta_X(T). && \text{(by def. of } \delta_X) \end{aligned}$$

For the same reason, $\delta^{\varphi'}_X(T) = \varphi'(T) - \varphi'(X) + \delta_X(T)$. Therefore,

$$\begin{aligned} \delta^{\varphi'}_X(T) - \delta^{\varphi}_X(T) &= \big(\varphi'(T) - \varphi'(X) + \delta_X(T)\big) - \big(\varphi(T) - \varphi(X) + \delta_X(T)\big) \\ &= \big(\varphi'(T) - \varphi(T)\big) - \big(\varphi'(X) - \varphi(X)\big). \end{aligned}$$

This concludes the proof. ◀

▶ **Lemma 14.** *Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1})$ be any STN, and let $\hat{\varphi}$ be the least feasible schedule of $\mathcal{N}$. Fix some $X \in \mathcal{T}$ and some real value $l_X \geq \hat{\varphi}(X)$. Let $\mathcal{N}' = (\mathcal{T}', \mathcal{C}'_{\mathtt{t}_1})$ be the auxiliary STN obtained by introducing a corresponding lower-bound $\mathtt{t}_1$-constraint over X, i.e.,*

$$\mathcal{T}' \triangleq \mathcal{T} \cup \{z\}, \qquad \mathcal{C}'_{\mathtt{t}_1} \triangleq \mathcal{C}_{\mathtt{t}_1} \cup \big\{(z - T \leq 0) \mid T \in \mathcal{T}\big\} \cup \big\{(z - X \leq -l_X)\big\}.$$

Let $\mathcal{N}^{\hat{\varphi}}$ be the STN reweighted according to the reduced-costs weight transformation $w^{\hat{\varphi}}$, and let $\delta^{\hat{\varphi}}_X(T)$ be the length of the shortest path in $\mathcal{N}^{\hat{\varphi}}$ from (any) $T \in \mathcal{T}$ to X.

Then, for every $T \in \mathcal{T}$, the least feasible schedule $\hat{\varphi}'$ of $\mathcal{N}'$ is given by:

$$\hat{\varphi}'(T) = \hat{\varphi}(T) + \max\big(0, l_X - \hat{\varphi}(X) - \delta^{\hat{\varphi}}_X(T)\big).$$

Proof. Let w.l.o.g. $\hat{\varphi}'(z) = 0$. In order to become feasible for $\mathcal{N}'$ we claim, for every $T \in \mathcal{T}$, that the least feasible schedule $\hat{\varphi}(T)$ must be increased by at least $\max\big(0, l_X - \hat{\varphi}(X) - \delta^{\hat{\varphi}}_X(T)\big)$ time units (because of the lower-bound constraint $(z - X \leq -l_X) \in \mathcal{C}'_{\mathtt{t}_1}$). Indeed, for any $T \in \mathcal{T}$ that reaches X in $\mathcal{N}$, the $\mathtt{t}_1$-constraint $(X - T \leq \delta_X(T))$ (which is induced by telescoping all of the $\mathtt{t}_1$-constraints along any shortest path from T to X) must be satisfied. On the other hand, by Lemma 13 (applied to $\hat{\varphi}$ and to the anywhere-zero[†] schedule), it holds $\hat{\varphi}(X) - \hat{\varphi}(T) = \delta_X(T) - \delta^{\hat{\varphi}}_X(T)$. This can be seen as follows: if $\hat{\varphi}(T)$ is kept fixed, then $\hat{\varphi}(X)$ can be increased by at most $\delta^{\hat{\varphi}}_X(T)$ time units without breaking the induced constraint $(X - T \leq \delta_X(T))$. Here, $\hat{\varphi}(X)$ must be increased by at least $l_X - \hat{\varphi}(X)$ time units in order to satisfy $(z - X \leq -l_X) \in \mathcal{C}'_{\mathtt{t}_1}$, so $\hat{\varphi}(T)$ must be increased by at least the amount said above.

Next, we claim this increase also preserves feasability, i.e., it is the *least feasible increase*. For ease of notation, let $f(z) \triangleq 0$ and $f(T) \triangleq \hat{\varphi}(T) + \max\big(0, l_X - \hat{\varphi}(X) - \delta^{\hat{\varphi}}_X(T)\big)\ \forall T \in \mathcal{T}$.

[†] The anywhere-zero schedule ζ is that defined as, $\zeta(T) = 0$ for every $T \in \mathcal{T}$.

In order to prove that f satifies all the constraints in $\mathcal{C}_{\mathtt{t}_1}$, pick any $(B - A \leq w_{A,B}) \in \mathcal{C}_{\mathtt{t}_1}$. By hypothesis, it holds:

$$\hat{\varphi}(B) - \hat{\varphi}(A) \leq w_{A,B}. \tag{1}$$

For the sake of the argument, let us define: $\Delta_{A,B} \triangleq \big(f(B) - \hat{\varphi}(B)\big) - \big(f(A) - \hat{\varphi}(A)\big)$.

So, the following holds:

$$f(B) - f(A) = \hat{\varphi}(B) - \hat{\varphi}(A) + \Delta_{A,B}. \tag{2}$$

Then, either one of the following two cases holds:

- If $l_X - \hat{\varphi}(X) \leq \delta_X^{\hat{\varphi}}(B)$, then $f(B) = \hat{\varphi}(B)$, so $\Delta_{A,B} \leq 0$. Therefore,

$$\begin{aligned} f(B) - f(A) &\leq \hat{\varphi}(B) - \hat{\varphi}(A) && \text{(by (2))} \\ &\leq w_{A,B}. && \text{(by (1))} \end{aligned}$$

- If $l_X - \hat{\varphi}(X) > \delta_X^{\hat{\varphi}}(B)$, it is easy to check that $\Delta_{A,B} \leq \delta_X^{\hat{\varphi}}(A) - \delta_X^{\hat{\varphi}}(B)$.
  By definition of $\delta_X^{\hat{\varphi}}$ and since $(B - A \leq w_{A,B}) \in \mathcal{C}_{\mathtt{t}_1}$, then $\delta_X^{\hat{\varphi}}(A) \leq \delta_X^{\hat{\varphi}}(B) + w_{A,B}^{\hat{\varphi}}$.
  Therefore,

$$\begin{aligned} f(B) - f(A) &\leq \hat{\varphi}(B) - \hat{\varphi}(A) + \delta_X^{\hat{\varphi}}(A) - \delta_X^{\hat{\varphi}}(B) \\ &\leq \hat{\varphi}(B) - \hat{\varphi}(A) + w_{A,B}^{\hat{\varphi}} = w_{A,B}. \end{aligned}$$

So, in either case, $f(B) - f(A) \leq w_{A,B}$.

Finally, clearly $f(X) = l_X$, so $(z - X \leq -l_X) \in \mathcal{C}'_{\mathtt{t}_1}$ is also satisfied.

This proves f is a feasible schedule of $\mathcal{N}'$. All in, it is the least feasible, i.e., $f = \hat{\varphi}'$. ◀

With this two facts in mind, the description of `t2DTP()` can now proceed more smoothly.

Recall that, firstly, the STN $\mathcal{N}_{\mathtt{t}_1}$ is checked. If $\mathcal{N}_{\mathtt{t}_1}$ is already inconsistent, so it is $\mathcal{N}_{\mathtt{t}_2}$; otherwise, $\hat{\varphi}_{\mathcal{N}}$ is the least feasible schedule of $\mathcal{N}_{\mathtt{t}_1}$. So, $w^{\hat{\varphi}_{\mathcal{N}}} \geq 0$ for every constraint in $\mathcal{C}_{\mathtt{t}_1}$. Now, for each target node $X \in \mathcal{T}$, the Dijkstra algorithm on input $(\mathcal{N}^{\hat{\varphi}_{\mathcal{N}}}, X)$ computes $\delta_X^{\hat{\varphi}_{\mathcal{N}}}(T)$. The whole distance matrix $\{\delta_X^{\hat{\varphi}_{\mathcal{N}}}(T)\}_{T \in \mathcal{T}, X \in \mathcal{T}}$ is computed here, and kept stored in memory. Multiple-sources single-target shortest paths are needed, actually, but these can be easily computed with the traditional Dijkstra's algorithm, e.g., just reverse the direction of all arcs in the input network and treat the single-target node as if it were a single-source. What follows aims, if there's still an unsatisfied $\mathtt{t}_2$-constraint $c_{\mathtt{t}_2} \in \mathcal{C}_{\mathtt{t}_2}$, at increasing the candidate schedule f by the least necessary amount satisfying both $c_{\mathtt{t}_2}$ and the whole $\mathcal{C}_{\mathtt{t}_1}$. So, let us initialize $f \leftarrow \hat{\varphi}_{\mathcal{N}}$. Then the following iterates.

While $\exists$ some $X \in \mathcal{T}$ and $c_X = \bigvee_{i=1}^{k}(l_i \leq X \leq u_i) \in \mathcal{C}_{\mathtt{t}_2}$ s.t. $f(X)$ doesn't satisfy c_X:

if $f(X) > u_k (= \max_i u_i)$, then $\mathcal{N}_{\mathtt{t}_2}$ is inconsistent (see Theorem 15); otherwise, let i^* be the smallest $i \in [1, k]$ such that $f(X) < l_i$. By Lemma 13 and given f, then δ_X^f is given by:

$$\delta_X^f(T) \leftarrow \delta_X^{\hat{\varphi}_0}(T) + \big(f(T) - \hat{\varphi}_0(T)\big) - \big(f(X) - \hat{\varphi}_0(X)\big),\ \forall\, T \in \mathcal{T}. \tag{rule-δ}$$

So, by Lemma 14, the following updating rule:

$$f(T) \leftarrow f(T) + \max\big(0, l_{i^*} - f(X) - \delta_X^f(T)\big),\ \forall\, T \in \mathcal{T}. \tag{rule-f}$$

yelds the least feasible schedule for the next auxiliary STN $\mathcal{N}'_{\mathtt{t}_1}$ obtained by adding the new lower-bound $\mathtt{t}_1$-constraint $(z - X \leq -l_{i^*})$. At each iteration of the while-loop $\mathcal{N}'_{\mathtt{t}_1}$ is enriched with an additional lower-bound $\mathtt{t}_1$-constraint as above. So, $\mathcal{N}'_{\mathtt{t}_1}$ has $|\mathcal{T}'| = |\mathcal{T}| + 1$ time-points (z included) and at most $|\mathcal{C}'_{\mathtt{t}_1}| \leq |\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{C}_{\mathtt{t}_2}|$ $\mathtt{t}_1$-constraints (one $\mathtt{t}_1$-constraint per $c_{\mathtt{t}_2} \in \mathcal{C}_{\mathtt{t}_2}$

is enough, as for each $c_{\mathtt{t}_2}$ only its greatest lower-bound counts). If the while-loop completes without ever finding $\mathcal{N}_{\mathtt{t}_2}$ to be inconsistent (because, eventually, $f(X) > u_k (= \max_i u_i)$ for some $X \in \mathcal{T}$ at some point), then the last updating of f yelds the least feasible schedule of $\mathcal{N}_{\mathtt{t}_2}$ (as shown below in Theorem 15). This concludes the description of `t2DTP()`.

Notice that, during the whole computation, the scheduling values can only increase monotonically upwards – like in a value-iteration process.

▶ **Theorem 15.** *`t2DTP()` is correct, i.e., on any input $\mathtt{t}_2$DTN $\mathcal{N}_{\mathtt{t}_2} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2})$, it returns a feasible schedule $\varphi : \mathcal{T} \to \mathbb{R}$, if $\mathcal{N}_{\mathtt{t}_2}$ is consistent; otherwise, it recognizes $\mathcal{N}_{\mathtt{t}_2}$ as inconsistent.*

Proof. Let $\iota = 0, 1, 2, \ldots, \iota_h$ be all the iterations of the while-loop of `t2DTP()`, where ι_h is assumed to be the last iteration where the updating `rule-`f is applied.

For every iteration $\iota \in [1, \iota_h]$, the auxiliary STN $\mathcal{N}'^{(\iota)}_{\mathtt{t}_1}$ is formally defined as:

$$\mathcal{N}'^{(\iota)}_{\mathtt{t}_1} \triangleq (\mathcal{T} \cup \{z\}, \mathcal{C}'^{(\iota)}_{\mathtt{t}_1}), \text{ where } z \text{ is the } \textit{zero time-point}, \text{ and ...}$$

$$\mathcal{C}'^{(\iota)}_{\mathtt{t}_1} \triangleq \mathcal{C}_{\mathtt{t}_1} \cup \big\{(z - T \leq 0) \mid T \in \mathcal{T}\big\} \cup \big\{(z - X^{(\gamma)} \leq -l^{(\gamma)}_{i^*}) \mid 1 \leq \gamma \leq \iota\big\},$$

where, for all $\gamma \leq \iota$, $X^{(\gamma)}$ is the (unique) $X \in \mathcal{T}$ appearing in some $\mathtt{t}_2$-constraint that is considered at the while-loop's γ-th iteration, and $l^{(\gamma)}_{i^*}$ is its corresponding lower-bound.

Also, let $f^{(\iota)}$ be the candidate schedule as updated by `rule-`f during the ι-th iteration.

By applying Lemma 13 and 14 repeatedly, for each iteration ι, it holds that $f^{(\iota)}$ is the least feasible schedule of $\mathcal{N}'^{(\iota)}_{\mathtt{t}_1}$. This is the key invariant at the heart of `t2DTP()`.

Concerning actual correctness, firstly, assume that `t2DTP()` recognizes $\mathcal{N}_{\mathtt{t}_2}$ as inconsistent.

If $\mathcal{N}_{\mathtt{t}_1}$ was already inconsistent (cfr. Theorem 2), so $\mathcal{N}_{\mathtt{t}_2}$ is too. Otherwise, the inconsistency of $\mathcal{N}_2$ really holds because of these two facts jointly: (i) the key invariant mentioned above; and, (ii) at the end of the while-loop, it must be $f(X) > u_k (= \max_i u_i)$ for some $\mathtt{t}_2$-constraint $c_X = \bigvee_{i=1}^{k} (l_i \leq X \leq u_i) \in \mathcal{C}_{\mathtt{t}_2}$. Indeed notice that, by (i), no possible feasible schedule $g < f$ can be neglected (discarded) during the upward monotone (value-iteration like) updates of the schedules; and, by (ii), no possible schedule $g \geq f$ can ever satisfy $c_X \in C_2$. So, $\mathcal{N}_{\mathtt{t}_2}$ is really inconsistent.

Secondly, assume that $\mathcal{N}_{\mathtt{t}_2}$ is recognized as consistent, by returning a schedule $f^{(\iota_h)}$.

Since `t2DTP()` can do that only after the above while-loop completes, the exit condition of the latter ensures that $f^{(\iota_h)}$ satisfies every constraint in $\mathcal{C}_{\mathtt{t}_2}$. Moreover, the key invariant implies that $f^{(\iota_h)}$ is the least feasible schedule of $\mathcal{N}'^{(\iota_h)}_{\mathtt{t}_1}$, so that $f^{(\iota_h)}$ satisfies all of the $\mathtt{t}_1$-constraints in $\mathcal{C}_{\mathtt{t}_1}$. These two combined, $f^{(\iota_h)}$ is the least feasible schedule of $\mathcal{N}_{\mathtt{t}_2}$. So, $\mathcal{N}_{\mathtt{t}_2}$ is indeed consistent. ◀

The next result asserts that `t2DTP()` always halts in time polynomial in the input size.

▶ **Theorem 16.** *Suppose that `t2DTP()` runs on input $\mathtt{t}_2$DTN $\mathcal{N}_{\mathtt{t}_2} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2})$.*
Then, `t2DTP()` halts in time $O\big(|\mathcal{T}| \cdot |\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{C}_{\mathtt{t}_2}| \cdot (|\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{T}| \cdot \log |\mathcal{T}|) + |\mathcal{T}| \cdot d_{\mathcal{C}_{\mathtt{t}_2}}\big)$.

Proof. Solving the STP $\mathcal{N}_{\mathtt{t}_1} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1})$ with BF-VI takes $O(|\mathcal{T}| \cdot |\mathcal{C}_{\mathtt{t}_1}|)$ time (cfr. Theorem 2). Computing the shortest paths distance matrix $\{\delta^{\hat{\varphi}_{\mathcal{N}}}_X(T)\}_{T \in \mathcal{T}, X \in \mathcal{T}}$ takes $|\mathcal{C}_{\mathtt{t}_2}|$ calls to the Dijkstra algorithm (one per $X \in \mathcal{T}$ participating in some $\mathtt{t}_2$-constraint), so, $O(|\mathcal{C}_{\mathtt{t}_2}| \cdot (|\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{T}| \cdot \log |\mathcal{T}|))$ total time. Checking the while-loop exit condition (i.e., wether there exists some unsatisfied $c_X \in \mathcal{C}_{\mathtt{t}_2}$), can be done in $O(|\mathcal{T}| \cdot d_{\mathcal{C}_{\mathtt{t}_2}})$ total time (because there are at most $d_{\mathcal{C}_{\mathtt{t}_2}}$ iterations and each check can be done in $O(|\mathcal{T}|)$ time). At each iteration of the while-loop, applying `rule-`δ and `rule-`f to all $T \in \mathcal{T}$ takes $O(|\mathcal{T}|)$ time per iteration, and

we have at most $d_{\mathcal{C}_{\mathtt{t}_2}}$ of them; so, notice that it takes only $O(1)$ time per single application of the rules.

Therefore, the overall time complexity of `t2DTP()` on any input $\mathcal{N}_{\mathtt{t}_2} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2})$ is:

$$\mathtt{Time}_{\mathtt{t_2DTP()}}(\mathcal{N}_{\mathtt{t}_2}) = O\big(|\mathcal{T}| \cdot |\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{C}_{\mathtt{t}_2}| \cdot (|\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{T}| \cdot \log|\mathcal{T}|) + |\mathcal{T}| \cdot d_{\mathcal{C}_{\mathtt{t}_2}}\big).$$

This is a strongly polynomial time, i.e., not depending on the magnitude of the arc weights. ◀

4 Faster Deterministic Algorithm for RDTPs

With our brand new $\mathtt{t}_2$DTPs algorithm in mind, let us now focus on solving RDTPs fastly. Given an input RDTP $\mathcal{N} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2} \cup \mathcal{C}_{\mathtt{t}_3})$, we firstly solve the $\mathtt{t}_2$DTP $\mathcal{N}_{\mathtt{t}_2} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2})$ with `t2DTP()` (cfr. Section 3). If $\mathcal{N}_{\mathtt{t}_2}$ is already inconsistent, we're done as $\mathcal{N}$ is too. Otherwise, the key idea is that of checking the consistency of all the $\mathtt{t}_3$-constraints by making one single reduction call to the 2-SAT problem (which can be solved in linear-time [1]).

For this reason, the universe of boolean variables is $\{x_c\}_{c \in \mathcal{C}_{\mathtt{t}_3}}$, i.e., we have one variable per $c \in \mathcal{C}_{\mathtt{t}_3}$. Let d', d'' be the first and second disjunct of any given $c \in \mathcal{C}_{\mathtt{t}_3}$ (respectively), the intended interpretation being that x_c is `true` iff d' is satisfied (and d'' can be anything), whereas x_c is `false` iff d' is unsatisfied and d'' is satisfied.

The 2-CNF formula $\text{CL}_{\mathcal{N}}$ is built as follows. Basically, for each $c \in \mathcal{C}_{\mathtt{t}_3}$ and each disjunct d of c, we enforce the binding requirement of satisfying all the temporal constraints in $\{d\} \cup \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2}$, and we check whether this implies that some other disjunct $\tilde{d}$ of any other $\mathtt{t}_3$-constraint $\tilde{c} \neq c$ becomes unsatisfiable as a consequence. More precisely, we check whether satisfying $\{d\} \cup \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2}$ implies that some weight $\tilde{u}$ must become a *strict* lower-bound for the scheduling value of some $\tilde{X} \in \mathcal{T}$ that appears in some other $\mathtt{t}_3$-disjunct $\tilde{d} = (\tilde{l} \leq \tilde{X} \leq \tilde{u})$. This is formalized in Definition 17 (below). If that is the case, a binary clause asserting the above implication[‡] is added to $\text{CL}_{\mathcal{N}}$. Let us formally describe the details of this construction.

▶ **Definition 17.** Given any RDTP $\mathcal{N} = (\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2} \cup \mathcal{C}_{\mathtt{t}_3})$, initially $\text{CL}_{\mathcal{N}}$ is an empty set of binary clauses. For each $\mathtt{t}_3$-constraint of $\mathcal{N}$, e.g., for each $c = d'_c \vee d''_c \in \mathcal{C}_{\mathtt{t}_3}$ where $d'_c = (l_1 \leq X_i \leq u_1)$ and $d''_c = (l_2 \leq X_j \leq u_2)$, some $i < j$, $\text{CL}_{\mathcal{N}}$ is populated as follows:

1. Consider the $\mathtt{t}_2$DTP $\mathcal{N}[d'_c]_{\mathtt{t}_2}$ in which d'_c is added to $\mathcal{C}_{\mathtt{t}_1}$ as a pair of $\mathtt{t}_1$-constraints, i.e.,

$$\mathcal{N}[d'_c]_{\mathtt{t}_2} \triangleq \Big(\mathcal{T} \cup \{z\}, \big(\mathcal{C}_{\mathtt{t}_1} \cup \{(z - X_i \leq -l_1), (X_i - z \leq u_1)\} \cup \{z - T \leq 0 \mid T \in \mathcal{T}\}\big) \cup \mathcal{C}_{\mathtt{t}_2}\Big).$$

   If $\mathcal{N}[d'_c]_{\mathtt{t}_2}$ is consistent, let $\hat{\varphi}[d'_c]$ be its least feasible schedule; otherwise, add the unary clause $\neg x_c$ to $\text{CL}_{\mathcal{N}}$. For each $\tilde{c} \neq c$ in $\mathcal{C}_{\mathtt{t}_3}$, e.g., $\tilde{c} = (\tilde{l}_1 \leq X_{\tilde{i}} \leq \tilde{u}_1) \vee (\tilde{l}_2 \leq X_{\tilde{j}} \leq \tilde{u}_2) \in \mathcal{C}_{\mathtt{t}_3}$,
   - if $\hat{\varphi}[d'_c](X_{\tilde{i}}) > \tilde{u}_1$ then add the implication $x_c \Rightarrow \neg x_{\tilde{c}}$ (i.e., clause $\neg x_c \vee \neg x_{\tilde{c}}$) to $\text{CL}_{\mathcal{N}}$;
   - if $\hat{\varphi}[d'_c](X_{\tilde{j}}) > \tilde{u}_2$ then add the implication $x_c \Rightarrow x_{\tilde{c}}$ (i.e., clause $\neg x_c \vee x_{\tilde{c}}$) to $\text{CL}_{\mathcal{N}}$.
2. Consider the $\mathtt{t}_2$DTP $\mathcal{N}[d''_c]_{\mathtt{t}_2}$ in which d''_c is added to $\mathcal{C}_{\mathtt{t}_1}$ (similarly as above). If $\mathcal{N}[d''_c]_{\mathtt{t}_2}$ is consistent, let $\hat{\varphi}[d''_c]$ be its least feasible schedule; otherwise, add the unary clause x_c to $\text{CL}_{\mathcal{N}}$. Again, for each $\mathtt{t}_3$-constraint $\tilde{c} \neq c$ of $\mathcal{N}$, e.g., $\tilde{c} = (\tilde{l}_1 \leq X_{\tilde{i}} \leq \tilde{u}_1) \vee (\tilde{l}_2 \leq X_{\tilde{j}} \leq \tilde{u}_2)$: if $\hat{\varphi}[d''_c](X_{\tilde{i}}) > \tilde{u}_1$ then add the implication $\neg x_c \Rightarrow \neg x_{\tilde{c}}$ (i.e., clause $x_c \vee \neg x_{\tilde{c}}$) to $\text{CL}_{\mathcal{N}}$; and if $\hat{\varphi}[d''_c](X_{\tilde{j}}) > \tilde{u}_2$ then add the implication $\neg x_c \Rightarrow x_{\tilde{c}}$ (i.e., the clause $x_c \vee x_{\tilde{c}}$) instead.

[‡] Here, recall the rule of material implication $p \to q \leftrightarrow \neg p \vee q$.

So, if the 2-SAT problem instance $\text{CL}_{\mathcal{N}}$ is unsatisfiable, the input RDTP $\mathcal{N}$ is inconsistent. Otherwise, for every $c = d' \vee d'' \in \mathcal{C}_{\mathtt{t}_3}$ we get at least one feasible $\mathtt{t}_2$DTP: either $\mathcal{N}[d'_c]_{\mathtt{t}_2}$, which is related to the first disjunct $\{d'\} \cup \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2}$; or $\mathcal{N}[d''_c]_{\mathtt{t}_2}$, which is related to the second $\{d''\} \cup \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2}$ (according to whether x_c is `true` or not in the satisfying assignment of $\text{CL}_{\mathcal{N}}$). Then we compute the pointwise-maximum schedule taken among all of those. Formally,

▶ **Definition 18.** Let $\phi : \{x_c\}_{c \in \mathcal{C}_{\mathtt{t}_3}} \to \{\mathtt{true}, \mathtt{false}\}$ be any satisfying assignment of $\text{CL}_{\mathcal{N}}$. For every $c = d'_c \vee d''_c \in \mathcal{C}_{\mathtt{t}_3}$, let us define:

$$d_c^{\phi} \triangleq \begin{cases} d'_c, & \text{if } \phi(x_c) = \mathtt{true}; \\ d''_c, & \text{otherwise.} \end{cases} \qquad \text{then,} \qquad \check{\varphi}_{\mathcal{N}}(T) \triangleq \max_{c \in \mathcal{C}_{\mathtt{t}_3}} \hat{\varphi}[d_c^{\phi}](T), \ \ \forall T \in \mathcal{T},$$

where $\hat{\varphi}[d_c^{\phi}]$ denotes the least feasible schedule of the consistent $\mathtt{t}_2$DTP $\mathcal{N}[d_c^{\phi}]_{\mathtt{t}_2}$.

The above pointwise-maximum schedule $\check{\varphi}_{\mathcal{N}}$ turns out to be feasible for the input RDTP $\mathcal{N}$, as we show next. It is assumed we are given an RDTP $\mathcal{N}$ for which $\text{CL}_{\mathcal{N}}$ is satisfiable.

▶ **Proposition 19.** *Given $\mathcal{N}$ as above, the schedule $\check{\varphi}_{\mathcal{N}}$ satisfies every $c \in \mathcal{C}_{\mathtt{t}_1}$.*

Proof. Let $c_{\mathtt{t}_1} = (Y - X \leq w_{X,Y}) \in \mathcal{C}_{\mathtt{t}_1}$ be any $\mathtt{t}_1$-constraint, some $X, Y \in \mathcal{T}$ and $w \in \mathbb{R}$. Pick any $c_Y^* \in \arg\max_{c \in \mathcal{C}_{\mathtt{t}_3}} \hat{\varphi}[d_c^{\phi}](Y)$. Clearly, $\max_{c \in \mathcal{C}_{\mathtt{t}_3}} \hat{\varphi}[d_c^{\phi}](X) \geq \hat{\varphi}[d_{c_Y^*}^{\phi}](X)$. Therefore:

$$\begin{aligned} \check{\varphi}_{\mathcal{N}}(Y) - \check{\varphi}_{\mathcal{N}}(X) &= \max_{c \in \mathcal{C}_{\mathtt{t}_3}} \hat{\varphi}[d_c^{\phi}](Y) - \max_{c \in \mathcal{C}_{\mathtt{t}_3}} \hat{\varphi}[d_c^{\phi}](X) \\ &\leq \hat{\varphi}[d_{c_Y^*}^{\phi}](Y) - \hat{\varphi}[d_{c_Y^*}^{\phi}](X) \leq w, \end{aligned}$$

where the very last inequality holds because $\hat{\varphi}[d_{c_Y^*}^{\phi}]$ is feasible for $(\mathcal{T}, \mathcal{C}_{\mathtt{t}_1})$. So, $\check{\varphi}_{\mathcal{N}}$ satisfies $c_{\mathtt{t}_1}$. ◀

▶ **Proposition 20.** *Given $\mathcal{N}$ as above, the schedule $\check{\varphi}_{\mathcal{N}}$ satisfies every $c \in \mathcal{C}_{\mathtt{t}_2}$.*

Proof. Let $c_{\mathtt{t}_2} = \bigvee_{i=1}^{k} (l_i \leq X \leq u_i) \in \mathcal{C}_{\mathtt{t}_2}$ be any $\mathtt{t}_2$-constraint, some $X \in \mathcal{T}$, $l_i, u_i \in \mathbb{R}$. Pick any $c_X^* \in \arg\max_{c \in \mathcal{C}_{\mathtt{t}_3}} \hat{\varphi}[d_c^{\phi}](X)$. By definition $\hat{\varphi}[d_{c_X^*}^{\phi}]$ is a feasible schedule of $\mathcal{N}[d_{c_X^*}^{\phi}]_{\mathtt{t}_2}$, thus it is feasible for $(\mathcal{T}, \mathcal{C}_{\mathtt{t}_2})$ too. Therefore,

$$l_q \leq \hat{\varphi}[d_{c_X^*}^{\phi}](X) \leq u_q, \text{ for some } q \in \{1, \ldots, k\}.$$

Since, $\check{\varphi}_{\mathcal{N}}(X) = \varphi[d_{c_X^*}^{\phi}](X)$, then $\check{\varphi}_{\mathcal{N}}(X) \in [l_q, u_q]$ for the same q. So, $\check{\varphi}_{\mathcal{N}}$ satisfies $c_{\mathtt{t}_2}$. ◀

▶ **Proposition 21.** *Given $\mathcal{N}$ as above, the schedule $\check{\varphi}_{\mathcal{N}}$ satisfies every $c \in \mathcal{C}_{\mathtt{t}_3}$.*

Proof. Let $c_{\mathtt{t}_3} = (l_1 \leq X \leq u_1) \vee (l_2 \leq Y \leq u_2) \in \mathcal{C}_{\mathtt{t}_3}$ be any $\mathtt{t}_3$-constraint, some $X, Y \in \mathcal{T}$, $X < Y$ and $l_1, l_2, u_1, u_2 \in \mathbb{R}$. Assume w.l.o.g. $\phi(x_{c_{\mathtt{t}_3}}) = \mathtt{true}$. Then, $l_1 \leq \hat{\varphi}[d_{c_{\mathtt{t}_3}}^{\phi}](X) \leq u_1$.

If $c_{\mathtt{t}_3} \in \arg\max_{c \in \mathcal{C}_{\mathtt{t}_3}} \hat{\varphi}[d_c^{\phi}](X)$, then $\check{\varphi}_{\mathcal{N}}(X) = \hat{\varphi}[d_{c_{\mathtt{t}_3}}^{\phi}](X) \in [l_1, u_1]$; so, $\check{\varphi}_{\mathcal{N}}$ would satisfy $c_{\mathtt{t}_3}$. Otherwise, $c_{\mathtt{t}_3} \notin \arg\max_{c \in \mathcal{C}_{\mathtt{t}_3}} \hat{\varphi}[d_c^{\phi}](X)$, and assume $\check{\varphi}_{\mathcal{N}}(X) \notin [l_1, u_1]$ towards a contradiction. Pick any $c_X^* \in \arg\max_{c \in \mathcal{C}_{\mathtt{t}_3}} \hat{\varphi}[d_c^{\phi}](X)$. All these hypotheses combined:

$$\check{\varphi}_{\mathcal{N}}(X) = \hat{\varphi}[d_{c_X^*}^{\phi}](X) > u_1.$$

Therefore, ϕ must satisfy either $p \Rightarrow \neg x_{c_{\mathtt{t}_3}}$ or $\neg p \Rightarrow \neg x_{c_{\mathtt{t}_3}}$, for some boolean variable p (where the actual case depends on the actual value of $d_{c_X^*}^{\phi}$). Since ϕ satisfies either p or $\neg p$, then ϕ must satisfy $\neg x_{c_{\mathtt{t}_3}}$; i.e., $\phi(x_{c_{\mathtt{t}_3}}) = \mathtt{false}$. This is absurd, as we assumed $\phi(x_{c_{\mathtt{t}_3}}) = \mathtt{true}$.

The proof of the other case, in which $\phi(x_{c_{\mathtt{t}_3}}) = \mathtt{false}$ is initially assumed, is symmetric. So, $\check{\varphi}_{\mathcal{N}}(X)$ satisfies $c_{\mathtt{t}_3}$. ◀

Let us mention that our algorithm is called `RDTP()`, basically, it aims at computing $\hat{\varphi}_{\mathcal{N}}$ as above; if it fails in that (either because $\mathcal{N}_{\mathtt{t}_2}$ is already inconsistent or $\textsc{Cl}_{\mathcal{N}}$ is unsatisfiable), it recognizes the input RDTP $\mathcal{N}$ as inconsistent. Now, we can prove this is correct and fast.

▶ **Theorem 22.** *`RDTP()` is correct, i.e., on any RDTN $\mathcal{N} = (\mathcal{T}, \mathcal{C}_{t_1} \cup \mathcal{C}_{t_2} \cup \mathcal{C}_{t_3})$, it returns a feasible schedule $\hat{\varphi}_{\mathcal{N}} : \mathcal{T} \to \mathbb{R}$, if $\mathcal{N}$ is consistent; otherwise, $\mathcal{N}$ is recognized as inconsistent.*

Proof. Recall that $\mathcal{N}$ is recognized as inconsistent only if $(\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2})$ is already inconsistent or if the 2-SAT problem instance $\textsc{Cl}_{\mathcal{N}}$ is unsatisfiable. In the former case, since $(\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2})$ is inconsistent, so it is $\mathcal{N}$. In the latter, by construction of $\textsc{Cl}_{\mathcal{N}}$, it is not possible to satisfy all the constraints in $\mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2} \cup \mathcal{C}_{\mathtt{t}_3}$ (otherwise, the reader can check, it would've been possible to construct a satisfying assignment for $\textsc{Cl}_{\mathcal{N}}$, straightforwardly); so, $\mathcal{N}$ is really inconsistent.

On the other side, by Propositions 19, 20 and 21, schedule $\hat{\varphi}_{\mathcal{N}}$ is really feasible for $\mathcal{N}$. ◀

The next result asserts that the halting time is strongly polynomial in the input size.

▶ **Theorem 23.** *Let `RDTP()` run on any input RDTP $\mathcal{N} = (\mathcal{T}, \mathcal{C}_{t_1} \cup \mathcal{C}_{t_2} \cup C_3)$. Its always halts within time $O\big(|\mathcal{T}| \cdot |\mathcal{C}_{t_1}| + |\mathcal{C}_{t_2}| \cdot (|\mathcal{C}_{t_1}| + |\mathcal{T}| \cdot \log |\mathcal{T}|) + |\mathcal{T}| \cdot d_{\mathcal{C}_{t_2}} \cdot |\mathcal{C}_{t_3}| + |\mathcal{C}_{t_3}|^2\big)$.*

Proof. By Theorem 16, $(\mathcal{T}, \mathcal{C}_{\mathtt{t}_1} \cup \mathcal{C}_{\mathtt{t}_2})$ takes $O\big(|\mathcal{T}| \cdot |\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{C}_{\mathtt{t}_2}| \cdot (|\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{T}| \cdot \log |\mathcal{T}|) + |\mathcal{T}| \cdot d_{\mathcal{C}_{\mathtt{t}_2}}\big)$ time to be checked. Using that solution as an initial candidate, solving the two $\mathtt{t}_2$DTPs $\mathcal{N}[d'_c]_{\mathtt{t}_2}$ and $\mathcal{N}[d''_c]_{\mathtt{t}_2}$, for each $c \in \mathcal{C}_{\mathtt{t}_3}$ where $c = d'_c \vee d''_c$, it takes $O\big(|\mathcal{T}| \cdot |\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{C}_{\mathtt{t}_2}| \cdot (|\mathcal{C}_{\mathtt{t}_1}| + |\mathcal{T}| \cdot \log |\mathcal{T}|) + |\mathcal{T}| \cdot d_{\mathcal{C}_{\mathtt{t}_2}} \cdot |\mathcal{C}_{\mathtt{t}_3}|\big)$ total time. Next, for each $c, \tilde{c} \in \mathcal{C}_{\mathtt{t}_3}$ such that $\tilde{c} \neq c$, eventually adding the corresponding clauses to $\textsc{Cl}_{\mathcal{N}}$ takes $O(1)$ time per clause; so, $\textsc{Cl}_{\mathcal{N}}$ is built in total time $O(|\mathcal{C}_{\mathtt{t}_3}|^2)$. Since $|\textsc{Cl}_{\mathcal{N}}| = O(|\mathcal{C}_{\mathtt{t}_3}|^2)$, solving the 2-SAT problem on input $\textsc{Cl}_{\mathcal{N}}$ takes time $O(|\mathcal{C}_{\mathtt{t}_3}|^2)$ (e.g., with the algorithm of [1]). Finally, computing d^{ϕ}_c and $\check{\varphi}_{\mathcal{N}}$ takes $O(|\mathcal{T}| \cdot |\mathcal{C}_{\mathtt{t}_3}|)$ time. All in, the above mentioned time complexity of `RDTP()` follows. ◀

5 NP-completeness of Multi-Tail & Multi-Head $\mathtt{t}_3$HyTPs

This section enquiries the tractability frontier of RDTPs by considering HyTNs [3], where the basic idea is that of blending the two models together and see what happens to the complexity of the corresponding temporal problems. Two restricted kinds of disjunctive temporal problems, TAIL-$\mathtt{t}_3$HyTP and HEAD-$\mathtt{t}_3$HyTP, are both proven to be NP-complete. The former problem is that of deciding whether a multi-tail $\mathtt{t}_3$HyTN (i.e., a temporal network in which the constraints can be modeled only by multi-tail hyperarcs and by $\mathtt{t}_3$-constraints) is consistent or not. The latter, HEAD-$\mathtt{t}_3$HyTP, is the same as the former but considers multi-head hyperarcs instead. Let us now focus on TAIL-$\mathtt{t}_3$HyTP.

▶ **Theorem 24.** TAIL-$\mathtt{t}_3$HyTP *is NP-complete in a strong sense, i.e., even if the input $(\mathcal{T}, \mathcal{A} \cup \mathcal{C}_{\mathtt{t}_3})$ are restricted to satisfy $w_A(\cdot) \in \{-1, 0, 1\}$, $|T_A| \leq 2$ for every $A \in \mathcal{A}$, and every $\mathtt{t}_3$-constraint $(l_i \leq X \leq u_i) \vee (l_j \leq Y \leq u_j) \in \mathcal{C}_{\mathtt{t}_3}$ has all zero-valued lower/upper-bounds.*

Proof. We claim that if $\mathcal{H} = (\mathcal{T}, \mathcal{A} \cup \mathcal{C}_{\mathtt{t}_3})$ is an integer-weighted and consistent multi-tail $\mathtt{t}_3$HyTN, it admits an integer-valued feasible schedule $s : \mathcal{T} \to \{-T, \ldots, T\}$ where $T = \sum_{A \in \mathcal{A}, v \in V} |w_A(v)| + \sum_{c \in \mathcal{C}_{\mathtt{t}_3}, c = (l_1 \leq X \leq u_1) \vee (l_2 \leq Y \leq u_2)} (|l_1| + |u_1| + |l_2| + |u_2|)$. Indeed let s be a feasible schedule (integer-valued or not) of $\mathcal{H}$, and consider the projection HyTN $\mathcal{H}^s \triangleq (\mathcal{T}, \mathcal{A}')$, for $\mathcal{A}' \triangleq \mathcal{A} \cup \bigcup_{c \in \mathcal{C}_{\mathtt{t}_3}} A^s_c$, where for every $c = (l_1 \leq X \leq u_1) \vee (l_2 \leq Y \leq u_2) \in \mathcal{C}_{\mathtt{t}_3}$ we pick the following pair of $\mathtt{t}_1$-constraints:

$$A^s_c \triangleq \begin{cases} \big\{(Z - X \leq -l_1), (X - Z \leq u_1)\big\}, & \text{if } l_1 \leq s(X) \leq u_1; \\ \big\{(Z - Y \leq -l_2), (Y - Z \leq u_2)\big\}, & \text{otherwise.} \end{cases}$$

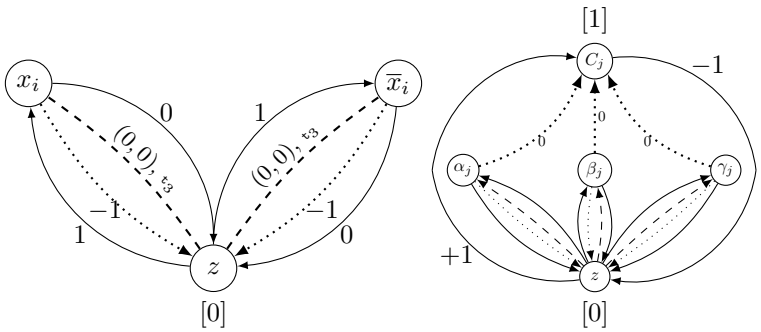


Figure 4 Variable and clause gadgets (at left and right, respectively) used in Theorem 24.

By construction of $\mathcal{H}^s$, s is a feasible for HyTN $\mathcal{H}^s$. So, by Proposition 7, $\mathcal{H}^s$ admits an integer-valued feasible schedule s' bounded by $-T$ and $+T$ as above. By contruction of $\mathcal{H}^s$, s' is feasible for $\mathcal{H}$ too.

Moreover, any such integer-valued feasible schedule can be verified in strongly polynomial time w.r.t. the size of the input; hence, TAIL-$\mathtt{T}_3$HYTP is in NP.

To show that the problem is NP-hard, we describe a reduction from 3-SAT.

Let us consider a boolean 3-CNF formula with $n \geq 1$ variables and $m \geq 1$ clauses:

$\varphi(x_1, \ldots, x_n) = \bigwedge_{i=1}^{m}(\alpha_i \vee \beta_i \vee \gamma_i)$, where $C_i = (\alpha_i \vee \beta_i \vee \gamma_i)$ is the i-th clause of φ and each $\alpha_i, \beta_i, \gamma_i \in \{x_j, \overline{x}_j \mid 1 \leq j \leq n\}$ is either a positive or a negative literal.

We associate to φ a multi-tail $\mathtt{t}_3$HyTN $\mathcal{H}_\varphi = (\mathcal{T}, \mathcal{A} \cup \mathcal{C}_{\mathtt{t}_3})$, where each boolean variable x_i occurring in φ gets represented by two time-points, x_i and $\overline{x}_i$. $\mathcal{T}$ also contains a time-point z that represents the reference initial time-point for $\mathcal{H}_\varphi$, i.e., the first time-point that has to be executed at time zero. Moreover, for each pair x_i and $\overline{x}_i$, $\mathcal{H}_\varphi$ contains:

a multi-tail hyperarc with tails $\{x_i, \overline{x}_i\}$, both weighted -1, and head in z.

a $\mathtt{t}_3$-constraint $\big((0 \leq x_i \leq 0) \vee (0 \leq \overline{x}_i \leq 0)\big) \in \mathcal{C}_{\mathtt{t}_3}$. If $\mathcal{H}_\varphi$ is consistent, the multi-tail hyperarc and the $\mathtt{t}_3$-constraint associated to $x, \neg x$ assures that $\mathcal{H}_\varphi$ admits an integer feasible schedule s (as we mentioned above) such that $s(x_i)$ and $s(\overline{x}_i)$ are coherently set with values in $\{0, 1\}$. In this way, s is forced to encode a truth assignment on the x_i's.

The HyTN $\mathcal{H}_\varphi$ contains also a time-point C_j for each clause C_j of φ; each C_j is connected by a multi-tail hyperarc with head in C_j and tails over the literals occurring in C_j and by two standard and opposite arcs with time-point z as displayed in Fig. 4 (right). This assures that if $\mathcal{H}_\varphi$ admits a feasible schedule s, then s assigns scheduling time 1 at least to one of the time-point representing the literals connected with the multi-tail hyperarc.

Fig. 4 depicts the gadgets. A more formal definition of $\mathcal{H}_\varphi$ is given in Appendix A.

The reader can check that $|\mathcal{T}| = 1 + 2n + m = O(m + n)$, $m_{\mathcal{A}} = O(m + n)$, $|\mathcal{C}_{\mathtt{t}_3}| = O(n)$; therefore, the transformation is linearly bounded.

We next show that φ is satisfiable if and only if $\mathcal{H}_\varphi$ is consistent.

Any truth assignment $\nu : \{x_1, \ldots, x_n\} \to \{\mathtt{true}, \mathtt{false}\}$ satisfying φ can be translated into a feasible schedule $s : \mathcal{T} \to \mathbb{Z}$ of $\mathcal{H}_\varphi$ as follows. For time-point z, let $s(z) = 0$, and let $s(C_j) = 1$ for each $j = 1, \ldots, m$; then, for each $i = 1, \ldots, n$, let $s(x_i) = 1$ and $s(\overline{x}_i) = 0$ if the truth value of x_i, $\nu(x_i)$, is $\mathtt{true}$, otherwise let $s(x_i) = 0$ and $s(\overline{x}_i) = 1$. It is simple to verify that, using this schedule s, all the constraints comprising each single gadget are satisfied and, therefore, the network is consistent. So, $\mathcal{H}_\varphi$ is consistent.

Vice versa, assume that $\mathcal{H}_\varphi$ is consistent. Then, it admits an integer-valued feasible schedule s (as we mentioned above). After the translation $s(v) \triangleq s(v) - s(z)$, we can assume that $s(z) = 0$. Hence, $s(C_j) = 1$ for each $j = 1, \ldots, m$, as enforced by the two standard

arcs incident at C_j in the clause gadget, and $\{s(x_i), s(\overline{x}_i)\} = \{0, 1\}$ for each $i = 1, \ldots, n$, as enforced by the constraints comprising the variable gadgets. Therefore, the feasible schedule s can be translated into a truth assignment $\nu : \{x_1, \ldots, x_n\} \to \{\texttt{true}, \texttt{false}\}$ defined by $\nu(x_i) = \texttt{true}$ if $s(x_i) = 1$ (and $s(\overline{x}_i) = 0$); $\nu(x_i) = \texttt{false}$ if $s(x_i) = 0$ (and $s(\overline{x}_i) = 1$) for every $i = 1, \ldots, n$. So, φ is satisfiable.

To conclude, we observe that any hyperarc $A \in \mathcal{A}$ of $\mathcal{H}_\varphi$ has weights $w_A(\cdot) \in \{-1, 0, 1\}$, size $|A| \leq 3$, and any $\texttt{t}_3$-constraint $c = (l_i \leq X \leq u_i) \vee (l_j \leq Y \leq u_j) \in \mathcal{C}_{\texttt{t}_3}$ has zero lower and upper-bounds (i.e., $l_i = u_i = l_j = u_j = 0$). Since any hyperarc with three tails can be replaced by two hyperarcs each having at most two tails, the consistency problem remains NP-Complete even if $|A| \leq 2$ for every $A \in A$. ◀

In order to prove that HEAD-$\texttt{T}_3$HyTP is also NP-complete, we could proceed with an argument similar to that of Theorem 24. However, we also observe that the same result follows as an immediate corollary of the following inter-reducibility between the two models.

▶ **Definition 25.** A multi-tail (multi-head) RHyTN is any temporal network in which the constraints can be modeled only by multi-tail (multi-head) hyperarcs and by $\{\texttt{t}_2, \texttt{t}_3\}$ disjunctive temporal constraints.

The problem of checking whether a given RHyTN is consistent is named RHyTP. Observe,

▶ **Proposition 26.** *Multi-head and multi-tail RHyTPs are inter-reducible by means of* log-*space, linear-time, local-replacement reductions. Particularly, multi-head and multi-tail* $\texttt{t}_3$*HyTPs are inter-reducible by such reductions. (The proof is in Appendix A)*

Therefore, by Proposition 26, it follows that HEAD-$\texttt{T}_3$HyTP is also strongly NP-complete.

6 Pseudo-Polynomial Time Algorithm for $\texttt{t}_2$HyTPs

We end by studying multi-tail and multi-head $\texttt{t}_2$HyTNs (i.e., temporal networks in which the temporal constraints can be only $\texttt{t}_2$ disjunctive temporal constraints and either only multi-tail or multi-head hyperarcs). It turns out that checking the corresponding temporal problems, TAIL-$\texttt{T}_2$HyTP and HEAD-$\texttt{T}_2$HyTP, lies in NP ∩ co-NP and admits pseudo-polynomial time algorithms. By Proposition 26, it is sufficient to focus on multi-head $\texttt{t}_2$HyTPs only. The corresponding pseudo-polynomial time algorithm is named $\texttt{t}_2\texttt{HyTP()}$, and described below – notice that it generalizes $\texttt{t}_2\texttt{DTP()}$. Given any integer-weighted multi-head $\texttt{t}_2$HyTPs $\mathcal{H}_{\texttt{t}_2} = (\mathcal{T}, \mathcal{A} \cup \mathcal{C}_{\texttt{t}_2})$ in input, we firstly solve the HyTP $\mathcal{H} = (\mathcal{T}, \mathcal{A})$ with the VI algorithm of Theorem 10. If $\mathcal{H}$ is recognized as inconsistent, the algorithm halts. Otherwise, let φ be the least feasible schedule of $\mathcal{H}$. Then proceed as follows:

While $\exists$ some $X \in \mathcal{T}$ and $c_X = \bigvee_{i=1}^{k}(l_i \leq X \leq u_i) \in \mathcal{C}_{\texttt{t}_2}$ s.t. $\varphi(X)$ doesn't satisfy c_X:

If $\varphi(X) > u_k (= \max_i u_i)$, then $\mathcal{H}_{\texttt{t}_2}$ is recognized as inconsistent; otherwise, let i^* be the smallest $i \in [1, k]$ such that $\varphi(X) < l_i$. Firstly, we increase the value of $\varphi(X)$ up to l_{i^*}, i.e., update $\varphi(X) \leftarrow l_{i^*}$. Secondly, the VI algorithm of Theorem 10 is invoked on input $(\mathcal{H}, \varphi)$, so, then, φ becomes the schedule returned by that run of VI. The process iterates so on and so forth, and if the while-loop completes without recognizing $\mathcal{H}_{\texttt{t}_2}$ as inconsistent, φ is returned. The correctness and the time complexity are asserted below. (The proof is in Appendix A)

▶ **Theorem 27.** $\texttt{t}_2\texttt{HyTP()}$ *is correct, i.e., running on any integer-weighted multi-head* $\texttt{t}_2$*HyTP* $\mathcal{H}_{\texttt{t}_2} = (\mathcal{T}, \mathcal{A} \cup \mathcal{C}_{\texttt{t}_2})$*, an integer-valued feasible schedule* $\varphi : \mathcal{T} \to \mathbb{Z}$ *is returned, in case* $\mathcal{H}_{\texttt{t}_2}$ *is consistent; otherwise,* $\mathcal{H}_{\texttt{t}_2}$ *is correctly recognized as inconsistent.*

Moreover, the corresponding time complexity is pseudo-polynomial, i.e.,

$$\mathit{Time}_{t_2\mathit{HyTP}()}(\mathcal{H}_{t_2}) = O\big((|\mathcal{T}| + |\mathcal{A}|) \cdot m_{\mathcal{A}} \cdot W_{\mathcal{A},\mathcal{C}_{t_2}}\big),$$
$$\textit{where } W_{\mathcal{A},\mathcal{C}_{t_2}} \triangleq \max\Big(\max_{A\in\mathcal{A}}\max_{h\in A}|w_A(h)|,\ \max_{\substack{l_j \textit{ appears in any} \\ \vee_{i=1}^{k}(l_i\leq X\leq u_i)\in\mathcal{C}_{t_2}}} l_j\Big).$$

Finally, since `t2HyTP()` is correct, it is possible to establish the following complexity result.

▶ **Theorem 28.** TAIL-$\mathtt{t}_2$HyTP, HEAD-$\mathtt{t}_2$HyTP $\in$ NP$\cap$*co*-NP. *(The proof is in Appendix A)*

7 Conclusions and Future Works

A deeper combinatorial comprehension on the algorithmics of RDTPs led to a new elementary deterministic strongly polynomial time procedure for solving them, significantly improving the asymptotic running times suggested by Kumar before. In future works we'd like to investigate further on possible generalizations/extensions of the proposed algorithms, aiming at covering some compatible (or even wider) subclasses of the disjunctive temporal constraints problem.

References

1 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

2 Richard Bellman. On a routing problem. *Quarterly of Applied Maths*, 16(1):87–90, 1958.

3 Carlo Comin, Roberto Posenato, and Romeo Rizzi. Hyper temporal networks - A tractable generalization of simple temporal networks and its relation to mean payoff games. *Constraints*, 22(2):152–190, 2017. doi:10.1007/s10601-016-9243-0.

4 Rina Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, CA, US, 2003.

5 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. doi:10.1016/0004-3702(91)90006-6.

6 Yves Deville, Olivier Barette, and Pascal Van Hentenryck. Constraint satisfaction over connected row-convex constraints. *Artificial Intelligence*, 109(1):243–271, 1999. doi:10.1016/S0004-3702(99)00012-0.

7 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec 1959. doi:10.1007/BF01386390.

8 Manolis Koubarakis. Chapter 19 - temporal csps. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of AI*, pages 665–697. Elsevier, 2006. doi:10.1016/S1574-6526(06)80023-4.

9 T. K. Sathish Kumar. Tractable classes of metric temporal problems with domain rules. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 847–852. AAAI Press, 2006.

10 T. K. Satish Kumar. On the tractability of restricted disjunctive temporal problems. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, pages 110–119, 2005.

11 T. K. Satish Kumar, Marcello Cirillo, and Sven Koenig. Simple temporal problems with taboo regions. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI'13, pages 548–554. AAAI Press, 2013.

12 Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann, San Francisco, CA, USA, 2004.

13 Angelo Oddi and Amedeo Cesta. Incremental forward checking for the disjunctive temporal problem. In *Proceedings of the 14th European Conference on Artificial Intelligence*, ECAI'00, pages 108–112, Amsterdam, The Netherlands, The Netherlands, 2000. IOS Press.

14 A.K. Pani and G.P. Bhattacharjee. Temporal representation and reasoning in artificial intelligence: A review. *Mathematical and Computer Modelling*, 34(1):55–80, 2001.

15 E. Schwalb and L. Vila. Temporal constraints: A survey. *Constraints*, 3(2):129–149, 1998.

16 Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.

17 Ioannis Tsamardinos and Martha E Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151(1):43–89, 2003.

A Appendix: Omitted Proofs

The appendix proceeds by offering additional missing proofs.

Proof of Proposition 26. We show the reduction from multi-tail to multi-head hypergraphs; the converse direction is symmetric. Informally, all the arcs are reversed (so that what was multi-tail becomes multi-head), and, contextually, the time-axis is inverted (to account for the inversion of the direction of all arcs). Finally, all $\mathtt{t}_2$ and $\mathtt{t}_3$-constraints are also reversed.

Given a multi-tail RHyTN $\mathcal{H} = (\mathcal{T}, \mathcal{A} \cup \mathcal{C}_{\mathtt{t}_2} \cup \mathcal{C}_{\mathtt{t}_3})$, we associate to $\mathcal{H}$ a multi-head RHyTN $\mathcal{H}' = (\mathcal{T}, \mathcal{A}' \cup \mathcal{C}'_2 \cup \mathcal{C}'_3)$ by reversing all multi-tail hyperarcs, all $\mathtt{t}_2$ and $\mathtt{t}_3$-constraints. Formally,

$$\mathcal{A}' \triangleq \Big\{(v, S, w) \mid (S, v, w) \in \mathcal{A}\Big\}, \quad \mathcal{C}'_2 \triangleq \Big\{ \bigvee_{i=1}^{k} (-u_i \leq X \leq -l_i) \mid \bigvee_{i=1}^{k} (l_i \leq X \leq u_i) \in \mathcal{C}_{\mathtt{t}_2} \Big\},$$
$$\mathcal{C}'_3 \triangleq \Big\{ \big((-u_1 \leq X \leq -l_1) \vee (-u_2 \leq Y \leq -l_2)\big) \mid \big((l_1 \leq X \leq u_1) \vee (l_2 \leq Y \leq u_2)\big) \in \mathcal{C}_{\mathtt{t}_3} \Big\}.$$

We claim that $\mathcal{H}$ is consistent if and only if $\mathcal{H}'$ is consistent. To prove it, we note that each schedule s for $\mathcal{H}$ can be associated, with a flip of the time direction, to the schedule $s' \triangleq -s$. Then, it holds that s is feasible for $\mathcal{H}$ if and only if s' is feasible for $\mathcal{H}'$. Indeed, s satisfies the constraint represented by an hyperarc $A = (T_A, h_A, w_A) \in \mathcal{A}$, i.e.,

$s(h_A) \leq \max_{v \in T_A}\{s(v) + w_A(v)\}$,

or, equivalently, $-s(h_A) \geq \min_{v \in T_A}\{-s(v) - w_A(v)\}$, if and only if s' (that is, $-s$) satisfies the constraint represented by the reversed hyperarc $A' = (h_A, T_A, w_A)$, i.e., if and only if:

$s'(h_A) \geq \min_{v \in T_A}\{s'(v) - w_{A'}(v)\}$.

Next, s satisfies a $\mathtt{t}_2$-constraint $\bigvee_{i=1}^{k}(l_i \leq X \leq u_i)$ *iff* $l_i \leq s(X) \leq u_i$ holds for some $i \in [1, k]$, or equivalently, *iff* $-u_i \leq -s(X) \leq -l_i$; this happens *iff* s' satisfies the constraint represented by the reversed disjunct $(-u_i \leq X \leq -l_i)$, i.e., *iff* $-u_i \leq s'(X) \leq -l_i$.

Finally, s satisfies a $\mathtt{t}_3$-constraint $((l_1 \leq X \leq u_1) \vee (l_2 \leq Y \leq u_2))$ *iff* either $l_1 \leq s(X) \leq u_1$ or $l_2 \leq s(Y) \leq u_2$, or equivalently, either $-u_1 \leq -s(X) \leq -l_1$ or $-u_2 \leq -s(Y) \leq -l_2$; this happens *iff* s' satisfies the constraint represented either by the reversed disjunct $(-u_1 \leq X \leq -l_1)$ or by $(-u_2 \leq Y \leq -l_2)$, i.e., *iff* either $-u_1 \leq s'(X) \leq -l_1$ or $-u_2 \leq s'(Y) \leq -l_2$. ◀

Formal definition of $\mathcal{H}_\varphi$ in the proof of Theorem 24. More formally, $\mathcal{H}_\varphi = (\mathcal{T}, \mathcal{A} \cup \mathcal{C}_{\mathtt{t}_3})$ is:

- $\mathcal{T} = \{z\} \cup \{x_i \mid 1 \leq i \leq n\} \cup \{\overline{x}_i \mid 1 \leq i \leq n\} \cup \{C_j \mid 1 \leq j \leq m\}$;
- $\mathcal{A} = \bigcup_{i=1}^{n} \mathrm{Var}_i \cup \bigcup_{j=1}^{m} \mathrm{Cla}_j$, where:

- $\mathrm{Var}_i = \Big\{(z, x_i, 1), (x_i, z, 0), (z, \overline{x}_i, 1), (\overline{x}_i, z, 0),$
  $\big(\{x_i, \overline{x}_i\}, z, [w(x_i), w(\overline{x}_i)] = [-1, -1]\big)\Big\}$.
  This is for the variable gadget of x_i as depicted in Fig. 4 (left);
- $\mathrm{Cla}_j = \Big\{(z, C_j, 1), (C_j, z, -1),$
  $(\{\alpha_j, \beta_j, \gamma_j\}, C_j, [w(\alpha_j), w(\beta_j), w(\gamma_j)] = [0, 0, 0])\Big\}$.
  This defines the clause gadget for clause $C_j = (\alpha_i \vee \beta_i \vee \gamma_i)$ as in Fig. 4 (right).

- $\mathcal{C}_{\mathtt{t}_3} = \bigcup_{i=1}^{n} \mathrm{Var}'_i$, where:
  - $\mathrm{Var}'_i = \big\{\big((0 \le x_i \le 0) \vee (0 \le \overline{x}_i \le 0)\big)\big\}$.
    This completes the variable gadget of x_i as depicted in Fig. 4 (left); ◀

Proof of head-$\mathtt{t}_2$HyTP, tail-$\mathtt{t}_2$HyTP $\in$ NP. We claim that if $\mathcal{H} = (\mathcal{T}, \mathcal{A} \cup \mathcal{C}_{\mathtt{t}_2})$ is an integer-weighted and consistent multi-tail $\mathtt{t}_2$HyTN, it admits an integer-valued feasible schedule $s : \mathcal{T} \to \{-T, \ldots, T\}$ where $T = \sum_{A \in \mathcal{A}, v \in V} |w_A(v)| + \sum_{c \in \mathcal{C}_{\mathtt{t}_2}, c = \bigvee_{i=1}^{k} (l_i \le X \le u_i)} (|l_i| + |u_i|)$. Indeed, let s be a feasible schedule (integer-valued or not) of $\mathcal{H}$, and consider the projection HyTN:

$$\mathcal{H}^s \triangleq (\mathcal{T} \cup \{z\}, \mathcal{A}'),$$
$$\mathcal{A}' \triangleq \mathcal{A} \cup \{(z - T \le 0) \mid T \in \mathcal{T}\} \cup \bigcup_{c \in \mathcal{C}_{\mathtt{t}_2}} A^s_c.$$

where for every $c = \bigvee_{i=1}^{k} (l_i \le X \le u_i) \in \mathcal{C}_{\mathtt{t}_2}$ this pair of $\mathtt{t}_1$-constraints is taken:

$$A^s_c \triangleq \big\{(z - X \le -l_i), (X - z \le u_i) \mid \text{ for the smallest } i \in \{1, \ldots, k\} \text{ s.t. } l_i \le s(X) \le u_i\big\}.$$

By construction of $\mathcal{H}^s$, s is a feasible for HyTN $\mathcal{H}^s$. So, by Proposition 7, $\mathcal{H}^s$ admits an integer-valued feasible schedule s' bounded as above. By construction of $\mathcal{H}^s$, s' is feasible for $\mathcal{H}$ too.

Any such integer-valued feasible schedule can be verified in strongly polynomial time w.r.t. the size of the input, simply by checking the actual consistency of each constraint in $\mathcal{A} \cup \mathcal{C}_{\mathtt{t}_2}$; hence, HEAD-$\mathtt{T}_2$HyTP is in NP. Thus, by Proposition 26, TAIL-$\mathtt{T}_2$HyTP $\in$ NP. ◀

▶ **Definition 29** (Hypercycle). We recall from [3] that a *hypercycle* $\mathcal{C}_0$ in a HyTN $\mathcal{H}$ is actually a pair $(S, \mathcal{C}_0)$ with $S \subseteq \mathcal{T}$ and $\mathcal{C}_0 \subseteq \mathcal{A}$ such that:
1. $S = \cup_{A \in \mathcal{C}_0} A$ and $S \neq \emptyset$;
2. $\forall v \in S$ there exists an unique $A \in \mathcal{C}_0$ such that $t_A = v$.
Every infinite path in a cycle $(S, \mathcal{C})$ contains, at least, one *finite cyclic sequence* $v_i, v_{i+1}, \ldots, v_{i+p}$, where $v_{i+p} = v_i$ is the only repeated node in the sequence. A cycle $(S, \mathcal{C}_0)$ is *negative* if for any finite cyclic sequence $v_1, v_2, \ldots, v_p$, it holds that $\sum_{t=1}^{p-1} w_{a(v_t)}(v_{t+1}) < 0$, where $a(v)$ denotes the unique arc $A \in \mathcal{C}_0$ with $t_A = v$ as required in previous item 2.

▶ **Definition 30** (Certified Least Feasible Schedule (CLFS)). Given any integer-weighted multi-head HyTN $\mathcal{H} = (\mathcal{T}, \mathcal{A})$, a *certified least feasible schedule (CLFS)* for $\mathcal{H}$ is a pair $\varphi_{\mathrm{cert}} \triangleq \{\varphi, \mathcal{F}\}$, where $\varphi : \mathcal{T} \to \mathbb{Z}$ is a feasible schedule of $\mathcal{H}$, and $\mathcal{F} \triangleq \{\mathcal{C}_X\}_{X_\mathcal{T}}$ is a family of hypercycles of $\mathcal{H}$ (which works as a certificate of minimality for φ, as follows): for every $X \in \mathcal{T}$, $\mathcal{C}_X$ is a negative hypercycle of the auxiliary HyTN $\mathcal{H}_X$ obtained from $\mathcal{H}$ just by adding one $\mathtt{t}_1$-constraint requiring X to be scheduled strictly before time $\varphi(X)$,

$$\mathcal{H}_X \triangleq \big(\mathcal{T} \cup \{z\}, \mathcal{A} \cup \{(z - T \le 0) \mid T \in \mathcal{T}\} \cup \{X - z \le \varphi(X) - 1\}\big).$$

φ_{cert} can be verified in strongly polynomial time, because negative hypercycles can be checked so (as shown e.g., in Lemma 3 of [3]) and feasiblity of φ can be checked by inspection. The

soundness of CLFSs follows from the proof argument of Proposition 7 (i.e., the idea in this proof is – cfr. Lemma 2 in [3] – to project the feasible HyTN over a conservative graph and then, in that setting, to exploit the integrality properties of potentials as prescribed e.g., by the Bellman-Ford algorithm) plus the fact that the universe of feasible schedules of any given multi-head HyTN is closed under pointwise-minimum, i.e., given two feasible schedules s_1, s_2, the schedule $s(u) \triangleq \min(s_1(u), s_2(u))\ \forall u \in \mathcal{T}$, is still feasible (also notice that, in multi-tail HyTNs, the pointwise-maximum works instead).

With Definitions 29 and 30 in mind, we can proceed with the following proof.

Proof of head-$\mathtt{t}_2$HyTP, tail-$\mathtt{t}_2$HyTP $\in$ co-NP. To show HEAD-$\mathtt{T}_2$HyTP $\in$ co-NP, we shall exhibit certificates (of inconsistent networks) which we can verify in strongly polynomial time.

The basic idea, in order to construct such certificates, is to consider what happens during the execution of algorithm `t2HyTP()`, assuming the input instance $\mathcal{H} = (\mathcal{T}, \mathcal{A} \cup \mathcal{C}_{\mathtt{t}_2})$ is inconsistent. If the HyTN $\mathcal{H}_0 \triangleq (\mathcal{T}, \mathcal{A})$ is already inconsistent, then it admits a negative hypercycle $\mathcal{C}_0$ (see e.g., Theorem 4 in [3]). Moreover, $\mathcal{C}_0$ can be checked in strongly polynomial time (see e.g., Lemma 3 in [3]), so $\mathcal{C}_0$ is already a valid certificate of inconsistency. Otherwise, let φ_0 be a CLFS of $\mathcal{H}_0$, then there must exist some $X_0 \in \mathcal{T}$ and $c_{X_0} = \bigvee_{i=1}^{k}(l_i \leq X_0 \leq u_i) \in \mathcal{C}_{\mathtt{t}_2}$ s.t. $\varphi_0(X_0)$ doesn't satisfy c_{X_0}. If $\varphi_0(X_0) > u_k(= \max_i u_i)$, then (φ_0, c_{X_0}) is a valid certificate of inconsistency; otherwise, let i_0^* be the smallest $i \in [1, k]$ such that $\varphi_0(X_0) < l_i$. Let $\mathcal{H}_1$ be the auxiliary HyTN obtained from $\mathcal{H}_0$ just by adding one $\mathtt{t}_1$-constraint requiring X_0 to be scheduled at or after time $l_{i_0^*}$:

$$\mathcal{H}_1 \triangleq \big(\mathcal{T} \cup \{z\}, \mathcal{A} \cup \{(z - T \leq 0) \mid T \in \mathcal{T}\} \cup \{z - X_0 \leq -l_{i_0^*}\}\big).$$

Then, let φ_1 be a CLFS of $\mathcal{H}_1$ (notice φ_1 exists because $\mathcal{H}_0$ was assumed to be feasible). Again, there must exist some $X_1 \in \mathcal{T}$ and $c_{X_1} = \bigvee_{i=1}^{k}(l_i \leq X_1 \leq u_i) \in \mathcal{C}_{\mathtt{t}_2}$ s.t. $\varphi_1(X_1)$ doesn't satisfy c_{X_1}. Again, if $\varphi_1(X_1) > u_k(= \max_i u_i)$, then $(\varphi_0, c_{X_0}, \varphi_1, c_{X_1})$ is a valid certificate of inconsistency; otherwise, we can construct yet another auxiliary HyTN $\mathcal{H}_2$ by adding one $\mathtt{t}_1$-constraint requiring X_1 to be scheduled at or after time $l_{i_1^*}$, for appropriate i_1^* defined similarly as before. The construction iterates inductively and, generally, it leads to a sequence of the following kind (where the $\{\varphi_i\}_{i=0}^{N}$ are all CLFSs of the auxiliary HyTNs):

$$\Big((\varphi_0, c_{X_0}), (\varphi_1, c_{X_1}), \ldots, (\varphi_N, c_{X_N})\Big),$$

where, notice, it's length is at most $N \leq d_{\mathcal{C}_{\mathtt{t}_2}}$ (because, for each iteration of the construction, one disjunct of some $\mathtt{t}_2$-constraint is ruled out forever). Each element of the sequence can be verified in strongly polynomial time, and the length of the same sequence is strongly polynomial; plus, Theorem 27 implies the correctness of these certificates. This proves that HEAD-$\mathtt{T}_2$HyTP $\in$ co-NP. Thus, by Proposition 26, TAIL-$\mathtt{T}_2$HyTP $\in$ co-NP too. ◀

Proof of Theorem 27. The correctness argument is similar to that for proving correctness of `t2DTP()`, the details are simpler in this case because the only algorithm that is used to update the schedule φ monotonically is the VI algorithm of Theorem 10 (instead of Bellman-Ford and multiple calls to Dijkstra as it was for `t2DTP()`); indeed, the VI algorithm of Theorem 10 also provides the *least* feasible schedule in case the input HyTN is consistent, thus a similar (actually simpler) correctness argument still holds. Also the time complexity of `t2HyTP()` is a direct consequence of the complexity of the VI algorithm of Theorem 10, where the maximal weight measure W is increased to $W_{\mathcal{A},\mathcal{C}_{\mathtt{t}_2}}$ (as defined above) in order to take into account the lower-bound constraints (i.e., those of type $(z - X \leq -l_{i^*})$) that are (implicitly) introduced in $\mathcal{A}$ during the main while-loop; notice that, during the computation, the VI algorithm

is invoked on input $(\mathcal{H}, \varphi)$ so that at each iteration the scheduling values are initialized to those of the previous iteration (this ensures that they are always updated monotonically upwards during the whole computation, thus amortizing the total cost among all iterations). Plus, at each iteration at least one scheduling value is increased (i.e., $\varphi(X)$ is increased to l_{i^*} to satisfy the last $(z - X \leq -l_{i^*})$). Finally, checking the while-loop's condition takes time $O(d_{\mathcal{C}_{\mathtt{t}_2}} \cdot |\mathcal{T}|)$ total time, and since $d_{\mathcal{C}_{\mathtt{t}_2}} \leq |\mathcal{T}| \cdot W_{\mathcal{A},\mathcal{C}_{\mathtt{t}_2}}$, then $O(d_{\mathcal{C}_{\mathtt{t}_2}} \cdot |\mathcal{T}|) = O(|\mathcal{T}|^2 \cdot W_{\mathcal{A},\mathcal{C}_{\mathtt{t}_2}})$ (which, notice, it is not a bottleneck asymptotically). So, the $\mathtt{Time}_{\mathtt{t_2HyTP()}}$ bound holds. ◀

Algebraic Operators for Processing Sets of Temporal Intervals in Relational Databases

Andreas Dohr
University of Bonn, Institute of Computer Science III, Germany
andreas.dohr@uni-bonn.de

Christiane Engels
University of Bonn, Institute of Computer Science III, Germany
engelsc@cs.uni-bonn.de

Andreas Behrend
University of Bonn, Institute of Computer Science III, Germany
behrend@cs.uni-bonn.de

Abstract

The efficient management of temporal data has become increasingly important for many database applications. Most commercial systems already allow the management of temporal data but the operational support for processing this data is still rather limited. One particular reason is that many extension proposals typically require considerable modifications of the underlying database engine. In this paper, we propose a lightweight solution where temporal operators are realized using a library of user-defined functions. This way the complexity of temporal queries can be drastically reduced leading to more readable and less error-prone code without touching the database system. Our experiments show that the proposed operators significantly outperform temporal queries formulated in pure SQL. In addition, we investigate the possibility to incorporate algebraic optimization strategies directly into our operator definitions which allow for further performance improvements.

2012 ACM Subject Classification Information systems → Database management system engines, Information systems → Information systems applications

Keywords and phrases Temporal Databases, Relational Operators, Situation Calculus

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.11

1 Introduction

The support for temporal data is continuously extended by almost every commercial database system reflecting its importance for many practical applications. Today's systems typically provide an automated version control of the data and mechanisms for fast history access known as time travel. However, many temporal operations are still implemented within the application layer because database systems offer no comprehensive support for efficiently performing operations over temporal data, yet. In particular, the fact that temporal data is usually represented by means of temporal intervals rather than single time points is usually neglected and no specific data type like period nor operator support is provided in standard SQL. Consequently, the task of processing sets of temporal intervals has to be realized by complex low-level SQL queries or is simply left to tools or applications outside the database.

For querying temporal intervals, the sequenced semantics seems to be the most natural choice [6]. Formulating sequenced statements over this data in standard SQL, however, often leads to very complex expressions [9] which are difficult to maintain and can hardly be optimized by the database system.

As an example, we consider an air traffic scenario in which various events (e.g. changes of altitude, speed, and heading of an aircraft) are continuously monitored and stored in a database. If we are interested in the time interval during which a plane was in cruising mode but did not perform a maneuver, various subintervals have to be computed as a cruising period may coincide with several maneuver periods. The inherent complexity of the reasoning becomes even more apparent as soon as more than two conditions are compared this way. This is due to the fact that all possible sequences of time points have to be covered by the respective database query as the systems basically support an event-based calculus, only. Still missing is the possibility to generate sets of input intervals which are processed by set-oriented operators within the FROM clause. To this end, TABLE functions can be used which allow for modeling the required many-to-many mapping. As an example, the following query detects cruising phases without maneuvering for flights operated by Lufthansa using the table function TDIFF:

```
SELECT Start,End FROM TABLE (TDIFF(vw_LHcruising,vw_LHmaneuvering)) Period;
```

The complexity of the query is drastically reduced as all time point comparisons are hidden in the table function which accesses two views containing cruising and maneuvering phases of Lufthansa flights, respectively.

In this paper, we discuss the development of a library of such temporal operators which form the basis for realizing a situation calculus inside a temporal database system. In particular, our contributions are as follows:

- We introduce formal definitions of the most important operators like TUNION, TDIFF, TSECT, THULL, TCOMPL, or TCROP in a systematic way.
- We show the possibility of optimizing sequences of temporal operators similar to algebraic optimization rules for relational expressions.
- We provide an evaluation of various implementation alternatives for temporal operators showing the feasibility of our proposal.

The paper is organized as follows: After discussing related work in Section 2 we specify a set of temporal operators and corresponding algebraic optimization rules in Sections 3 and 4. Different implementations of two selected temporal operators are discussed in Section 5 and evaluated in Section 6. Finally, we draw a conclusions in Section 7.

2 Related Work

Our work is motivated by the general idea of supporting situation awareness inside a database system [13]. The workspace manager of Oracle provides the well-known relationship operators proposed by Allen [2] which can be employed for this purpose. When querying a history of situations, however, the comparison of sets of intervals is often needed, and Allen's operators are not suited anymore [4]. The lack of database support for monitoring the temporal development of data led to specialized systems such as moving object, data stream or complex event processing (CEP) systems. Especially CEP appeared to be well-suited for handling the dynamics of an application. While the employed event-based calculus is useful for general temporal reasoning over discrete changes, it is not well-suited for modeling the continuous changes between domain-specific situations represented as anchored temporal intervals. Even CEP systems with a richer data model for temporal abstractions such as Microsoft's StreamInsight [1], Naiad [17] or SEEP [12] are limited with respect to processing sets of temporal intervals.

As an alternative approach, operational support for processing sets of temporal intervals has been proposed in [9], [10]. In these works the authors propose to reduce a temporal operator to its nontemporal counterpart by using two primitives, the *temporal normalizer*

for group based operators (e.g. projection, aggregation, union) and the *temporal aligner* for tuple based operators (product, joins). This represents an elegant generic approach and even provides the possibility to use the DBMS optimizer for the resulting nontemporal expressions. However, rewriting temporal operators this way leads to unspecialized versions which perform mostly not as efficient as the more specialized table functions proposed in this work. In addition, several extensions to a relational DBMS kernel have to be implemented in order to use these temporal features which is problematic for proprietary DBMS. No support for temporal operations in non-sequenced semantics is provided.

3 Temporal Operators

In this section we present temporal operators for processing sets of temporal intervals which have been already mentioned in [4]. We extend this work by providing a formal basis for the specification of these operators from which suitable implementations are derived in Section 5. We first define temporal operators working on sets of intervals in Section 3.1. Afterwards, these operators are used to specify temporal operators for period-stamped temporal relations in Section 3.2. All our proposed operators act on table level, meaning that they have table-valued in- and/or output. For illustrating our approach, we exemplarily define

- three point-based binary operators TUNION, TDIFF, and TSECT,
- two point-based unary operators THULL and TCOMPL,
- four interval-based unary operators TMIN, TMAX, TFIRST, and TLAST,
- and a binary interval-based operator TCROP with its point-based analog TCROPC.

3.1 Interval Operators

In the following we use half-open time intervals $I = [I.s, I.e)$ including the start point $I.s$ and excluding the end point $I.e$ of interval I. We denote the interval length by $I.\ell := I.e - I.s$ for $I.e \neq \infty$ and $I.\ell := \infty$ otherwise. For the definition of the point-based operators we use *coalescing* of a set of temporal intervals $\mathcal{I}$ specified as follows:

$$Coal(\mathcal{I}) := \{[I_1.s, I_k.e) \mid \exists \text{ sequence } I_1 < \ldots < I_k \in \mathcal{I} \text{ s.t. } I_i.e \in I_{i+1}, 1 \leq i \leq k-1, \\ \text{and } \nexists\, I \in \mathcal{I} \text{ s.t. } I_1.s \in \text{int}(I) \text{ or } I_k.e \in I\},$$

where $I_1 < I_2 :\Leftrightarrow I_1.s < I_2.s \wedge I_1.e < I_2.e$ and $\text{int}(I) := (I.s, I.e)$ denotes the interior of interval I. Because of the first condition we only coalesce intervals that are either overlapping or touching. The second condition ensures that the sequence and therefore the resulting interval cannot be further extended. An example is shown in Figure 1. Note that this definition of coalescing is equivalent to the standard definition of temporal coalescing given for example in [7].

For the binary operators union, difference and intersection we have chosen a point-based rather than interval-based model (cf. [5]). By using coalescing, we represent the required timestamps (i.e. minimum requirements) with a maximal interval partition.

▶ **Definition 1** (Operators TUNION, TDIFF, and TSECT)**.** Let $\mathcal{I}$ and $\mathcal{I}'$ be finite sets of right-open intervals. Then the temporal union of the two interval sets $\mathcal{I}$ and $\mathcal{I}'$ is defined as $\text{TUNION}(\mathcal{I}, \mathcal{I}') := Coal(\mathcal{I} \cup \mathcal{I}')$. Their temporal difference is

$$\text{TDIFF}(\mathcal{I}, \mathcal{I}') := Coal(\{J \subseteq I \mid I \in \mathcal{I}, \forall I' \in \mathcal{I}' : J \cap I' = \emptyset, \\ \text{and J is maximal with this property}\}),$$

and the temporal intersection is given by $\text{TSECT}(\mathcal{I}, \mathcal{I}') := Coal(\{I \cap I' \mid I \in \mathcal{I}, I' \in \mathcal{I}'\})$.

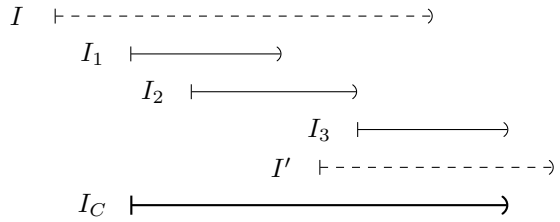


Figure 1 Example of the coalescing operator, $I_C = Coal(\{I_1, I_2, I_3\})$. If there were intervals I or I', the sequences $I < I_3$ and $I_1 < I_2 < I'$ respectively lead to an extended coalesced interval I_C.

Temporal union is defined as coalescing of the input interval sets whereas temporal difference comprises all subintervals of $\mathcal{I}$ not covered by $\mathcal{I}'$ in coalesced form. Similarly, temporal intersection is defined as the coalesced common time periods of both sets. Note that if both input sets are coalesced, i.e. $Coal(\mathcal{I}) = \mathcal{I}$ and $Coal(\mathcal{I}') = \mathcal{I}'$, then $\text{TSECT}(\mathcal{I}, \mathcal{I}') = \{I \cap I' \,|\, I \in \mathcal{I}, I' \in \mathcal{I}'\}$, and if $\mathcal{I}$ is coalesced, then $\text{TDIFF}(\mathcal{I}, \mathcal{I}') = \{J \subseteq I \,|\, I \in \mathcal{I}, \forall I' \in \mathcal{I}' : J \cap I' = \emptyset$, and J is maximal with this property$\}$ [7]. As a last example of point-based operators, the two unary operators THULL and TCOMPL are defined. A sample application of all introduced point-based operators is depicted in Figure 2.

▶ **Definition 2** (Operators THULL and TCOMPL)**.** We define the hull of a finite set of right-open intervals $\mathcal{I}$ as the time period enclosed by the given intervals $\text{THULL}(\mathcal{I}) := [\min_{I \in \mathcal{I}} I.s, \max_{I \in \mathcal{I}} I.e)$. The complement of $\mathcal{I}$ with respect to its hull is given by $\text{TCOMPL}(\mathcal{I}) := \text{TDIFF}(\{\text{THULL}(\mathcal{I})\}, \mathcal{I})$.

The list of operators is complemented by four interval-based operators which return the shortest and longest as well as the first and last set of intervals from $\mathcal{I}$, respectively.

▶ **Definition 3** (Operators TMIN and TMAX)**.** For a finite set of right-open intervals $\mathcal{I}$, we define the operators TMIN and TMAX returning the shortest respectively longest intervals of $\mathcal{I}$ as $\text{TMIN}(\mathcal{I}) := \arg\min_{I \in \mathcal{I}} I.\ell$ and $\text{TMAX}(\mathcal{I}) := \arg\max_{I \in \mathcal{I}} I.\ell$.

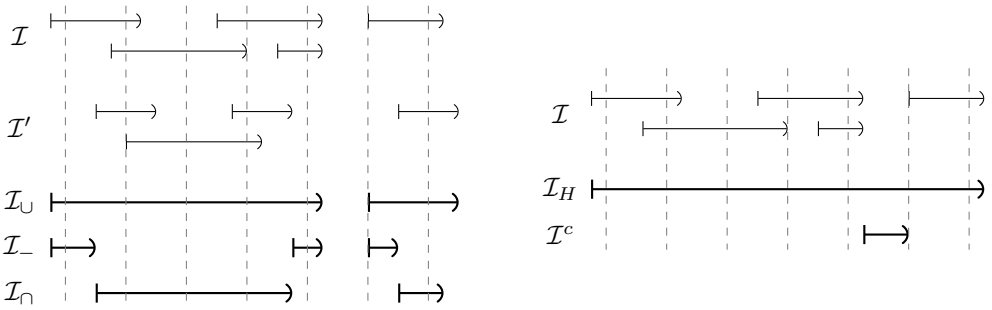
▶ **Definition 4** (Operators TFIRST and TLAST)**.** For a finite set of right-open intervals $\mathcal{I}$, we define the operators TFIRST and TLAST returning the intervals of $\mathcal{I}$ having the earliest start (latest end) point as $\text{TFIRST}(\mathcal{I}) := \arg\min_{I \in \mathcal{I}} I.s$ and $\text{TLAST}(\mathcal{I}) := \arg\max_{I \in \mathcal{I}} I.e$.

Finally, the TCROP operator crops the input intervals with respect to a given interval CR. This operator is especially useful for viewing a snapshot of some relation, i.e. all Lufthansa flights performed today. TCROPC is its coalesced analog.

▶ **Definition 5** (TCROP and TCROPC)**.** For a finite set of right-open intervals $\mathcal{I}$ and a non-empty, right-open interval CR we define $\text{TCROP}(\mathcal{I}, CR) := \{I \cap CR \,|\, I \in \mathcal{I}\}$ and $\text{TCROPC}(\mathcal{I}, CR) := Coal(\{I \cap CR \,|\, I \in \mathcal{I}\})$.

3.2 Operators for Period-Stamped Relations

In a temporal database tuple, temporal intervals are typically associated with other 'non-temporal' attribute values which have to be processed, too. Therefore, the introduced operators are extended to temporal relations with explicit, non-temporal attributes A and timestamp T. We consider $\pi_{B,T}(R)$ for $B \subseteq A$ a subset of the non-temporal attributes of the relation R. The proposed operators will act on all value equivalent tuples of $\pi_{B,T}(R)$, i.e.



(a) TUNION $\mathcal{I}_\cup$, TDIFF $\mathcal{I}_-$, and TSECT $\mathcal{I}_\cap$ **(b)** THULL $\mathcal{I}_H$ and TCOMPL $\mathcal{I}^c$

Figure 2 Sample applications of point-based operators for sets of right-open intervals $\mathcal{I}$ and $\mathcal{I}'$.

performing temporal aggregation with respect to B. Note that for $B = \emptyset$ the operations are directly performed on the temporal intervals and for $B = A$ value equivalence on relational level is considered. We denote by $R_T^{\bar{b}} := \pi_T(\sigma_{B=\bar{b}}(R))$ the timestamps of all value equivalent tuples w.r.t. B with value $\bar{b}$. First, we define the extended coalescing operator w.r.t $B \subseteq A$:

$$Coal(R, B) := \left\{(\bar{b}, t) \,|\, \bar{b} \in \pi_B(R), t \in Coal(R_T^{\bar{b}})\right\}.$$

The three point-based operators TUNION, TDIFF, and TSECT for sets of intervals are extended in the following to relational operators having two period-stamped relations and a subset $B \subseteq A$ of the non-temporal attributes as input:

▶ **Definition 6.** For two period-stamped relations R and S with non-temporal attributes A, timestamp T, and $B \subseteq A$ we set

$$\text{TUNION}(R, S, B) := \left\{(\bar{b}, t) \,|\, \bar{b} \in \pi_B(R \cup S), t \in \text{TUNION}(R_T^{\bar{b}}, S_T^{\bar{b}})\right\},$$

$$\text{TDIFF}(R, S, B) := \left\{(\bar{b}, t) \,|\, \bar{b} \in \pi_B(R), t \in \text{TDIFF}(R_T^{\bar{b}}, S_T^{\bar{b}})\right\},$$

$$\text{TSECT}(R, S, B) := \left\{(\bar{b}, t) \,|\, \bar{b} \in \pi_B(R) \cap \pi_B(S), t \in \text{TSECT}(R_T^{\bar{b}}, S_T^{\bar{b}})\right\}.$$

The remaining unary operators on relation level can be directly specified via their interval equivalent:

▶ **Definition 7.** For two period-stamped relations R and S with non-temporal attributes A, timestamp T, and $B \subseteq A$ we set $\text{TOP}(R, B) := \left\{(\bar{b}, t) \,|\, \bar{b} \in \pi_B(R), t \in \text{TOP}(R_T^{\bar{b}})\right\}$ with TOP$\in$ {THULL, TCOMPL, TMIN, TMAX, TFIRST, TLAST}.

Similarly, the operators TCROP and TCROPC are extended:

▶ **Definition 8.** For a period-stamped relation R with non-temporal attributes A, timestamp T, $B \subseteq A$, CR a non-empty right-open interval, and TOP $\in$ {TCROP, TCROPC} we set $\text{TOP}(R, B, CR) := \left\{(\bar{b}, t) \,|\, \bar{b} \in \pi_B(R), t \in \text{TOP}(R_T^{\bar{b}}, CR)\right\}$.

4 Algebraic Optimization

Similar to the non-temporal relational operators, algebraic optimization can be performed for our temporal operators. Let's consider the following example. We are interested in all flight phases excluding landings performed yesterday by a Southwest Airlines' aircraft. The following SQL query employing the TDIFF- and TCROP-operators can be employed:

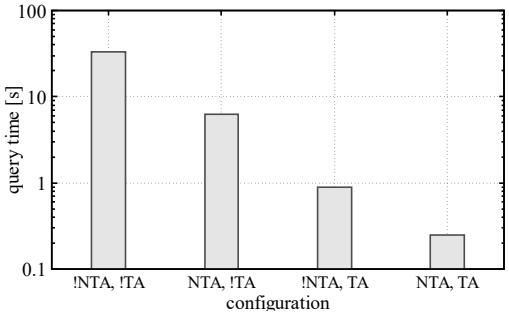


Figure 3 Performance of selection pushing for non-temporal attributes (NTA) and the timestamp attribute (TA).

```
SELECT aircraft, T FROM TABLE
  TCROP(TDIFF(flight,landing,{aircraft,airline})),
       {aircraft,airline}, yesterday) flight_phases
WHERE airline = 'Southwest Airlines'
```

Before calculating the temporal difference it makes sense to evaluate the selection to Southwest Airlines, i.e. to push the selection. Likewise we might also push the condition that the flight phase was performed yesterday. As this refers to the timestamp caution is advised. Pushing the temporal selection in the example using around 1.7 million entries from the dataset "Airline On-Time Statistics and Delay Causes" [18] considerably reduces the execution time as depicted in Figure 3. We focus in the following on the formal basis of these kinds of optimizations and introduce selection pushing as algebraic optimization rule for our temporal operators.

In [7] the following so called *conditional coalescing rule* states that selections σ^T can be pushed through the coalescing operator acting on value equivalent tuples (i.e. $Coal(R, A)$ for $A := attr(R)$ in our terminology) if the selection condition c does not contain any temporal attribute: $\sigma_c^T(Coal(R, A)) \equiv Coal(\sigma_c^T(R, A))$. This rule naturally extends to $B \subseteq A$ provided that the selection condition refers to attributes in B only. As our operators are based on coalescing and/or act on value equivalent tuples w.r.t. B this allows us to push selections – still provided that the selection condition refers to attributes in B only. Regarding pushing of selections referring to the timestamp T, we studied three kinds of selection criteria:

1. A restriction on the start point of the timestamp, i.e. T.s ∈ Q for a given right-open interval Q.
2. A restriction on the end point of the timestamp, i.e. T.e ∈ Q' for a given right-open interval Q'.
3. A restriction on the length of the timestamp, i.e. T.ℓ ∈ D for a given right-open interval D.

Furthermore, we distinguished whether the input relations of the operators are in coalesced form and whether the output relation will be cropped to an interval CR. In the following we will focus on the three point-based binary operators TUNION, TDIFF and TSECT.

The example in Figure 4 illustrates that due to the coalescing applied in our temporal operators we cannot simply push selections like those specified above through the operators. Given a set of right-open intervals $\mathcal{I}$ we want to select all intervals of $Coal(\mathcal{I})$ starting in Q. Naively selecting all intervals of $\mathcal{I}$ starting in Q as depicted in $\mathcal{S}$tart does not lead to the solution $\mathcal{R}$es. This is because two of the intervals will be merged during coalescing with an interval starting before Q violating the selection condition, and the other interval will

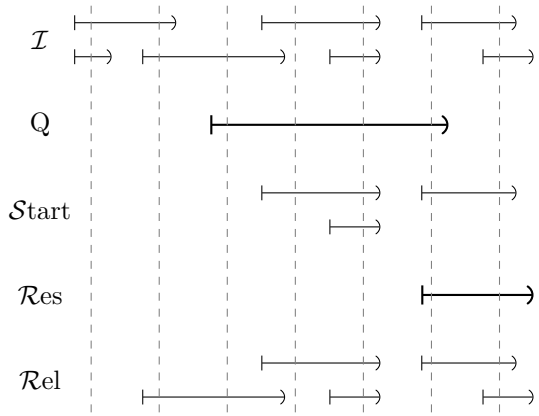


Figure 4 Example of selection pushing for temporal operators. $\mathcal{S}$tart comprises all intervals of $\mathcal{I}$ starting in Q, $\mathcal{R}$es contains the result set of $\sigma_{I.s \in Q}(Coal(\mathcal{I}))$, and $\mathcal{R}$el comprises the intervals of $\mathcal{I}$ relevant to compute $\mathcal{R}$es.

be extended during coalescing with an interval starting after Q, so that only the modified interval is contained in the result set $\mathcal{R}$es. But the first two intervals of $\mathcal{I}$ are not relevant at all to compute $\mathcal{R}$es because they end before Q starts. $\mathcal{R}$el comprises the intervals of $\mathcal{I}$ relevant to compute $\mathcal{R}$es.

Generalizing the idea of intervals relevant to compute the result of a selection referring to the timestamp, we cannot push the selection through the temporal operator, but we can introduce selections applied before the temporal operator on the basis of the selection condition in order to reduce the input size of the operator. For the question which intervals are starting during a specified interval Q over an uncoalesced input only those intervals are relevant that

- start in Q and are therefore possibly part of an interval of the result set,
- end in Q (or later) and may extend intervals starting in Q to violate the selection condition,
- or start after Q and may extend intervals starting in Q such that the extended, coalesced interval is part of the result set.

In particular, intervals ending before Q, i.e. satisfying I.e $<$ Q.s, are not relevant and can be discarded. This is true for all three operators TUNION, TDIFF, and TSECT giving the following rule:

In the case of uncoalesced input relations with selection condition $T.s \in Q$, a selection $\sigma_{T.e \geq Q.s}$ may be introduced and pushed through the temporal operator.

For TUNION, it does not make any difference whether the input relations are coalesced or not, as coalescing is involved anyway. But for TDIFF and TSECT, we can refine the above rule in cases with coalesced input relations as no further coalescing has to be performed (cf. Section 3.1). So if R is in coalesced form, we can introduce and push a stronger selection condition for R through TDIFF(R,S,B), namely $\sigma_{T \cap Q \neq \emptyset}$. Similarly, if R or S is coalesced, we can introduce and push the same selection condition for R or S respectively through TSECT(R,S,B).

If the output is cropped to an interval CR we can further reduce the input relations. Again considering the case with uncoalesced input first, we can introduce and push a new selection $\sigma_{T.s < CR.e}$. Everything happening after CR.e is irrelevant if the resulting intervals are cropped to CR even though there were intervals starting in Q. In addition, we can adjust

Table 1 Introduced selection conditions in the different settings for the operators TUNION, TDIFF, and TSECT. *Coalesced* only refers to relation R in TDIFF(R,S,B) and relations R and S in TSECT(R,S,B).

Setting	$T.s \in Q$	$T.e \in Q'$
uncoalesced	$\sigma_{T.e \geq Q.s}$	$\sigma_{T.s \leq Q'.e}$
coalesced	$\sigma_{T \cap Q \neq \emptyset}$	$\sigma_{T \cap Q' \neq \emptyset}$
cropped to CR	$\sigma_{T.e \geq Q.s}$	$\sigma_{T.s \leq Q'.e}$
+ uncoalesced	$\sigma_{T.s < CR.e}$	$\sigma_{T.e > CR.s}$
cropped to CR	$\sigma_{T \cap Q \neq \emptyset}$	$\sigma_{T \cap Q' \neq \emptyset}$
+ coalesced	$\sigma_{T \cap CR \neq \emptyset}$	$\sigma_{T \cap CR \neq \emptyset}$

the query interval Q and the crop interval CR according to the following formulas

$$CR^*.s := max(CR.s, Q.s) \quad \text{and} \quad Q^*.e := min(CR.e, Q.e)$$

as all resulting intervals start in Q and the output is cropped to CR. In the coalesced case we can again push a stronger selection condition, namely $\sigma_{T \cap CR \neq \emptyset}$ for coalesced R in TDIFF(R,S,B) and coalesced R or S in TSECT(R,S,B).

A summary of the introduced selection conditions in the different settings for $T.s \in Q$ – as well as those for $T.e \in Q'$ which are exactly inverted – is given in Table 1.

For queries referring to the timestamp length, i.e. $T.\ell \in D$, we can push selections only in the coalesced cases of TSECT and R in TDIFF(R,S,B). In these cases the input intervals can only get shorter when the temporal operator is applied. So only intervals longer than D.s are relevant and the condition $\sigma_{T.\ell \geq D.s}$ can be introduced and pushed. If in addition the output gets cropped the selection $\sigma_{T \cap CR \neq \emptyset}$ can be introduced again.

The presented rules for introducing and pushing selections are designed in such a way that for a combined selection $\sigma_{T.s \in Q \wedge T.e \in Q'}$ all suitable selections according to Table 1 can be introduced and pushed.

5 Implementation

In this section we discuss the implementations of two selected temporal operators. As examples we choose the temporal union operator (TUNION) and the temporal difference operator (TDIFF). We present several algorithms to realize the proposed semantics of those two operators. We implement the TUNION operator using *Single Scan*, *Index Based* and *Sorting Based* algorithms. The TDIFF operator is implemented by *Union First* and *Sorting Once* algorithms.

To use these implementations, there is no need to modify database systems regarding parser trees, cost functions, etc. To make the application of temporal operators transparent to the user in a high degree, we are using Oracle User-Defined Functions [20] for implementation. The implementation is not limited to Oracle Database products. Using primarily user defined table functions and basic concepts it is relatively easy to implement the operators in any common database system. For instance, in Section 6 an implementation in PostgreSQL was used to compare TUNION and TDIFF with existing temporal operators using Temporal Postgres [19].

5.1 Temporal Union

The temporal union operator (TUNION) merges right-open intervals using the concept of coalescing introduced decades ago ([7], [11], [15]) that do have overlapping relationships as defined by James F. Allen (i.e. *contains*, *equals*, *starts*, *finishes*, *meets*, *overlaps*) [2]. While the input sets can contain those overlapping relationships the output sets only include relationships of the kind *before* or *after* due to the construction of this operator.

Using an algorithm described in [22] as well as two basic algorithms we present three implementations for temporal union designed as Oracle User-Defined Functions. Pure SQL:1992 queries [21] are neither used nor evaluated due to their already shown poor performance [22].

In contrast to the definition from Section 3, the following implementations use one interval set as input instead of two sets due to simplicity reasons. The implementation with two input sets is analogue.

5.1.1 Single Scan

The most efficient algorithm proposed in [22] uses partitioning of data and windowing functions to coalesce time intervals of equal, nontemporal attributes. The concepts of windowing functions were introduced in the SQL:2003 standard.

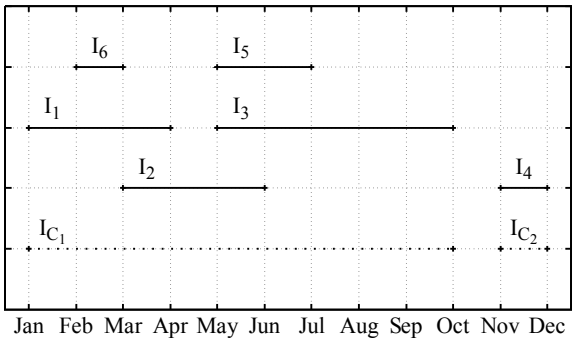
For a table R, containing start and end points of intervals and additional nontemporal columns, the temporal columns are queried into a temporary table that holds the timestamps ts, two additional columns named $Start\_ts$ and End_ts and columns for the nontemporal data. For a start point set $Start\_ts = 1$ and $End\_ts = 0$, for an end point set $Start\_ts = 0$ and $End\_ts = 1$.

The temporary table is queried to build differences of the sums of the start and end points assigned to a certain timestamp. Compute the difference of sums for ts at time t_i and denote it as $Current\_Total\_ts$. Compute the difference of sums for ts at time t_{i-1} and denote this as $Previous\_Total\_ts$. For efficient querying order the timestamps and limit the scope of the analytic windows function. If $Current\_Total\_ts = 0$ the number of start and end points at time t_i are equal and ts is an end point of a result interval. If $Previous\_Total\_ts = 0$ the number of start and end points at time t_{i-1} were equal and ts is an start point of a result interval. In a last step all those identified timestamps are combined to create the final result set of coalesced intervals.

5.1.2 Index Based

The index based implementation of the temporal union operator is using built-in indexes on the start points and end points of given intervals in order to enhance searching for large intervals overlapping many small intervals. Finding these large intervals the algorithm can skip processing the smaller intervals contained. Starting with the lowest start point $I_i.s$, the algorithm searches for the maximum end point $I_k.e$ that has an start point $I_k.s$ smaller than the given end point $I_i.e$. The interval is extended by this endpoint and this extended interval is used for further exploration of new end points. If none is found, $[I_i.s, I_k.e)$ is stored as new interval and the start point of a new interval is searched with $I_l.s > I_k.e$. With this new start point another iteration starts finding a maximum end point. Indexing in this context is used to support finding the minimal start points and maximal end points. These can be computed in almost constant time. To keep the implementation simple and compatible with existing systems, we decided to use traditional b-trees, so that only little changes have to be applied to an existing database schema.

a)



b)

I.s	I.e
2018-3-1	2018-5-31
2018-11-1	2018-11-30
2018-5-1	2018-6-30
2018-2-1	2018-2-28
2018-5-1	2018-9-30
2018-1-1	2018-3-31

c)

R.time	R.type
2018-1-1	start
2018-2-1	start
2018-2-28	start
2018-3-1	end
2018-3-31	end
2018-5-1	start
2018-5-1	start
2018-5-31	end
2018-6-30	end
2018-9-30	end
2018-11-1	start
2018-11-30	end

d)

I_C.s	I_C.e
2018-1-1	2018-9-30
2018-11-1	2018-11-30

Figure 5 TUNION Sorting Algorithm.

5.1.3 Sorting Based

The sorting based algorithm is using a fairly simple approach. It is somehow similar to the single scan algorithm but does not need SQL:2003 windowing functions. Let $\mathcal{I}$ be a set of right-open intervals (see Figure 5 a). Sort the start and end points of these intervals (b), hold them in a result set $\mathcal{R}$ and assign a type to them (*start* and *end*, (c)). After that, sweep over the sorted result set with a single running counter. By increasing or decreasing the counter depending on the type of time point one can create all resulting intervals in a single run (d). The counter is increased if a point with type *start* is recognized and decreased if point of type *end* is recognized. If the counter equals 0, a new result interval can be added to the results $\mathcal{I}_\mathcal{C}$. The sorting based TUNION is by default almost independent of the dataset's structure. The sorting is applied to all starting and all end points of every interval. Therefore by having n intervals we have to order $2n$ time points in logarithmic complexity which results in a runtime complexity of $O(n \log(n))$.

5.2 Temporal Difference

The temporal difference operator (TDIFF) is used to compute the difference of two sets of right-open intervals $\mathcal{I}$, $\mathcal{I}'$. The resulting set of intervals $\mathcal{R}$ consists of all intervals created by the points of the intervals of the first set which are not elements of the intervals of the second set. We will present two strategies for implementing the temporal difference in this section.

5.2.1 Union First

To implement a *Union First* algorithm we utilize the already proposed temporal union operator by applying it on the two input sets $\mathcal{I}$, $\mathcal{I}'$ separately. Reducing the input sets with this operator ensures all remaining intervals to be distinct and ordered in their respective result tables. Now we can move two nested cursors over these remaining intervals and compare them to get the resulting subintervals $\mathcal{R}$. Comparing the intervals $[I_i.s, I_i.e)$ and $[I'_j.s, I'_j.e)$, three different relationships can hold.

- $I_i.e < I'_j.s$
  The first interval $[I_i.s, I_i.e)$ is before the second $[I'_j.s, I'_j.e)$ not overlapping with it. A new start point $R_k.s = I_i.s$ in the result intervals of R is created if $R_k.s = null$ holds. Set $R_k.e = I_i.e$ and move to $[I_{i+1}.s, I_{i+1}.e)$.

- $I_i.s < I'_j.s =< I_i.e$

  The two intervals are overlapping or adjacent and the first interval starts before the second interval start. If $R_k.s = null$ add the start point of the first interval a new result interval start point, $R_k.s = I_j.s$, and set $R_k.e = I'_j.s$. If the end point of the first interval is after the end point of the second interval, move to the next interval $[R_{k+1}.s, R_{k+1}.e)$ $= [null, null)$. Set $[R_{k+1}.s, R_{k+1}.e) = [I'_j.e, I_i.e)$. Move to $[I'_{j+1}.s, I'_{j+1}.e)$.

- $I_i.s >= I'_j.s$

  If the end point of the first interval is before or at the end point of the second interval, no results are created or updated. Move to $[I_{i+1}.s, I_{i+1}.e)$. If the end point of the second interval is after or at the start point of the first interval, the result can be updated: $[R_k.s, R_k.e) = [I'_j.e, I_i.e)$. If the end point of the second is before the start point of the first interval, set $[R_k.s, R_k.e) = [I_i.e, I_i.s)$.

5.2.2 Sorting Once

Implementing a *Sorting Once* algorithm is similar to the TUNION operator based on sorting: querying the start and end points of the input interval sets, assigning types to the points, ordering them and sweeping over the ordered points one time only. We query the start and end points of the two sets of right-open intervals $\mathcal{I}$, $\mathcal{I}'$ and assign those points to four different types: $start1$, $start2$, $end1$, $end2$. To log the occurrence of those types we use two counters: $counter1$ is increased if a type $start1$ is recognized and decreased if the type is $end1$; $counter2$ is increased if a type $start2$ is recognized and decreased if the type is $end2$. Initial values are 0. To avoid creating false intervals when start or end points in $\mathcal{I}$ and $\mathcal{I}'$ are identical, ordering of types must follow the order $I.s$, $I'.s$, $I.e$, $I'.e$, which can be accomplished by simply adjusting names of the assigned types: $1start2$, $2start1$, $3end1$, $4end2$. In the following, the ordered list of points of $\mathcal{I}$ and $\mathcal{I}'$ is denoted $\mathcal{L}$, the types for entries are $L.t$ and time points $L.p$, the resulting interval set is denoted as $\mathcal{R}$. After ordering $\mathcal{L}$ by $L.p$ and $L.t$, we sweep over $\mathcal{L}$ and evaluate by type:

- $L_i.t = 2start1$

  Start point of an interval from $\mathcal{I}$ can be a start point in result interval $R_k.s$. If $counter1 = 0$ and $counter2 = 0$ set $R_k.s = L_i.p$, increment $counter1$ and i.

- $L_i.t = 1start2$

  Start point of an interval from $\mathcal{I}'$ can be an end point in the result interval. If $counter2 = 0$, set $T_k.e = L_i.p$. If $R_k.s \neq 0$, increment k, (create a result) and set $R_k.s = 0$ and $T_k.e = 0$. Finish by incrementing $counter2$ and i.

- $L_i.t = 4end2$

  An end point of an interval from $\mathcal{I}'$ can be start point in the result interval. If $counter1 > 0$ and $counter2 = 1$, there is a period in the first input relation that is ended by the end point of the second. $L_i.p$ will be start point of the result interval. Decrement $counter2$, increment i.

- $L_i.t = 3end1$

  An end point of an interval from the $\mathcal{I}$ can be an end point in the result interval. If $counter1 = 1$, this time points closes a period in $\mathcal{I}$. $L_i.p$ will be end point in result interval, $T_k.e = L_i.p$. $If R_k.s \neq 0$ increment k, (create a result) and set $R_k.s = 0$ and $T_k.e = 0$. Finish by decrementing $counter$, incrementing i.

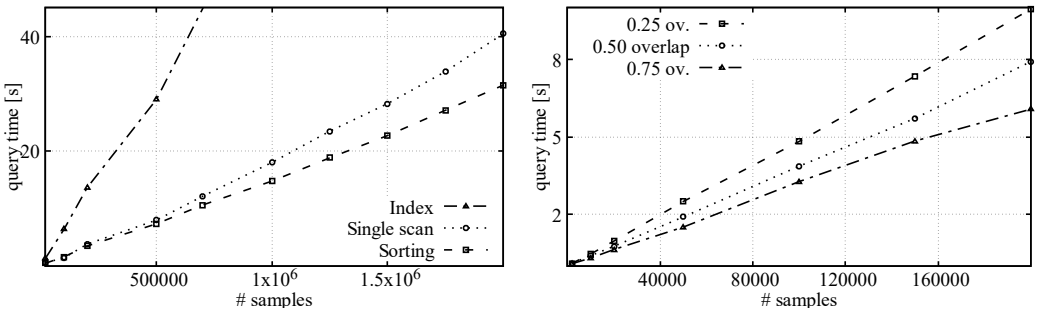


Figure 6 a) TUNION Comparison. b) TUNION Indexing on Statistical Data.

6 Evaluation

In this section we evaluate the implementations of the two selected temporal operators discussed in Section 5, TUNION and TDIFF. We use two kinds of datasets, statistically generated datasets and a real world dataset. In generated data the properties of overlapping temporal intervals can be adjusted by changing the length or position of intervals. This enables us to adapt the relationships of the generated intervals, i.e. change the ratio of overlapping, starting or containing intervals.

The real world dataset "Airline On-Time Statistics and Delay Causes" is provided by the Bureau of Transportation Statistics [18] and specifies the departure and landing timestamps of flights in the USA. It consists of roughly eight million entries per year and the flight times can be observed at a granularity of minutes.

For benchmark purposes we mainly use the commercial product Oracle Database 12c Standard Edition running on a Windows 7 Professional operating system. In order to compare our operators to the difference and union described in [9], we reimplemented TUNION and TDIFF with the open source product PostgreSQL 9.4 on Ubuntu 16.04 operating system (see Section 6.1.2 and 6.2.3). The software components are all executed on virtual hardware that has 8 GB of RAM and 3x3.4 GHz CPU cores assigned to and is running on solid state discs.

6.1 Temporal Union

Besides the real world dataset, different generated datasets with single nontemporal attributes are employed for the evaluation as well. These are comparable to the dataset of flights where a flight has an assigned flight number as nontemporal attribute and two timestamps for departure and arrival. The maximum size of input tables is two million samples. The overall result of comparing the different implementations of the TUNION operator is that sorting and single scan algorithms have the best overall performance (see Figure 6 a)) whereas the index-based implementation using basic b-trees performs very poorly.

Using Solid State Discs for evaluation does not affect the measurements significantly. This is due to the implementation of TUNION using mostly sequential disc reads.

6.1.1 Index vs. Sorting

Obviously, the index-based algorithm is highly dependent on the dataset's structure by design. If there are large intervals overlapping smaller ones, the loop searching for maximum end points of merged intervals is executed fewer times. This is tested by creating statistical datasets that have different amounts of large intervals compared to the sample size. Figure 6

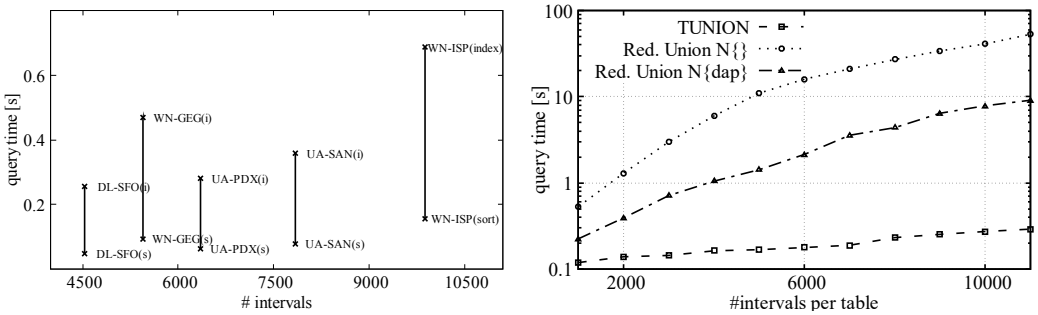


Figure 7 a) Index vs. Sorting on Flight Data. b) TUNION vs. Reduced Union.

b) shows the query over this dataset instanced with 200,000 intervals with no nontemporal attributes and data having 25%, 50% and 75% overlapping intervals.

Additionally, we take a look at the real life dataset ([18]). Airports departing flights mostly to airports located in short distances have less large flight time intervals than airports, which are in central regions or hubs with long routes to fly. This results in less overlaps and thus in higher execution times of indexed TUNION.

Figure 7 a) shows some combinations of airports and airlines queried with indexing and sorting. In this figure the upper points represent the query time using indexing, and the lower points show the query times using sorting. The lesser the distance between upper and lower points is, the more intervals are overlapped by large intervals and the more indexing approximates sorting.

6.1.2 TUNION vs. Reduced Union

To compare the execution times of [9] where temporal union is reduced to its nontemporal counterpart by temporal primitives (i.e. split, align, normalize) with our approach we took the proposed temporal extension for PostgreSQL [19] and reimplemented the TUNION-operator in this environment [14] as UDFs. Using the known flights dataset, Figure 7 b) shows the comparison of the sorting-based TUNION and the reduced union based on the normalizer function. We compared two scenarios $\mathcal{N}\{\}$ and $\mathcal{N}\{dap\}$, where in the first setting without a normalization attribute all overlapping intervals have to be split. In the latter the destination airport is used as normalization attribute which partitions the dataset by ten airports.

The reduced union produces a much larger set of results due to the temporal normalizers nature, which breaks every interval into disjoint subintervals in order to use standard SQL union on these subintervals and thus decreasing performance in comparison to sorting TUNION.

The results are only comparable to a certain degree, since reducing using temporal primitives and creating a set of very specialized operators do not follow the same approach. The first approach shows more aspects of generality, given that temporal selection, projection, aggregation, difference, union, and intersection can be reduced to nontemporal operators using a small set of temporal primitives. Our approach concedes this generality and focuses on very special operators with efficient implementations. Regarding the TUNION implementation, the use of a temporal splitter is not necessary.

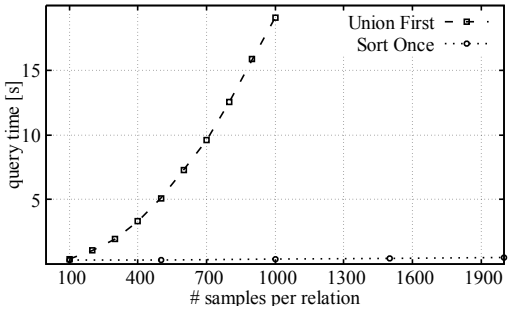


Figure 8 TDIFF - Union First vs. Sort Once.

6.2 Temporal Difference

Similar to Section 6.1 we use a real world dataset as well as generated datasets with a sample size of 500,000 in order to evaluate TDIFF operator variants.

6.2.1 Statistical Data

For the evaluation of TDIFF we first generate a dataset in a way that for every interval in the first interval set $\mathcal{I}$ we assign an interval in the second interval set $\mathcal{I}'$ that is neither adjacent nor overlapping. Modulating the position of the intervals of $\mathcal{I}'$ we then can change the relationship from 1) I'_i *after* I_i to 2) I_i *overlaps* I'_i to 3) I'_i *finishes* I_i to 4) I'_i *during* I_i, and use these scenarios to compute averages for various sample sizes.

Because of the structure of the *Union First* algorithm and the nested loop to be executed, the query time has to be in $O(n\log(n) + m\log(m) + nm)$ where n is the size of $\mathcal{I}$ and m is the size of $\mathcal{I}'$. Assuming that both input sets have similar sizes and few overlapping intervals that can be coalesced in the first step, this implementation performs rather poorly.

In comparison the *Sort Once* strategy has a much better performance. This is due to the fact that after the sorting in $O(n\log(n) + m\log(m))$ time the algorithm sweeps in linear $O(n + m)$ time over the ordered time points.

Figure 8 shows the query times of *Union First* and *Sort Once*. We are using only 0.5% of entries of the 500k dataset for visualization purposes. We are discarding the *Union First* algorithm further on and evaluate with *Sort Once* only in the next section.

6.2.2 Real World Dataset

Suppose an airline company wants to get an overview of the landing delays of its flights. The company is especially interested in when they are happening, and how long those delays are. The desired delay intervals can be elegantly computed by applying TDIFF($\mathcal{I}', \mathcal{I}$) to interval sets, where $\mathcal{I}$ are all flights assigned to that airline from the flight plan and $\mathcal{I}'$ are the time intervals of planned departure and and actual landing data. Using this scenario, Figure 9 a) shows the query times of different airlines in the first quarter of the year 2008 based on the *Sort Once* algorithm. The query times on a real world dataset show similarity to those of generated averaged data we used in the previous section.

6.2.3 TDIFF vs. Reduced Difference

In [9] it is shown how the temporal difference can be reduced to its nontemporal difference operator. Figure 9 b) shows the comparison of the TDIFF operator and the nontemporal

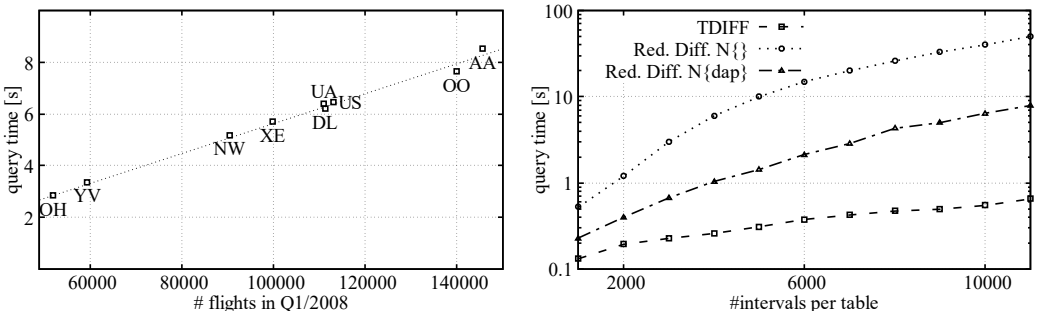


Figure 9 a) TDIFF - Sort Once with Flight Data. b) TDIFF vs. Reduced Difference.

difference with normalizer function using the combined generated datasets as well as the known flights dataset analogue to Section 6.1.2. Likewise, the reduced difference produces a much larger set of results due to the temporal normalizers nature, which breaks every interval into disjoint subintervals in order to use standard SQL minus operator on these subintervals and thus decreasing performance in comparison to TDIFF.

7 Conclusions

In this work we discussed the development of a library of temporal operators for processing sets of temporal intervals. To this end, we provided the formal description of various operators and discussed their implementation by means of table functions. The resulting operators allow to formulate complex temporal queries in an elegant way and open the possibility for further algebraic optimization. In fact, the provided operators can be seen as a foundation of a situation calculus necessary for achieving situation-awareness inside a database system. We have studied and evaluated different algorithms for implementing the operators TUNION and TDIFF without the need for considerable changes to an existing database schema or even database kernel. One particular interesting finding is the fact that the index-based algorithm performed poorly in comparison with the other implementations. However, we assume that performance can be increased dramatically by using specialized index structures as proposed in ([3], [8], [16]). Improving existing algorithms of the proposed temporal operators by implementing specific data structures as well as extending the library of table functions is subject of future work.

References

1 Mohamed H. Ali, Badrish Chandramouli, Balan Sethu Raman, and Ed Katibah. Spatio-temporal stream processing in microsoft streaminsight. *IEEE Data Eng. Bull.*, 33(2):69–74, 2010.

2 James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

3 Chuan-Heng Ang and Kok-Phuang Tan. The interval b-tree. *Information Processing Letters*, 53(2):85–89, 1995.

4 Andreas Behrend, Philip Schmiegelt, and Andreas Dohr. Supporting situation awareness in spatio-temporal databases. *Datenbank-Spektrum*, 16(3):207–218, 2016.

5 Michael H. Böhlen, Renato Busatto, and Christian S. Jensen. Point-versus interval-based temporal data models. In *Proc. of ICDE*, pages 192–200, 1998.

6 Michael H. Böhlen, Christian S. Jensen, and Richard T. Snodgrass. Temporal statement modifiers. *ACM Trans. Database Syst.*, 25(4):407–456, 2000.

7 Michael H. Böhlen, Richard T. Snodgrass, and Michael D. Soo. Coalescing in temporal databases. In *VLDB 1996*, pages 180–191, 1996.

8 Tolga Bozkaya and Z. Meral Özsoyoglu. Indexing valid time intervals. In *DEXA 1998*, pages 541–550, 1998.

9 Anton Dignös, Michael H. Böhlen, and Johann Gamper. Temporal alignment. In *SIGMOD*, pages 433–444, 2012.

10 Anton Dignös, Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Extending the kernel of a relational DBMS with comprehensive support for sequenced temporal queries. *ACM Trans. Database Syst.*, 41(4):26:1–26:46, 2016.

11 Curtis E. Dyreson. Temporal coalescing with now, granularity, and incomplete information. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 169–180. ACM, 2003. doi:10.1145/872757.872779.

12 Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, and Peter R. Pietzuch. Integrating scale out and fault tolerance in stream processing using operator state management. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 725–736. ACM, 2013. URL: http://dl.acm.org/citation.cfm?id=2463676, doi:10.1145/2463676.2465282.

13 Dieter Gawlick, Eric S. Chan, Adel Ghoneimy, and Zhen Hua Liu. Mastering situation awareness: The next big challenge? *SIGMOD Record*, 44(3):19–24, 2015.

14 The PostgreSQL Global Development Group. PostgreSQL 9.4.12 Documentation, 2017. [Online; accessed 28-May-2017]. URL: https://www.postgresql.org/docs/9.4/static/index.html.

15 Christian S. Jensen, James Clifford, Shashi K. Gadia, Arie Segev, and Richard T. Snodgrass. A glossary of temporal database concepts. *SIGMOD Record*, 21(3):35–43, 1992.

16 Hans-Peter Kriegel, Marco Pötke, and Thomas Seidl. Managing intervals efficiently in object-relational databases. In *VLDB 2000*, pages 407–418, 2000.

17 Derek Gordon Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: a timely dataflow system. In *ACM SIGOPS 2013*, pages 439–455, 2013.

18 Bureau of Transportation Statistics. Airline On-Time Statistics and Delay Causes, 2017. [Online; accessed 7-February-2017]. URL: https://www.transtats.bts.gov.

19 University of Zurich. Temporal PostgreSQL, 2017. [Online; accessed 28-May-2017]. URL: http://tpg.inf.unibz.it/downloads/postgresql-9.6beta3-temporal.tar.gz.

20 Oracle. Oracle Database 12c Release 2 Online Documentation, 2017. [Online; accessed 7-February-2017]. URL: http://docs.oracle.com/database/122/index.htm.

21 Richard T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 1999.

22 Xin Zhou, Fusheng Wang, and Carlo Zaniolo. Efficient temporal coalescing query support in relational database systems. In *DEXA 2006*, pages 676–686, 2006.

Deciding the Consistency of Branching Time Interval Networks

Marco Gavanelli
Department of Engineering
University of Ferrara, Italy
marco.gavanelli@unife.it
https://orcid.org/0000-0001-7433-5899

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
https://orcid.org/0000-0002-9221-879X

Abstract

Allen's Interval Algebra (IA) is one of the most prominent formalisms in the area of qualitative temporal reasoning; however, its applications are naturally restricted to linear flows of time. When dealing with nonlinear time, Allen's algebra can be extended in several ways, and, as suggested by Ragni and Wölfl [20], a possible solution consists in defining the Branching Algebra (BA) as a set of 19 basic relations (13 basic linear relations plus 6 new basic nonlinear ones) in such a way that each basic relation between two intervals is completely defined by the relative position of the endpoints on a tree-like partial order. While the problem of deciding the consistency of a network of IA-constraints is well-studied, and every subset of the IA has been classified with respect to the tractability of its consistency problem, the fragments of the BA have received less attention. In this paper, we first define the notion of convex BA-relation, and, then, we prove that the consistency of a network of convex BA-relations can be decided via path consistency, and is therefore a polynomial problem. This is the first non-trivial tractable fragment of the BA; given the clear parallel with the linear case, our contribution poses the bases for a deeper study of fragments of BA towards their complete classification.

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

Keywords and phrases Constraint programming, Consistency, Branching time

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.12

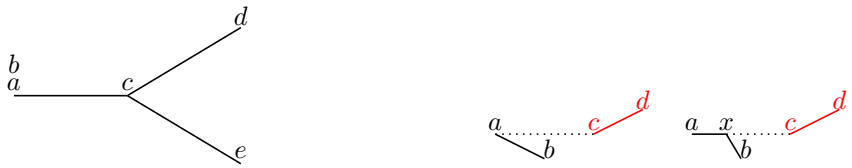
Acknowledgements G. Sciavicco acknowledges partial support by the Italian INDAM GNCS project *Formal methods for verification and synthesis of discrete and hybrid systems.*

1 Introduction

Allen's Interval Algebra [1] (IA) is one of the most prominent formalisms in the area of qualitative temporal, and also spatial, reasoning. However, its applications are naturally restricted to linear flows of time. Allen's algebra is considered one of the most influential formalisms in qualitative reasoning, and it has found application in a wide range of contexts,

such as scheduling, planning, database theory, 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 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. The most relevant problem in IA is deciding whether a network can be satisfied, that is, deciding if every variable of a network can be instantiated to an interval without violating any constraint. The consistency of a network of constraints is archetypical of the class of constraints satisfaction problems (CSP), because a network is a conjunction of constraints; other consistency problems, even in temporal algebras, are more general, and allow some form of disjunction. A very wide range of programming techniques, strategies, and heuristics have been and are being studied to devise efficient implementations; in the particular case of the IA, whose consistency problem is NP-complete, two main strategies have been mainly adopted, that are based either on clever brute-force enumerating algorithms (see, e.g. [13, 23]), or on tractable fragments of the algebra, which are interesting on their own [14] as well as heuristics, aimed to reduce the branching factor in branch-and-bound approaches [15, 19]. Among the several tractable fragments of the IA, a particularly interesting one is known as IA_{convex}, introduced in the early years of this line of research, and encompassing 44 basic and non-basic relations [25]. To obtain IA_{convex}, van Beek and Cohen [25] study, first, the simpler Point Algebra (PA), which has only three basic relations, and define the notion of convexity for a PA-relation; having identified the set of PA_{convex}-relations, they define the set IA_{convex} as the maximal subset of IA-relations with the following property: every network of IA_{convex}-constraints can be translated into an equi-satisfiable network of PA_{convex}-constraints. Then, using the properties of such a translation, they prove that the path consistency algorithm is complete for deciding the consistency of a IA_{convex}-network, obtaining the same result for PA_{convex}-networks as a corollary. Since then, many other tractable fragments of the IA have been discovered, and, in fact, we know the status of every fragment of the IA with respect to the tractability/intractability of the corresponding consistency problem [14].

Branching time has been less studied from the algebraic point of view, in sharp contrast with the huge amount of research on point-based and interval-based temporal logics, such as CTL, CTL*, ATL, or branching PNL [3, 4, 11]. The Branching Point Algebra (BPA) has been studied in [9, 12], and the computational behaviour of the consistency problem of the BPA and its fragments has been analyzed in [5], where, in particular, a polynomial algorithm to decide the consistency of a network of BPA-constraints is given. In [20], the authors define a branching version of Allen's IA, which we refer to as BA (Branching Algebra) 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 basic relations, and also studied from the (first-order) expressive power point of view in [10], is characterized by basic relations whose semantics cannot be always written in the language of endpoints, therefore requiring quantification; on the other hand, the basic relations of the second set are coarser, still jointly exhaustive and mutually exclusive, and their point-based semantics depends only on the relative position of the endpoints. The latter set (BA_{basic}), composed of 19 basic relations, is therefore preferable in many aspects. As expected, the consistency problem for the full BA is NP-complete, and we know only one tractable fragment of it that includes at least one nonlinear relation, that is, the set BA_{basic} itself, since a network of BA_{basic}-constraints can be fully translated into a network of BPA-constraints. In this paper:



(a) (b)

Figure 1 A pictorial representation of the four basic branching point relations, in which $a = b$, $a < c$, $d > c$, and $d||e$ (left-hand side), and two branching relations that need quantification (right-hand side).

(i) in the the spirit of [25], we define the notion of convex branching point relation and the notion of convex branching interval relation, and prove that, as in the linear case, the consistency of a network of convex branching interval (and therefore point) relations can be decided enforcing its path consistency, and

(ii) following [22], 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 convex relations instead of basic relations.

This paper is organized as follows. First we give some necessary preliminaries and notation. In Section 3 we give the main result of this paper, that is, we define convex branching relations and show that the consistency problem can be decided by enforcing path consistency in convex networks. Then, in Section 4, we give some experimental evidence that convex branching relations can be used to speed up the process of deciding branching relation networks, before concluding.

2 Preliminaries

Notation. Let $(\mathcal{T}, <)$ be a partial order, whose elements are generally denoted by $a, b, \ldots$, and where $a||b$ denotes that a and b are incomparable with respect to the ordering relation $<$. We use $x, y, \ldots$ to denote variables in the domain of points. 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 $a||b$ then there exists no $c \in \mathcal{T}$ such that $c > a$ and $c > b$ (that is, it is a tree). In a branching model $(\mathcal{T}, <)$, any maximal linearly ordered subset $\mathcal{B}$ of $\mathcal{T}$ will be called *branch*. There are four basic relations that may hold between two points on a branching model: *equals* $(=)$, *incomparable* $(||)$, *less than* $(<)$, and *greater than* $(>)$; the first two are symmetric, while the last two are inverse of each other. These relations are depicted in Fig. 1a, 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 $I, J, \ldots$, and we use $X, Y, \ldots$ to indicate variables in the domain of intervals. Following Allen [1], we adopt the so-called *strict interpretation* by asking that intervals with coincident endpoints are excluded. In the case of linear time, a theory that encompasses both intervals and *points* has been presented in [7]. There are several ways to define basic relations between intervals on a branching order. Following [10], 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

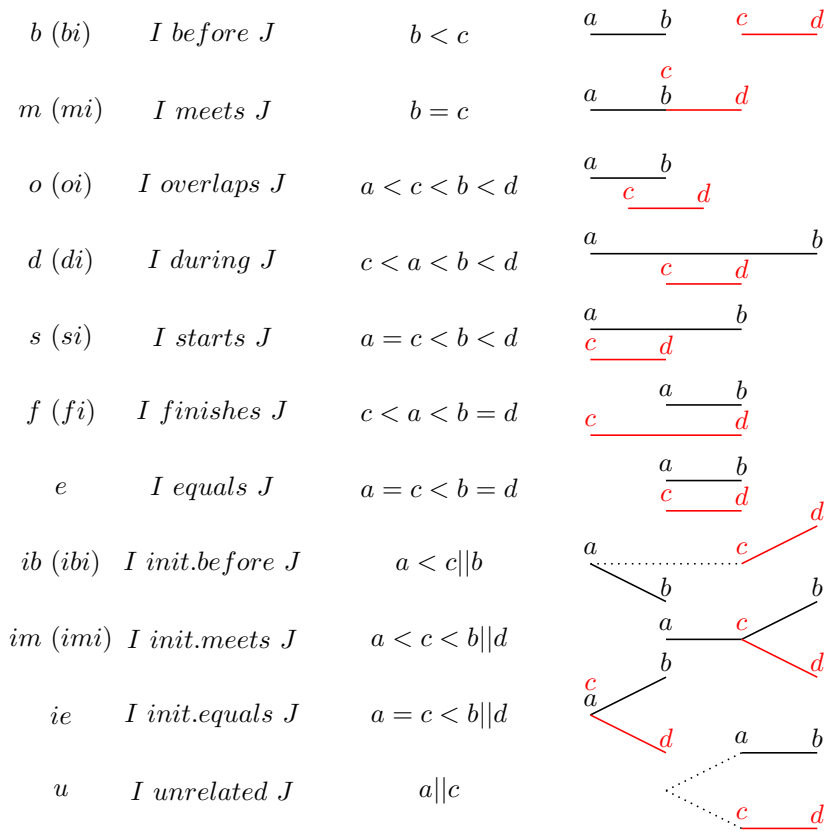


Figure 2 A pictorial representation of the nineteen basic branching interval relations. In this picture, we assume $a < b$ and $c < d$. Solid lines are actual intervals, dashed lines complete the underlying tree structure.

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 Fig. 1b we see that, in order to distinguish the two situations, we need to quantify of the existence, or non-existence, of points, comparable with c, 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 Wölfl [20] introduce a set of coarser relations, characterized by being translatable to point-based relations using only the language of endpoints, that is, without quantification. These 19 relations are depicted in Fig. 2, and form the set of *basic branching interval relations* BA_{basic}; for each relation, the symbol in brackets corresponds to its inverse one, 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 fine relations. For example, the relation im is the disjunction of the two relations in Fig. 1b.

Operations and algebras. The set BPA of *branching point relations* is the set of all possible non-empty disjunctions of basic branching point relations, and it encompasses $2^4 - 1 = 15$ relations. Similarly, the set BA of *branching interval relations* is the set of all possible disjunctions of basic branching interval relations, and it encompasses $2^{19} - 1$

Table 1 Composition of basic BPA-relations (left-hand side), and of basic PA-relations (right-hand side).

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

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

relations. In general, given the basic relations $r_1, \ldots, r_l$, we denote by $R = \{r_1, \ldots, r_l\}$ the disjunctive relation $r_1 \vee \ldots \vee r_l$; thus, a relation is seen as a set, and a basic relation as a singleton. A BPA-*constraint* is an object of the type xRy, where x, y are point variables and $R \in BPA$; analogously, a BA-*constraint* is an object of the type XRY, where X, Y are interval variables and $R \in BA$. There are three basic operations with relations: (Boolean) intersection, inverse, and composition. The *inverse* of a relation $R = \{r_1, \ldots, r_l\}$ is the relation $R^{-1} = \{r_1^{-1}, \ldots, r_l^{-1}\}$, where, for each i, r_i^{-1} is the inverse basic relation of the basic relation r_i. In our notation, for example, *bi* (*later*) is 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: $R_1 \circ R_2 = \{r \mid \exists r_1 \in R_1 \exists r_2 \in R_2 (r = r_1 \circ r_2)\}$. Clearly, to compute the composition of two non-basic relations we base ourselves on the composition between basic relations. As for the set BPA_{basic} (resp., the set PA_{basic}), the composition table can be easily computed 'by hand', as in Tab. 1, left-hand side (resp., right-hand side), where we use the abbreviations $lin = \{<, =, >\}$ and $? = lin \cup \{||\}$. The result of composing two relations in the set BA_{basic} (resp., IA_{basic}) can be computed automatically from Tab. 1, and it is fully reported in [21] (resp., [2]).

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, and two networks N and M are said to be *equi-satisfiable* if it happens that N is consistent if and only if M is consistent. Given a constraint (s, t) in a network N, we say that $r \in (s, t)$ is *feasible* if there exists a model of N such that s and t are realized respecting r, and a constraint (s, t) is said to be *minimal* if every $r \in (s, t)$ is feasible and (s, t) cannot be extended with other feasible relations; establishing the minimal constraints for every constraint in a network is called the *minimal labels* problem. Enforcing the minimal label in a network implies deciding its consistency, but, in general, we may have a consistent network with non-minimal labels. The operations of inverse, intersection, and composition can be used to design a constraint satisfaction problem (CSP) technique to decide the consistency of a A-network. The sets BPA and BA are called, respectively, the *branching point algebra* and the *branching interval algebra*, and they extend, respectively, the *interval algebra* IA and the *point algebra* PA. Since the consistency problem for the IA is NP-complete [26], the problem of finding tractable fragments of it is interesting, and it has been largely studied

in the recent literature [1, 25, 14]. The branching setting presents a similar situation, as the consistency problem for the BA is NP-complete as well, but, in contrast with the linear case, only one tractable fragment is known, that is, BA_{basic} [20]; the consistency of a network of basic branching interval constraints can be decided by translating it into an equi-satisfiable network of BPA-constraints, for which a deterministic polynomial consistency algorithm exists [5].

Local consistency. On the one hand, the BA-consistency problem is in NP because there exists a simple non-deterministic algorithm that solves it, which, given a BA-network $N = (V, E)$, guesses the relative position of $2 \cdot |V|$ points and checks if every constraint is respected. On the other hand, these problems are often solved 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; it is said to be *strongly* k-consistent if it is k'-consistent for every $k' \leq k$ (see [17]); if a network is strongly k-consistent, then it must also have minimal labels. 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) does not imply consistency; this is true for BA-networks as well as for IA-networks. In [25], however, it is proven that enforcing path consistency is equivalent to computing the minimal labels of a IA_{basic}-network, which, in turn, allows one to check the existence of a model. This property of path consistency is shown for a more general set of relations, called *convex interval relations*, which are defined starting from the set PA_{convex} of *convex point relations*, and, in particular, it is proved that the set IA_{convex} (the *convex interval algebra*) includes IA_{basic}, and that its consistency problem (and, as a corollary, the consistency of a PA_{convex}-constraints) can be decided by path consistency. This result, particularly interesting for us, has the following consequences. First, IA_{convex} is a fragment of the IA with a tractable (in fact, cubic time) consistency problem. Second, one can implement a simple branch-and-bound algorithm to decide the consistency of any IA-network, based on IA_{basic}: at each step, the algorithm tries one basic relation for each relation, and then forces the path consistency of the resulting network; if at any step the network is path consistent, it returns *true*, and if every combination has been tried and enforcing path consistency has always resulted in an empty relation, it returns *false*. Third, the set IA_{convex} 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 IA_{convex}-network, that particular branch can be decided by simply enforcing path consistency.

In the following, we shall define the algebra BA_{convex} of convex branching relations. We shall prove that, since BA_{convex} extends BA_{basic}, and since enforcing path consistency can be used to decide the consistency of a BA_{convex}-network, one can apply the same schema, effectively lifting all above results to the setting of branching time.

A motivating scenario. *Scheduling* is the problem of distributing computing resources (such as processor time, bandwidth, or memory) to various processes, threads, data flows, and applications that need them. In robotics, scheduling is used to organize tasks to be assigned

to robots of various kinds, in such a way that all physical and subjective constraints are met. A recent application of Allen's Interval Algebra to the scheduling of tasks for a robot has been proposed by Mudrová and Hawes [18]. The authors propose a scheduling technique that takes into account a series of constraints, including *deadlines* and *processing time* for each one of a series of tasks that a robot is asked to complete. They propose to apply a consistency checking algorithm to the network of qualitative constraints that underlies the scheduling problem, in order to prune any ordering of the tasks that does not meet the qualitative constraints, and to be able to select, systematically, possible models of the problem. To each of the models, then, a successive phase of quantitative constraint checking is applied.

Mudrová and Hawes solve the scheduling problem of a *single robot*, which encompasses assuming the time to be linear and introducing the additional constraint that no two tasks can overlap. Using Branching Algebra instead of Interval Algebra as part of the scheduling algorithm, and using the relation $||$ to relax, when possible, the non-overlapping constraint, one may obtain, instead, branching models among the solutions. A branching model of the scheduling can be interpreted as a scheduling in which *more than one robot* is involved, that is, in which every new branch implicitly refers to a new robot being activated (at the branching point), and, if we assume that *bootstrapping* a robot has some fixed cost (higher than maintaining a robot active), then it makes sense to look for a branching scheduling as we have defined it, that is, tree-like (in which branches never join again). Thus, Branching Algebra allows one to generalize this scheduling problem to a more complex scenario, which cannot be easily handled in the original formulation.

3 Convex Branching Interval Relations

We start by defining the concept of convex relation in the branching setting. In this section, we operate with translations from interval constraints to point constraints; when necessary, for an interval variable X, we use the symbols X^-, X^+ to denote the point variables that correspond to its endpoints.

▶ **Definition 1.** The *convex branching point algebra* is the set of relations:

$$BPA_{convex} = \{\{=\}, \{<\}, \{<, =\}, \{<, =, >\}, \{=, >\}, \{>\}, \{||\}\}.$$

Each relation of set BPA_{convex} is called *convex branching point relation.*

Observe that BPA_{convex} extends the convex point algebra PA_{convex} as defined in [25] by adding the relation $\{||\}$. While the set PA_{convex} is closed under composition, inverse, and intersection, the set BPA_{convex} is closed under under inverse and intersection only; this, however, does not prevent us from applying constraint propagation algorithms such as path consistency enforcing.

▶ **Definition 2.** The *convex branching interval algebra* BA_{convex} is the set of all (and only) BA-relations R such that the constraint XRY can be translated to an equi-satisfiable conjunction of constraints between the endpoints of X and Y using BPA_{convex}-relations only. A branching relation with such a property is called *convex branching interval relation.*

For example, in the linear setting, $\{b, m\}$ is convex because, if $X = [X^-, X^+]$ and $Y = [Y^-, Y^+]$, the constraint $X\{b, m\}Y$ is equivalent to the conjunction of the constraints $X^- < X^+$, $Y^- < Y^+$, and $X_1^+\{<, =\}Y^-$; conversely, the relation $\{b, bi\}$ is not convex, because translating it results in a disjunction of point-based constraints. Clearly, the set BA_{convex} extends the set IA_{convex} of convex interval relations as defined in [25]. The

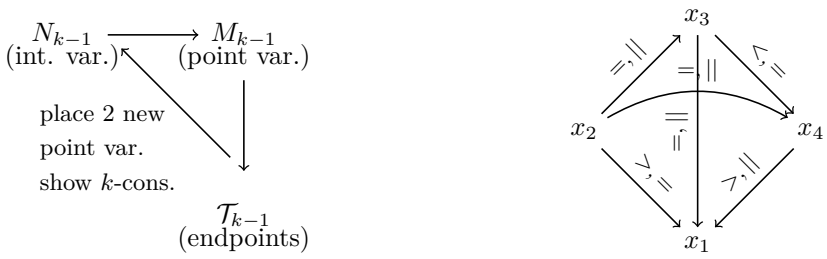


Figure 3 A general view of the strategy for the inductive case of Theorem 5 (left-hand side), and a path-consistent network with non-minimal labels (right-hand side).

crucial property of a BA_{convex}-network N is that it can be translated to an equi-satisfiable BPA_{convex}-network M (notice that this is not true for a generic BA-network: non-convex constraints may result in disjunctions that cannot be represented in the language of full BPA-networks), such that, if N is k-consistent for any k, then M is $2 \cdot k$-consistent. As it can be easily checked, there are precisely 91 convex branching interval relations. In the linear case, the following results hold [25].

▶ **Theorem 3.** *Enforcing path consistency in a IA_{convex}-network is sufficient to compute its minimal labels.*

▶ **Corollary 4.** *Enforcing path consistency in a PA_{convex}-network is sufficient to compute its minimal labels.*

Our purpose in this section is to generalize the above theorem to the branching case.

We want to prove that we can decide the consistency of a BA_{convex}-network by enforcing its path consistency. To this end, we prove that enforcing path consistency of a BA_{convex}-network actually enforces the minimal labels on each constraint by proving that, in fact, enforcing path consistency of a network entails enforcing its strongly k-consistency for every k. As we have already observed, this allows us to check the consistency of a network.

▶ **Theorem 5.** *Enforcing path consistency in a BA_{convex}-network is sufficient to compute its minimal labels.*

Proof. Let N be a BA_{convex}-network, and let M be the BPA_{convex}-network that results from translating N in the language of endpoints. We assume that path consistency has been forced on N, and we want to show that N is strongly k-consistent for every k; since a strongly k-consistent network must have minimal labels, we have the result. Let us proceed by induction. As base case, we know that N is k-consistent for $k \leq 3$. As for the inductive case, we suppose now that N is $k-1$-consistent and we prove that it is also k-consistent. Consider a subset S of $k-1$ interval variables in N. Let us call N_{k-1} the sub-network, with the $k-1$ interval variables $X_1, \ldots, X_{k-1}$, corresponding to the projection of N over set S, and let us call M_{k-1} the corresponding BPA_{convex}-network whose variables are precisely the $2 \cdot (k-1)$ endpoints of $X_1, \ldots, X_{k-1}$. Our strategy, as sketched in Fig. 3a, can be summarized as follows: since N_{k-1} is consistent by hypothesis, M_{k-1} must be consistent as well, that is, it must realized in a branching model $\mathcal{T}_{k-1}$; if we pick the point variables corresponding to the endpoints of any k-th interval variable and accommodate them in $\mathcal{T}$ showing that every constraint is respected, then we obtain a branching model for k interval variables, proving that N_k is also consistent. Let X be any interval variable in N different

from $X_1, \ldots, X_{k-1}$, and let XR_iX_i the BA_{convex}-relation between the variables X and X_i, for each i. Let M_k the BPA_{convex}-network obtained by adding to M_{k-1} the point variables X^-, X^+, the constraint $X^- < X^+$, and every constraint between the endpoints of X and the endpoints of $X_1, \ldots, X_{k-1}$ that results from translating the constraints of the type XR_iX_i. On $\mathcal{T}_{k-1}$ we can identify the set $\mathcal{R} = \{a_1, \ldots, a_n\}$ with the following characteristics: for each i, a_i is the realization of some point variable y in M_{k-1} (that is, a_i is the realization of some endpoint of the interval variables $X_1, \ldots, X_{k-1}$) and that, for every point variable $y \in M_{k-1}$, realized in some point $a \in \mathcal{T}_{k-1}$, it is not the case that $a < a_i$. Indeed, consider the branching model $\mathcal{T}_{k-1}$: since it must be a tree, it may be the case that, in order to realize two variables that are constrained to be incomparable to each other, a greatest common predecessor must be added; therefore, if projected to the points that realize some point variable in N_{k-1}, $\mathcal{T}_{k-1}$ is a *forest of trees*, rather then a tree. Every point in $\mathcal{R}$ is the *root* of one of the trees in $\mathcal{T}_{k-1}$; let us call c their greatest common predecessor. Now, let $x_1, \ldots, x_m$ be the point variables that have been realized in $a_1, \ldots a_n$ (observe that $n \leq m \leq k-1$: two variables may have been realized in the same root, and m cannot exceed k_1 because, at most, every interval variable has its left endpoint realized in a root). We want to show, first, that the point variable X^- can be successfully realized on $\mathcal{T}_{k-1}$, and we proceed case by case.

- Suppose, first, that $(x_l, X^-) = \{||\}$ for every point variable x_l realized in some root. In this case, we realize X^- with a new point a such that $a||a_i$ for each root a_i, and that $c < a$. To prove that this is a consistent choice, consider any point variable y of M_{k-1} realized at some point $b \geq a_i$ for some root a_i. Suppose that a_i is the realization of some point variable x_l, which means that $(y, x_l) \subseteq lin$. If $< \in (y, x_l)$, then $(y, x_l) \circ (x_l, X^-) = \{<, ||\}$. By intersection with BA_{convex}, then either $(y, X^-) = \{<\}$ or $(y, X^-) = \{||\}$; in the first case, however, we obtain, by path consistency, that $(X^-, x_l) \in lin$, which is a contradiction. Therefore, $(y, X^-) = \{||\}$. If , on the other hand, $< \notin (y, x_l)$, then, $(y, x_l) \subseteq \{>, =\}$, and, since $\{>, =\} \circ \{||\} = \{||\}$, it must be the case that $(y, X^-) = \{||\}$.
- Suppose, now, that $(x_l, X^-) \subseteq lin$ for some point variable x_l realized in some root a_i. Observe, first, that if $(X^-, x_l) = \{<\}$, then we can select the subset $\mathcal{R}' \subseteq \mathcal{R}$ such that, for each $x'_l \in \mathcal{R}'$, we have that $(X^-, x'_l) = \{<\}$; in this case, by the argument in the above case, for each $x''_l \in \mathcal{R} \setminus \mathcal{R}'$, we have that $(X^-, x'_l) = \{||\}$. Consider each a_j that is the realization of some variable in $\mathcal{R}'$: we realize X^- in a point $a > c$, such that a is less than every such a_j, and incomparable with every other root in $\mathcal{R} \setminus \mathcal{R}'$. If, otherwise, $(X^-, x_l) \subseteq \{>, =\}$, then, for each x'_l realized in some root $a_j \neq a_i$, we must have $(X^-, x'_l) = \{||\}$. In this case, we can say that a_i is the root of the tree in which we have to realize X^-; let us call it $\mathcal{T}_{a_i}$. Observe that, by the same argument as in the above case, wherever we realize X^- in $\mathcal{T}_{a_i}$, this realization is consistent with any point that belongs to some $\mathcal{T}_{a_j}$ with $a_j \neq a_i$. Now, we consider the point $b \in \mathcal{T}_{a_i}$ which is the least point (greater than or equal to a_i) with at least two immediate successors b_1, b_2 such that $b_1||b_2$, if it exists. We have the following cases.

  - Suppose that b does not exists. This means that T_{a_i} is linearly ordered. Let b' be the least point (greater than a_i), such that is the realization of some variable y such that $(y, X^-) = \{||\}$. If there is no such b', then, by Theorem 3, we can find a realization for X^- consistent with $\mathcal{T}_{a_i}$; since we already know that such a realization is consistent with every other tree, we conclude that it is consistent. If b' exists, then we realize X^- in a point a such that $a||b'$ and that $a > d$ where d is the immediate predecessor of b'. By the argument used in the first case, this choice must be consistent with $\mathcal{T}_{a_i}$, and therefore it must be consistent.

- Suppose, now, that b exists. If y is realized in b, and $\{<, =, ||\} \cap (X^-, y) \neq \emptyset$, then we proceed as in the previous case. If, on the other hand, $(X^-, y) = \{>\}$, we have the following two cases. First, if $< \in (X^-, y)$ for every y_j realized in some point b_j such that b_j is immediate successor of b, then realize X^- in a point a such that $b < a$ and that $a < b_j$ for every immediate successor b_j of b, which must be a consistent choice, given that, by path consistency, the relation between X^- and every variable realized in a point greater than b must contain $<$. If, for some immediate successor b_j of b, which is the realization of some variable y, it is the case that $< \notin (X^-, y)$, then we can treat every immediate successor b_j of b as the root of some sub-tree of T_{a_i}, and therefore we can apply the same entire argument, recursively.

Having realized the variable X^-, the network M_{k-1} enriched with X^- (and all relative constraints) must be consistent. By reapplying the entire argument, we can show that any other point variable can be consistently realized in the resulting network; if we choose X^+ among these, we prove that the original network N is, in fact k-consistent, completing the induction. ◀

▶ **Corollary 6.** *Enforcing path consistency in a BPA_{convex}-network is sufficient to compute its minimal labels.*

It would be natural, at this point, to ask whether the set BPA_{convex} can be enriched while retaining the above nice properties. A natural candidate in this perspective would be the set of all BPA-relations such that, for each R, it is the case that $R \cap lin$ (which is a linear relation) is convex. This set is closed under composition, inverse, and intersection, allowing one to approach its consistency problem in the same way as we did in this work; in particular, it would be possible to define the set of BA-relations that can be translated in this language, obtaining, in fact, a greater set of branching relation whose minimal labels could be enforced by path consistency. Unfortunately, there is an easy counter-example to this claim, shown in Fig. 3b, in which we have a path consistent network but with non-minimal labels. Therefore, if there exists any extension of BPA_{convex} whose minimal labels can be enforced by path-consistency, it cannot include at least one of the relations in Fig. 3b.

Observe that enforcing the minimal labels via path consistency is not the only way to prove that the consistency of a network can be decided via path consistency, and it is certainly not the only way to prove that a fragment of relations is tractable. For example, the tractability of the consistency problem for a network of IA-relations in the *ORD-Horn* fragment is proven via embedding into the Horn fragment of propositional logic [14]; as an another example, the tractability of the consistency problem for a network of full BPA-relations is proven with a specialized algorithm in [5], and it is the basis for Ragni and Wölfl's result about the tractability of the consistency problem for a network of BA_{basic}-relations.

4 Experiments

In order to evaluate the usefulness of the convex fragment as a heuristics for the task of checking the consistency of a BA-network, we devised a set of experiments, following the classical methodologies in the literature.

To generate a random set of instances, we used a (modification of) a technique suggested by Renz and Nebel [22] that consists of the following steps. Given a number n of nodes, an average density d and a probability p, we generate a random instance as follows:

Algorithm 1 Backtracking algorithm.

```
function CONSISTENT(P, Split)
    enforce generalized arc consistency on P
    if there is a variable ν_XY such that D_XY = ∅ then
        return false
    else
        choose an unprocessed variable ν_XY such that D_XY ∉ Split
        if there is no such variable then
            return true
        {D_1, ..., D_p}=PARTITION(D_XY, Split)
        for all D_i ∈ {D_1, ..., D_p} do
            P' = P_{D_XY / D_i}
            if CONSISTENT(P', Split) then
                return true
        return false
```

(i) we generate a graph with n nodes, and select $\frac{d \cdot n \cdot (n-1)}{2}$ edges at random;
(ii) for each selected edge (s,t), we generate a BA-relation R by selecting, with probability p, each BA_{basic}-relation to be inserted in R, and
(iii) to each non-selected edge (s,t), we assign the universal relation.

As much as the solver is concerned, we based ourselves on the general strategy of *constraint logic programming on finite domains*, by means of which we are able to check the satisfiability of a temporal network. The solver itself is based on the dual CSP, encoded with translation into ternary constraints, as proposed by Condotta et al. [6]. In particular, given an A-network $N = (V, E)$, we define a CSP P as a triple $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, such that:

- the set $\mathcal{V}$ contains a CSP variable ν_{XY} for each pair of variables X, Y in V;
- the domain $\mathcal{D}_{XY}$ of each variable ν_{XY} is precisely the constraint (X, Y), and
- for each triple of variables, $\mathcal{C}$ contains a ternary constraint, so that each ternary constraint will be satisfied by a triple $(r, r_1, r_2) \in BA^3_{basic}$ if and only if $r \in r_1 \circ r_2$.

As noted by Condotta et al. [6], enforcing the (generalized) arc consistency on the problem P is equivalent to enforcing path consistency on the original A-network. Since, as we know, both path consistency and (generalized) arc consistency are incomplete algorithms for BA-networks, a backtracking search is applied, and to each node of the search tree, (generalized) arc consistency is enforced. The generic schema of a backtracking algorithm can be described as in Alg. 1 [19].

In Alg. 1, the family of sets *Split* plays a key role. When solving the general CSP, without exploiting any tractable fragment of the BA, *Split* can be thought as containing all singletons, each one of them corresponding to a single BA_{basic}-relation. Therefore, the technique to solve a problem P consists of simply choosing a variable, substitute its domain with one of its components creating a new problem P', and recursively solve P'. This algorithm is correct because the search terminates with a node with basic relations only, for which path consistency is a complete method. When a bigger fragment for which enforcing path consistency is known to be complete for consistency, we can take advantage from it by setting *Split* to be the family of its relations; in such a case Alg. 1 stops the search even if the domain of some of the CSP variables is not a singleton, obtaining, *de facto*, a smaller branching factor of the search tree. Unfortunately, establishing if a set can be partitioned into smaller sets taken from some family of sets corresponds to the *set partitioning*

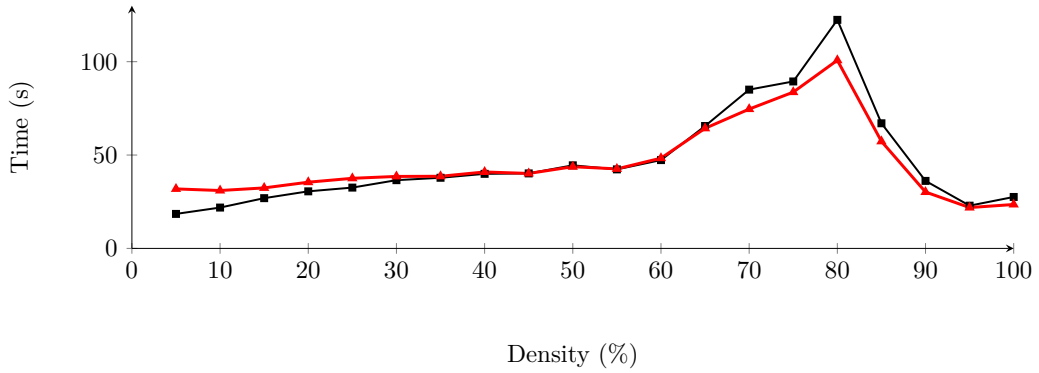


Figure 4 Geometric mean of the computation time to solve the instances varying the density d of the network. Number of nodes $n = 15$, 20 instances solved per point. The black squares show the running time of the backtracking algorithm with $Split = BA_{basic}$, while the red triangles represent the curve with $Split = BA_{convex}$.

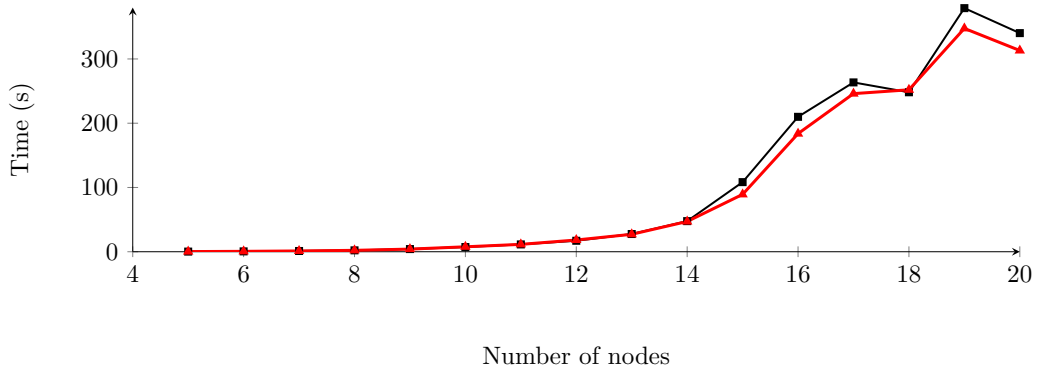


Figure 5 Geometric mean of the computation time to solve the instances varying the number of nodes n of the network. Density $d = 70\%$, 20 instances solved per point. The black squares show the running time of the backtracking algorithm with $Split = BA_{basic}$, while the red triangles represent the curve with $Split = BA_{convex}$.

problem, which is NP-complete in general (in Alg. 1 this step is encoded into a function Partition$(\mathcal{D}, Split)$). In our case, the domain can be partitioned into basic relations (so a polynomial solution exists), but such a solution is inconvenient, as it does not exploit the tractability of the fragment. Since this problem should be solved in every node of the search tree, a quick, although non-optimal, method is mandatory to obtain reasonable efficiency. A first solution would be to pre-compute the (possibly, optimal) solutions of the set-partitioning for each possible subset of the domains; this would generate a table of size $2^{|BA_{basic}|}$, which may fit into the main memory of modern computers, but poses the problem of efficiently accessing to the data structure that contains it. Another option is to use a greedy method to quickly provide a possibly non-optimal partitioning (note, also, that it is not necessary for Partition$(\mathcal{D}, Split)$ to return the complete partitioning, as each of the sets $\mathcal{D}_1, \ldots, \mathcal{D}_p$ can be generated on demand). We decided to follow the latter approach and to store the set of convex relations into a *trie*, which is a data structure whose access time depends on the order in which the elements of a set are stored, but that resulted efficient in practice.

The algorithm was implemented in the constraint logic programming system ECL^{i}PS^e [24], using the CLP(FD) library. To implement the ternary constraints we used the `propia`

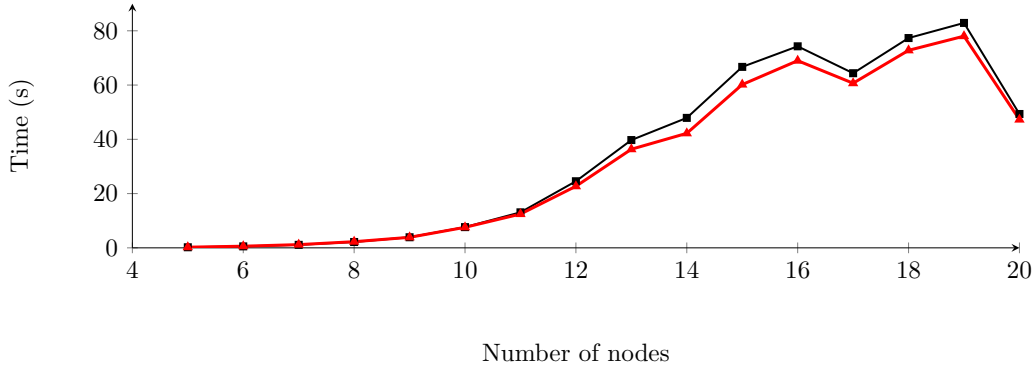


Figure 6 Geometric mean of the computation time to solve the instances varying the number of nodes n of the network. Each point is the geometric average of 140 instances, obtained with density varying from 70% to 100%. The black squares show the running time of the backtracking algorithm with $Split = BA_{basic}$, while the red triangles represent the curve with $Split = BA_{convex}$.

library [16] that provides a general and very declarative way to implement new constraints, although we are aware that more efficient implementations could be possible. The objective of the experimentation was discussing the relative improvement given by the exploitation of the convex fragment, rather then evaluating the absolute performances of our implementation. All experiments were run on a Intel Core i7-3720QM CPU @ 2.60GHz running ECL^{i}PS^e Version 6.1 #224 on Linux Mint 18.1 Serena 64 bits, and using only one core. Timeout was fixed to 10 minutes.

In Fig. 4, we fixed the number of vertices of the A-network to $n = 15$, the probability $p = 1/2$ and varied the constraint density d from 5% to 100%. Each point in the curve represents the geometric mean obtained running 20 instances. Statistically speaking, a small number of *very difficult* networks may be produced in a set of 20 random instances; the neat effect on the computation time of such instances can be softened using the geometric mean rather then the, more common, arithmetic mean [8]. The shape of the curve shows the expected *phase transition*: when the density is low, most of the instances are easily satisfiable, while to high density correspond networks for which proving unsatisfiability is easy. The phase transition occurs at a density around $d = 80\%$, in which both satisfiability and unsatisfiability are hard to prove. The curves in Fig. 4 (in which the red curve represents the performance of the algorithm when the convex fragment is taken into account) show that exploiting the convex fragment is particularly convenient for hard problems near the phase transition, while the overhead that, implicitly, is introduced in such a solution makes it not worth for easily satisfiable problems. In Fig. 5 we fixed the density to a point close to phase transition ($d = 70\%$), and varied the number of nodes in the graph, from 3 to 20. Each point is the geometric mean of 20 runs. At 70% density, most problems under 20 nodes are difficult, and, again, exploiting the convex fragment is convenient with respect to the expected performance. Finally, we investigated the computation time of the two solutions varying the number of nodes (from 5 to 20) independently of the density. To this end, we generated, for each number of nodes, 140 instances with densities varying from 70% to 100%, and considered the geometric mean of the time needed to solve them. In Fig. 6 we show the result of such an analysis, that proves that a certain improvement in computation time exists when the convex fragment is taken into account.

5 Conclusions

Allen's Interval Algebra is one of the most prominent formalisms in the area of qualitative temporal reasoning. However, its applications are naturally restricted to linear flows of time, raising the question of whether one can reason about branching (tree-like) flows of time in a similar manner. We considered, in this paper, the set of 19 branching relations suggested by Ragni and Wölfl, which enjoy the desirable characteristics of being expressible in the language of endpoints without quantification. Ragni and Wölfl have shown that the consistency problem for a network of branching relations is intractable (as expected), while the consistency problem for a network of basic branching relations is polynomial. In clear parallelism with the linear case, we defined the set of convex branching relations, which extends the set of basic branching relations, and we proved that enforcing path consistency of a network of convex relations is sufficient to decide its consistency, effectively providing the first non-trivial tractable (via path consistency) fragment of the branching algebra. As another consequence of this work, we made it possible to treat the consistency problem of a network of constraints as a constraint propagation problem, allowing not only the possibility of quick implementations using well-known libraries, but, also, the possibility of implementing a clever branch-and-bound algorithm for a generic network that exploits the tractability of convex relations as an heuristics. Finally, we tested such a solution, giving experimental evidence of the expected improvement.

The most interesting open problems at the moment include, among other, the question of whether the convex branching algebra is maximal with respect to tractability of the network consistency problem (which seems unlikely) and/or with respect to the possibility of enforcing the minimal labels of a network via path consistency, the question of whether other popular and well-behaved fragments of the interval algebra in the linear case can be generalized to the branching setting preserving their computational behaviour, and the question of whether efficient enumerating algorithms can be devised for the branching case as it has been done in the linear case. We already know that there is a set of branching point relations which could be considered a natural generalization of convex branching point relations and whose minimal labels problem cannot be solved by path consistency; however, fragments of the branching point algebra that strictly include BA_{convex} and for which the minimal labels of a network can be enforced by path consistency are still possible.

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 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

4 D. Bresolin, A. Montanari, and P. Sala. An optimal tableau for right propositional neighborhood logic over trees. In *Proc. of TIME 2008: 15th International Symposium on Temporal Representation and Reasoning*, pages 110–117. IEEE, 2008.

5 M. Broxvall. The point algebra for branching time revisited. In *Proc. of KI2001: Advances in Artificial Intelligence*, volume 2174 of *Lecture Notes in Artificial Intelligence*, pages 106–121. Springer, 2001.

6 J.F. Condotta, D. D'Almeida, C. Lecoutre, and L. Saïs. From qualitative to discrete constraint networks. In *Proc. of the Workshop on Qualitative Constraint Calculi held with KI 2006*, pages 54–64, 2006.

7 W. Conradie, S. Durhan, and G. Sciavicco. An integrated first-order theory of points and intervals: Expressive power in the class of all linear orders. In *Proc. of TIME 2012: 19th International Symposium on Temporal Representation and Reasoning*, pages 47–51. IEEE, 2012.

8 M.J. Dent and R.E. Mercer. A new model of hard binary constraint satisfaction problems. In *Proc. of AI 96: 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, volume 1081 of *Lecture Notes in Computer Science*, pages 14–25. Springer, 1996.

9 I. Düntsch, H. Wang, and S. McCloskey. Relations algebras in qualitative spatial reasoning. *Fundamenta Informaticae*, 39(3):229–248, 1999.

10 S. Durhan and G. Sciavicco. Allen-like theory of time for tree-like structures. *Information and Computation*, 259(3):375–389, 2018.

11 E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. MIT Press, 1990.

12 R. Hirsch. Expressive power and complexity in algebraic logic. *Journal of Logic and Computation*, 7(3):309–351, 1997.

13 P. Jonsson and V. Lagerkvist. An initial study of time complexity in infinite-domain constraint satisfaction. *Artificial Intelligence*, 245:115–133, 2017.

14 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.

15 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.

16 T. Le Provost and M. Wallace. Generalized constraint propagation over the CLP scheme. *Journal of Logic Programming*, 16(3):319–359, 1993.

17 A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

18 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.

19 B. Nebel. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class. *Constraints*, 1(3):175–190, 1997.

20 M. Ragni and S. Wölfl. 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.

21 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.

22 J. Renz and B. Nebel. Efficient methods for qualitative spatial reasoning. *Journal of Artifiacial Intelligence Resoning*, 15:289–318, 2001.

23 J. Renz and B. Nebel. Qualitative spatial reasoning using constraint calculi. In M. Aiello, I. Pratt-Hartmann, and J.F.A.K. van Benthem, editors, *Handbook of Spatial Logic*, pages 161–215. Springer, 2007.

24 J. Schimpf and K. Shen. Ecl$^{\text{i}}$ps$^{\text{e}}$ - from LP to CLP. *Theory and Practice of Logic Programming*, 12(1-2):127–156, 2012.

25 P. van Beek and R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.

26 M. B. Vilain and H.A. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. of AAAI 1986: 5th National Conference on Artificial Intelligence*, pages 377–382, 1986.

A Game-Theoretic Approach to Timeline-Based Planning with Uncertainty

Nicola Gigante[1]
University of Udine, Udine, Italy
gigante.nicola@spes.uniud.it

Angelo Montanari[1]
University of Udine, Udine, Italy
angelo.montanari@uniud.it

Marta Cialdea Mayer[1]
University of Roma Tre, Rome, Italy
cialdea@ing.uniroma3.it

Andrea Orlandini
National Research Council, Rome, Italy
andrea.orlandini@istc.cnr.it

Mark Reynolds[2]
The University of Western Australia, Perth, Australia
mark.reynolds@uwa.edu.au

Abstract

In timeline-based planning, domains are described as sets of independent, but interacting, components, whose behaviour over time (the set of timelines) is governed by a set of temporal constraints. A distinguishing feature of timeline-based planning systems is the ability to integrate planning with execution by synthesising control strategies for *flexible plans*. However, flexible plans can only represent *temporal uncertainty*, while more complex forms of nondeterminism are needed to deal with a wider range of realistic problems. In this paper, we propose a novel game-theoretic approach to timeline-based planning problems, generalising the state of the art while uniformly handling temporal uncertainty and nondeterminism. We define a general concept of timeline-based *game* and we show that the notion of winning strategy for these games is strictly more general than that of control strategy for dynamically controllable flexible plans. Moreover, we show that the problem of establishing the existence of such winning strategies is decidable using a doubly exponential amount of space.

2012 ACM Subject Classification Computing methodologies → Planning under uncertainty

Keywords and phrases Timeline-based planning with uncertainty, strategic games, decidability

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.13

Related Version A full version of the paper is available at [17], `https://arxiv.org/abs/1807.04837`.

[1] N. Gigante and A. Montanari's work was mainly done while on leave at *The University of Western Australia*. The work was supported by the GNCS project *Formal methods for verification and synthesis of discrete and hybrid systems* (N. Gigante, A. Montanari, M. Cialdea Mayer) and the PRID project *ENCASE - Efforts in the uNderstanding of Complex interActing SystEms* (N. Gigante, A. Montanari).
[2] Mark Reynolds acknowledges the support of Australian Research Council funding (DP140103365).

1 Introduction

In the timeline-based approach to planning [21], the world is modelled as a set of independent, but interacting, components, whose behaviour over time, the *timelines*, is governed by a set of temporal constraints. This approach differs from the classical *action-based* planning, relying on PDDL [13], for its more declarative nature and its focus on temporal reasoning (see, *e.g.* [9, 11, 14, 15, 16, 21]). Timeline-based systems have been successfully deployed in a number of complex scenarios, ranging from space operations [3, 7, 21] to manufacturing [5, 24]. An important feature of timeline-based planning systems is their ability to integrate planning with execution by means of *flexible plans*, which represent envelopes of possible solutions that differ in the execution times and/or the duration of tasks. Flexible plans allow the controller to handle the *temporal uncertainty* involved in dealing with partially controllable elements and the external environment. Cialdea Mayer *et al.* [9] rigorously defined the concept of timeline-based planning specification as well as *dynamically controllable* flexible plans, which can be executed guaranteeing to satisfy the problem constraints while reactively handling any *temporal uncertainty* in the uncontrollable behaviour. A technique for synthesising dynamic control strategies is shown in [8].

Other forms of uncertainty, such as *nondeterminism* (*i.e.*, *which* tasks the environment chooses to perform), are not supported: even for external variables, completely controlled by the environment, their evolution is known up to temporal uncertainty only. This choice to focus on temporal reasoning and temporal uncertainty is coherent with the history and scope of timeline-based systems. However, it is not completely reflected into the grammar of modelling languages used in timeline-based systems, which are expressive enough to model complex scenarios that require the system to handle non-temporal nondeterminism. In such cases, current systems often employ a *re-planning* stage as part of their execution cycle (see, *e.g.* [24]): any mismatch between the expected and actual behaviours of the environment results into a revision of the flexible plan, which then can resume execution. Unfortunately, the cost of such a re-planning phase may be incompatible with the requirements of real-time execution and, more importantly, if a wrong choice is made by the original flexible plan, the re-planning might happen too late to be able to recover a controllable state of the system. Hence, knowledge engineers have to explicitly account for this problem if they want to avoid unnecessary failures and costly re-planning during execution, which make the system less effective and more complex to use.

Nondeterministic planning issues have been extensively investigated within the action-based planning framework following different approaches such as, for instance, reactive planning systems [4], deductive planning [23], POMDP [18], and model checking [12]. More recently, fully observable nondeterministic (FOND) planning problems have been addressed [19, 20] also considering temporally extended goals [6, 22]. However, action-based planning does not support flexible plans and temporal uncertainty, and it does not take into account controllability issues. Recently, SMT-based techniques have been exploited to deal with uncontrollable durations in strong temporal planning [10]; however, dynamic controllability issues are not addressed.

This paper defines the novel concept of *timeline-based planning game*, a game-theoretic generalisation of the timeline-based planning problem with uncertainty, which uniformly treats both temporal uncertainty and general nondeterminism. In these games, the controller tries to satisfy the given temporal constraints no matter what the choices of the environment are. We compare the proposed games with the current approaches based on flexible plans. In particular, we show how current timeline-based modelling languages can express problems

that, only seeming to involve temporal uncertainty at first, in fact model scenarios which would require the controller to handle non-temporal nondeterminism. We show that these problems do not admit dynamically controllable flexible plans (as defined in [9]), but do admit winning strategies when seen as instances of timeline-based games. A study of the decidability and complexity of the problem of establishing the existence of a winning strategy for a given timeline-based planning game concludes the paper.

The paper is structured as follows. Section 2 briefly recaps the basic definitions, and Section 3 discusses the limitations of the approach employed in state-of-the-art timeline-based planning systems. Then, Section 4 defines timeline-based planning games, and shows its greater generality with respect to the current approach. Finally, Section 5 addresses decidability and complexity issues. Section 6 concludes discussing future developments.

2 Timeline-based planning

This section introduces timeline-based planning and describes the state of the art of the field with regards to how uncertainty is handled by current timeline-based systems. As a representative of the modelling languages used by existing systems, we chose the formal language introduced in [9]. We first introduce the basic concepts of the framework, without considering uncertainty, as studied in [15, 16]; then, we add uncertainty to the picture and discuss how it is handled by current systems.

2.1 Basic definitions

State variables are the basic building blocks of the timeline-based planning framework.

▶ **Definition 1** (State variables). A *state variable* is a tuple $x = (V_x, T_x, D_x, \gamma_x)$, where:

- V_x is the *finite domain* of the variable;
- $T_x : V_x \to 2^{V_x}$ is the *value transition function*, which maps each value $v \in V_x$ to the set of values that can follow it;
- $D_x : V_x \to \mathbb{N}^+ \times (\mathbb{N}^+ \cup \{+\infty\})$ is a function that maps each $v \in V_x$ to the pair $(d_{min}^{x=v}, d_{max}^{x=v})$ of the minimum and maximum duration of any interval over which $x = v$;
- $\gamma_x : V_x \to \{\mathsf{c}, \mathsf{u}\}$ is a function called *controllability tag* (see Section 2.2).

Which value is taken by a state variable over a specified time interval is described by means of *tokens*. The behaviour of a state variable over time is modelled by a finite sequence of tokens, called a *timeline*.

▶ **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 $x = (V_x, T_x, D_x, \gamma_x)$ is a finite sequence $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ of tokens for x.

For any token $\tau_i = (x, v_i, d_i)$ in a timeline $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ we can define the functions $\mathsf{start\text{-}time}(\mathsf{T}, i) = \sum_{j=1}^{i-1} d_j$ and $\mathsf{end\text{-}time}(\mathsf{T}, i) = \mathsf{start\text{-}time}(\mathsf{T}, i) + d_i$, hence mapping each token to the corresponding time interval $[\mathsf{start\text{-}time}, \mathsf{end\text{-}time})$ (right extremum excluded). As an example, the time interval associated with the token $\tau_1 = (x, 2, 5)$ is $[0, 5)$. When there is no ambiguity, we write $\mathsf{start\text{-}time}(\tau_i)$ and $\mathsf{end\text{-}time}(\tau_i)$ to denote, respectively, $\mathsf{start\text{-}time}(\mathsf{T}, i)$ and $\mathsf{end\text{-}time}(\mathsf{T}, i)$. The *horizon* of a timeline $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ is defined as $\mathcal{H}(\mathsf{T}) = \mathsf{end\text{-}time}(\tau_k)$. A timeline T can be empty, in which case we define its horizon as $\mathcal{H}(\mathsf{T}) = 0$.

The problem domain and the goal are modelled by a set of temporal constraints, called *synchronisation rules*. For the sake of space, we do not provide a detailed account of their syntax. Informally, each synchronisation rule has the following form:

$$rule := a_0[x_0 = v_0] \longrightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k, \ with$$
$$\mathcal{E}_i := \exists a_1[x_1 = v_1]a_2[x_2 = v_2] \ldots a_n[x_n = v_n] \,.\, \mathcal{C}$$

where $x_0, \ldots, x_n$ are state variables and $v_0, \ldots, v_n$ are values, with $v_i \in V_{x_i}$ for all i. Each rule thus 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 be empty (T), 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 $a \leq^{e_1,e_2}_{[l,u]} b$, where a and b are token names, $e_1, e_2 \in \{\mathsf{s}, \mathsf{e}\}$, $l \in \mathbb{N}$, and $u \in \mathbb{N} \cup \{+\infty\}$. Intuitively, each atom relates the start (s) or end (e) of the two tokens, and l and u are respectively a lower and upper bound to the distance between the two endpoints. Pointwise atoms relating a single endpoint with a specific point in time are also possible, *e.g.*, $a \leq^{\mathsf{s}}_{[l,u]} t$. An atom $a \leq^{e_1,e_2}_{[l,u]} b$ is *bounded* if $u \neq +\infty$, and *unbounded* otherwise. With these basic atomic relations, one can express all the Allen's relations over time intervals [1]. As an example, one can define $a \,\mathsf{meets}\, b$ as $a \leq^{\mathsf{e},\mathsf{s}}_{[0,0]} b$, or $a \,\mathsf{during}\, b$ as $a \leq^{\mathsf{s},\mathsf{s}}_{[0,+\infty]} b \wedge b \leq^{\mathsf{e},\mathsf{e}}_{[0,+\infty]} a$. Moreover, one can constrain the duration of tokens, *e.g.*, writing $\mathsf{duration}(a) = k$ or $\mathsf{duration}(a) \geq k$ as a shorthand for, respectively, $a \leq^{\mathsf{s},\mathsf{e}}_{[k,k]} a$ and $a \leq^{\mathsf{s},\mathsf{e}}_{[k,+\infty]} a$. Moreover, disjunctions in synchronisation rules allows one to express some forms of conditional (if/then/else) statements.

In the simplest setting, 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. A solution to such a problem is simply a set of timelines that satisfy the rules.

▶ **Definition 3** (Timeline-based planning problem)**.** A *timeline-based planning problem* is a pair $P = (SV, S)$, where SV is a set of state variables and S is a set of synchronisation rules over SV.

▶ **Definition 4** (Solution plan)**.** A *scheduled solution plan* for a problem $P = (SV, S)$ is a set Γ of scheduled timelines, one for each $x \in SV$, such that $v_{i+1} \in T_x(v_i)$ and $d^{x=v}_{min} \leq d_i \leq d^{x=v}_{max}$ for all tokens $\tau_i = (x_i, v_i, d_i) \in \mathsf{T}_x$, and all the rules in S are satisfied.

Note that the controllability tags γ_x of state variables are ignored in Definition 4, as it only considers the problem of satisfying the synchronisation rules and the issues arising from the execution of the plan are not considered. Even without considering any form of uncertainty, timeline-based planning is already quite a hard problem. Indeed, we know from [16] that the problem of deciding whether there exists a solution plan for a given timeline-based planning problem is EXPSPACE-complete.

As an example, consider Figure 1, which shows a possible solution for a problem with two state variables, x_{cam} and x_{dir}, that respectively represent the on/off state of a camera and its direction. The transition function $T_{x_{\mathsf{dir}}}$ of the second variable is such that the camera can only stay still or move counterclockwise, that is, $T_{x_{\mathsf{dir}}}(\leftarrow) = \{\leftarrow, \downarrow\}$. The first rule states the system requirement that the camera must remain switched off at least four time steps after each use, to let the components cool down. The objective of the system is that of performing some shoots of a given duration, provided the camera is pointed in the right direction: a shoot downwards lasting two time steps, and a shoot toward left lasting three time steps, in

$$
\begin{aligned}
a[x_{\mathsf{cam}} = \mathsf{on}] &\longrightarrow \exists b[x_{\mathsf{cam}} = \mathsf{off}] \,.\, a \text{ meets } b \wedge \mathsf{duration}(b) \geq 4 \\
\top &\longrightarrow \exists a[x_{\mathsf{cam}} = \mathsf{on}]\; b[x_{\mathsf{dir}} = \downarrow] \,.\, \mathsf{duration}(a) = 2 \wedge a \text{ during } b \\
\top &\longrightarrow \exists a[x_{\mathsf{cam}} = \mathsf{on}]\; b[x_{\mathsf{dir}} = \leftarrow] \,.\, \mathsf{duration}(a) = 3 \wedge a \text{ during } b \\
\top &\longrightarrow \exists a[x_{\mathsf{dir}} = \uparrow] \,.\, a =^{\mathsf{s}} 0
\end{aligned}
$$

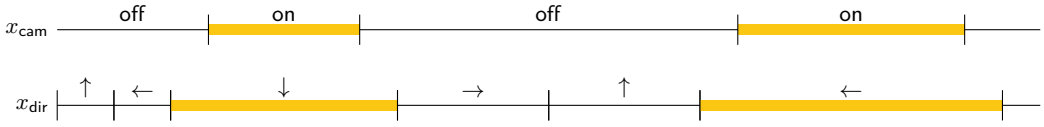


Figure 1 An example of timeline-based planning problem. Two state variables are used to represent the on/off state of a camera x_{cam} and its pointing direction x_{dir}. The transition function of x_{dir} forces the camera to only move counterclockwise.

an arbitrary order. This objective is encoded by the second and third synchronisation rules, which are triggerless. The last rule expresses the fact that the camera is initially pointed upwards.

2.2 Timeline-based planning with uncertainty

A distinguishing feature of existing timeline-based planning systems is their ability to integrate planning and execution, accounting for the unavoidable uncertainty that comes from the interaction with the environment where the plan is executed. We now recall how timeline-based planning problems with uncertainty are defined in [9].

Two different sources of uncertainty can be represented by this model. The first comes from *external variables*, which are completely under the control of the environment, in contrast to *controlled variables* (also called *planned* variables), which are dealt with the system. The planner is not allowed to decide anything regarding the behaviour of external variables. The second is the duration of *uncontrollable tokens*. The *controllability tag* γ_x, associated with each state variable x, states whether a token where $x = v$ is *controllable* ($\gamma_x(v) = \mathsf{c}$) or *uncontrollable* ($\gamma_x(v) = \mathsf{u}$). The execution of uncontrollable tokens is planned and performed by the system, but their duration cannot be known in advance, *e.g.*, one may go shopping without knowing how much time he/she will have to wait at the counter. In both cases, only *temporal* uncertainty is considered in the current approach.

To deal with the uncertainty inherent in the execution of the plan, timeline-based planning systems make use of the concept of *flexible timeline*.

▶ **Definition 5** (Flexible timeline)**.** A *flexible token* is a tuple $\tau = (x, v, [e, e'], [d, d'])$, where x is a state variable, $v \in V_x$, $[e, e'] \in \mathbb{N} \times \mathbb{N}$ is the interval of possible token *end times*[3], and $[d, d'] \in \mathbb{N}^+ \times \mathbb{N}^+$ is the interval of possible token *durations*.

A *flexible timeline* for a state variable $x = (V_x, T_x, D_x, \gamma_x)$ is a finite sequence $\mathsf{T}_x = \langle \tau_1, \ldots, \tau_k \rangle$ of flexible tokens for x where $[e_1, e'_1] = [d_1, d'_1]$, and $[e_i, e'_i] \subseteq [e_{i-1} + d_i, e'_{i-1} + d'_i]$ for all $1 < i \leq k$.

A flexible timeline represents a set of different timelines which differ in the precise timings of the described events. Tokens, timelines, and plans will also be referred as *scheduled* tokens, timelines and plans, to better differentiate them from flexible ones. A scheduled timeline

[3] Flexible tokens report their end times rather than their start times because in this way a flexible timeline can precisely constrain its horizon.

$\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$ is an *instance* of a flexible timeline $\mathsf{T}' = \langle \tau'_1, \ldots, \tau'_k \rangle$ if for each $\tau_i = (x, v, d)$ and $\tau'_i = (x', v', [e, e'], [d, d'])$, we have $x = x'$, $v = v'$, $d \in [d, d']$, and $\mathsf{end\text{-}time}(\tau_i) \in [e, e']$.

Flexibility can be naturally lifted from timelines to plans. The notion of *flexible plan* is formally defined as follows.

▶ **Definition 6** (Flexible plan)**.** A *flexible plan* over a set of state variables SV is a pair $\Pi = (\Gamma, \mathcal{R})$, where Γ is a set of *flexible timelines*, exactly one for each $x \in SV$, and $\mathcal{R}$ is a set of *atoms* over the tokens occurring in Γ.

Given a flexible plan $\Pi = (\Gamma, \mathcal{R})$, a scheduled plan Γ' is an *instance* of Π if the timelines in Γ' are instances of those in Γ, and they satisfy the atoms in $\mathcal{R}$.

A flexible plan can be viewed as a tentative set of solutions to a planning problem where the precise timing of execution and the duration of tokens are chosen during execution. The set of atoms that comes together with the set of flexible timelines allow the plan to specify additional constraints over the tokens that compose the timelines. Flexible plans can in particular be used to describe the expected behaviour of external variables, of which one may know the future evolution only up to some temporal uncertainty. It is worth pointing out that a flexible plan is *unconditional*, *i.e.*, a single plan is committed to a specific sequence of state variable values, and the only freedom left concerns token durations.

A timeline-based planning problem with uncertainty is formally defined as follows.

▶ **Definition 7** (Timeline-based planning problem with uncertainty)**.** A *timeline-based planning problem with uncertainty* is a tuple $P = (SV_C, SV_E, S, \mathcal{O})$, where:

- SV_C and SV_E are the sets of, respectively, the *controlled* and the *external* variables;
- S is a set of synchronisation rules over $SV_C \cup SV_E$;
- the *observation* $\mathcal{O} = (\Gamma_E, \mathcal{R}_E)$ is a flexible plan over SV_E specifying the behaviour of external variables.

Definition 7 differs from Definition 3 in two main respects: it splits the set of variables into controlled and external ones and it includes a flexible plan describing the temporally uncertain behaviour of external variables.

To solve the problem, one has to find a set of flexible timelines for the controlled variables such that the rules can be satisfied by a suitable set of instances.

▶ **Definition 8** (Flexible solution plan)**.** Let $P = (SV_C, SV_E, S, \mathcal{O})$, with $\mathcal{O} = (\Gamma_E, \mathcal{R}_E)$ be a timeline-based planning problem with uncertainty. A flexible plan $\Pi = (\Gamma, \mathcal{R})$ over $SV_C \cup SV_E$ is a *flexible solution plan* for P if:

1. Π agrees with $\mathcal{O}$, that is, $\Gamma_E \subseteq \Gamma$ and $\mathcal{R}_E \subseteq \mathcal{R}$;
2. the plan does not restrict the duration of uncontrollable tokens, that is, for any $\mathsf{T} \in \Gamma$ and any token $\tau = (x, v, [e, e'], [d, d']) \in \mathsf{T}$, if $\gamma_x(v) = \mathsf{u}$, then $d = d_{min}^{x=v}$ and $d' = d_{max}^{x=v}$;
3. any *instance* of Γ is a scheduled solution plan for the timeline-based planning problem $P' = (SV_C \cup SV_E, S)$, and there exists at least one such instance.

Note that, despite the name, which is borrowed from [9], the observation $\mathcal{O}$ is rather an *a priori* description of the environment behaviour, which is supposed to be completely known up to the given temporal uncertainty. Usual definitions of planning problems involve the specification of a maximum bound on the *horizon* of the solution plans. For the sake of generality, we omit this parameter, as it can be expressed by suitable synchronisation rules.

According to Item 3 of Definition 8, any instance of a flexible plan satisfies the synchronisation rules of the problem. However, there is no guarantee that one such instance exists for each possible instance of the external timelines. In other words, Definition 8 does

not guarantee that a flexible solution plan can be executed in any possible scenario. Thus, a control strategy is needed to determine how to schedule controllable tasks. Because of space concerns, we cannot present all the details of the definition of control strategy, which is thoroughly illustrated in [9]. Informally, it can be thought of as a function σ which chooses how to schedule the start time of tokens for controlled variables, and the end time of controllable tokens, during execution. A flexible solution plan Π is said to be *weakly controllable* if for each possible schedule of tokens for external variables and of uncontrollable tokens, there is a control strategy σ such that following σ during execution results into an instance of Π. It is said to be *strongly controllable* if, conversely, a single control strategy σ exists which results into an instance of Π whatever the schedule of the endpoints under its control by the environment is. Finally, it is said to be *dynamically controllable* if the controlled endpoints can be scheduled by taking into account, at any given time, only the past history of the execution in such a way that an instance of Π is obtained. Given its generality and wider applicability, dynamic controllability is definitely the most interesting form of controllability. Notice that these concepts, whose definition in the context of flexible plans for timeline-based planning is given in [9], have analogous counterparts in the context of temporal networks [25].

Timeline-based systems which aim at handling both planning and execution cannot simply produce flexible plans, but have to ensure a chosen degree of controllability of the produced plans. As an example, the PLATINUm system [24] employs a two-phase process where a flexible plan is first produced and then checked for the existence of a dynamic control strategy. Dynamic controllability of a flexible plan can be checked, for instance, via a reduction to timed game automata [8]. Since uncontrollable flexible plans are not suitable to be executed, the problem is considered to be solved only when a dynamically controllable flexible solution plan is found, together with its dynamic control strategy.

3 Limitations of the current approach

The focus of timeline-based systems on temporal reasoning and temporal uncertainty clearly emerges from the previous section. This focus has its roots in the history of the paradigm and the typical application scenarios where timeline-based systems have been employed. The exclusive focus on temporal uncertainty is especially evident in the treatment of external variables: their behaviour is supposed to be completely known excepting only the precise timing of specific events. Consider, for example, a satellite in orbit doing some measurements and transmitting the results back to Earth. In such a domain, external variables might be used to represent visibility windows where the different ground stations can be reached by the satellite. The precise timing of those windows is uncertain, but everything else is known (even months) in advance. Nevertheless, to handle the case of a mismatch between the expected and the observed behaviour of the environment, systems such as PLATINUm employ a feedback loop where a *failure manager* is in charge of triggering, if needed, a *re-planning* (or plan repair) phase, which should produce a new flexible plan, with a suitable control strategy, taking into account the newly acquired observations (the name of the $\mathcal{O}$ component in Definition 7 comes from this scenario).

In contrast to domains as the above one, other applications might require to re-plan more frequently. As an example, in robotics scenarios such as those discussed in [24], where the planned system interacts with a human agent, one cannot hope to represent with temporal uncertainty alone all the possible variability of the behaviour of the external environment. In these scenarios, most often re-plans get triggered to handle the unpredicted outcomes

of generally nondeterministic choices that the external agents can make, rather than to fix problems in the domain model. This observation motivates us in proposing an interpretation of timeline-based problems that is able to handle both temporal uncertainty and general nondeterminism, extending and generalising the current approach based on flexible plans.

As a matter of fact, it turns out that the domain description languages commonly in use, here exemplified by the formal syntax defined in Section 2, can easily express scenarios where the need to handle general nondeterminism arise in problems which apparently only involved temporal uncertainty. To see how this may happen, consider a simple timeline-based planning problem with uncertainty $P = (SV_C, SV_E, S, \mathcal{O})$, with a single state variable x, with $V_x = \{v_1, v_2, v_3\}$, $SV_E = \varnothing$, and S consisting of the following synchronisation rules:

$$a[x = v_1] \longrightarrow \exists b[x = v_2] \,.\, a \leq^{\mathsf{e},\mathsf{s}}_{[0,0]} b \wedge a \leq^{\mathsf{s},\mathsf{e}}_{[0,5]} a \vee \exists c[x = v_3] \,.\, a \leq^{\mathsf{e},\mathsf{s}}_{[0,0]} c \wedge a \leq^{\mathsf{s},\mathsf{e}}_{[6,10]} a$$
$$\top \longrightarrow \exists a[x = v_1] \,.\, a =^{\mathsf{s}} 0$$

Suppose that $D_x(v) = [1, 10]$ for all $v \in V_x$, and that $\gamma_x(v_1) = \mathsf{u}$ and $\gamma_x(v_2) = \gamma_x(v_3) = \mathsf{c}$, that is, tokens where $x = v_1$ are uncontrollable. The rules require the execution to start with a token where $x = v_1$, followed by a token where either $x = v_2$ or $x = v_3$ depending on the duration of the first token. This scenario is, intuitively, trivial to control. The system must execute $x = v_1$ as a first token due to the second rule. Then, the environment controls its duration, and the system simply has to wait for the token to end, and then execute either $x = v_2$ or $x = v_3$ depending on how long the first token lasted. However, there are no flexible plans that represent this simple strategy, since each given plan must fix the value of every token in advance. To guarantee the satisfaction of the rules, the choice of which value to assign to x on the second token must be made during the execution, but this is not possible because of the exclusively sequential nature of flexible plans. In this case, therefore, the problem would be considered as unsolvable, even if the goals stated by the rules seem simple to achieve.

The issues discussed here come from the lack of a proper support for general nondeterminism in the framework of flexible plans. However, the last example shows that this is not just a missing feature of current systems, but rather a class of scenarios that can be easily modelled by (the syntax of) timeline-based description languages but whose solutions are not captured by the commonly considered semantics. The *timeline-based planning games* defined in the next section provide a clean way to express the solution to this kind of scenario, providing a semantics to timeline-based planning problems with uncertainty capable of modelling both temporal uncertainty and general nondeterminism in a uniform way. Moreover, they handle the external variables in the most general way, without assuming any *a priori* knowledge of their future behaviour.

4 Timeline-based planning games

Let us now introduce *timeline-based planning games*. They generalise dynamic control strategies for flexible plans while suitably handling the limitations discussed in Section 3.

▶ **Definition 9** (Timeline-based planning game)**.** A *timeline-based planning game* is a tuple $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$, where SV_C and SV_E are the sets of, respectively, the *controlled* and the *external* variables, and $\mathcal{S}$ and $\mathcal{D}$ are two sets of synchronisation rules, respectively called *system* and *domain* rules, involving variables from both SV_C and SV_E.

Intuitively, a timeline-based planning game $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ is a turn-based, two-player game played by the controller, *Charlie*, and the environment, *Eve*. By playing the game, the players progressively build the timelines of a *scheduled plan* (see Definition 4). At

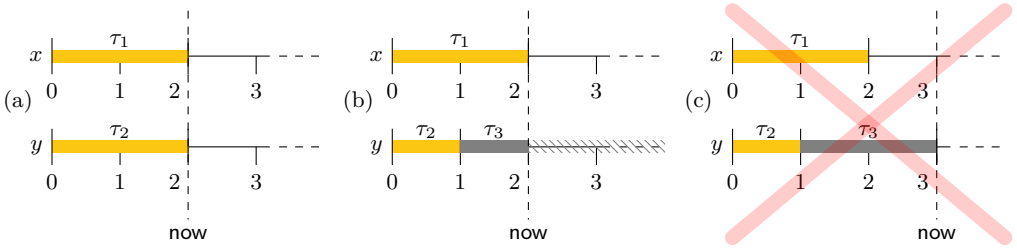


Figure 2 Examples of partial plans: (a) two tokens for variables x and y where $\mathsf{end\text{-}time}(\tau_1) = \mathsf{end\text{-}time}(\tau_2) = 2$ and $\mathsf{now} = 2$; (b) $\mathsf{end\text{-}time}(\tau_1) = 2$, $\mathsf{end\text{-}time}(\tau_2) = 1$, while τ_3 continues; (c) incorrect partial plan, because τ_3 continues over the end of τ_1 and a successor for τ_1 is unspecified.

each round, each player makes a move deciding which tokens to start and/or to stop and at which time. Both players are constrained by the set $\mathcal{D}$ of *domain* rules, which describe the basic rules governing the world. Domain rules replace and generalise the *observation* $\mathcal{O}$ of Definition 7, allowing one to freely model the interaction between the system and the environment. They are not intended to be *Eve*'s (or *Charlie*'s) *goals*, but, rather, a background knowledge about the world that can be assumed to hold at any time. Since neither player can violate $\mathcal{D}$, the strategy of each player may safely assume the validity of such rules. In addition, *Charlie* is responsible for satisfying the set $\mathcal{S}$ of *system* rules, which describe the rules governing the controlled system, including its goals. *Charlie* wins if, assuming *Eve* behaves according to the domain rules, he manages to construct a plan satisfying the system rules. In contrast, *Eve* wins if, while satisfying the domain rules, she prevents *Charlie* from winning, either by forcing him to violate some system rule, or by indefinitely postponing the fulfilment of his goals.

Let us now formally describe the way in which a play of a (timeline-based) planning game evolves. First of all, we observe that at any given time during the play, the plan will be partially built, waiting for some tokens to be completed. A *partial plan* is a plan where the last token on each timeline may be unfinished (*open* token). A timeline whose last token is open is called an *open* timeline.

▶ **Definition 10** (Open timeline)**.** Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a planning game and let $SV = SV_C \cup SV_E$. An *open token* for G is a pair $\tau = (x, v)$, where $x \in SV$ and $v \in V_x$. An *open timeline* for $x \in SV$ is a non-empty finite sequence of tokens $\mathsf{T} = \langle \tau_1, \ldots, \tau_{k-1}, \tau_k \rangle$, where $\langle \tau_1, \ldots, \tau_{k-1} \rangle$ is a scheduled timeline for x and $\tau_k = (x, v_k)$ is an open token.

We will refer to tokens and timelines as defined in Definition 2 as *closed* tokens and *closed* timelines, respectively. In an open timeline $\mathsf{T} = \langle \tau_1, \ldots, \tau_k \rangle$, only $\mathsf{start\text{-}time}(\tau_k)$ is defined for its last open token τ_k: $\mathsf{start\text{-}time}(\tau_0) = 0$ and $\mathsf{start\text{-}time}(\tau_i) = \mathsf{end\text{-}time}(\tau_{i-1})$ for $i > 1$. Recall that $\mathcal{H}(\mathsf{T})$ for a closed timeline is the $\mathsf{end\text{-}time}$ of its last token and that $\mathcal{H}(\mathsf{T}) = 0$ for empty timelines. For an open timeline $\mathsf{T} = \langle \tau_1, \ldots, \tau_{k-1}, \tau_k \rangle$, we define its horizon as the $\mathsf{end\text{-}time}$ of its last *closed* token, *i.e.*, $\mathcal{H}(\mathsf{T}) = \mathsf{end\text{-}time}(\tau_{k-1})$ (which is equal to $\mathsf{start\text{-}time}(\tau_k)$).

▶ **Definition 11** (Partial plan)**.** Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a planning game and let $SV = SV_C \cup SV_E$. A *partial plan* for G is a pair $\Pi = (\Gamma, \mathsf{now})$, where Γ is a set of timelines, either open or closed, one for each $x \in SV$, and $\mathsf{now} \in \mathbb{N}$ is the current time, such that:

1. $\mathcal{H}(\mathsf{T}) < \mathsf{now}$ for any *open* timeline $\mathsf{T} \in \Gamma$;
2. $\mathcal{H}(\mathsf{T}) = \mathsf{now}$ for any *closed* timeline $\mathsf{T} \in \Gamma$;

For a partial plan $\Pi = (\Gamma, \mathsf{now})$, we write $\mathsf{T} \in \Pi$ to mean $\mathsf{T} \in \Gamma$, and we define $\mathcal{H}(\Pi) = \max_{\mathsf{T} \in \Pi} \mathcal{H}(\mathsf{T})$ and $\mathsf{now}(\Pi) = \mathsf{now}$. The set of all possible partial plans for a given game G is denoted as $\mathbf{\Pi}_G$, or simply $\mathbf{\Pi}$ where G is understood.

Definition 11 implies that time cannot advance after the end of a token without specifying its successor, and ensures that the partial plan has been built only up to now. Figure 2 shows an example of invalid partial plan where a token continues after the end of another and the successor of the latter is not specified, forming an invalid gap in the description of the timelines. It also implies that if any empty, thus closed, timeline is present, then all timelines are empty. Thus, a unique well-defined *empty* partial plan exists, which we will denote by $\Pi_\varnothing$, with all empty timelines and $\mathsf{now}(\Pi_\varnothing) = 0$.

Players incrementally build a partial plan by extending the initially empty partial plan $\Pi_\varnothing$. This is done by means of actions that specify which tokens to start and/or end.

▶ **Definition 12** (Action). Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a planning game and let $SV = SV_C \cup SV_E$. An *action* α for G is a term of the form $\mathsf{start}(x, v)$ or $\mathsf{end}(x, v)$, where $x \in SV$ and $v \in V_x$. The set of actions is partitioned into the set A_C of *Charlie*'s actions and the set A_E of *Eve*'s actions. An action belongs to A_C (resp., A_E) if it is of the form $\mathsf{start}(x, v)$, for some $x \in SV_C$ (resp., $x \in SV_E$) and $v \in V_x$, or of the form $\mathsf{end}(x, v)$, for some $x \in SV$, $v \in V_x$, and $\gamma_x(v) = \mathsf{c}$ (resp., $\gamma_x(v) = \mathsf{u}$).

Definition 12 can be read as follows. When an action $\mathsf{start}(x, v)$ or $\mathsf{end}(x, v)$ is executed by a player, a token for the variable x, where $x = v$, respectively starts or ends. Ending a task (a token) and starting the next one are two different actions, even if, as it will be precisely stated later, the end of a token must be immediately followed by the start of the next one. Depending on who owns the variable and the involved value, each action can be executed only by a specific player. More precisely, players can start tokens for the variables that they own, and end the tokens that hold values that they control.

It is worth noticing that, in contrast to the original definition of timeline-based planning problems with uncertainty (Definition 7), Definition 12 admits cases where $x \in SV_E$ and $\gamma_x(v) = c$ for some $v \in V_x$, that is, cases where *Charlie* may control the duration of a variable that belongs to *Eve*. This situation is symmetrical to the more common one where *Eve* controls the duration of a variable that belongs to *Charlie*, and there is no need, in our setting, to impose any asymmetry.

Actions are combined into *moves* that can start and/or end multiple tokens at once.

▶ **Definition 13** (Move). Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a timeline-based planning game. A *move* μ is a term of the form $\mathsf{wait}(t)$ or $\mathsf{play}(t, A)$, where $t \in \mathbb{N}$ is the *timestamp* of the move, $A \subseteq \mathsf{A}_C$ or $A \subseteq \mathsf{A}_E$, and for each $x \in SV_C \cup SV_E$, *at most one* action in A involves x. The set of moves is partitioned into the set M_C of *Charlie*'s moves and the set M_E of *Eve*'s moves. A move μ belongs to M_C if it is either of the form $\mu = \mathsf{wait}(t)$ or of the form $\mu = \mathsf{play}(t, A)$ and $A \subseteq \mathsf{A}_C$, while it belongs to M_E only if it is of the form $\mu_E = \mathsf{play}(t, A)$ and $A \subseteq \mathsf{A}_E$.

According to Definition 13, *Charlie* can either execute some set of actions A, by playing a $\mathsf{play}(t, A)$ move, or wait until some given time t, by playing $\mathsf{wait}(t)$. In contrast, *Eve* has a unique kind of move available, *i.e.*, $\mathsf{play}(t, A)$, which executes the actions in A at time t.

We are now ready to introduce the fundamental notion of round.

▶ **Definition 14** (Round). A *round* ρ is a pair of moves $(\mu_C, \mu_E) \in \mathsf{M}_C \times \mathsf{M}_E$. Let Π be a partial plan. A round $\rho = (\mu_C, \mu_E)$ is *applicable* to Π if the following conditions are met.

1. Integrity conditions:
   - **a.** any action of the form $\mathsf{start}(x, v)$ is executed by either μ_C or μ_E if and only if $\mathsf{T}_x \in \Pi$ is a closed timeline;

b. for any action of the form $\mathsf{end}(x, v)$, executed by μ_C or μ_E, either an action $\mathsf{start}(x, v)$ is being executed by either moves, or $\mathsf{T}_x \in \Pi$ is an open timeline $\mathsf{T}_x = \langle \tau_1, \ldots, \tau_k \rangle$ and its open token is $\tau_k = (x, v)$.

2. Timing conditions:

a. if $\mu_C = \mathsf{play}(t_C, A_C)$ and $\mu_E = \mathsf{play}(t_E, A_E)$, then $t_C = t_E = \mathsf{now}(\Pi)$;

b. if $\mu_C = \mathsf{wait}(t_C)$ and $\mu_E = \mathsf{play}(t_E, A_E)$, then $t_C > \mathsf{now}(\Pi)$ and $t_E \leq t_C$.

A single move from either player is *applicable* to Π if there exists at least one move from the other player such that the round combining the two moves is applicable to Π.

The conditions that make a round applicable can be interpreted as follows. Item 1 ensures that the actions played by each move of the round are consistent with the current state of the timelines in the partial plan. In particular, a start action has to follow a end one, and start actions cannot be played on timelines that are already open. Note that both $\mathsf{start}(x, v)$ and $\mathsf{end}(x, v)$ for the same x and v can be played at the same round, possibly by two different players, provided that the timeline was previously closed. In this case, the move builds a token of unitary length (see Definition 15). Item 2 constrains the timings of the moves of a round: if *Charlie* does not wait, then he has to play immediately, that is, $t_C = \mathsf{now}(\Pi)$; otherwise, he can wait until an arbitrary future time point t_C. In both cases, *Eve* must play at a timestamp $t_E \leq t_C$. This restriction has the following meaning. The advancement of time during the game is determined mostly by *Charlie*, who can make it advance one step at the time, by playing at each round, or skip some time steps at once without playing anything, by waiting. In both cases, *Eve*'s moves must specify what the environment is doing, if anything, in the meantime, hence the requirement that $t_E \leq t_C$. If $t_E < t_C$, then time advances only up to t_E, so that at the next round *Charlie* can timely reply to *Eve*'s move.

The next definition specifies the effects of players' moves.

▶ **Definition 15** (Outcome of a round)**.** Let Π be a partial plan and $\rho = (\mu_C, \mu_E)$ be an applicable round. The *outcome* of the application of ρ to Π is a partial plan $\rho(\Pi)$, which is obtained from Π by applying the following ordered sequence of steps:

1. for each $\mu \in \{\mu_C, \mu_E\}$, if $\mu = \mathsf{play}(t, A)$, then, for any $\mathsf{start}(x, v) \in A$, an open token $\tau = (x, v)$ is appended to T_x;

2. for each $\mu \in \{\mu_C, \mu_E\}$, if $\mu = \mathsf{play}(t, A)$, then, for any $\mathsf{end}(x, v) \in A$, the last open token $\tau = (x, v)$ of T_x (possibly added at the previous step) is replaced by a closed token $\tau' = (x, v, d)$, where $d = t - \mathsf{start\text{-}time}(\tau) + 1$;

3. $\mathsf{now}(\rho(\Pi)) = \min(t_C, t_E) + 1$, where either $\mu_C = \mathsf{play}(t_C, A_C)$ or $\mu_C = \mathsf{wait}(t_C)$, and $\mu_E = \mathsf{play}(t_E, A_E)$.

The effects of start and end actions are defined in Items 1 and 2, respectively. The steps are intended to be applied one after the other in order to handle the case where both $\mathsf{start}(x, v)$ and $\mathsf{end}(x, v)$ are played in the same round. Time advances according to Item 3, depending on the timestamps of the moves. Note that if no wait move is played, the new current time corresponds to the horizon of the resulting partial plan.

A play of a planning game is just a sequence of rounds applied to the empty plan $\Pi_\varnothing$.

▶ **Definition 16** (Play)**.** Let G be a planning game and Π_0 be a partial plan (we call it an initial partial plan). A *play* for G from Π_0 is a sequence $\overline{\rho} = \langle \rho_0, \ldots, \rho_k \rangle$ of rounds such that ρ_0 is applicable to Π_0 and ρ_{i+1} is applicable to $\Pi_{i+1} = \rho_i(\Pi_i)$, for $1 \leq i < k$.

Let $\overline{\rho} = \langle \rho_0, \ldots, \rho_k \rangle$ be a play and Π be a partial plan. We denote by $\overline{\rho}(\Pi) = \rho_k(\ldots \rho_0(\Pi))$ the *outcome* of $\overline{\rho}$ applied to Π. Where the initial partial plan is not mentioned, it is understood that the play is applied to $\Pi_\varnothing$. Each non-empty partial plan Π can be *closed* to form a

scheduled plan Π' (see Definition 4) by closing all the open tokens of open timelines at time $\mathsf{now}(\Pi)$. In the following, when the context is clear, we will interchangeably speak of a partial plan as a scheduled plan by implicitly referring to its closure. It can be easily checked that rounds as specified in Definition 13 suffice to build any possible scheduled plan over the game variables, *i.e.*, for any partial plan Π, there exists a play $\overline{\rho}$ such that $\Pi = \overline{\rho}(\Pi_\varnothing)$.

▶ **Definition 17** (Strategy for *Charlie*)**.** A *strategy for Charlie* is a function $\sigma_C : \mathbf{\Pi} \to \mathsf{M}_C$ that maps any given partial plan Π to a move μ_C applicable to Π.

▶ **Definition 18** (Strategy for *Eve*)**.** A *strategy for Eve* is a function $\sigma_E : \mathbf{\Pi} \times \mathsf{M}_C \to \mathsf{M}_E$ that, given a partial plan Π and *Charlie*'s move μ_C applicable to Π, returns the next *Eve*'s move μ_E such that $\rho = (\mu_C, \mu_E)$ is applicable to Π.

A play $\overline{\rho}$ is said to be *played according to* a strategy σ_C for *Charlie*, starting from some initial partial plan Π_0, if $\rho_i = (\sigma_C(\Pi_{i-1}), \mu_E^i)$, for some μ_E^i, for all $0 < i < |\overline{\rho}|$, and to be played according to a strategy σ_E for *Eve* if $\rho_i = (\mu_C^i, \sigma_E(\Pi_{i-1}, \mu_C^i))$, for all $0 < i < |\overline{\rho}|$. For any pair of strategies (σ_C, σ_E) and any $k \geq 0$, there is a unique run $\overline{\rho}_k(\sigma_C, \sigma_E)$ of length k played according to σ_C and σ_E.

It is worth to note that, according to our definition of strategy, *Charlie* can base his decisions only on the previous rounds of the game, not including *Eve*'s move at the current round. Together with the fact that time strictly increases of at least one time unit at each round, this implies that *Charlie* has to wait at least one time unit to react to a move by *Eve*. This models the realistic assumption that the *sense-reason-react* loop of the controller needs a finite amount of time to be executed, and is coherent with the semantics of dynamic control strategies of flexible plans from [9].

Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a planning game. We define two timeline-based planning problems, $P_\mathcal{D} = (SV, \mathcal{D})$ and $P_G = (SV, \mathcal{D} \cup \mathcal{S})$, for G. A partial plan is *admissible* if it is a solution plan for $P_\mathcal{D}$, and *successful* if it is a solution plan for P_G. Similarly a play $\overline{\rho}$ is admissible or successful if its outcome $\overline{\rho}(\Pi_\varnothing)$ is, respectively, admissible or successful.

▶ **Definition 19** (Admissible strategy for *Eve*)**.** A strategy σ_E for *Eve* is *admissible* if for each strategy σ_C for *Charlie*, there is a $k \geq 0$ such that the play $\overline{\rho}_k(\sigma_C, \sigma_E)$ is admissible.

▶ **Definition 20** (Winning strategy for *Charlie*)**.** Let σ_C be a strategy for *Charlie*. We say that σ_C is a *winning strategy* for *Charlie* if for any *admissible* strategy σ_E for *Eve*, there exists an $n \geq 0$ such that the play $\overline{\rho}_n(\sigma_C, \sigma_E)$ is successful.

We say that *Charlie wins* the game G if he has a winning strategy, while *Eve wins* the game if such a strategy does not exist. Let us consider again the problem described in Section 3 to show a simple winning strategy. The problem can be viewed as a game with the rules in Figure 1 belonging to $\mathcal{S}$ and $\mathcal{D}$ empty. After playing $\mathsf{start}(x, v_1)$ at the beginning, *Charlie* only has to wait for *Eve* to play $\mathsf{end}(x, v_1)$, and then play $\mathsf{start}(x, v_2)$ or $\mathsf{start}(x, v_3)$ according to the current timestamp.

A more involved example can be obtained by considering two variables $x \in SV_C$ and $y \in SV_E$, with $V_x = V_y = \{go, stop\}$, unit duration, and rules as follows.

$$\mathcal{S} = \left\{ \begin{array}{c} a[x = stop] \longrightarrow \exists b[y = stop] \,.\, b \leq^{\mathsf{e},\mathsf{s}}_{[0,0]} a \\ \top \longrightarrow \exists a[x = stop] \,.\, \top \end{array} \right\}$$

$$\mathcal{D} = \left\{ \quad \top \longrightarrow \exists a[y = stop] \,.\, \top \quad \right\}$$

Here, *Charlie*'s ultimate goal is to realise $x = stop$, but this can only happen after *Eve* realised $y = stop$. This is guaranteed to happen, since we consider only admissible strategies. Hence, the winning strategy for *Charlie* only chooses $x = go$ until *Eve* chooses $y = stop$, and

then wins by executing $x = stop$. If $\mathcal{D}$ was instead empty, a winning strategy would not exist since a strategy that never chooses $y = stop$ would be admissible. This would therefore be a case where *Charlie* looses because *Eve* can indefinitely postpone his victory.

Let us compare now the concept of *dynamic controllability* of flexible plans, as defined in [9], with the existence of winning strategies for timeline-based planning games, and show the greater generality of the latter concept. Proofs are omitted because of space concerns, but are available in an extended report [17].

The first step is to back the claim of this greater generality. Indeed, it can be shown that for any timeline-based planning problem with uncertainty $P = (SV_C, SV_E, S, \mathcal{O})$ there is an *associated timeline-based planning game* $G_P = (SV_C, SV_E, \mathcal{D}, \mathcal{S})$, such that a dynamically controllable flexible solution plan for P gives us a winning strategy for $\mathcal{G}$. To this aim, we need a way to represent as a game any given planning problem with uncertainty. Intuitively, this can be done by encoding the observations $\mathcal{O}$ into suitable domain rules. The game associated with a problem therefore mimics the exact setting described by the problem. What follows shows that such a game does indeed exist, and that there is a close relationship between its winning strategies and the dynamically controllable flexible plans for the problem.

▶ **Theorem 21.** *Let P be a timeline-based planning problem with uncertainty. If P admits a* dynamically controllable flexible solution plan*, then Charlie has a winning strategy for the associated timeline-based planning game G_P.*

Theorem 21 shows that the proposed framework can represent any timeline-based planning problem with uncertainty, and that the notion of winning strategy for a game subsumes that of dynamically controllable flexible plan. Moreover, it can be seen that the game setting is strictly more expressive, *i.e.*, there are cases where no dynamically controllable flexible plans exist, but a winning strategy can be found. This is the case with the example problem discussed in Section 3, which has an easy winning strategy when seen as a game, while it has no dynamically controllable flexible plan. Therefore, one can prove the following theorem.

▶ **Theorem 22.** *There exists a timeline-based planning problem with uncertainty $P = (SV_C, SV_E, S, \mathcal{O})$ such that there are no dynamically controllable flexible plans for P, but Charlie has a winning strategy for the associated planning game G_P.*

5 Decidability and complexity

A natural question about timeline-based planning games is whether a winning strategy for a given game can effectively be found. In this section, we positively answer the question, showing that a winning strategy, if it exists, can be found with an algorithm that runs in doubly exponential space. Proofs are omitted because of space concerns, but are available in an extended report [17].

The proposed algorithm makes use of the concept of *concurrent game structure* (CGS) [2], in particular, of the specific subclass of *turn-based synchronous game structures* (simply *game structures*, from now on), that, in the case of two players only, can be defined as follows.

▶ **Definition 23** (Turn-based synchronous game structure [2])**.** A *turn-based synchronous game structure* is a tuple $S = (Q_1, Q_2, M_1, M_2, \mathcal{P}, \pi, \delta_C, \delta_E)$, where:

1. Q_1 and Q_2 are the finite sets of *states* belonging to Player 1 and Player 2, respectively;
2. M_1 and M_2 are the finite sets of *moves* available to Player 2 and Player 2, respectively;
3. $\mathcal{P}$ is a finite set of *proposition letters*;
4. $\pi : Q_1 \cup Q_2 \to 2^{\mathcal{P}}$ provides the set of proposition letters that hold at any given state;
5. $\delta_1 : Q_1 \times M_1 \times Q_2$ and $\delta_2 : Q_2 \times M_2 \times Q_1$ are the *transition relations*, that specify the states reachable from a given one by applying a move by either player.

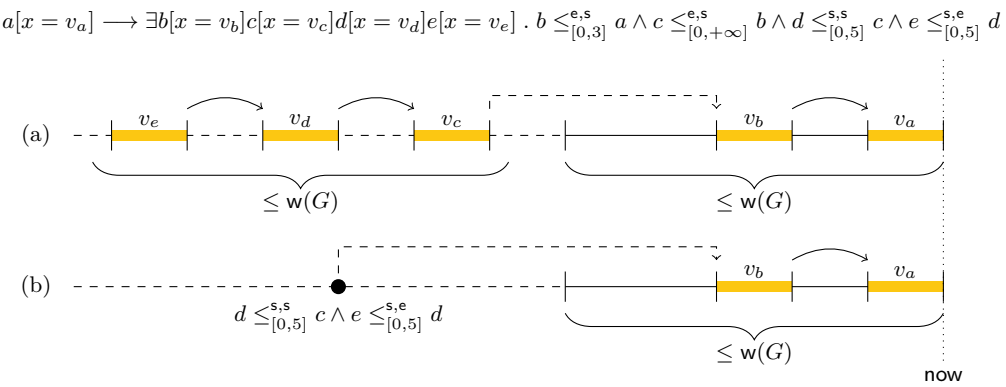


Figure 3 How to represent the distant history of a play of a game G in a compact way: (a) the complete description of the timeline; and (b) the compact description of the distant history.

We proceed in two steps. We first provide a suitable encoding of a timeline-based planning game G into a corresponding game structure, and then we exploit existing machinery for ATL^* model checking [2] to find a winning strategy for G.

Let us focus on the first step. To encode a timeline-based planning game into a game structure, one may think of interpreting partial plans as states, marking with some proposition letter d those which satisfy the domain rules and with w those where the system rules are satisfied. However, such an encoding can, in principle, lead to an infinite state space, as the number of possible partial plans is infinite: the set of moves available to each player is infinite, as they specify a timestamp; moreover, at any given time, a rule triggered by one of the current tokens may require to look arbitrarily back in the past in order to check (the possibility of) its satisfaction. We now show how to constrain the state space to be finite.

First of all, we observe that arbitrarily large timestamps can only be introduced by wait moves played by *Charlie*, as the timestamp of any play move played by *Charlie* is forced to be equal to now and the timestamp of any *Eve*'s move is bounded by the one of a *Charlie*'s move. Now, even though wait moves are useful for *Charlie* to skip a given amount of time without executing a sequence of empty play moves, they can always be replaced by such a sequence, and thus it can be easily shown that if a winning strategy exists, another one exists which does not use any wait move. Hence, w.l.o.g., we can safely restrict ourselves to a *finite* set of moves of form $\mathsf{play}(t, A)$, with $t = \mathsf{now}$.

Let us focus now on the partial plans to show that not every single detail of their distant past has to be remembered. Consider a timeline-based planning game $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$, and let the *window* of G, $\mathsf{w}(G)$, be the product of all the non-zero lower and upper bounds appearing in any bounded atom of any rule, and all the non-zero lower and upper bounds on the length of tokens of any state variable. The window of the game gives a coarse upper bound on how far from a given point a chain of bounded atoms can look, which is a fundamental parameter of any timeline-based planning problem, *e.g.*, it has been exploited in [16] to study the complexity of the plan existence problem.

For the sake of clarification, consider the above half of Figure 3, which shows a timeline satisfying a particular synchronisation rule. The satisfaction of bounded and unbounded atoms of the rule by tokens are represented by solid and dashed arrows, respectively. The depicted tokens can be partitioned in two blocks, which satisfy two different groups of bounded atoms. Once the first group has been witnessed, only its existence need to be remembered, not the precise position or length of its tokens, since any bounded atom involving

the trigger cannot be affected by anything farther than the window $\mathsf{w}(G)$. Hence, while the events happening at most $\mathsf{w}(G)$ time steps before now need to be remembered in detail, the satisfaction of the other group of atoms can be remembered symbolically as shown in the bottom half of Figure 3, *i.e.*, by remembering the existence of the tokens satisfying each required group of atoms, and their relative ordering. The same has to be done to remember future requests triggered by tokens appeared in the past.

It can be easily checked that $\mathsf{w}(G) \in \mathcal{O}(2^{|G|})$, and that the number of bits needed to store the succinct representation of the history and of future requests is exponential as well. Hence, we can build a data structure of size at most *exponential* in the size of G, that, during the whole play, can be used to represent the current state and the past history of the game.

Let Π be a partial plan. We denote by $[\Pi]$ its succinct representation, and by $[\mathbf{\Pi}_G]$ the set of the succinct representations of all the possible partial plans on G. For any round ρ, we can easily specify its application to the succinct representation of a partial plan in such a way that $\rho([\Pi]) = [\rho(\Pi)]$. Similarly, given $[\Pi]$, we can easily check whether Π is a solution plan for a given timeline-based planning problem.

We are now ready to define the *game structure* $\mathcal{G}$ associated with a planning game G.

▶ **Definition 24** (Game structure for a planning game)**.** Let $G = (SV_C, SV_E, \mathcal{S}, \mathcal{D})$ be a timeline-based planning game. The corresponding game structure $\mathcal{G}$ is a tuple $(Q_C, Q_E, M_C, M_E, \mathcal{P}, \pi, \delta_C, \delta_E)$, where:

1. $Q_C = [\mathbf{\Pi}_G]$ and $Q_E \subseteq [\mathbf{\Pi}_G] \times \mathsf{M}_C$;
2. $M_C = \mathsf{M}_C$ and $M_E = \mathsf{M}_E$;
3. $\mathcal{P} = \{d, w\}$, where d and w are proposition letters,
4. $\pi(q_E) = \varnothing$, for any $q_E \in Q_E$, and $d \in \pi([\Pi])$ (resp., $w \in \pi([\Pi])$ if and only if Π is a solution plan for $P_{\mathcal{D}}$ (resp., P_G);
5. $([\Pi], \mu_C, ([\Pi], \mu_C)) \in \delta_C$ if and only if $\mu_C = \mathsf{play}(t, A)$ is applicable to Π and $t = \mathsf{now}$;
6. $(([\Pi], \mu_C), \mu_E, [\Pi']) \in \delta_E$ if and only if $\rho = (\mu_C, \mu_E)$ is applicable to Π and $\rho([\Pi]) = [\Pi']$.

The winning condition for *Charlie* (see Definition 20) can be expressed by means of the following ATL^* formula, where Player 1 is interpreted as *Charlie* and Player 2 as *Eve*:[4]

$$\varphi \equiv \langle\!\langle 1 \rangle\!\rangle [\![2]\!] \big([\![1]\!] \, \mathsf{F}\, d \rightarrow \mathsf{F}(d \wedge w) \big)$$

The formula asks for the existence of a strategy σ_C for *Charlie* ($\langle\!\langle 1 \rangle\!\rangle$) such that, for all strategies σ_E for *Eve* ($[\![2]\!]$), if σ_E is admissible ($[\![1]\!] \, \mathsf{F}\, d$), *i.e.*, for all *Charlie*'s strategies, there is a future point where d holds, then there is future point where $d \wedge w$ holds ($\mathsf{F}(d \wedge w)$), *i.e.*, *Charlie* wins. By applying known ATL^* model-checking algorithms on the game structure encoding the planning game and the above formula, one may solve the problem.

It can be proved that any strategy satisfying the above ATL^* formula corresponds to a winning strategy for *Charlie* on the original timeline-based planning game. The proposed compact representation of partial plans has exponential size, and thus the game structure has a doubly exponential number of states. Since ATL^* model-checking of fixed-size formulae over game structures is PTIME-complete [2], the following theorem holds.

▶ **Theorem 25.** *Let G be a timeline-based planning game. The problem of establishing whether Charlie has a winning strategy on G belongs to* 2EXPSPACE.

[4] ATL^* is a branching-time temporal logic similar to CTL^*, that allows one to quantify over the strategies of the different players of a game structure and to differentiate between paths played according to those strategies.

6 Conclusions and future work

This paper defines *timeline-based planning games*, a novel game-theoretic approach to timeline-based planning with uncertainty. Unlike current formulations based on dynamic controllability of flexible plans, the proposed one can uniformly deal with both temporal uncertainty and general nondeterminism, and it is strictly more expressive. We showed that a winning strategy can be found in doubly exponential space. Whether there exists a matching complexity lower bound, how to synthesise a finite-state machine implementing such a strategy, and how hard the synthesis problem is are still open issues.

References

1 J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002.

3 J. Barreiro, M. Boyce, M. Do, J. Frank, M. Iatauro, T. Kichkaylo, P. Morris, J. Ong, E. Remolina, T. Smith, and D. Smith. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *Proc. of the 4th International Competition on Knowledge Engineering for Planning and Scheduling*, 2012.

4 M. Beetz and D. McDermott. Improving robot plans during their execution. In *Proc. of the International Conference on Artificial Intelligence Planning Systems (AIPS)*, 1994.

5 S. Borgo, A. Cesta, A. Orlandini, and A. Umbrico. A Planning-Based Architecture for a Reconfigurable Manufacturing System. In *Proc. of the 26th International Conference on Automated Planning and Scheduling*, pages 358–366, 2016.

6 A. Camacho, E. Triantafillou, C. Muise, J. A. Baier, and S. A. McIlraith. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *Proc. of the 31st AAAI Conference on Artificial Intelligence*, 2017.

7 A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *Proc. of the 21st Conference on Innovative Applications of Artificial Intelligence*, pages 66–71, 2009.

8 M. Cialdea Mayer and A. Orlandini. An Executable Semantics of Flexible Plans in Terms of Timed Game Automata. In *Proc. of the 22nd International Symposium on Temporal Representation and Reasoning*, pages 160–169, 2015.

9 M. Cialdea Mayer, A. Orlandini, and A. Umbrico. Planning and Execution with Flexible Timelines: a Formal Account. *Acta Informatica*, 53(6-8):649–680, 2016.

10 A. Cimatti, M. Do, A. Micheli, M. Roveri, and D. E. Smith. Strong temporal planning with uncontrollable durations. *Artificial Intelligence*, 256:1–34, 2018.

11 A. Cimatti, A. Micheli, and M. Roveri. Timelines with Temporal Uncertainty. In *Proc. the 27th AAAI Conference on Artificial Intelligence*, 2013.

12 A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1):35–84, 2003.

13 M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

14 J. Frank. What is a timeline? In *Proc. of the 4th Workshop on Knowledge Engineering for Planning and Scheduling*, pages 31–38, 2013.

15 N. Gigante, A. Montanari, M. Cialdea Mayer, and A. Orlandini. Timelines are Expressive Enough to Capture Action-based Temporal Planning. In *Proc. of the 23rd International Symposium on Temporal Representation and Reasoning*, pages 100–109, 2016.

16 N. Gigante, A. Montanari, M. Cialdea Mayer, and A. Orlandini. Complexity of timeline-based planning. In *Proc. of the 27th International Conference on Automated Planning and Scheduling*, pages 116–124, 2017.

17 N. Gigante, A. Montanari, M. Cialdea Mayer, A. Orlandini, and M. Reynolds. A game-theoretic approach to timeline-based planning with uncertainty. *CoRR*, abs/1807.04837, 2018. URL: `http://arxiv.org/abs/1807.04837`.

18 L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

19 C. Muise, S. A. McIlraith, and J. C. Beck. Improved non-deterministic planning by exploiting state relevance. In *Proc. of the 22nd International Conference on Automated Planning and Scheduling*, 2012.

20 C. Muise, S.A. McIlraith, and V. Belle. Non-deterministic planning with conditional effects. In *Proc. of the 24th International Conference on Automated Planning and Scheduling*, 2014.

21 N. Muscettola. HSTS: Integrating Planning and Scheduling. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 6, pages 169–212. Morgan Kaufmann, 1994.

22 F. Patrizi, N. Lipovetzky, and H. Geffner. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence*, 2014.

23 S. Steel. Action under uncertainty. *Journal of Logic and Computation*, 4(5):767–795, 1994.

24 A. Umbrico, A. Cesta, M. Cialdea Mayer, and A. Orlandini. PLATINUm: A New Framework for Planning and Acting in Human-Robot Collaborative Scenarios. In *Proc. of the 16th International Conference of the Italian Association for Artificial Intelligence*, pages 498–512, 2017.

25 T. Vidal and H. Fargier. Handling Contingency in Temporal Constraint Networks: from Consistency to Controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

Sound-and-Complete Algorithms for Checking the Dynamic Controllability of Conditional Simple Temporal Networks with Uncertainty

Luke Hunsberger
Department of Computer Science, Vassar College, NY, USA
hunsberger@vassar.edu

Roberto Posenato
Department of Computer Science, University of Verona, Verona, Italy
roberto.posenato@univr.it
https://orcid.org/0000-0003-0944-0419

Abstract

A Conditional Simple Temporal Network with Uncertainty (CSTNU) is a data structure for representing and reasoning about time. CSTNUs incorporate *observation time-points* from Conditional Simple Temporal Networks (CSTNs) and *contingent links* from Simple Temporal Networks with Uncertainty (STNUs). A CSTNU is *dynamically controllable* (DC) if there exists a strategy for executing its time-points that guarantees the satisfaction of all relevant constraints no matter how the uncertainty associated with its observation time-points and contingent links is resolved in real time. This paper presents the first sound-and-complete DC-checking algorithms for CSTNUs that are based on the propagation of labeled constraints and demonstrates their practicality.

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

Keywords and phrases Temporal Networks, Conditional Simple Temporal Problem with Uncertainty, Dynamic Controllability, Checking Algorithm

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.14

1 Introduction

A Conditional Simple Temporal Network with Uncertainty (CSTNU) is a data structure for representing and reasoning about time in domains where some constraints may apply only in certain scenarios and some events may have uncontrollable, but bounded durations [13]. They were defined to represent important features of, for example, (1) *workflow systems* used to automate medical-treatment processes [16],[7]; and (2) *planning systems* when uncertain durations are present [17]. A CSTNU may include *observation time-points* and *contingent links.* An observation time-point represents a test action whose execution generates a truth value for a corresponding propositional letter. A contingent link represents an action with an uncertain, but bounded duration. Observation time-points and contingent links both involve uncertainty *and* uncontrollability, since the outcomes of tests and contingent durations are not known in advance and are not controlled by the scheduling agent; they are only *observed* during execution.

CSTNUs generalize Conditional Simple Temporal Networks (CSTNs) [23] and Simple Temporal Networks with Uncertainty (STNUs) [21]. The *dynamic controllability* (DC) property for CSTNUs generalizes the corresponding properties for CSTNs and STNUs. In brief, a CSTNU is DC if there exists a strategy for executing its time-points that guarantees

the satisfaction of all relevant constraints no matter how the uncertainty associated with its observation time-points and contingent links is resolved during execution. The *DC-checking problem* for CSTNUs is that of determining whether arbitrary CSTNUs are DC.

Combi et al. [5, 6] presented a variety of sound constraint-propagation rules for CSTNUs, but did not address completeness. Following their approach, but focusing on CSTNs, Hunsberger et al. [14] presented the first practical sound-and-complete DC-checking algorithm for CSTNs. Recently, they presented a faster version of their algorithm, called the π-DC-checking algorithm [12], that will play a role in this paper. In other approaches, Hunsberger and Posenato [11] presented an algorithm that views the DC-checking problem for CSTNs as a two-player game, searching an abstract game tree to find a "winning" strategy, guided by Monte-Carlo Tree Search and Limited Discrepancy Search; and Cimatti et al. [4] reduced the DC-checking problem for CSTNUs (and a broader class of networks) to a controller-synthesis problem for timed game automata, but have not shown whether that approach can be made practical for CSTNUs.

Contribution. This paper presents the first *practical sound-and-complete* DC-checking algorithms for CSTNUs. The first algorithm reduces the DC-checking problem for CSTNUs to the DC-checking problem for CSTNs; the second propagates constraints directly in the input CSTNU. The paper proves that both algorithms are correct and empirically evaluates their performance.

2 Conditional STNs with Uncertainty (CSTNUs)

This section recalls the definition of a well-defined CSTNU, which allows contingent links (as in an STNU) and observation time-points (as in a CSTN). The presentation combines and extends definitions from earlier work [18, 13, 6, 12]. The notion of a *streamlined* temporal network from recent work on CSTNs [2] is then applied to CSTNUs.

▶ **Definition 1** (P-Labels). For a set $\mathcal{P}$ of propositional letters, a *p-label* is a (possibly empty) conjunction of (positive or negative) literals from $\mathcal{P}$. The *empty p-label* is notated $\boxdot$. For any p-label ℓ, and any $p \in \mathcal{P}$, if $\ell \models p$ or $\ell \models \neg p$, we say that p *appears* in ℓ. For p-labels, ℓ_1 and ℓ_2, if $\ell_1 \models \ell_2$, we say that ℓ_1 *entails* ℓ_2. If $\ell_1 \wedge \ell_2$ is satisfiable, we say that ℓ_1 and ℓ_2 are *consistent.* $\mathcal{P}^*$ denotes the set of all *satisfiable* p-labels with literals from $\mathcal{P}$.

▶ **Definition 2** (CSTNU). A *Conditional Simple Temporal Network with Uncertainty* is a tuple, $\langle \mathcal{T}, \mathcal{P}, L, \mathcal{C}, \mathcal{OT}, \mathcal{O}, \mathcal{L} \rangle$, where:

- $\mathcal{T}$ is a finite set of real-valued variables, called *time-points*;
- $\mathcal{P}$ is a finite set of propositional letters;
- $L\colon \mathcal{T} \to \mathcal{P}^*$ assigns p-labels to time-points;
- $\mathcal{C}$ is a set of *labeled* constraints, each of the form, $(Y - X \leq \delta,\ \ell)$, where $X, Y \in \mathcal{T}$, $\delta \in \mathbb{R}$, and $\ell \in \mathcal{P}^*$;
- $\mathcal{OT} \subseteq \mathcal{T}$ is a set of *observation* time-points;
- $\mathcal{O}\colon \mathcal{P} \to \mathcal{OT}$ is a bijection from propositional letters to observation time-points;
- $\mathcal{L}$ is a set of *contingent links* each of the form (A, x, y, C), where: $A \in \mathcal{T}, C \in \mathcal{T} \setminus \mathcal{OT}$, $A \not\equiv C$ are called the *activation* and *contingent* time-points, respectively; $L(A) = L(C)$; $0 < x < y < \infty$; and distinct contingent links have distinct contingent time-points.

By convention, for each $p \in \mathcal{P}$, $\mathcal{O}(p)$ (i.e., the observation time-point whose execution determines the truth value for p) may be denoted by $P?$.

Hunsberger *et al.* [14] called a CSTN *well-defined* if the p-labels on its time-points and constraints satisfied certain properties. CSTNs that are not well defined turn out to be useless. Definition 3 extends well-definedness to CSTNUs.

▶ **Definition 3** (Well-defined CSTNU)**.** A CSTNU is *well defined* if:

1. for each $(Y - X \leq \delta, \ell) \in \mathcal{C}$, $\ell \models L(X) \wedge L(Y)$;
2. for each $T \in \mathcal{T}$, and each p appearing in $L(T)$:
   a. $\ell \models L(P?)$; and
   b. $(P? - T \leq -\epsilon, L(T)) \in \mathcal{C}$, for some $\epsilon > 0$;
3. for each $(Y - X \leq \delta, \ell) \in \mathcal{C}$, and each p appearing in ℓ, $\ell \models L(P?)$.

Cairo *et al.* [2] showed that if $\mathcal{S}$ is a well-defined CSTN (i.e., satisfies properties 1–3 above), then there is a CSTN $\mathcal{S}_{\boxdot}$ such that:

1. $\mathcal{S}_{\boxdot}$ does *not* have any labels on its time-points, and
2. $\mathcal{S}_{\boxdot}$ is DC if and only if $\mathcal{S}$ is DC.

$\mathcal{S}_{\boxdot}$ is called a *streamlined* CSTN. We say that $\mathcal{S}_{\boxdot}$ is *DC-equivalent* to $\mathcal{S}$.
This result extends easily to CSTNUs.

▶ **Definition 4** (Streamlined CSTNU)**.** Given a well-defined CSTNU $\mathcal{S}$, its *streamlined* version $\mathcal{S}_{\boxdot}$ is the same as $\mathcal{S}$ except that its time-points have no p-labels.

It is straightforward to show that if $\mathcal{S}$ is a *well-defined* CSTNU, then $\mathcal{S}$ is DC if and only if its streamlined version $\mathcal{S}_{\boxdot}$ is DC. For this reason, this paper restricts attention to streamlined CSTNUs, which simplifies definitions and proofs of the main results with no loss of generality.

Each CSTNU has a corresponding graph whose nodes represent time-points, and whose edges represent various kinds of temporal constraints. To accommodate the propositional labels (p-labels) from CSTN graphs and the alphabetic labels (a-labels) from STNU graphs, each edge in a CSTNU graph may be annotated by both p-labels *and* a-labels. This paper treats a-labels more rigorously than in prior work in anticipation of how they are conjoined and modified during constraint propagation. In particular, an a-label is liberally defined to allow *conjunctions of upper-case letters* that may arise during constraint propagation.

▶ **Definition 5** (A-Letters, A-Labels)**.** If $C_1, \ldots, C_k$ are the contingent time-points for a CSTNU $\mathcal{S}$, then $\mathcal{A} = \{c_1, \ldots, c_k, C_1, \ldots, C_k\}$ is the set of *alphabetic letters (a-letters)* for $\mathcal{S}$; $c_1, \ldots, c_k$ are the *lower-case* (LC) a-letters; and $C_1, \ldots, C_k$ are the *upper-case* (UC) a-letters. An *a-label*, $\aleph$, is a set of a-letters that: (1) is empty, notated as $\diamond$; (2) contains exactly one LC a-letter, notated as c_i; or (3) contains *one or more* UC a-letters, notated as $C_{i_1} \ldots C_{i_m}$. The set of all a-labels with letters from $\mathcal{A}$ is denoted by $\mathcal{A}^*$. The set of *UC a-labels* (that is, a-labels that contain zero or more UC a-letters) is denoted by $\mathcal{A}_u^*$. For any $\aleph, \aleph' \in \mathcal{A}_u^*$, their *conjunction* is given by their union (i.e., $\aleph\aleph' = \aleph \cup \aleph'$).

Edges in a CSTNU graph are annotated by triples, called a *labeled values,* that generalize: (1) the numerical weights that appear on edges in STN graphs; (2) the *lower-case* and *upper-case* a-letters on edges in STNU graphs; and (3) the p-labels on edges in CSTN graphs. Since any pair of time-points, X and Y, may participate in multiple constraints, an edge from X to Y may have multiple labeled values, each of the form, $\langle \delta_i, \aleph_i, \ell_i \rangle$.

▶ **Definition 6** (Labeled values)**.** A *labeled value* is a triple, $\langle \delta, \aleph, \ell \rangle$, where: $\delta \in \mathbb{R}$, $\aleph \in \mathcal{A}^*$, and $\ell \in \mathcal{P}^*$.

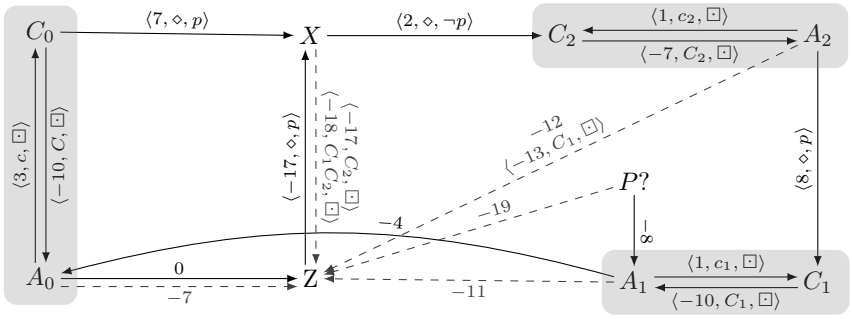


Figure 1 A CSTNU graph with 3 contingent links and one observation time-point $P?$

In a CSTNU graph, an *ordinary* STN constraint, $Y - X \leq \delta$, is represented by an edge $X \xrightarrow{\langle \delta, \diamond, \boxdot\rangle} Y$; and a conditional constraint, $(Y - X \leq \delta, \ell)$, is represented by an edge $X \xrightarrow{\langle \delta, \diamond, \ell\rangle} Y$. Edges associated with contingent links require further introduction.

In an *STNU* graph, each contingent link (A, x, y, C) gives rise to two edges: a *lower-case* edge from $A \xrightarrow{c:x} C$ that represents the possibility that the duration $C - A$ might take on its minimum value x; and an *upper-case* edge from $C \xrightarrow{C:-y} A$ that represents that $C - A$ might take on its maximum value y. In a CSTNU graph, the lower-case edge is $A \xrightarrow{\langle x, c, \boxdot\rangle} C$ and the upper-case edge is $C \xrightarrow{\langle -y, C, \boxdot\rangle} A$.

Figure 1 shows the graph for a sample CSTNU. Z is a (non-contingent, non-observation) time-point whose value is fixed at 0. Each time-point X is implicitly constrained to occur at or after Z. Labeled values such as $\langle -11, \diamond, \boxdot\rangle$ are abbreviated as simply -11. The dashed edges represent constraints obtained by Algorithm 2, described in section 6; it determines that this CSTNU is not dynamically controllable.

3 Dynamic Controllability for CSTNUs

The truth values of propositions in a CSTNU are not known in advance; they are incrementally revealed as observation time-points are executed. Similarly, the durations of contingent links are only observed as the contingent time-points happen to execute. However, a *dynamic strategy* for executing the time-points in a CSTNU can *react* to observations and contingent executions in real time. A *viable* strategy is one that guarantees that all relevant constraints will be satisfied no matter which truth values and durations are revealed over time. A CSTNU with a dynamic and viable strategy is *dynamically controllable* (DC).

Like much recent work on STNUs and CSTNs [18, 14, 1], this paper defines the DC property for CSTNUs to allow execution strategies to react *instantaneously* to observations, instead of requiring arbitrarily small delays. It generalizes the DC semantics for STNUs [18] and the π-DC semantics for CSTNs [1].

This paper focuses on CSTNUs whose sets of contingent and observation time-points are distinct – with no loss of generality because any contingent observation time-point could be represented by two time-points, a contingent time-point C and an observation time-point $P?$, constrained to occur simultaneously. An instantaneously reactive strategy could wait for C to execute and then execute $P?$ at the same time.

Preliminaries. A *scenario* s specifies a truth value for each proposition, and a *situation* ω specifies a duration for each contingent link. A *drama* is then a scenario-situation pair (s, ω). The projection of a CSTNU onto a drama (s, ω) is the *STN* obtained by restricting attention to the constraints whose labels are true under s and assigning each contingent duration to the value specified by ω.

▶ **Definition 7** (Scenario/Situation/Drama/Projection)**.** A *scenario* is a function, $s\colon \mathcal{P} \to \{\top, \bot\}$, that assigns a truth value to each $p \in \mathcal{P}$. A scenario also determines the truth value, $s(\ell)$, for any p-label $\ell \in \mathcal{P}^*$. The set of all scenarios over $\mathcal{P}$ is denoted by $\mathcal{I}$. If (A_1, x_1, y_1, C_1), $\ldots$, (A_k, x_k, y_k, C_k) are the contingent links for a CSTNU $\mathcal{S}$, then $\Omega = [x_1, y_1] \times \ldots \times [x_k, y_k]$ is called the *space of situations* for $\mathcal{S}$, and any $\omega = (\omega_1, \ldots, \omega_k) \in \Omega$ is called a *situation.* A *drama* is any pair $(s, \omega) \in \mathcal{I} \times \Omega$, where $s \in \mathcal{I}$ is a scenario, and $\omega \in \Omega$ is a situation. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O}, \mathcal{L} \rangle$ be any CSTNU, and (s, ω) any drama for $\mathcal{S}$, where $\omega = (\omega_1, \ldots, \omega_k)$. The *projection* of $\mathcal{S}$ onto (s, ω) – denoted by $Prj(\mathcal{S}, s, \omega)$ – is the STN, $(\mathcal{T}, \mathcal{C}_s)$, where:[1]

$$\mathcal{C}_s = \{(Y - X \leq \delta) \mid \text{ for some } \ell, (Y - X \leq \delta, \ell) \in \mathcal{C} \text{ and } s(\ell) = \top\} \cup \{(C_i - A_i = \omega_i) \mid (A_i, x_i, y_i, C_i) \in \mathcal{L}\}$$

3.1 Execution strategies

Cairo et al. [1] introduced the π-DC semantics for CSTNs that, unlike prior versions [14, 8], does not permit a kind of circular dependency among simultaneous observations. A π-dynamic strategy must specify, for each scenario, both a *schedule* for the time-points, and an *order of dependency* among the observation time-points. *This section extends the π-DC semantics to cover CSTNUs.*

▶ **Definition 8** (Schedule)**.** A *schedule* for a set $\mathcal{T}$ of time-points is a *complete* mapping, $\psi\colon \mathcal{T} \to \mathbb{R}$. For any $X \in \mathcal{T}$, and any schedule ψ, the execution time for X in ψ is denoted by $[\psi]_X$. The set of schedules for $\mathcal{T}$ is denoted by Ψ.

▶ **Definition 9** (Order of Dependency)**.** Let $\mathcal{OT} = \{P_1?, \ldots, P_k?\}$ be a set of observation time-points. Any permutation π over $(1, 2, \ldots, k)$ effectively specifies an order for those observation time-points. For any $P? \in \mathcal{OT}$, let $\pi(P?) \in \{1, 2, ..., k\}$ denote the (integer) position of $P?$ in the order determined by π; and let Π_k denote the set of all permutations over $(1, 2, \ldots, k)$.

▶ **Definition 10** (π-Execution Strategy)**.** A *π-execution strategy* for a CSTNU $\mathcal{S}$ with k contingent links is a mapping, $\sigma\colon (\mathcal{I} \times \Omega) \to (\Psi \times \Pi_k)$, where for each drama $r = (s, \omega) \in \mathcal{I} \times \Omega$, $\sigma(s, \omega)$ is a pair (ψ_r, π_r) such that $\psi_r\colon \mathcal{T} \to \mathbb{R}$ is a schedule, and $\pi_r \in \Pi_k$ determines an order of dependency among the observation time-points. For convenience, π_r is extended such that $\pi_r(C) = 0$ for each contingent time-point C, and $\pi_r(X) = \infty$ for each non-contingent, non-observation time-point X.

The strategy σ is *viable* if for each drama $r = (s, \omega)$, the schedule ψ_r is a solution to the projection $Prj(\mathcal{S}, s, \omega)$. And σ is *coherent* if for each drama $r = (s, \omega)$, and any $P?$ and $Q?$ in $\mathcal{OT}$, $[\psi_r]_{P?} < [\psi_r]_{Q?}$ implies $\pi_r(P?) < \pi_r(Q?)$ (i.e., if ψ_r *schedules* $P?$ before $Q?$, then π_r *orders* $P?$ before $Q?$).

▶ **Definition 11** (π-History)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O}, \mathcal{L} \rangle$ be a CSTNU with k contingent links; σ a π-execution strategy for $\mathcal{S}$; $r = (s, \omega)$ a drama; $(\psi_r, \pi_r) = \sigma(s, \omega)$; $t \in \mathbb{R}$; and

[1] $C_i - A_i = \omega_i$ abbreviates the pair of constraints, $C_i - A_i \leq \omega_i$ and $A_i - C_i \leq -\omega_i$.

Table 1 Morris-Muscettola rules for DC-checking STNUs.

Rule	Conditions	Pre-existing and generated edges
No Case (NC):		$W \xleftarrow{v} Y \xleftarrow{u} X$; $W \xleftarrow{u+v} X$
Upper Case (UC):		$A \xleftarrow{C:v} Y \xleftarrow{u} X$; $A \xleftarrow{C:u+v} X$
Lower Case (LC):	$(v < 0)$	$X \xleftarrow{v} C \xleftarrow{c:u} A$; $X \xleftarrow{u+v} A$
Cross Case (CC):	$(D \not\equiv C$ and $v < 0)$	$X \xleftarrow{D:v} C \xleftarrow{c:u} A$; $X \xleftarrow{D:u+v} A$
Label Removal (LR):	$(v \geq -x)$	$C \xleftarrow{c:x} A \xleftarrow{C:v} X$; $C \xleftarrow{v} X$

$d \in \{1, 2, \ldots, k; \infty\}$. Then $\mathcal{H}(t, d, s, \omega, \sigma) = (\mathcal{H}_s, \mathcal{H}_\omega)$ is the π-*history* of (t, d) for the drama (s, ω) and strategy σ, where:

$$\mathcal{H}_s = \{(p, s(p)) \mid P? \in \mathcal{OT},\ [\psi_r]_{P?} \leq t,\ \text{and } \pi_r(P?) < d\};$$
$$\mathcal{H}_\omega = \{(A, C, [\psi_r]_C - [\psi_r]_A) \mid \exists x, y \text{ such that } (A, x, y, C) \in \mathcal{L},\ \text{and } [\psi_r]_C \leq t\}.$$

$\mathcal{H}_s$ specifies the truth values of all propositions p observed *before* time t in the schedule ψ_r, as well as those observed *at* time t if $P?$ is ordered *before* position d by the permutation π_r. And $\mathcal{H}_\omega$ specifies the durations of all contingent links that completed *at or before* time t in the schedule ψ_r.

▶ **Definition 12** (π-Dynamic Execution Strategy)**.** A π-execution strategy, σ, for a CSTNU $\mathcal{S}$, is called π-*dynamic* if for every pair of dramas, (s_1, ω_1) and (s_2, ω_2), and every *non-contingent* (but possibly observation) time-point X:

let:	$(\psi_1, \pi_1) = \sigma(s_1, \omega_1)$ and $(\psi_2, \pi_2) = \sigma(s_2, \omega_2)$,
let:	$t = [\psi_1]_X$, and $d = \pi_1(X) \in \{1, 2, \ldots, \lvert\mathcal{OT}\rvert; \infty\}$.
if:	$\mathcal{H}(t, d, s_1, \omega_1, \sigma) = \mathcal{H}(t, d, s_2, \omega_2, \sigma)$
then:	$[\psi_2]_X = t$ and $\pi_2(X) = d$.

Thus, if, in the drama (s_1, ω_1), σ executes X at time t and position d, and the relevant histories are the same then, in the drama (s_2, ω_2), σ must also execute X at t and d.

▶ **Definition 13** (π-DC)**.** A CSTNU, $\mathcal{S}$, is π-*dynamically controllable* (π-DC) if there exists a π-execution strategy for $\mathcal{S}$ that is both viable and π-dynamic.

4 DC-Checking for STNUs and CSTNs

This section summarizes the DC-checking algorithms for STNUs and CSTNs, due to Morris and Muscettola [20] and Hunsberger and Posenato [12], respectively, that play important roles in our *new* CSTNU DC-checking algorithms.

4.1 DC checking for STNUs

Table 1 lists the five constraint-propagation rules for STNUs due to Morris and Muscettola [20].[2] The *No Case* rule captures standard constraint propagation for STNs. The *Upper Case* rule generates a *conditional* constraint that guards against the possibility that the contingent duration $C - A$ might take on its maximum value. It can be glossed as: "While

[2] Later STNU algorithms [18, 19] use techniques that do not readily transfer to CSTNUs.

C remains unexecuted, X must *wait* at least $-u - v$ after the execution of A."[3] The *Lower Case* rule generates a constraint that guards against $C - A$ taking on its minimum value. The *Cross Case* rule generates a conditional constraint that guards against one contingent duration $C - A$ taking on its minimum value, while another $D - X$ takes on its maximum value. The *Label Removal* rule specifies when a conditional constraint has the force of an unconditional constraint.

The Morris-Muscettola DC-checking algorithm applies the rules from Table 1 in at most $O(N^2)$ rounds, at a cost of $O(N^3)$ per round. Afterward, it computes the *AllMax* STN, which is the *STN* projection in which each contingent link is set to its maximum duration. The *AllMax* STN is computed from the *fully propagated* STNU by: (1) removing all lower-case edges; and (2) removing the upper-case letters from all (original or generated) upper-case edges. If the *AllMax* STN is consistent, then the STNU is declared to be DC.

4.2 π-DC checking for CSTNs

Hunsberger et al. [14] presented a 6-rule IR-DC-checking algorithm for CSTNs ("IR" for "instantaneous reaction") that is based on the propagation of labeled constraints. Hunsberger and Posenato [12] subsequently introduced a faster, 3-rule version of their algorithm, called the π-DC-checking algorithm, which is used in this paper. The π-DC-checking algorithm generates constraints whose labels may include *q-literals,* such as $?p$, that indicate that a constraint need only hold as long as the value of p is unknown.

▶ **Definition 14** (Q-literals, q-labels)**.** If $p \in \mathcal{P}$, then $?p$ is a *q-literal,* a *q-label* is a (possibly empty) conjunction of literals and/or q-literals, and $\mathcal{Q}^*$ denotes the set of all q-labels. For example, $p(?q)\neg r$ and $(?q)(?r)t\neg u$ are both q-labels.

The $\star$ operator extends ordinary conjunction to q-labels. Intuitively, if constraint C_1 is labeled by p, and constraint C_2 is labeled by $\neg p$, then *both* C_1 and C_2 must hold as long as p is unknown, which is represented by $p \star \neg p \ = ?p$.

▶ **Definition 15** ($\star$)**.** The operator, $\star$: $\mathcal{Q}^* \times \mathcal{Q}^* \to \mathcal{Q}^\star$, is defined thusly. First, for any $p \in \mathcal{P}$, $p \star p = p$ and $\neg p \star \neg p = \neg p$; otherwise, for any $p_1, p_2 \in \{p, \neg p, ?p\}$, $p_1 \star p_2 = ?p$. Next, for any $\ell_1, \ell_2 \in \mathcal{Q}^*$, $\ell_1 \star \ell_2 \in \mathcal{Q}^*$ denotes the conjunction obtained by applying $\star$ in pairwise fashion to matching literals from ℓ_1 and ℓ_2, and conjoining any unmatched literals. For example: $(p\neg q(?r)t) \star (qr\neg s) = p(?q)(?r)\neg st$.

Table 2 lists the sound-and-complete propagation rules for the π-DC-checking algorithm for CSTNs. The LP rule implements ordinary STN constraint propagation except that the labels, α and β, from the parent edges are conjoined in the generated edge. The qR_0 rule stipulates that a lower-bound constraint on $P?$ cannot depend on the value of p determined by executing $P?$. The qR_3^* rule specifies when an occurrence of $p, \neg p$ or $?p$ can be removed from a propositional label. The qR_3^* rule can generate edges whose labels are q-labels.

The π-DC-checking algorithm applies the rules from Table 2 until either (Non-DC) a negative self-loop with a consistent label is found; or (DC) no new edges can be generated. The completeness proof for the π-DC-checking algorithm shows how, in positive instances, to construct the *earliest-first strategy,* whose execution decisions are based on tracking the *current partial scenario* and computing *effective lower bounds* for unexecuted time-points. The *spreading lemma* ensures that lower-bound execution constraints are already present

[3] Wait constraints are only relevant if the wait time, $-u - v$, is positive (i.e., if $u + v < 0$).

Table 2 Constraint-propagation rules for π-DC-checking CSTNs.

Rule	Conditions	Pre-existing and Generated Edges
LP:	$u + v < 0$, $\alpha\beta \in \mathcal{P}^*$	$\mathrm{Z} \xleftarrow{\langle v,\beta\rangle} Y \xleftarrow{\langle u,\alpha\rangle} X$; $\mathrm{Z} \xleftarrow{\langle u+v,\alpha\beta\rangle} X$
qR$_0$:	$w < 0$, $\alpha \in \mathcal{Q}^*$	$\mathrm{Z} \xleftarrow{\langle w,\alpha\tilde{p}\rangle} P?$; $\mathrm{Z} \xleftarrow{\langle w,\alpha\rangle} P?$
qR$_3^*$:	$w < 0$, $\alpha,\beta \in \mathcal{Q}^*$	$Y \xrightarrow{\langle v,\beta\tilde{p}\rangle} \mathrm{Z} \xleftarrow{\langle w,\alpha\rangle} P?$; $Y \xrightarrow{\langle \max\{v,w\},\alpha\star\beta\rangle} \mathrm{Z}$

In each rule, $X, Y \in \mathcal{T}$; $P? \in \mathcal{OT}$; and $\mathrm{Z} = 0$. In qR$_0$ and qR$_3^*$, $\tilde{p} \in \{p, \neg p, ?p\}$; and p does not appear in α or β (in any form).

in the fully propagated network, courtesy of the qR$_0$ and qR$_3^*$ rules. The proof also shows that upper-bound execution constraints cannot generate negative loops in the relevant STN projection. Termination is guaranteed by inserting a global upper bound (or *horizon*), whose value is $h = nM$, where $n = |\mathcal{T}|$ and M is the maximum absolute value of any negative edge in the network. The horizon constraints do not affect the DC property, assuming that all edge weights are rational [2].

An upper bound for the computational complexity of the algorithm can be obtained assuming that each possible labeled value of each edge heading to Z must be updated M times and propagated to all other edges: $O(|\mathcal{T}|(3^{|\mathcal{P}|}|\mathcal{T}|M)) = O(M|\mathcal{T}|^2 3^{|\mathcal{P}|})$. Although exponential in the worst case, it has been shown to be practical across a variety of networks.

5 Algorithm 1: Reducing CSTNU-DC to CSTN-DC

This section introduces a novel DC-checking algorithm for CSTNUs that first transforms its input CSTNU $\mathcal{S}$ into a *DC-equivalent* CSTN $\mathcal{S}'$, and then applies the π-DC-checking algorithm for CSTNs to $\mathcal{S}'$. The transformation for contingent links is illustrated below. For each contingent link, (A, x, y, C), a new observation time-point $P_c?$ is introduced that is constrained to occur exactly x after A. Executing $P_c?$ generates a value for p_c that determines whether the duration $C - A$ shall be x or y.[4] If $p_c = \top$, then C must co-occur with $P_c?$ (i.e., x after A); otherwise, C must execute exactly $y - x$ after $P_c?$ (i.e., y after A). Because the CSTNU has been transformed into a CSTN, the horizon value $h = nM$ can be applied to that CSTN without affecting the DC property. The computational cost of this CSTNU-to-CSTN transformation is $O(|\mathcal{L}|)$ (i.e., linear).

$$A \underset{\langle -x,\boxdot\rangle}{\overset{\langle x,\boxdot\rangle}{\longleftrightarrow}} P_c? \underset{\langle 0,\boxdot\rangle,\ \langle x-y,\neg p_c\rangle}{\overset{\langle 0,p_c\rangle,\ \langle y-x,\boxdot\rangle}{\longleftrightarrow}} C$$

6 Algorithm 2: Propagating in the CSTNU

This section introduces a novel DC-checking algorithm for CSTNUs that propagates constraints in the CSTNU using the rules in Table 3. The names of the rules reflect the STNU and CSTN rules from Tables 1 and 2 that they generalize, except that z! is a new kind of rule that *forward propagates* upper-case a-labels. The z! rule is not needed for DC-checking STNUs,

[4] Cairo and Rizzi [3] proved that restricting contingent durations to be either the minimum or maximum value, but nothing in between, does not affect the DC property.

Table 3 Constraint-propagation rules for CSTNU Algorithm 2.

Rule	Conditions	Pre-existing and Generated Edges
(zLp/Nc/Uc)	$u + v < 0$, $\alpha\beta \in \mathcal{P}^*$	$\mathrm{Z} \xleftarrow{\langle v, \aleph, \beta\rangle} Y \xleftarrow{\langle u, \diamond, \alpha\rangle} X$; $\langle u+v, \aleph, \alpha\beta\rangle$
(zLc/Cc)	$x + v < 0$, $C \notin \aleph$, $\beta \in \mathcal{P}^*$	$\mathrm{Z} \xleftarrow{\langle v, \aleph, \beta\rangle} C \xleftarrow{\langle x, c, \boxdot\rangle} A$; $\langle x+v, \aleph, \beta\rangle$
(z!)	$-y + v < 0$, $\beta \in \mathcal{P}^*$	$\mathrm{Z} \xleftarrow{\langle v, \aleph, \beta\rangle} A \xleftarrow{\langle -y, C, \boxdot\rangle} C$; $\langle -y+v, C\aleph, \beta\rangle$
(zLr)	$m = \max\{v, w-x\}$, $C \notin \aleph\aleph_1$, $\beta, \gamma \in \mathcal{Q}^*$	$Y \xrightarrow[\langle m, \aleph\aleph_1, \beta\star\gamma\rangle]{\langle v, C\aleph, \beta\rangle} \mathrm{Z} \xleftarrow{\langle w, \aleph_1, \gamma\rangle} A \xrightarrow{\langle x, c, \boxdot\rangle} C$
(zqR_0)	$w < 0$; $\tilde{p} \in \{p, \neg p, ?p\}$; $\tilde{p}\beta \in \mathcal{Q}^*$,	$\mathrm{Z} \xleftarrow[\langle w, \aleph, \beta\rangle]{\langle w, \aleph, \beta\tilde{p}\rangle} P?$
(zqR_3^*)	$w < 0$; $\tilde{p} \in \{p, \neg p, ?p\}$; $\tilde{p}\beta, \gamma \in \mathcal{Q}^*$,	$Y \xrightarrow[\langle \max\{v,w\}, \aleph\aleph_1, \beta\star\gamma\rangle]{\langle v, \aleph, \beta\tilde{p}\rangle} \mathrm{Z} \xleftarrow{\langle w, \aleph_1, \gamma\rangle} P?$

$\mathrm{Z} = 0$; $A, C, X, Y \in \mathcal{T}$; C is contingent; $P? \in \mathcal{OT}$; $\aleph, \aleph_1 \in \mathcal{A}_u^*$.

Algorithm 2: CSTNU-DC-CH($\mathcal{S}$).

Input: $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O}, \mathcal{L}\rangle$: a CSTNU instance
Output: the dynamic controllability status of $\mathcal{S}$.
G = graph for $\mathcal{S}$
$h = M|\mathcal{T}|$, where M is the maximum absolute value of any negative edge
foreach $X \in \mathcal{T}$ **do**
  Add the edges, $\mathrm{Z} \xrightarrow{\langle h, \diamond, \boxdot\rangle} X$ and $X \xrightarrow{\langle 0, \diamond, \boxdot\rangle} \mathrm{Z}$, to G.
do
  $G = \mathrm{zqR}_0(G)$ // Label Modification
  $G = \mathrm{zqR}_3^*(G)$
  $G = zLp/Nc/Uc(G)$ // Edge Generation
  $G = z!(G)$
  $G = zLc/Cc(G)$
  $G = zLr(G)$
  if *(any negative self-loop with a p-label has been found)* **then return** *not DC*
while *(rules continue to generate new edges)*
return *DC*

but is needed to ensure completeness for CSTNU DC-checking. Note that z! and zqR_3^* can generate *conjunctions* of upper-case a-labels, which can be handled by all of the other rules.

To ensure termination, the algorithm inserts the same horizon constraints seen earlier, then it exhaustively applies the rules from Table 3. It outputs *not DC* if a negative self-loop with a consistent p-label is found; otherwise, *DC*. The pseudo-code for the algorithm is given in Algorithm 2.

We begin with relevant definitions, then prove soundness and completeness.

▶ **Definition 16** (Precedes). Let σ be a π-dynamic strategy, (s, ω) any drama, and $(\psi, \pi) = \sigma(s, \omega)$. For any $X, Y \in \mathcal{T}$, if either $[\psi]_X < [\psi]_Y$ or ($[\psi]_X = [\psi]_Y$ and $\pi(X) < \pi(Y)$) then we say that X *precedes* Y in (ψ, π), notated $X \prec^{\pi}_{\psi} Y$.

If $X \prec^{\pi}_{\psi} P?$, then the decision to execute X cannot depend on the observation of p. In

addition, if X and Y are distinct time-points, and at least one of them is an observation time-point, then $X \prec^{\pi}_{\psi} Y$ if and only if $\neg(Y \prec^{\pi}_{\psi} X)$.

▶ **Definition 17** (Satisfy a Labeled Constraint)**.** A π-execution strategy σ *satisfies the labeled constraint* $(Y - X \leq \delta, \ell)$, where $\ell \in \mathcal{P}^*$, if, for each drama (s, ω), either $s(\ell) = \bot$ or $[\psi]_Y - [\psi]_X \leq \delta$, where $(\psi, \pi) = \sigma(s, \omega)$.

▶ **Definition 18** (Satisfy a Contingent Link)**.** A π-execution strategy σ *satisfies the contingent link* (A_i, x_i, y_i, C_i) if for each drama (s, ω), $[\psi]_{C_i} - [\psi]_{A_i} = \omega_i$. In such a case, we also say that σ satisfies the lower-case and upper-case edges associated with that contingent link.

From Defns. 7 and 12, it follows that a viable π-execution strategy σ must satisfy all of the (original) labeled constraints in $\mathcal{S}$ (before any constraint propagation) and all of the (original) lower- and upper-case edges in $\mathcal{S}$.

A constraint-propagation rule is sound if whenever a viable and dynamic σ satisfies the pre-existing edge(s) in that rule, σ must also satisfy the edge generated by that rule. Now, the rules in Table 3 only generate edges pointing at Z, which represent lower-bound constraints; however, the generated edges may have a-labels with multiple UC letters and q-labels (e.g., see rules z!, zLr and zqR$_3^*$); and many of the rules can propagate such labeled values. Therefore, the semantics of satisfying a lower-bound edge must accommodate such a-labels and q-labels.

▶ **Definition 19** (Satisfy a Lower-Bound Constraint)**.** A π-execution strategy σ *satisfies the lower-bound constraint* $(Y \geq \delta, \langle \aleph, \beta \rangle)$ represented by the edge from Y to Z labeled by $\langle -\delta, \aleph, \beta \rangle$, where $\beta \in \mathcal{Q}^*$, and $\aleph \in \mathcal{A}_u^*$, if for each drama (s, ω), *any* of the following hold, where $(\psi, \pi) = \sigma(s, \omega)$:

(1) $[\psi]_Y \geq \delta$;
(2) for some $C_i \in \aleph$, where $(A_i, x_i, y_i, C_i) \in \mathcal{L}$, $\omega_i < y_i$
(i.e., the i^{th} contingent link does *not* take on its maximum duration);
(3) for some $p \in \beta$, $s(p) = \bot$;
(4) for some $\neg p \in \beta$, $s(p) = \top$; or
(5) for some $?p \in \beta$, $P? \prec^{\pi}_{\psi} Y$.

If $(\mathrm{Z} - Y \leq -\delta, \beta)$ (i.e., $(Y \geq \delta, \beta)$) is a labeled constraint in a CSTNU (prior to any propagation), then $\beta \in \mathcal{P}^*$, and the corresponding edge in the graph has the labeled value $\langle -\delta, \diamond, \beta \rangle$. For this edge, clauses (2) and (5) in Defn. 19 are vacuous, whence satisfaction reduces to: $[\psi]_Y \geq \delta$ or $s(\beta) = \bot$. Thus, Defn. 19 reduces to Defn. 17 for original labeled edges that happen be lower-bound edges.

More generally, it will be useful to note that for any lower-bound edge labeled by $\langle -\delta, \aleph, \beta \rangle$, where $\beta \in \mathcal{P}^*$, satisfaction (i.e., Defn. 19) reduces to:

(i) $[\psi]_Y \geq \delta$;
(ii) for some $C_i \in \aleph$, where $(A_i, x_i, y_i, C_i) \in \mathcal{L}$, $\omega_i < y_i$; or
(iii) $s(\beta) = \bot$. (‡)

▶ **Definition 20** (Soundness)**.** A constraint-propagation rule is *sound* if whenever a *viable* and π-*dynamic* execution strategy σ satisfies the rule's pre-existing (parent) edges, it also satisfies the rule's generated (child) edge.

Note. In each of the soundness proofs below, σ is assumed to be a viable and π-dynamic strategy that satisfies the parent edges in the rule under consideration. Note, too, that soundness proofs for the (zqR$_0$), (zqR$_3^*$) and (zLp/Nc/Uc) rules are skipped to save space.

▶ **Lemma 21.** *The (zLc/Cc) rule from Table 3 is sound.*

Proof. Suppose σ does *not* satisfy the generated edge from A to Z in rule (zLc/Cc). Since the p-label $\alpha\beta$ on the generated edge is consistent, it follows from Defn. 19 that there is some drama (s, ω) such that *all* of the following hold:

(¬ i) $[\psi]_A < -x - v$;
(¬ ii) for each $C_i \in \aleph$, where $(A_i, x_i, y_i, C_i) \in \mathcal{L}$, $\omega_i = y_i$; and
(¬ iii) $s(\alpha\beta) = \top$.

First, (¬ iii) implies that $s(\alpha) = \top$ and $s(\beta) = \top$. Therefore, since σ satisfies the edge from C to Z, (¬ ii) implies that $[\psi]_C \geq -v$, by Defn. 19. Next, let ω' be the same as ω except that the contingent link AC takes on its *minimum* value x; and let $(\psi', \pi') = \sigma(s, \omega')$. Since σ is viable, $[\psi']_C - [\psi']_A = x$. However, since $C \notin \aleph$, (¬ ii) also holds for ω'; thus, $[\psi']_C \geq -v$ must hold. And, since the only difference between (s, ω) and (s, ω') is the duration of the contingent link AC, the first difference between ψ and ψ' must occur when C executes, which happens *after* A executes. Thus, $[\psi]_A = [\psi']_A = [\psi']_C - x \geq -v - x$, contradicting (¬ i). ◀

▶ **Lemma 22.** *The (z!) rule from Table 3 is sound.*

Proof. Let (s, ω) be any drama for which all $C_i \in C\aleph$ take on their maximum durations (i.e., $\omega_i = y_i$), and such that $s(\alpha\beta) = \top$; and let $(\psi, \pi) = \sigma(s, \omega)$. Then $s(\beta) = \top$ and all $C_i \in \aleph$ take on their maximum durations. Therefore, since σ satisfies the parent edge from A to Z, it follows that $[\psi]_A \geq -v$. Next, since $s(\alpha) = \top$, and C takes on its maximum duration, then $[\psi]_C = [\psi]_A - y \geq -v - y$. Thus, σ satisfies the generated edge from C to Z. ◀

▶ **Lemma 23.** *The (zLr) rule from Table 3 is sound.*

Proof. Suppose that σ does *not* satisfy the generated edge in rule (zLr). Then, by Defn. 19, there is a drama (s, ω) for which *all* of the following hold:

(1†) $[\psi]_Y < -m$;
(2†) for each $C_i \in \aleph\aleph_1$, where $(A_i, x_i, y_i, C_i) \in \mathcal{L}$, $[\psi]_{C_i} - [\psi]_{A_i} = y_i$;
(3†) for each $p \in \beta \star \gamma$, $s(p) = \top$;
(4†) for each $\neg p \in \beta \star \gamma, s(p) = \bot$; and
(5†) for each $?p \in \beta \star \gamma$, $\neg(P? \prec^{\pi}_{\psi} Y)$.

where $(\psi, \pi) = \sigma(s, \omega)$. In addition, since σ is valid and satisfies the parent edge from Y to Z, one of the following must hold, by (‡), above:

(1) $[\psi]_Y \geq -v$;
(2) for some $C' \in C\aleph$, where $(A', x', y', C') \in \mathcal{L}$, $[\psi]_{C'} - [\psi]_{A'} < y'$;
(3) for some $p \in \beta$, $s(p) = \bot$;
(4) for some $\neg p \in \beta$, $s(p) = \top$; or
(5) for some $?p \in \beta$, $P? \prec^{\pi}_{\psi} Y$.

Now, (1) contradicts (1†), since $-v \geq -m$. (2) holding for some $C' \in \aleph$ would contradict (2†). And (5) contradicts (5†), since $?p \in \beta$ implies $?p \in \beta \star \gamma$. Therefore, either (2) holds for $C' = C$ (i.e., $[\psi]_C - [\psi]_A < y$) or some instance(s) of (3) or (4) hold(s).

Suppose some instance(s) of (3) or (4) hold(s). For (3), if $p \in \beta$ and $s(p) = \bot$, then to avoid contradicting (3†), we must have $?p \in \beta \star \gamma$, which, by (5†), implies that $\neg(P? \prec^{\pi}_{\psi} Y)$. We can assume Y and $P?$ are distinct because any occurrence of p in β could be removed by (zqR$_0$). Therefore, $Y \prec^{\pi}_{\psi} P?$ must hold. A similar argument applies to any occurrence of $\neg p \in \beta$ that makes (4) hold. Thus, Y must precede any $P?$ for which p or $\neg p$ makes (3) or (4) hold, respectively.

Let s' equal s, except that: if $p \in \beta$ makes (3) hold, then $s'(p) = \top$; and if $\neg p \in \beta$ makes (4) hold, then $s'(p) = \bot$. Since σ satisfies the parent edge from Y to Z, one or more clauses from Defn. 19 must hold for $(\psi', \pi') = \sigma(s', \omega)$. By construction, (3) and (4) do *not* hold for s'; thus, one of the following must hold:

(1′) $[\psi']_Y \geq -v$;

(2′) for some $C' \in C\aleph, [\psi']_{C'} - [\psi']_{A'} < y'$; or

(5′) for some $?p \in \beta$, $P? \prec^{\pi'}_{\psi'} Y$.

Now (ψ, π) and (ψ', π') each determine a sequence of events that can be ordered, first by execution time and, second, for simultaneous events, by order of dependence. Let t' be the *earliest* time at which the two sequences differ. By construction, it must be where some $[\psi]_{R?} = [\psi']_{R?} = t'$, but $s(r) \neq s'(r)$. Furthermore, $Y \prec^{\pi}_{\psi} R?$ and, thus, by the definition of t', $[\psi']_Y = [\psi]_Y$. But then $(1^\dagger)$ implies that $[\psi']_Y = [\psi]_Y < -m \leq -v$, whence (1′) is false. As for (5′), if $?q \in \beta$ (and hence $?q \in \beta \star \gamma$) and $Q? \prec^{\pi'}_{\psi'} Y$, then $[\psi']_{Q?} \leq t'$, whence $[\psi']_{Q?} = [\psi]_{Q?}$ and, thus, $Q? \prec^{\pi}_{\psi} Y$, which contradicts $(5^\dagger)$. Thus, (2′) must hold for some $C' \in C\aleph$. Since σ is valid, and the contingent durations in ω did not change from (s, ω) to (s', ω), (2′) and (2) are equivalent. Thus, the only possibility is that (2) holds for $C' = C$ (i.e., $[\psi]_C - [\psi]_A < y$).

Next, suppose that $[\psi]_Y < [\psi]_C$. Let ω^+ be the same as ω except that $C - A = y$. It is not hard to check that in the drama (s', ω^+), conditions $(1^\dagger)$–$(5^\dagger)$ all hold, but that *none* of the conditions (1′)–(5′) can hold. (Changing to the situation ω^+ removed the last possibility (i.e., that $C - A < y$).) But that contradicts that σ satisfies the parent edge from Y to Z. Thus, $[\psi]_Y \geq [\psi]_C$.

Next, since σ satisfies the edge from A to Z, by Defn. 19, one of these must hold:

(1_A) $[\psi]_A \geq -w$;

(2_A) for some $C' \in \aleph_1$, $[\psi]_{C'} - [\psi]_{A'} < y'$;

(3_A) for some $p \in \gamma$, $s(p) = \bot$;

(4_A) for some $\neg p \in \gamma$, $s(p) = \top$; or

(5_A) for some $?p \in \gamma$, $P? \prec^{\pi}_{\psi} A$.

Now, $(2^\dagger)$ contradicts (2_A). And $(5^\dagger)$ contradicts (5_A). (We can assume that A and $P?$ are distinct since, otherwise, rule (zqR_0) could have been used to remove any occurrence of $P?$ from γ.) And $[\psi]_A \leq [\psi]_C - x \leq [\psi]_Y - x < -m - x \leq -w$ implies (1_A) is false. (The inequalities follow from σ being viable, from $[\psi]_C \leq [\psi]_Y$, from $(1^\dagger)$, and $m = \max\{v, w - x\}$.) Finally, any $?p$ making (5_A) true yields $[\psi]_{P?} \leq [\psi]_A < [\psi]_C \leq [\psi]_Y$, whence $P? \prec^{\pi}_{\psi} Y$, contradicting $(5^\dagger)$. Thus, (3_A) or (4_A) must hold.

Now, suppose that some p makes both (3) and (3_A) true. Then $p \in \beta \star \gamma$ and $s(p) = \bot$, contradicting $(3^\dagger)$. Thus, the letters that make (3) true, if any, must be distinct from the letters that make (3_A) true, if any. Similarly, the letters that make (4) true must be distinct from those that make (4_A) true. In addition, any p that makes (3) true requires $s(p) = \bot$, which implies that p cannot simultaneously make (4_A) true; and any p that makes (4) true cannot simultaneously make (3_A) true. In short, the letters that make (3) or (4) true are *distinct* from those that make (3_A) or (4_A) true. And, to avoid contradicting $(3^\dagger)$ or $(4^\dagger)$, for any $\pm p$ that makes (3), (4), (3_A) or (4_A) true, $?p$ must be in $\beta \star \gamma$, whence $(5^\dagger)$ yields that Y must precede $P?$ (i.e., $Y \prec^{\pi}_{\psi} P?$).

So, let s'' be the same as s' except that if p makes either (3_A) or (4_A) true, then $s''(p) \neq s'(p) = s(p)$. (Since s and s' only differ on letters that make (3) or (4) true, and since those letters are distinct from the letters making (3_A) or (4_A) true, s and s' must agree on all letters that make (3_A) or (4_A) true.) Let $(\psi'', \pi'') = \sigma(s'', \omega)$. Let t'' be the first time when the events in (ψ'', π'') and (ψ, π) differ. Then for some $U?$, $[\psi'']_{U?} = [\psi]_{U?} = t''$ and $s''(u) \neq s(u)$; and Y precedes $U?$ in (ψ'', π'') and (ψ, π). Therefore, $[\psi'']_Y = [\psi']_Y = [\psi]_Y \leq t'' \leq t'$.

Since σ satisfies the edge from A to Z, one or more clauses from Defn. 19 must hold for that edge. By construction, the only candidates are:
($1''_A$) $[\psi'']_A \geq -w$;
($2''_A$) for some $C' \in \aleph_1$, $[\psi'']_{C'} - [\psi'']_{A'} < y'$; or
($5''_A$) for some $?p \in \gamma$, $P? \prec''_{\psi''} A$.
Since all events occurring before time t'' are executed identically by (ψ, π) and (ψ'', π''), (1_A) being false implies that $(1''_A)$ must also be false, since $[\psi]_A < [\psi]_Y \leq t''$. Similarly, (2_A) being false implies that $(2''_A)$ must also be false. Finally, if $?g \in \gamma$ makes $(5''_A)$ true, that contradicts $(5^\dagger)$, since $?g \in \beta \star \gamma$. Therefore, all cases lead to a contradiction. ◀

▶ **Theorem 24.** *The rules from Table 3 are complete for π-DC checking for CSTNUs.*

Proof. Let $\mathcal{S}$ be any CSTNU; let $\mathcal{S}^*$ be the CSTNU obtained by fully propagating $\mathcal{S}$ using the rules from Table 3; and suppose that no negative loop with a consistent p-label was found and, thus, the DC-checking algorithm returned *DC*. Let $\mathcal{S}^*_x$ be the *AllMax* CSTN obtained by deleting all LC edges from $\mathcal{S}^*$ and removing all UC a-labels from labeled values in $\mathcal{S}^*$. By construction, the *AllMax* CSTN must already be fully propagated. To see this, note that ignoring the a-labels in the (zLp/Nc/Uc), zqR_0 and zqR^*_3 rules for CSTNUs from Table 3 reduces them to the LP, qR_0 and qR^*_3 rules for CSTNs from Table 2, respectively. Since no negative loop with consistent p-label was found by the CSTNU DC-checking algorithm, none exist in the *AllMax* CSTN; hence it too must be DC.

Construct the *earliest-first strategy* σ for $\mathcal{S}$, as follows. Let α be the *current partial scenario* (CPS), initially $\boxdot$; and let $\mathcal{T}_u$ be the unexecuted time-points, initially $\mathcal{T}\backslash\{\text{Z}\}$. For each $X \in \mathcal{T}_u$, compute its *effective lower bound:* $\text{ELB}(X) = \max\{\delta \mid \exists (X \geq \delta, \ell) \in \mathcal{S}^*_x, \mathit{appl}(\ell, \alpha)\}$.[5] Let (Λ, χ) be the first execution decision: "if nothing happens before time Λ, then execute the time-points in χ", where $\Lambda = \min\{\text{ELB}(X) \mid X \in \mathcal{T}_u\}$; and $\chi = \{X \in \mathcal{T}_u \mid \text{ELB}(X) = \Lambda\}$ [9].

Case 1: No contingent time-point executes before time Λ. For each active contingent link, (A_i, x_i, y_i, C_i), raise its lower bound to $\Lambda - a_i$, where $a_i = [\sigma(s)]_{A_i}$. This cannot introduce any new constraints into $\mathcal{S}^*$ or $\mathcal{S}^*_x$; thus, both are still DC. And, since no ELB values have changed, (Λ, χ) is the earliest-first decision for the CSTN $\mathcal{S}^*_x$. Thus, inserting the relevant execution constraints cannot introduce any negative loops into any relevant STN projection [12]. Remove any executed time-points from $\mathcal{T}_u$; update the CPS α to include any new observations; and delete any labeled values that are inconsistent with those observations.

Case 2: A contingent time-point C executes at some time $t \leq \Lambda$. Update $\mathcal{S}^*$, as follows. First, replace the labeled value $\langle -y, C, \boxdot\rangle$ on the original UC edge from C to A with $\langle -\delta, \diamond, \boxdot\rangle$, where $\delta = t - [\sigma(s)]_A$ is the observed duration for the link (A, x, y, C); and replace the labeled value $\langle x, c, \boxdot\rangle$ on the original lower-case edge from A to C by $\langle \delta, \diamond, \boxdot\rangle$. The execution semantics ensures that $\delta \in [x, y]$. Second, *remove* any labeled value $\langle w, \aleph, \beta\rangle$ from $\mathcal{S}^*$ for which $C \in \aleph$. Third, for each $X \in \mathcal{T}_u$, insert a lower-bound constraint, $(X \geq t, \alpha)$. (Although $\mathit{ELB}(X, \alpha) \geq \Lambda \geq t$, the ELB value could have been due to a C-labeled edge which has since been removed.) Finally, fully propagate the modified $\mathcal{S}^*$ CSTNU.

Suppose a negative loop with consistent p-label is discovered in the modified $\mathcal{S}^*$. Any such loop must include the edge from A to C since, otherwise, nothing could prevent the

[5] $\mathit{appl}(\ell, \alpha)$ holds if ℓ is *applicable* given the CPS α [14]. Formally, $\mathit{appl}(\ell, \alpha)$ holds if each p that appears in *both* ℓ and α appears as p in both or as $\neg p$ in both.

Figure 2 A negative loop in the modified (left) and original (right) CSTNU $\mathcal{S}^*$.

corresponding loop in the original $\mathcal{S}^*$ from being generated, which would be even more negative. First, consider the graphs shown in Fig. 2, where irrelevant details (e.g., p-labels) have been omitted to improve clarity. The lower-bound $X \geq t = a + \delta$ in the modified $\mathcal{S}^*$ generates a negative loop only if $-f < 0$. But that lower bound on X can only be new/relevant if X's original ELB value arose from a C-labeled edge as illustrated on the righthand side, where $-t - \epsilon < -t$. But then the (zLr) rule would have generated the shaded labeled value in the figure, whence propagating backward from X to C to A to Z, courtesy of the (zLp/Nc/Uc), (z!) and (zLc/Cc) rules, would have generated a loop of length $(-a - x) + (-f) + x + a = -f < 0$ in the original, a contradiction. The only other possibility is if the path from C to Z in Fig. 2 included an occurrence of the (formerly UC) edge from C to A. This case would require a negative path from C to C, which would have made the original $\mathcal{S}^*$ non-DC, a contradiction. Thus, the modified $\mathcal{S}^*$ is necessarily DC. Compute ELB values based on the updated and propagated $\mathcal{S}^*_x$ graph and continue recursively.

The construction of the strategy will be complete (and the network still consistent) once all time-points have been executed. ◀

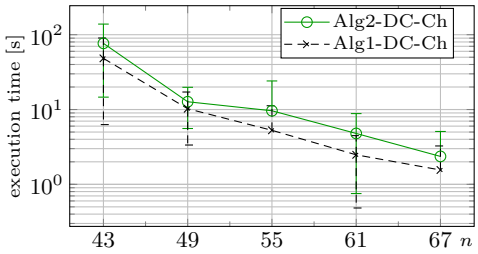
6.1 Computational complexity

An upper bound for the computational complexity of Algorithm 2 can be derived by adapting the original CSTN DC-checking algorithm complexity, while also considering the presence of a-labels. The three CSTN rules may have to consider all possible combinations of a-labels and q-labels. It is a matter of combinatoric operation to show that the complexity of such rules dominates the final upper bound, $O(M|\mathcal{T}|^2 3^{|\mathcal{P}|} 2^{|\mathcal{L}|})$, where M is the maximum absolute value of any negative weight in the graph.

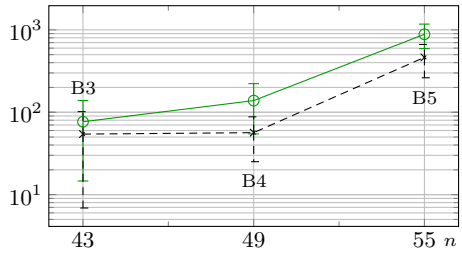
7 Empirical Evaluation

This section presents an empirical comparison of the performance of the two DC-checking algorithms introduced in this paper. `Alg1-DC-Ch` is our implementation of Algorithm 1 (cf. Sect. 5), which converts CSTNUs to CSTNs; `Alg2-DC-Ch` is our implementation of Algorithm 2 (cf. Sect. 6) which directly propagates CSTNU constraints. Both algorithms were implemented in Java and executed on a JVM 8 on a Linux machine with an Intel(R) Xeon(R) E5-2637 @ 3.5 GHz and 128GB of RAM. The source code is freely available [22].

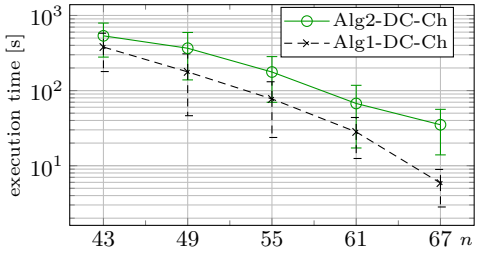
For testing, we built benchmarks with a structure similar to those proposed in earlier work [10] where each benchmark had DC CSTNs and non-DC CSTNs obtained from random workflow schemata generated by the ATAPIS toolset [15], parametrized by the number of activities, N, and the number of observations, $|\mathcal{P}|$. Here, we created three benchmarks, called B3, B4, and B5, each having 250 DC CSTNUs and 250 non-DC CSTNUs, obtained from random workflow schemata with (1) $N = 10$; (2) $|\mathcal{P}|$ equal to 3, 4 and 5, for B3, B4, and B5, respectively; and (3) representing activities as contingents links. Each benchmark is a composite of five sub-benchmarks, each having 50 instances generated by fixing also the number of parallel components in the generated workflow schemata. In particular, the



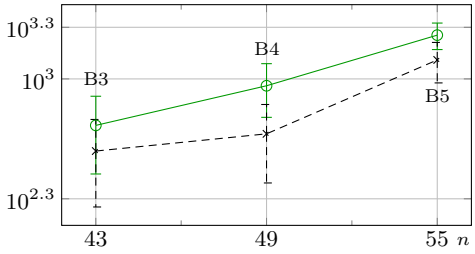
(a) Benchmark B3 for DC instances.



(b) Worst Case in B3, B4, and B5 for DC instances.



(c) Benchmark B3 for non-DC instances.



(d) Worst Case in B3, B4, and B5 for non-DC instances.

Figure 3 Experimental evaluation, $n = |\mathcal{T}|$.

maximum number of parallel components in a workflow was limited by the numbers of tasks and observations. Since the maximum number of observations was 5, we decided to consider, for each benchmark, 5 sub-benchmarks, each composed of CSTNU instances representing workflows where the number of parallel components was fixed at $0, 1, 2, 3$ or 4, respectively.

To conserve space, Figure 3a displays the average execution times of the two algorithms over all five sub-benchmarks of B3 considering DC instances. Each data point has a vertical error bar that represents a 95% confidence interval for the average execution time. With more parallel components, a random workflow can have more observations in parallel and, therefore, scenarios that are more independent of each other than scenarios that are nested. The corresponding CSTNU instance may have more nodes, but it can be easier to solve, as shown by the average execution times of the sub-benchmark in Figure 3a.

From the data in Figure 3a, and the results for benchmarks B4 and B5, it emerges that the most difficult instances are related to workflow schemata having no parallel connectors (i.e., instances belonging to the first sub-benchmark of each main benchmark). Therefore, Figure 3b reports only the average execution times obtained from instances of first sub-benchmark of B3, B4, and B5, respectively.

Figure 3c and Figure 3d show the results obtained when instances are non-DC. The two algorithms exhibit a worse performance checking non-DC instances than checking DC ones. In details, each algorithm require an average execution time that, at most, can be 8 times greater than the average execution time required for checking positive instances having the same order. We verified that such worse behavior is due to the fact that some instances have a similar configuration where a negative cycle emerge only after many iterations between lower bound and the global upper bound. We are investigating if it is possible to detect such configuration in advance in order to speed up the convergence.

However, Figure 3 demonstrates that Algorithm `Alg1-DC-Ch` tends to perform better, but the difference is not statistically significant.

8 Conclusions

This paper presented the first *practical, sound-and-complete* DC-checking algorithms for CSTNUs. The first algorithm converts an input CSTNU into a DC-equivalent CSTN, then runs an existing CSTN algorithm. The second directly propagates CSTNU constraints, using new rules that ensure completeness. An empirical evaluation demonstrated their practicality across a variety of benchmarks. Future work aims to determine whether adding new rules can speed up the algorithms.

References

1 Massimo Cairo, Carlo Comin, and Romeo Rizzi. Instantaneous reaction-time in dynamic-consistency checking of conditional simple temporal networks. In *23rd International Symposium on Temporal Representation and Reasoning (TIME 2016)*, pages 80–89, 2016. doi:10.1109/TIME.2016.16.

2 Massimo Cairo, Luke Hunsberger, Roberto Posenato, and Romeo Rizzi. A Streamlined Model of Conditional Simple Temporal Networks - Semantics and Equivalence Results. In *24th International Symposium on Temporal Representation and Reasoning (TIME 2017)*, volume 90 of *LIPIcs*, pages 10:1–10:19, 2017. doi:10.4230/LIPIcs.TIME.2017.10.

3 Massimo Cairo and Romeo Rizzi. Dynamic Controllability Made Simple. In *24th International Symposium on Temporal Representation and Reasoning (TIME 2017)*, volume 90 of *LIPIcs*, pages 8:1–8:16, 2017. doi:10.4230/LIPIcs.TIME.2017.8.

4 Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Dynamic controllability via timed game automata. *Acta Informatica*, 53(6–8):681–722, 2016. doi:10.1007/s00236-016-0257-2.

5 Carlo Combi, Luke Hunsberger, and Roberto Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. In *5th International Conference on Agents and Artificial Intelligence (ICAART-2013)*, volume 2, pages 144–156, 2013. doi:10.5220/0004256101440156.

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 *Agents and Artificial Intelligence*, volume 449 of *CCIS*, pages 314–331. Springer, 2014. doi:10.1007/978-3-662-44440-5_19.

7 Carlo Combi and Roberto Posenato. Towards temporal controllabilities for workflow schemata. In *17th International Symposium on Temporal Representation and Reasoning (TIME 2010)*, pages 129–136. IEEE Comp. Soc., 2010. doi:10.1109/TIME.2010.17.

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

9 Luke Hunsberger. Efficient execution of dynamically controllable simple temporal networks with uncertainty. *Acta Informatica*, 53(2):89–147, 2015. doi:10.1007/s00236-015-0227-0.

10 Luke Hunsberger and Roberto Posenato. Checking the Dynamic Consistency of Conditional Temporal Networks with Bounded Reaction Times. In *26th International Conference on Automated Planning and Scheduling, ICAPS 2016*, pages 175–183, 2016. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13108.

11 Luke Hunsberger and Roberto Posenato. A new approach to checking the dynamic consistency of conditional simple temporal networks. In *22nd International Conference*

on Principles and Practice of Constraint Programming, CP 2016, pages 268–286, 2016. doi:10.1007/978-3-319-44953-1_18.

12 Luke Hunsberger and Roberto Posenato. Simpler and faster algorithm for checking the dynamic consistency of conditional simple temporal networks. In *26th International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1324–1330, 2018. doi:10.24963/ijcai.2018/184.

13 Luke Hunsberger, Roberto Posenato, and Carlo Combi. The Dynamic Controllability of Conditional STNs with Uncertainty. In *PlanEx at ICAPS 2012*, pages 1–8, 2012.

14 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 *22nd International Symposium on Temporal Representation and Reasoning (TIME 2015)*, pages 4–18, 2015. doi:10.1109/TIME.2015.26.

15 Andreas Lanz and Manfred Reichert. Enabling time-aware process support with the atapis toolset. In *BPM Demo Sessions 2014*, volume 1295 of *CEUR Workshop Proceedings*, pages 41–45, 2014.

16 Richard Lenz and Manfred Reichert. It support for healthcare processes - premises, challenges, perspectives. *Data Knowl. Eng.*, 61(1):39–58, 2007. doi:10.1016/j.datak.2006.04.007.

17 Dian Liu, Hongwei Wang, Chao Qi, Peng Zhao, and Jian Wang. Hierarchical task network-based emergency task planning with incomplete information, concurrency and uncertain duration. *Knowledge-Based Systems*, 112:67–79, 2016. doi:10.1016/j.knosys.2016.08.029.

18 Paul Morris. A structural characterization of temporal dynamic controllability. In *CP 2006*, volume 4204, pages 375–389, 2006. doi:10.1007/11889205_28.

19 Paul Morris. Dynamic controllability and dispatchability relationships. In *Integration of AI and OR Techniques in Constraint Programming*, volume 8451 of *LNCS*, pages 464–479. Springer, 2014.

20 Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *AAAI-05/IAAI-05*, pages 1193–1198, 2005.

21 Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *IJCAI 2001*, pages 494–502, 2001.

22 Roberto Posenato. The CSTNU toolset. version 1.23. http://profs.scienze.univr.it/~posenato/software/cstnu, 2018.

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

Reducing ϵ-DC Checking for Conditional Simple Temporal Networks to DC Checking

Luke Hunsberger
Department of Computer Science, Vassar College, NY, USA
hunsberger@vassar.edu

Roberto Posenato
Department of Computer Science, University of Verona, Verona, Italy
roberto.posenato@univr.it
https://orcid.org/0000-0003-0944-0419

Abstract

Recent work on Conditional Simple Temporal Networks (CSTNs) has introduced the problem of checking the dynamic consistency (DC) property for the case where the reaction time of an execution strategy to observations is bounded below by some fixed $\epsilon > 0$, the so-called ϵ-DC-checking problem. This paper proves that the ϵ-DC-checking problem for CSTNs can be reduced to the standard DC-checking problem for CSTNs – without incurring any computational cost. Given any CSTN $\mathcal{S}$ with k observation time-points, the paper defines a new CSTN $\mathcal{S}_0$ that is the same as $\mathcal{S}$, except that for each observation time-point $P?$ in $\mathcal{S}$: (i) $P?$ is demoted to a non-observation time-point in $\mathcal{S}_0$; and (ii) a new observation time-point $P_0?$, constrained to occur exactly ϵ units after $P?$, is inserted into $\mathcal{S}_0$. The paper proves that $\mathcal{S}$ is ϵ-DC if and only if $\mathcal{S}_0$ is (standard) DC, and that the application of the ϵ-DC-checking constraint-propagation rules to $\mathcal{S}$ is equivalent to the application of the corresponding (standard) DC-checking constraint-propagation rules to $\mathcal{S}_0$. Two versions of these results are presented that differ only in whether a dynamic strategy for $\mathcal{S}_0$ can react *instantaneously* to observations, or only after some arbitrarily small, positive delay. Finally, the paper demonstrates empirically that building $\mathcal{S}_0$ and DC-checking it incurs no computational cost as the sizes of the instances increase.

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

Keywords and phrases Conditional Simple Temporal Networks, Dynamic Consistency, Temporal Constraints

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.15

1 Overview

A Conditional Simple Temporal Network (CSTN) is a data structure for reasoning about time in domains where some constraints may apply only in certain scenarios. For example, a patient who tests positive for a certain disease may need to receive care more urgently than someone who tests negative. In different research fields CSTNs have been used to model temporal reasoning tasks, for example, in planning [18, 21] and automating business processes [8, 15, 17].

Conditions in a CSTN are represented by propositional letters whose truth values are not controlled, but instead are *observed* in real time. Just as doing a blood test generates a positive or negative result that is only learned in real time, the execution of an *observation time-point* $P?$ in a CSTN generates a truth value for its corresponding propositional letter p.

An execution strategy for a CSTN specifies when the time-points will be executed, but cannot affect which truth values are generated by observations. However, a strategy can be *dynamic* in that its decisions can react to information from past observations. A CSTN is said to be dynamically consistent (DC) if it admits a dynamic strategy that guarantees the satisfaction of all relevant constraints no matter which outcomes are observed during execution.

Different varieties of the DC property have been defined that differ in how reactive a dynamic strategy can be. Originally, Tsamardinos *et al.* [20] stipulated that a dynamic strategy could react to an observation only after some positive delay, but that the delay could be arbitrarily small. Comin *et al.* [9] defined ϵ-DC, which assumes that a strategy's reaction time is bounded below by some fixed $\epsilon > 0$. Finally, Cairo *et al.* [2] defined π-DC, which allows a strategy to react instantaneously (i.e., after zero delay).

Several approaches to DC-checking algorithms have been presented in the literature to address the different flavors of DC [20, 4, 5, 9]. However, the only approach that has been demonstrated to be practical is the one based on the propagation of labeled constraints due to Hunsberger et al. [14, 11]. Different versions of their algorithm have been used to solve the (standard) DC-checking, ϵ-DC-checking, and π-DC-checking problems.

This paper makes the following contributions. First, it proves that the ϵ-DC-checking problem for CSTNs can be reduced to the standard DC-checking problem. The proof involves transforming the original CSTN instance $\mathcal{S}$ into a related CSTN, $\mathcal{S}_0$, and then showing that $\mathcal{S}$ is ϵ-DC if and only if $\mathcal{S}_0$ is (standard) DC. Second, the paper proves that the application of the ϵ-DC-checking constraint-propagation rules to $\mathcal{S}$ is equivalent to the application of the corresponding DC-checking constraint-propagation rules to $\mathcal{S}_0$. Third, it empirically compares the performance of (1) building $\mathcal{S}_0$ and DC-checking it versus (2) ϵ-DC-checking the original instance $\mathcal{S}$. On a suite of benchmark instances from the business application domain, the results demonstrate that building $\mathcal{S}_0$ and DC-checking it incurs no computational cost as the sizes of the instances increase.

In this way, the global contribution of the paper is to show that the ϵ-DC-checking problem is not a *new* problem (as has been suggested in recent papers), but is in fact reducible to the standard DC-checking problem. The empirical evaluation simply confirms that the ϵ-DC-checking algorithm and the algorithm based on transforming to a standard DC problem have comparable performances.

2 Background

Dechter *et al.* [10] introduced Simple Temporal Networks (STNs) to facilitate reasoning about time. An STN comprises real-valued variables, called *time-points,* and binary difference constraints on those variables. Typically, an STN includes a special time-point, Z, whose value is fixed at zero. A *consistent* STN is one that has a solution as a constraint satisfaction problem.

Tsamardinos *et al.* [20] introduced CSTNs, which augment STNs to include *observation time-points (OTPs)* and their associated *propositional letters.* In a CSTN, the execution of an OTP $P?$ generates a truth value for its associated propositional letter p. In addition, each time-point can be labeled by a conjunction of propositional literals specifying the scenarios in which that time-point must be executed. And since constraints among labeled time-points may similarly be applicable only in certain scenarios, later work generalized CSTNs to also include labels on constraints [13].

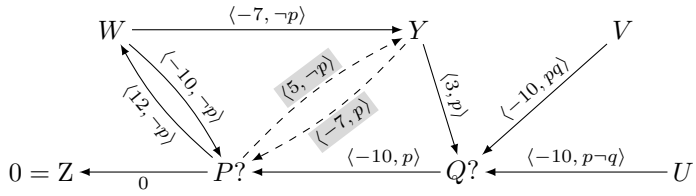


Figure 1 A sample CSTN.

Building on prior observations by Tsamardinos et al. [20], Hunsberger *et al.* [13, 6, 7] formalized properties that must be satisfied by the labels in a *well-defined* CSTN. But then Cairo *et al.* [3] showed that for any well-defined CSTN, no loss of generality results from subsequently *removing* the labels from its time-points, the result being a so-called *streamlined* CSTN. Therefore, to simplify our presentation and results, without any loss of generality, the rest of this paper restricts attention to streamlined CSTNs (i.e., CSTNs whose constraints can have propositional labels, but whose time-points cannot).

Fig. 1 shows a sample CSTN in its graphical form, where the nodes represent time-points, and the directed edges represent binary difference constraints. In the figure, Z is fixed at 0; and $P?$ and $Q?$ are OTPs whose execution generates truth values for p and q, respectively. The edge from U to $Q?$ being labeled by $p\neg q$ indicates that it applies only in scenarios where p is $\top$ and q is $\bot$. The dashed edges with shaded labels are generated by the DC-checking algorithm by Hunsberger *et al.* [14], to be discussed later on.

2.1 (Streamlined) CSTNs

The following definitions are from Hunsberger *et al.* [14], except that propositional labels appear only on constraints. Henceforth, the term CSTN shall refer to streamlined CSTNs.

▶ **Definition 1** (Labels). Given a set $\mathcal{P}$ of propositional letters: a *label* is a (possibly empty) conjunction of (positive or negative) literals from $\mathcal{P}$. The empty label is notated $\boxdot$; for any label ℓ, and any $p \in \mathcal{P}$, if $\ell \models p$ or $\ell \models \neg p$, then we say that p *appears* in ℓ; for any labels ℓ_1 and ℓ_2, if $\ell_1 \models \ell_2$, then ℓ_1 is said to *entail* ℓ_2; if $\ell_1 \wedge \ell_2$ is satisfiable, then ℓ_1 and ℓ_2 are called *consistent;* and $\mathcal{P}^*$ denotes the set of all *consistent* labels whose literals are drawn from $\mathcal{P}$.

▶ **Definition 2** (CSTN). A *Conditional Simple Temporal Network* (CSTN) is a tuple, $\langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$, where:

- $\mathcal{T}$ is a finite set of real-valued time-points (i.e., variables);
- $\mathcal{P}$ is a finite set of propositional letters (or propositions);
- $\mathcal{C}$ is a set of *labeled* constraints, each having the form, $(Y - X \leq \delta,\ \ell)$, where $X, Y \in \mathcal{T}$, $\delta \in \mathbb{R}$, and $\ell \in \mathcal{P}^*$;
- $\mathcal{OT} \subseteq \mathcal{T}$ is a set of observation time-points (OTPs); and
- $\mathcal{O}\colon \mathcal{P} \to \mathcal{OT}$ is a bijection that associates a unique OTP to each propositional letter.

In a CSTN graph, the OTP for p (i.e., $\mathcal{O}(p)$) is typically denoted by $P?$; and each labeled constraint, $(Y - X \leq \delta, \ell)$, is represented by an arrow from X to Y annotated by the *labeled value,* $\langle \delta, \ell \rangle$: $X \xrightarrow{\langle \delta, \ell \rangle} Y$. Since any time-points X and Y may participate in multiple constraints of the form, $(Y - X \leq \delta_i, \ell_i)$, the edge from X to Y may have multiple labeled values of the form, $\langle \delta_i, \ell_i \rangle$.

▶ **Definition 3** (Scenario). A function, $s\colon \mathcal{P} \to \{\top, \bot\}$, that assigns a truth value to each $p \in \mathcal{P}$ is called a *scenario.* For any label $\ell \in \mathcal{P}^*$, the truth value of ℓ determined by s is denoted by $s(\ell)$. $\mathcal{I}$ denotes the set of all scenarios over $\mathcal{P}$.

▶ **Definition 4** (Schedule). A *schedule* for a set of time-points $\mathcal{T}$ is a mapping, $\psi\colon \mathcal{T} \to \mathbb{R}$. The set of all schedules over $\mathcal{T}$ is denoted by Ψ.

The projection of a CSTN onto a scenario, s, is the STN obtained by restricting attention to the constraints whose labels are true under s (i.e., that must be satisfied in that scenario).

▶ **Definition 5** (Projection). Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$ be any CSTN, and s any scenario over $\mathcal{P}$. The *projection* of $\mathcal{S}$ onto s – notated $\mathcal{S}(s)$ – is the STN, $(\mathcal{T}, \mathcal{C}_s^+)$, where:

$$\mathcal{C}_s^+ = \{(Y - X \leq \delta) \mid \exists \ell, (Y - X \leq \delta, \ell) \in \mathcal{C} \text{ and } s(\ell) = \top\}$$

▶ **Definition 6** (Execution Strategy). An *execution strategy* for a CSTN $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$ is a mapping, $\sigma\colon \mathcal{I} \to \Psi$, from scenarios to schedules. The execution time for the time-point X in the schedule $\sigma(s)$ is denoted by $[\sigma(s)]_X$. An execution strategy σ for a CSTN $\mathcal{S}$ is *viable* if for each scenario s, the schedule $\sigma(s)$ is a solution to the projection $\mathcal{S}(s)$ (i.e., σ is guaranteed to satisfy all of the relevant constraints).

3 CSTN Reduction

This paper presents two related sets of results. Each set of results involves reducing a form of ϵ-DC checking to a form of standard DC checking (one of which involves instantaneous reaction). In each case, a given CSTN $\mathcal{S}$ is transformed (or *reduced*) to a related CSTN $\mathcal{S}_0$ such that the form of ϵ-DC checking for $\mathcal{S}$ is equivalent to the corresponding form of standard DC checking for $\mathcal{S}_0$. Although there are two different versions of this result, the translation from $\mathcal{S}$ to $\mathcal{S}_0$ is the same.

▶ **Definition 7** (Reduction CSTN, $\mathcal{S}_0$). Let $\mathcal{S}$ be any CSTN, and let $\epsilon > 0$ be arbitrary. The *reduction* of $\mathcal{S}$ is the CSTN $\mathcal{S}_0$ that is the same as $\mathcal{S}$ except that for each OTP $P?$ in $\mathcal{S}$, and its associated propositional letter p:

- $P?$ is demoted from an OTP in $\mathcal{S}$ to an ordinary time-point in $\mathcal{S}_0$;[1]
- $\mathcal{S}_0$ contains a new OTP $P_0?$ that is associated with the letter p in $\mathcal{S}_0$; and
- the constraint, $(P_0? = P? + \epsilon, \boxdot)$, is contained in $\mathcal{S}_0$.[2]

More formally, $\mathcal{S}_0 = \langle \mathcal{T} \cup \mathcal{OT}_0, \mathcal{P}, \mathcal{C} \cup \mathcal{C}_0, \mathcal{OT}_0, \mathcal{O}_0 \rangle$, where:

$$\begin{aligned} \mathcal{OT}_0 &= \{P_0? \mid P? \in \mathcal{OT}\}; \\ \mathcal{O}_0(p) = P_0? &\Leftrightarrow \mathcal{O}(p) = P?; \\ \mathcal{C}_0 &= \{(P_0? = P? + \epsilon, \boxdot) \mid P_0? \in \mathcal{OT}_0\}. \end{aligned}$$

Fig. 2 shows the reduction $\mathcal{S}_0$ that corresponds to the CSTN $\mathcal{S}$ from Fig. 1. Note that in any given instance, the value of ϵ will be fixed and known (e.g., $\epsilon = 3$).

3.1 Computational Complexity

Let $\epsilon > 0$ be arbitrary, and let $\mathcal{S}$ be any CSTN with k OTPs. The reduction of $\mathcal{S}$ to $\mathcal{S}_0$ involves $O(k)$ elementary operations for:

1. demoting original OTPs to non-OTPs;

[1] Although the demoted time-point is not an OTP in $\mathcal{S}_0$, it shall still be notated as $P?$ because keeping its name the same will simplify subsequent definitions and proofs.

[2] The constraint, $P_0? = P? + \epsilon$, abbreviates the pair of constraints, $P_0? - P? \leq \epsilon$ and $P? - P_0? \leq -\epsilon$.

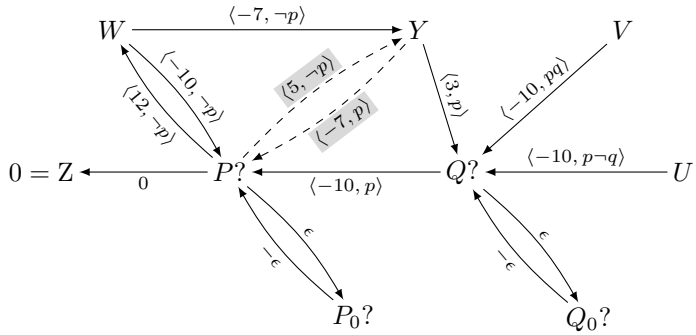


Figure 2 The reduction $\mathcal{S}_0$ corresponding to $\mathcal{S}$ from Fig. 1.

2. adding k new OTPs; and
3. inserting a pair of constraints between each demoted time-point $P?$ and the corresponding new OTP $P_0?$.

Therefore, the overall computational complexity for the reduction is $O(k)$.

4 Dynamic Strategies/Dynamic Consistency

The truth values of propositions in a CSTN are not known in advance, but a *dynamic* execution strategy can *react* to observations in real time. This paper addresses the following four flavors of dynamic strategy that differ in how reactive the strategy can be, ordered from most reactive to least reactive.

Type	Reaction Time, ρ
π-dynamic	$\rho \geq 0$
dynamic	$\rho > 0$
ϵ-dynamic	$\rho \geq \epsilon > 0$
$\hat{\epsilon}$-dynamic	$\rho > \epsilon > 0$

A π-dynamic strategy can react instantaneously to observations, but must specify an order of dependence among simultaneous observations [2]. The reaction times for a (standard) dynamic strategy can be arbitrarily small, but must be positive [20]. The reaction times for an ϵ-dynamic strategy must be greater than or equal to some fixed $\epsilon > 0$ [9]. The reaction times for an $\hat{\epsilon}$-dynamic strategy must be greater than some fixed $\epsilon > 0$. Since a CSTN is DC if and only if it has a viable and dynamic execution strategy, each distinct version of dynamic strategy gives rise to a distinct version of dynamic consistency: DC, π-DC, ϵ-DC, and $\hat{\epsilon}$-DC, respectively.

The paper will show that the $\hat{\epsilon}$-DC-checking problem can be reduced to (standard) DC checking; and that the ϵ-DC-checking problem can be reduced to π-DC checking. These reductions can be used to simplify the automated management of DC checking for CSTNs.

5 Reducing $\hat{\epsilon}$-DC Checking to (Standard) DC Checking

The following sections recall the relevant definitions for DC and $\hat{\epsilon}$-DC, and show that $\mathcal{S}$ is $\hat{\epsilon}$-DC if and only if $\mathcal{S}_0$ is DC.

5.1 Dynamic Strategies and (Standard) DC

(Standard) dynamic strategies were first defined by Tsamardinos *et al.* [20]. To facilitate comparisons with later definitions, the following are in the form given by Hunsberger *et al.* [14].

To begin, a *history* at time t comprises the truth values of all propositions that were observed *before* time t.

▶ **Definition 8** (History)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$ be any CSTN, s any scenario, σ any execution strategy for $\mathcal{S}$, and t any real number. The *history* of t in the scenario s, for the strategy σ – notated $Hist(t, s, \sigma)$ – is the set of observations made before time t according to the schedule $\sigma(s)$:

$$Hist(t, s, \sigma) = \{(p, s(p)) \mid P? \in \mathcal{OT} \text{ and } [\sigma(s)]_{P?} < t\}$$

▶ **Definition 9** (Dynamic Strategy)**.** An execution strategy σ for a CSTN $\mathcal{S}$ is called *dynamic* if for any scenarios s_1 and s_2, and any time-point X:

let: $t = [\sigma(s_1)]_X$
if: $Hist(t, s_1, \sigma) = Hist(t, s_2, \sigma)$
then: $[\sigma(s_2)]_X = t$.

In other words, if a dynamic strategy σ executes X at time t in scenario s_1, and the schedules $\sigma(s_1)$ and $\sigma(s_2)$ have the same history of *past* observations, then σ must also execute X at time t in s_2. That is, execution decisions can only depend on *past* observations, even if arbitrarily recent.

▶ **Definition 10** (Dynamic Consistency (DC))**.** A CSTN is *dynamically consistent* (DC) if there exists an execution strategy for it that is both viable and dynamic.

5.2 $\hat{\epsilon}$-Dynamic Strategies and $\hat{\epsilon}$-DC

The $\hat{\epsilon}$-dynamic consistency property has not been presented before in the literature. However, it differs only slightly from ϵ-DC, defined later on. Informally, an $\hat{\epsilon}$-dynamic strategy must schedule a time-point X at the same time t in two different scenarios s_1 and s_2 if both scenarios have the same history of *past* observations *before* time $t - \epsilon$ (i.e., ϵ units before the current time t).

▶ **Definition 11** ($\hat{\epsilon}$-History)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$ be any CSTN, s any scenario, σ any execution strategy for $\mathcal{S}$, t any real number, and $\epsilon > 0$. The *$\hat{\epsilon}$-history* of t in the scenario s, for the strategy σ, notated $\hat{\epsilon}Hist(t, s, \sigma)$, is the set of observations made *before* time $t - \epsilon$ according to $\sigma(s)$:

$$\hat{\epsilon}Hist(t, s, \sigma) = \{(p, s(p)) \mid P? \in \mathcal{OT} \text{ and } [\sigma(s)]_{P?} < t - \epsilon\}$$

▶ **Definition 12** ($\hat{\epsilon}$-Dynamic Execution Strategy)**.** Let $\epsilon > 0$; and let $\mathcal{S}$ be any CSTN. An execution strategy σ for $\mathcal{S}$ is called *$\hat{\epsilon}$-dynamic* if for any scenarios s_1 and s_2, and any time-point X:

let: $t = [\sigma(s_1)]_X$
if: $\hat{\epsilon}Hist(t, s_1, \sigma) = \hat{\epsilon}Hist(t, s_2, \sigma)$
then: $[\sigma(s_2)]_X = t$.

▶ **Definition 13** ($\hat{\epsilon}$-DC)**.** For any $\epsilon > 0$, a CSTN $\mathcal{S}$ is *$\hat{\epsilon}$-dynamically consistent* if it has a viable and $\hat{\epsilon}$-dynamic execution strategy.

The following theorem explicates the relation between $\hat{\epsilon}$-DC and DC.

▶ **Theorem 14.** *Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$ be any CSTN; let $\epsilon > 0$ be arbitrary; and let $\mathcal{S}_0 = \langle \mathcal{T} \cup \mathcal{OT}_0, \mathcal{P}, \mathcal{C} \cup \mathcal{C}_0, \mathcal{OT}_0, \mathcal{O}_0 \rangle$ be the reduction of $\mathcal{S}$. Then $\mathcal{S}$ is $\hat{\epsilon}$-DC if and only if $\mathcal{S}_0$ is DC.*

Proof. ($\Rightarrow$) Suppose that $\mathcal{S}$ is $\hat{\epsilon}$-DC. Then there exists an $\hat{\epsilon}$-dynamic and viable strategy σ for $\mathcal{S}$. Define a strategy σ_0 for the reduction $\mathcal{S}_0$, as follows. For any scenario s:

(1) For each $X \in \mathcal{T}$: let $[\sigma_0(s)]_X = [\sigma(s)]_X$

(1) For each $P_0? \in \mathcal{OT}_0$: let $[\sigma_0(s)]_{P_0?} = [\sigma(s)]_{P?} + \epsilon$

Since σ is viable for $\mathcal{S}$, σ satisfies all of the constraints in $\mathcal{C}$. And since, by (1) above, σ and σ_0 agree on all of the time-points in $\mathcal{T}$, σ_0 must also satisfy all of the constraints in $\mathcal{C}$. Finally, by (2) above, σ_0 must satisfy all of the constraints in $\mathcal{C}_0$. Therefore, σ_0 must be viable for $\mathcal{S}_0$.

Next, suppose that σ_0 is not dynamic. Then for some scenarios s_1 and s_2, and some time-point $X \in \mathcal{T} \cup \mathcal{T}_0$, $\mathit{Hist}(t, s_1, \sigma_0) = \mathit{Hist}(t, s_2, \sigma_0)$, but $[\sigma_0(s_2)]_X \neq t$, where $t = [\sigma_0(s_1)]_X$. With no loss of generality, assume that t is minimal for this circumstance. Then:

$$\begin{aligned}
\hat{\epsilon}\mathit{Hist}(t, s_2, \sigma) &= \{(p, s_2(p)) \mid P? \in \mathcal{OT} \text{ and } [\sigma(s_2)]_{P?} < t - \epsilon\} \\
&= \{(p, s_2(p)) \mid P_0? \in \mathcal{OT}_0 \text{ and } [\sigma_0(s_2)]_{P_0?} < t\}, \text{ by (2) above} \\
&= \mathit{Hist}(t, s_2, \sigma_0) \\
&= \mathit{Hist}(t, s_1, \sigma_0) \\
&= \{(p, s_1(p)) \mid P_0? \in \mathcal{OT}_0 \text{ and } [\sigma_0(s_1)]_{P_0?} < t\} \\
&= \{(p, s_1(p)) \mid P? \in \mathcal{OT} \text{ and } [\sigma(s_1)]_{P?} < t - \epsilon\}, \text{ by (2) above} \\
&= \hat{\epsilon}\mathit{Hist}(t, s_1, \sigma). \qquad (\dagger)
\end{aligned}$$

Now, if $X \in \mathcal{T}$, then $[\sigma(s_1)]_X = [\sigma_0(s_1)]_X = t$, but $[\sigma(s_2)]_X = [\sigma_0(s_2)]_X \neq t$, which, given that the relevant $\hat{\epsilon}$-histories are equal, contradicts that σ is $\hat{\epsilon}$-dynamic. Therefore, $X \notin \mathcal{T}$; thus, X must be some $R_0? \in \mathcal{OT}_0$, where $[\sigma(s_1)]_{R?} = [\sigma_0(s_1)]_{R_0?} - \epsilon = t - \epsilon \neq [\sigma_0(s_2)]_{R_0?} - \epsilon = [\sigma(s_2)]_{R?}$. Thus, the schedules $\sigma(s_1)$ and $\sigma(s_2)$ are different. Consider those schedules, each annotated with the relevant observations as they occur. Let $t^* \leq t - \epsilon$ be the earliest time at which the annotated schedules differ. There are two ways they can differ at t^*.

Case 1. Both schedules execute some OTP $Q?$ at t^*, but with different results (i.e., $s_1(q) \neq s_2(q)$). If $t^* < t - \epsilon$, it would contradict that $\hat{\epsilon}\mathit{Hist}(t, s_1, \sigma) = \hat{\epsilon}\mathit{Hist}(t, s_2, \sigma)$. Therefore, $t^* = t - \epsilon$.

Now, the definition of t^* ensures that $\hat{\epsilon}\mathit{Hist}(t^*, s_1, \sigma) = \hat{\epsilon}\mathit{Hist}(t^*, s_2, \sigma)$. But then $[\sigma(s_1)]_{R?} \neq [\sigma(s_2)]_{R?}$, shown earlier, contradicts that σ is $\hat{\epsilon}$-dynamic.

Case 2. One of the schedules executes some time-point Y at t^* while the other schedule executes Y at some later time: $[\sigma(s_i)]_Y = t^* < [\sigma(s_j)]_Y$, where $\{s_i, s_j\} = \{s_1, s_2\}$. But this, together with $\hat{\epsilon}\mathit{Hist}(t^*, s_1, \sigma) = \hat{\epsilon}\mathit{Hist}(t^*, s_2, \sigma)$, contradicts that σ is $\hat{\epsilon}$-dynamic.

($\Leftarrow$) Suppose that $\mathcal{S}_0$ is DC. Then there exists a viable and dynamic strategy σ_0 for $\mathcal{S}_0$. Let σ be the strategy for $\mathcal{S}$ such that for each scenario s, and each time-point $X \in \mathcal{T}$, $[\sigma(s)]_X = [\sigma_0(s)]_X$. Since σ_0 is viable for $\mathcal{S}_0$, it satisfies all of the constraints in $\mathcal{C} \cup \mathcal{C}_0$. And

since σ and σ_0 agree on all time-points in $\mathcal{T}$, it follows that σ satisfies all of the constraints in $\mathcal{C}$. Thus, σ is viable for $\mathcal{S}$.

Next, suppose that σ is not $\hat{\epsilon}$-dynamic. Then for some scenarios s_1 and s_2, and some time-point $X \in \mathcal{T}$, $\hat{\epsilon}Hist(t, s_1, \sigma) = \hat{\epsilon}Hist(t, s_2, \sigma)$, where $t = [\sigma(s_1)]_X$, but $[\sigma(s_2)]_X \neq t$. Arguing similarly to (†) above, it follows that $Hist(t, s_1, \sigma_0) = Hist(t, s_2, \sigma_0)$. And since $X \in \mathcal{T}$, $[\sigma_0(s_1)]_X = [\sigma(s_1)]_X = t \neq [\sigma(s_2)]_X = [\sigma_0(s_2)]_X$, contradicting that σ_0 is dynamic. ◀

6 Reducing ϵ-DC Checking to π-DC Checking

This section uses the same reduction of $\mathcal{S}$ to $\mathcal{S}_0$ to reduce the problem of ϵ-DC checking to that of π-DC checking.

6.1 ϵ-Dynamic Execution Strategy and ϵ-DC

The semantics for ϵ-DC is the same as that for $\hat{\epsilon}$-DC, except that an ϵ-history records the observations *at or before* time $t - \epsilon$, instead of *strictly before* $t - \epsilon$. Nonetheless, for easy reference, the full definitions are given below.

▶ **Definition 15** (ϵ-History)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$ be any CSTN, s any scenario, σ any execution strategy for $\mathcal{S}$, t any real number, and $\epsilon > 0$. The *ϵ-history* of t in the scenario s, for the strategy σ, notated $\epsilon Hist(t, s, \sigma)$, is the set of observations made *at or before* $t - \epsilon$ according to $\sigma(s)$:

$$\epsilon Hist(t, s, \sigma) = \{(p, s(p)) \mid P? \in \mathcal{OT} \text{ and } [\sigma(s)]_{P?} \leq t - \epsilon\}$$

▶ **Definition 16** (ϵ-Dynamic Execution Strategy)**.** Let $\epsilon > 0$. An execution strategy σ is *ϵ-dynamic* if for any scenarios s_1 and s_2, and any time-point X:

let:	$t = [\sigma(s_1)]_X$
if:	$\epsilon Hist(t, s_1, \sigma) = \epsilon Hist(t, s_2, \sigma)$
then:	$[\sigma(s_2)]_X = t$.

▶ **Definition 17** (ϵ-DC)**.** Given any $\epsilon > 0$, a CSTN is *ϵ-dynamically consistent* (ϵ-DC) if there exists an execution strategy for it that is both viable and ϵ-dynamic.

6.2 π-Dynamic Execution Strategy and π-DC

This section summarizes the π-DC semantics introduced by Cairo *et al.* [2] that allows a dynamic strategy to react instantaneously to observations, but requires an order of dependence among simultaneous observations.

▶ **Definition 18** (Order of dependence)**.** For any ordering $(P_1?, \ldots, P_k?)$ of observation time-points, where $k = |\mathcal{OT}|$, an *order of dependence* is a permutation π over $(1, 2, \ldots, k)$; and for each $P? \in \mathcal{OT}$, $\pi(P?) \in \{1, 2, ..., k\}$ denotes the integer position of $P?$ in that order. For any *non*-observation time-point X, we set $\pi(X) = \infty$. Finally, Π_k denotes the set of all permutations over $(1, 2, \ldots, k)$.

▶ **Definition 19** (π-Execution Strategy)**.** For any CSTN $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$, where $k = |\mathcal{OT}|$, a *π-execution strategy* for $\mathcal{S}$ is a mapping, $\sigma \colon \mathcal{I} \to (\Psi \times \Pi_k)$, such that for each scenario s, $\sigma(s)$ is a pair (ψ, π) where $\psi \colon \mathcal{T} \to \mathbb{R}$ is a schedule and $\pi \in \Pi_k$ is an order of dependence. For any $X \in \mathcal{T}$, $[\sigma(s)]_X$ denotes the execution time of X (i.e., $\psi(X)$); and for

any $P? \in \mathcal{OT}$, $[\sigma(s)]^{\pi}_{P?}$ denotes the position of $P?$ in the order of dependence (i.e., $\pi(P?)$). Finally, a π-dynamic strategy must be *coherent:* for any scenario s, and any $P?, Q? \in \mathcal{OT}$, $[\sigma(s)]_{P?} < [\sigma(s)]_{Q?}$ implies $[\sigma(s)]^{\pi}_{P?} < [\sigma(s)]^{\pi}_{Q?}$ (i.e., if $\sigma(s)$ *schedules* $P?$ before $Q?$, then it *orders* $P?$ before $Q?$).

▶ **Definition 20** (Viability)**.** The π-execution strategy $\sigma = (\psi, \pi)$ is called *viable* for the CSTN $\mathcal{S}$ if for each scenario s, the schedule $\psi(s)$ is a solution to the projection $\mathcal{S}(s)$.

▶ **Definition 21** (π-History)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$ be any CSTN. Let σ be any π-execution strategy for $\mathcal{S}$, s any scenario, t any real number, and $d \in \{1, 2, \ldots, |\mathcal{OT}|\} \cup \{\infty\}$ any integer position (or infinity). The π-*history* of (t, d) for the scenario s and strategy σ – denoted by $\pi Hist(t, d, s, \sigma)$ – is the set

$$\{(p, s(p)) \mid P? \in \mathcal{OT}, [\sigma(s)]_{P?} \leq t, \text{ and } \pi(P?) < d\}.$$

Thus, the π-history specifies the truth values of each proposition p that is observed *before* time t in the schedule ψ, or observed *at* time t if its corresponding observation time-point $P?$ is ordered *before* position d by the permutation π.

The following definition of a π-dynamic strategy is equivalent to that given by Cairo *et al.* [2]. (The straightforward proof has been omitted to save space.)

▶ **Definition 22** (π-Dynamic Strategy)**.** A π-execution strategy, σ, for a CSTN is π-*dynamic* if for every pair of scenarios, s_1 and s_2, and every time-point $X \in \mathcal{T}$:

let:	$t = [\sigma(s_1)]_X$, and $d = [\sigma(s_1)]^{\pi}_X$.
if:	$\pi Hist(t, d, s_1, \sigma) = \pi Hist(t, d, s_2, \sigma)$
then:	$[\sigma(s_2)]_X = t$ and $[\sigma(s_2)]^{\pi}_X = d$.

Thus, if σ executes X at time t and position d in scenario s_1, and the histories, $\pi Hist(t, d, s_1, \sigma)$ and $\pi Hist(t, d, s_2, \sigma)$, are the same, then σ must also execute X at time t and in position d in scenario s_2. (X may be an observation time-point.)

▶ **Definition 23** (π-Dynamic Consistency)**.** A CSTN, $\mathcal{S}$, is π-*dynamically consistent* (π-DC) if there exists a π-execution strategy for $\mathcal{S}$ that is both viable and π-dynamic.

Theorem 24 explicates the relationship between the ϵ-DC and π-DC properties. Its proof, which uses techniques similar to those used to prove Theorem 14, extended to accommodate the order of dependence, is omitted to save space.

▶ **Theorem 24.** *Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{OT}, \mathcal{O} \rangle$ be any CSTN; let $\epsilon > 0$ be arbitrary; and let $\mathcal{S}_0 = \langle \mathcal{T}_0, \mathcal{P}, \mathcal{C}_0, \mathcal{OT}_0, \mathcal{O}_0 \rangle$ be the reduction of $\mathcal{S}$. Then $\mathcal{S}$ is ϵ-DC if and only if $\mathcal{S}_0$ is π-DC.*

7 The π-DC- and ϵ-DC-Checking Algorithms

This section summarizes two versions of the constraint-propagation algorithm due to Hunsberger *et al.* [14][12]: the π-DC-checking algorithm and the ϵ-DC-checking algorithm. Each algorithm uses only three constraint-propagation rules.[3] This section proves that applying the rules for the ϵ-DC-checking algorithm to the CSTN $\mathcal{S}$ is equivalent to applying the rules for the π-DC-checking algorithm to the corresponding reduced CSTN $\mathcal{S}_0$.

[3] An earlier version of the π-DC-checking algorithm, called the IR-DC-checking algorithm (IR for "instantaneous reaction") used six rules, but recently, Hunsberger and Posenato [12] showed that three rules are sufficient. We applied similar techniques to create a three-rule version of the ϵ-DC-checking algorithm.

Table 1 Propagation rules for the π-DC-checking algorithm (above) and instances of their use (below).

LP:	$X \xrightarrow{\langle u,\alpha\rangle} Y \xrightarrow{\langle v,\beta\rangle} \mathrm{Z}$ $\langle u+v, \alpha\beta\rangle$	if $\alpha\beta \in \mathcal{P}^*$ and $u+v<0$
qR_0:	$P? \xrightarrow[\langle w,\alpha\rangle]{\langle w,\alpha\tilde{p}\rangle} \mathrm{Z}$	if $w<0$, $\tilde{p}\in\{p,\neg p,?p\}$, and $\alpha\in\mathcal{Q}^*$
qR_3^*:	$P? \xrightarrow{\langle w,\alpha\rangle} \mathrm{Z} \xleftarrow[\langle m,\alpha\star\beta\rangle]{\langle v,\beta\tilde{p}\rangle} Y$	if $w<0$, $\tilde{p}\in\{p,\neg p,?p\}$, and $\alpha,\beta\in\mathcal{Q}^*$
$X,Y\in\mathcal{T}$; $P?\in\mathcal{OT}$; $\mathrm{Z}=0$. In $\mathrm{qR}_0/\mathrm{qR}_3^*$, p does not appear in α or β; and $m=\max\{v,w\}$.		

LP:	$X \xrightarrow{\langle -3,pqr\rangle} Y \xrightarrow{\langle -4,rs\neg t\rangle} \mathrm{Z}$ $\langle -7, pqrs\neg t\rangle$
qR_0:	$P? \xrightarrow[\langle -9,qr\rangle]{\langle -9,(?p)qr\rangle} \mathrm{Z}$
qR_3^*:	$A? \xrightarrow{\langle -1,b\neg c\rangle} \mathrm{Z} \xleftarrow[\langle -1,b(?c)\rangle]{\langle -1,ac\rangle} B?$

7.1 The π-DC-Checking Algorithm

Table 1 lists the three constraint propagation rules used by the π-DC-checking algorithm. Note that the qR_3^* rule can generate a new kind of propositional label, called a *q-label* (defined below); and the qR_0 and qR_3^* rules can each be applied to q-labeled edges. Each q-label is a conjunction of *q-literals* (defined below). Whereas a constraint labeled by p must hold in all scenarios in which p is true, a constraint labeled by the q-literal $?p$ need only hold as long as the truth value of p is unknown (i.e., as long as $P?$ has not been executed).

▶ **Definition 25** (Q-literals, q-labels). A *q-literal* is a literal of the form $?p$, where $p\in\mathcal{P}$. A *q-label* is a conjunction of literals and/or q-literals. $\mathcal{Q}^*$ denotes the set of all q-labels. For any scenario s, and any q-literal $?p$, it is convenient to stipulate that $s \not\models ?p$.

For example, $p(?q)\neg r$ and $(?p)(?q)(?r)$ are both q-labels.

The $\star$ operator extends ordinary conjunction to accommodate q-labels. Intuitively, if the constraint C_1 is labeled by p, and C_2 is labeled by $\neg p$, then both C_1 and C_2 must hold as long as the value of p is unknown, represented by $p\star\neg p = ?p$.

▶ **Definition 26** ($\star$). The operator, $\star$: $\mathcal{Q}^*\times\mathcal{Q}^*\rightarrow\mathcal{Q}^\star$, is defined thusly. First, for any $p\in\mathcal{P}$, $p\star p = p$ and $\neg p\star\neg p=\neg p$; and for any $p_1,p_2\in\{p,\neg p,?p\}$, such that $p_1\neq p_2$, $p_1\star p_2 = ?p$. Next, for any $\ell_1,\ell_2\in\mathcal{Q}^*$, $\ell_1\star\ell_2\in\mathcal{Q}^*$ denotes the conjunction obtained by applying $\star$ in pairwise fashion to matching literals from ℓ_1 and ℓ_2, and conjoining any unmatched literals.

For example: $(p\neg q(?r)t)\star(qr\neg s) = p(?q)(?r)\neg st$.

7.2 The ϵ-DC-checking Algorithm

The ϵ-DC-checking algorithm uses the same rules as the π-DC-checking algorithm, except that in the qR_3^* rule, it uses $m=\max\{v,w-\epsilon\}$. For clarity, we shall refer to this version of the qR_3^* rule as qR_3^ϵ; and $\{\mathrm{LP},\mathrm{qR}_0,\mathrm{qR}_3^\epsilon\}$ shall be called the ϵ-DC-checking rules.

▶ **Theorem 27.** *Let $\epsilon>0$; let $\mathcal{S}$ be any CSTN; and let $\mathcal{S}^*$ be the CSTN that results from exhaustively applying the ϵ-DC-checking constraint-propagation rules to $\mathcal{S}$. Let $\mathcal{S}_0$ be the corresponding reduced CSTN for $\mathcal{S}$; and let $\mathcal{S}_0^*$ be the CSTN that results from exhaustively applying the π-DC-checking rules to $\mathcal{S}_0$. Then $\mathcal{S}^*$ and $\mathcal{S}_0^*$ are equivalent in the following sense:*

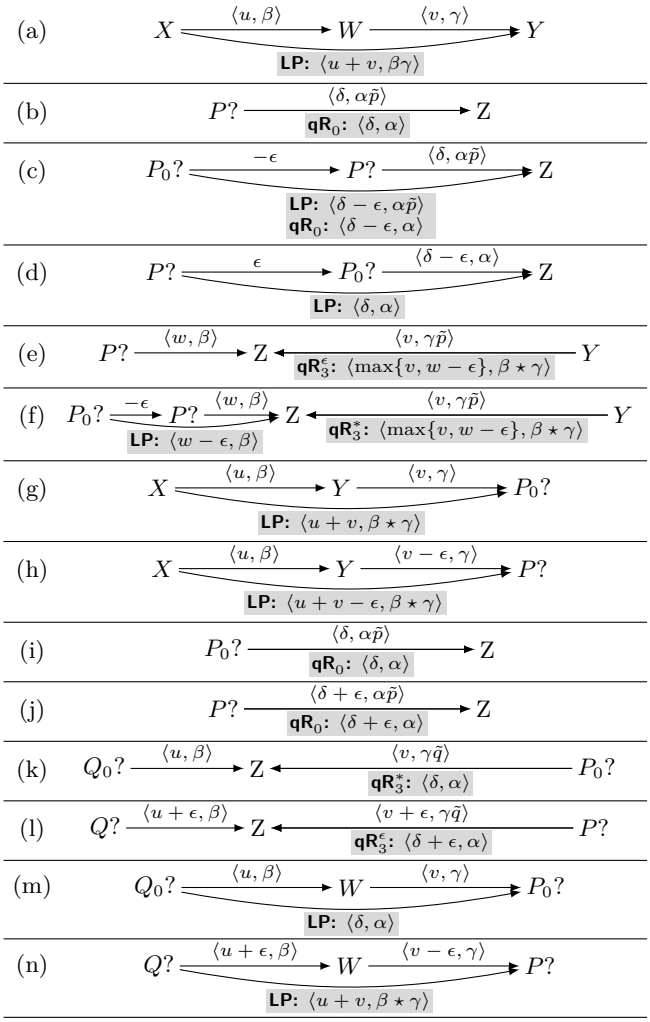


Figure 3 Constraint propagations for the proof of Theorem 27.

(1) *Every constraint in* $\mathcal{S}^*$ *is also in* $\mathcal{S}_0^*$.

(2) *For each* $P_0?, Q_0? \in \mathcal{OT}_0$, $X \in \mathcal{T} \backslash \mathcal{OT}_0$, $\delta \in \mathbb{R}$, *and* $\alpha \in \mathcal{Q}^*$:

- (a) $(P_0? - X \leq \delta, \alpha) \in \mathcal{S}_0^* \Rightarrow (P? - X \leq \delta - \epsilon, \alpha) \in \mathcal{S}^*$
- (b) $(X - P_0? \leq \delta, \alpha) \in \mathcal{S}_0^* \Rightarrow (X - P? \leq \delta + \epsilon, \alpha) \in \mathcal{S}^*$
- (c) $(P_0? - Q_0? \leq \delta, \alpha) \in \mathcal{S}_0^* \Rightarrow (P? - Q? \leq \delta, \alpha) \in \mathcal{S}^*$

Proof. **(Part 1)** Let Σ be some arbitrary sequence of applications of $\hat{\epsilon}$-DC-checking rules to edges from $\mathcal{S}^*$. Let $(Y - X \leq \delta, \alpha)$ in $\mathcal{S}^*$ be the *first* edge generated by that sequence that does *not* belong to $\mathcal{S}_0^*$. Call that constraint C. (Note that X and Y must be in $\mathcal{T}$ since C is in $\mathcal{S}^*$.)

Case 1.1: The constraint C was generated by applying the LP rule to edges in $\mathcal{S}^*$ (e.g., as shown in Fig. 3a, where $\delta = u + v$ and $\alpha = \beta\gamma$). But then, by assumption, the pre-existing edges (from X to W to Y) must also be in $\mathcal{S}_0^*$. Since the LP rule is also one of the DC-checking rules, the generated edge (from X to Y) must also be in $\mathcal{S}_0^*$. But that edge represents the constraint C, a contradiction.

Case 1.2: The constraint C was generated by the qR_0 rule (e.g., as shown in Fig. 3b). But then the pre-existing edge from $P?$ to Z must be in $\mathcal{S}_0^*$; and so is the edge from $P_0?$ to $P?$, as shown in Fig. 3c. Applying the LP rule to these edges, followed by the qR_0 rule, then yields the shaded labeled values for the edge from $P_0?$ to Z in $\mathcal{S}_0^*$, as shown in Fig. 3c. Next, applying the LP rule to the edges from $P?$ to $P_0?$ to Z, as shown in Fig. 3d, generates the edge from $P?$ to Z for $\mathcal{S}_0^*$. But that edge represents the constraint C, a contradiction.

Case 1.3: The constraint C was generated by the qR_3^ϵ rule (e.g., as shown in Fig. 3e, where $m = \max\{v, w - \epsilon\}$ and $\alpha = \beta \star \gamma$). But then the pre-existing edges (from $P_0?$ to $P?$ to Z) must be in $\mathcal{S}_0^*$, which allows the LP rule to generate the edge from $P_0?$ to Z in $\mathcal{S}_0^*$, shown in Fig. 3f, and then the qR_3^* rule to generate the shaded value for the edge from Y to Z in $\mathcal{S}_0^*$, which represents the constraint C, a contradiction.

(Part 2) Let Σ_0 be some arbitrary sequence of recursive applications of DC-checking rules to edges from $\mathcal{S}_0^*$. Let K in $\mathcal{S}_0^*$ be the *first* edge generated by that sequence that does *not* have a corresponding edge in $\mathcal{S}^*$ according to (2a), (2b) and (2c) in the statement of the theorem.

Case 2a: K has the form $(P_0? - X \leq \delta, \alpha)$. Given that $P_0? \not\equiv \text{Z}$, K can only have been generated by an application of the LP rule to edges in $\mathcal{S}_0^*$, as shown in Fig. 3g, where $\delta = u + v$ and $\alpha = \beta \star \gamma$. By assumption, the pre-existing edge from Y to $P_0?$ in $\mathcal{S}_0^*$ must have a corresponding edge from Y to $P?$ in $\mathcal{S}^*$. For example, if $Y \in \mathcal{T}$, then by (2a) there must be an edge from Y to $P?$ as shown in Fig. 3h. That edge enables the LP rule to then be used to generate the edge from X to $P?$ in $\mathcal{S}^*$, also shown in Fig. 3h. But that is the edge in $\mathcal{S}^*$ that corresponds to K, a contradiction. The case where Y is some $Q_0? \in \mathcal{OT}_0$ is even easier.

Case 2b.1: K has the form $(X - P_0? \leq \delta, \alpha)$ and was generated by applying the LP rule to edges in $\mathcal{S}_0^*$. This case is similar to Case 2a, but focusing on the lefthand side of the LP rule (i.e., the edge from X to Y) instead of the right.

Case 2b.2: K has the form $(\text{Z} - P_0? \leq \delta, \alpha)$ and was generated by applying the qR_0 rule to edges in $\mathcal{S}_0^*$, as shown in Fig. 3i. But then, by (2b), the pre-existing edge from $P_0?$ to Z has a corresponding edge in $\mathcal{S}^*$ from $P?$ to Z, leading to the propagation in Fig. 3j, which generates the edge in $\mathcal{S}^*$ that corresponds to K, a contradiction.

Case 2b.3: K has the form $(\text{Z} - P_0? \leq \delta, \alpha)$ and was generated by applying the qR_3^* rule to edges in $\mathcal{S}_0^*$, as shown in Fig. 3k, where $\delta = \max\{u, v\}$ and $\alpha = \beta \star \gamma$. By (2b), the pre-existing edges from $Q_0?$ to Z, and from $P_0?$ to Z in $\mathcal{S}_0$ have corresponding edges from $Q?$ to Z, and from $P?$ to Z in $\mathcal{S}^*$, as shown in Fig. 3l, leading to the generated edge from $P?$ to Z, where $\delta + \epsilon = \max\{u + \epsilon, v + \epsilon\}$. That edge corresponds to K, a contradiction.

Case 3: K has the form $(P_0? - Q_0? \leq \delta, \alpha)$. Since $P_0? \not\equiv \text{Z}$, then K must have been generated by an application of the LP rule. Fig. 3m illustrates the case where the intermediate time-point W is not in $\mathcal{OT}_0$, and where $\delta = u + v$ and $\alpha = \beta\gamma$. By (2b) and (2a), respectively, the pre-existing edges from $Q_0?$ to W to $P_0?$ have corresponding edges from $Q?$ to W to $P?$ in $\mathcal{S}^*$, leading to the propagation in Fig. 3n, where the edge from $Q?$ to $P?$ corresponds to K, a contradiction. The case where $W \in \mathcal{OT}_0$ is handled similarly. ◀

Theorem 27 shows that the ϵ-DC-checking and π-DC-checking algorithms perform equivalent constraint propagations. However, providing an analogous theorem that relates $\hat{\epsilon}$-DC-checking and standard DC-checking algorithms must be left to future work, since (1) no algorithm has yet been introduced for solving the $\hat{\epsilon}$-DC-checking problem; and (2) the sound 6-rule DC-checking algorithm for the standard DC-checking problem has not yet been proven complete [11].

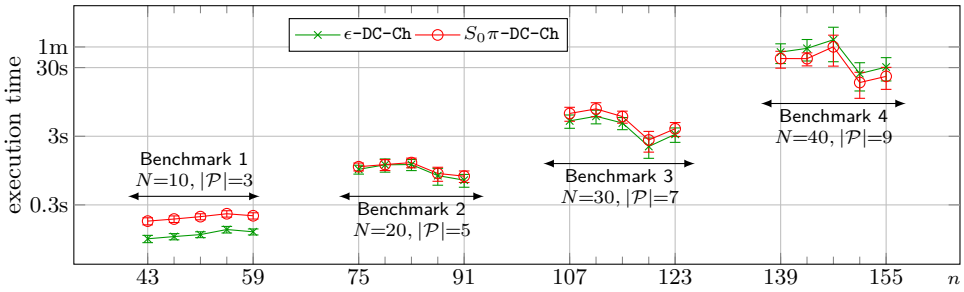


Figure 4 Execution time vs. number of time-points n.

8 Empirical Evaluation

Theorem 24 shows that the ϵ-DC-checking problem can be reduced to the π-DC-checking problem. In particular, the ϵ-DC-checking problem for a CSTN $\mathcal{S}$ is equivalent to the π-DC-checking problem for the reduced CSTN $\mathcal{S}_0$. This section compares the performance of implementations of an ϵ-DC-checking algorithm [11] and the π-DC-checking algorithm [12]. In what follows, the first algorithm is called `ϵ-DC-Ch` and the second is called $S_0\pi$`-DC-Ch` to reflect that it is applied to the reduced CSTN $\mathcal{S}_0$. Both implementations were obtained from Posenato [19], but we modified the ϵ-DC-checking algorithm to use only three rules, as discussed in Footnote 3 and Section 7.2. Algorithms and procedures for this evaluation were implemented in Java and executed on a JVM 8 in a Linux machine with two AMD Opteron 4334 CPUs and 64GB of RAM.

To facilitate comparisons with prior work, we tested both implementations on instances of the four benchmarks proposed by Hunsberger and Posenato [11]. Each benchmark has at least 60 DC and 60 non-DC CSTNs, obtained from random workflow schemata generated by the ATAPIS toolset [16]. The numbers of activities (N) and observations ($|\mathcal{P}|$) were varied, as shown in Fig. 4. Since non-DC networks were regularly solved one to two orders of magnitude faster than similarly sized DC ones, the rest of this section focuses on DC networks.

Fig. 4 displays the average execution times of the two algorithms over all four benchmarks. For $S_0\pi$`-DC-Ch`, the execution times include the time required to build S_0. Each data point represents the average of the execution times for instances of the given size. We extended the benchmarks, adding up to 50 DC instances, to generate tight error bars. In Figure 4, each error bar represents a 95% confidence interval for the average of the execution times.

The results demonstrate that, although `ϵ-DC-Ch` performs better in Benchmark 1, the performance difference becomes statistically insignificant as the sizes of the instances increase. The main reason for this behavior is that in small instances the number of observation time-points is quite small, which results in much less constraint propagation. As a result, the linear time required to build $\mathcal{S}_0$ and do the extra propagations can have a significant impact. In contrast, in larger instances, the number of observation time-points is larger, in which case the time to build $\mathcal{S}_0$ is dwarfed by the exponential time required for propagating constraints. The benchmark does not provide any larger instances because such instances are determined from random workflow instances where the maximum instance size is commonly limited to 30-40 tasks [16].

The experiments summarized in this section show that for all but the smallest CSTNs, there is no computational penalty associated with solving the ϵ-DC-checking problem by first computing the reduced CSTN $\mathcal{S}_0$ and then applying the π-DC-checking algorithm to it.

9 Conclusions

This paper presented a reduction of the $\hat{\epsilon}$-DC-checking problem to the (standard) DC-checking problem for CSTNs, and a reduction of the ϵ-DC-checking problem to the π-DC-checking problem. It also showed that the constraint-propagation rules for the π-DC-checking algorithm are equivalent to the rules for the ϵ-DC-checking algorithm. As a result, the paper showed that the ϵ-DC-checking problem for CSTNs can be easily represented within the standard CSTN framework (i.e., the ϵ-DC-checking problem is not a "new" problem, as has been suggested in recent related work). Furthermore, solving the ϵ-DC-checking problem by applying the π-DC-checking algorithm to the reduced CSTN incurs no computational penalty.

Results of this paper can be applied also in other possible variants of CSTNs. Bhargava et al. [1] defined the *delay controllability* of a Simple Temporal Network with Uncertainty (STNU). This modification allowed a network designer to insert a delay between the execution of a contingent time-point and the time that the executing agent learns of the duration of the corresponding contingent link. And the delay associated with each contingent link can be different. Similarly, a version of delay consistency for CSTNs could be defined to allow different values of epsilon for each observation time-point. In addition, the techniques used in this paper to reduce epsilon-DC for CSTNs to ordinary DC for CSTNs could be used to reduce delay controllability for STNUs to ordinary DC for STNUs.

References

1 Nikhil Bhargava, Christian Muise, Tiago Vaquero, and Brian Williams. Delay Controllability: Multi-Agent Coordination under Communication Delay. Technical Report MIT-CSAIL-TR-2018-002, MIT, 2018. URL: `http://hdl.handle.net/1721.1/113340`.

2 Massimo Cairo, Carlo Comin, and Romeo Rizzi. Instantaneous reaction-time in dynamic-consistency checking of conditional simple temporal networks. In *23rd International Symposium on Temporal Representation and Reasoning (TIME 2016)*, pages 80–89, 2016. `doi:10.1109/TIME.2016.16`.

3 Massimo Cairo, Luke Hunsberger, Roberto Posenato, and Romeo Rizzi. A Streamlined Model of Conditional Simple Temporal Networks - Semantics and Equivalence Results. In *24th International Symposium on Temporal Representation and Reasoning (TIME 2017)*, volume 90 of *LIPIcs*, pages 10:1–10:19, 2017. `doi:10.4230/LIPIcs.TIME.2017.10`.

4 A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, and M. Roveri. Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In *21st International Symposium on Temporal Representation and Reasoning (TIME 2014)*, pages 27–36. IEEE, 2014. `doi:10.1109/TIME.2014.21`.

5 A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, and M. Roveri. Dynamic controllability via Timed Game Automata. *Acta Informatica*, 53(6-8):681–722, 2016. `doi:10.1007/s00236-016-0257-2`.

6 Carlo Combi, Luke Hunsberger, and Roberto Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. In *5th International Conference on Agents and Artificial Intelligent (ICAART 2013)*, volume 2, pages 144–156, 2013. `doi:10.5220/0004256101440156`.

7 Carlo Combi, Luke Hunsberger, and Roberto Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty-revisited. In *Agents and Artificial Intelligence*, volume 449 of *CCIS*, pages 314–331. 2014. `doi:10.1007/978-3-662-44440-5_19`.

8 Carlo Combi and Roberto Posenato. Towards temporal controllabilities for workflow schemata. In *17th International Symposium on Temporal Representation and Reasoning (TIME 2010)*, pages 129–136, 2010. doi:10.1109/TIME.2010.17.

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

10 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1–3):61–95, 1991. doi:10.1016/0004-3702(91)90006-6.

11 Luke Hunsberger and Roberto Posenato. Checking the Dynamic Consistency of Conditional Temporal Networks with Bounded Reaction Times. In *26th International Conference on Automated Planning and Scheduling, ICAPS 2016*, pages 175–183, 2016. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13108.

12 Luke Hunsberger and Roberto Posenato. Simpler and faster algorithm for checking the dynamic consistency of conditional simple temporal networks. In *26th International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1324–1330. International Joint Conferences on Artificial Intelligence Organization, July 2018. doi:10.24963/ijcai.2018/184.

13 Luke Hunsberger, Roberto Posenato, and Carlo Combi. The Dynamic Controllability of Conditional STNs with Uncertainty. In *PlanEx at ICAPS 2012*, pages 1–8, 2012.

14 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 *22st International Symposium on Temporal Representation and Reasoning (TIME 2015)*, pages 4–18, 2015. doi:10.1109/TIME.2015.26.

15 Akhil Kumar and Russell R. Barton. Controlled violation of temporal process constraints – models, algorithms and results. *Information Systems*, 64:410 – 424, 2017. doi:10.1016/j.is.2016.06.003.

16 Andreas Lanz and Manfred Reichert. Enabling time-aware process support with the atapis toolset. In Lior Limonad and Barbara Weber, editors, *BPM Demo Sessions 2014*, volume 1295 of *CEUR Workshop Proceedings*, pages 41–45, 2014.

17 Andreas Lanz, Barbara Weber, and Manfred Reichert. Time patterns for process-aware information systems. *Requirements Engineering*, 19(2):113–141, 2014. doi:10.1007/s00766-012-0162-3.

18 Dian Liu, Hongwei Wang, Chao Qi, Peng Zhao, and Jian Wang. Hierarchical task network-based emergency task planning with incomplete information, concurrency and uncertain duration. *Knowledge-Based Systems*, 112:67 – 79, 2016. doi:10.1016/j.knosys.2016.08.029.

19 Roberto Posenato. A CSTN(U) consistency check algorithm implementation in Java. version 1.22. http://profs.scienze.univr.it/∼posenato/software/cstnu, 2017.

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

21 Peng Yu, Jiaying Shen, Peter Z. Yeh, and Brian Williams. Resolving over-constrained conditional temporal problems using semantically similar alternatives. In *25th International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 3300–3307, 2016.

On the Expressive Power of Hybrid Branching-Time Logics

Daniel Kernberger
School of Electrical Engineering and Computer Science
University of Kassel, Germany
daniel.kernberger@uni-kassel.de

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

Abstract

Hybrid branching-time logics are a powerful extension of branching-time logics like CTL, CTL* or even the modal μ-calculus through the addition of binders, jumps and variable tests. Their expressiveness is not restricted by bisimulation-invariance anymore. Hence, they do not retain the tree model property, and the finite model property is equally lost. Their satisfiability problems are typically undecidable, their model checking problems (on finite models) are decidable with complexities ranging from polynomial to non-elementary time. In this paper we study the expressive power of such hybrid branching-time logics beyond some earlier results about their invariance under hybrid bisimulations. In particular, we aim to extend the hierarchy of non-hybrid branching-time logics CTL, CTL$^+$, CTL* and the modal μ-calculus to their hybrid extensions. We show that most separation results can be transferred to the hybrid world, even though the required techniques become a bit more involved. We also present some collapse results for restricted classes of models that are especially worth investigating, namely linear, tree-shaped and finite models.

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

Keywords and phrases branching-time, μ-calculus, hybrid logics, expressiveness

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.16

1 Introduction

Temporal logics like LTL [14], CTL [7] and CTL* [9] are important specification formalisms for the behaviour of programs because they extend modal logic with the ability to reason about properties of unbounded or infinite computations. Their satisfiability and model checking problems are decidable, ranging from polynomial [7] to doubly exponential time [10]. This is somewhat remarkable given that typical temporal properties like "something happens infinitely often along some path" are not definable in First-Order Logic (FO). The key to decidability is bisimulation-invariance which is rooted in the modal nature of their logical operators. This, however, also limits the expressive power accordingly, for instance by not being able to distinguish a graph from its tree unfolding.

Hybrid logic [2] is the name known for a framework of extensions of modal logics with limited features of FO, aiming at increasing the expressiveness of modal logics whilst hopefully retaining most of its desirable computational properties [1]. Hybrid logics thus feature first-order variables and limited ways of manipulating them in the context of a modal or temporal logic whose evaluation in a Kripke structure can intuitively be understood as a search through

the graph. It then becomes possible to bind the current state of evaluation to a variable, to test for re-occurrence of such a state and to continue the evaluation at one such previously marked state. It is not hard to see that such features break bisimulation-invariance. Whilst this can be seen as undesirable for pure program specification purposes, hybrid logics have found some prominence in related fields like knowledge representation [3] etc.

The paper at hand presents some first results on a study of the expressive power of hybrid logics that results from an extension of well-known branching-time temporal logics – mostly CTL* and its fragments like CTL and CTL$^+$– with the aforementioned first-order features. It is part of a larger program to develop a model theory of hybrid branching-time logics. Previous work has investigated their model checking problems [12] (shown to range between polynomial space and non-elementary time/space) and introduced a small syntactical hierarchy of hybridisations of CTL* and its fragments. Thus, there is not just one hybrid CTL* but – depending on how one allows hybrid features to interact with state and path formulas – three versions of hybrid CTL*, distinguished also by computational complexity.

We briefly recall the construction of hybrid branching-time temporal logics in Sect. 2. In Sect. 3 we develop Ehrenfeucht-Fraïssé games for hybrid CTL as a standard tool for bounding the expressive power of a logic from above. These games can be seen as an extension of the games defined in [11] for hybrid CTL interpreted solely on trees. Sect. 4 then shows that some results known for branching-time logics can be lifted to their hybrid variants, at the expense of more involved constructions, though. Sect. 5 then compares hybrid branching-time logics to the extension of the modal μ-calculus with hybrid operators [13, 15]. It gives yet another argument for the distinction of the three versions of hybrid CTL* mentioned above: it is known already that – unlike the case of the non-hybrid logics – two of them cannot be translated into the hybrid μ-calculus. Here we show that the weakest of them can indeed. The paper then concludes in Sect. 6 with an overview of what is known now about the hierarchy of expressiveness amongst hybrid branching-time logics and a discussion on further work in this area.

2 The Full Hybrid Branching-Time Logic

Syntax. Let $Prop = \{p, q, \ldots\}$ be a finite set of atomic propositions and $Var = \{x, y, \ldots\}$ be a countable set of first-order *variables*. Formulas of the full hybrid branching time logic HCTL$^*_{\mathsf{pp}}$ are given by the grammar

$$\varphi := p \mid x \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid {\downarrow}x.\varphi \mid @_x\,\varphi$$
$$\psi := \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi\mathsf{U}\psi \mid {\downarrow}x.\psi \mid @_x\,\psi,$$

where $p \in Prop$ and $x \in Var$. The formulas derived by φ are called *state* formulas. Those generated by ψ are called *path* formulas. They can only occur as genuine subformulas in an HCTL$^*_{\mathsf{pp}}$ formula. We additionaly require the syntactic sanity restriction that formulas $@_x\,\psi$, where ψ is a genuine path formula, can only occur if there is no path quantifier E or A between $@_x\,\psi$ and the smallest ${\downarrow}x.\psi'$ in the syntax tree above $@_x\,\psi$.

The hybrid operators ${\downarrow}\,x$ and $@_x$ are called *binder* and *jump*. The atomic formula x is sometimes referred to as *variable test*. We are making use of the usual propositional abbreviations for $\mathtt{tt}$, $\mathtt{ff}$, $\wedge$, as well as the temporal ones $\mathsf{F}\psi := \mathtt{tt}\mathsf{U}\psi$, $\mathsf{G}\psi := \neg\mathsf{F}\neg\psi$, $\mathsf{A}\psi := \neg\mathsf{E}\neg\psi$, $\psi_1\mathsf{R}\psi_2 := \neg(\neg\psi_1\mathsf{U}\neg\psi_2)$ etc.

Semantics. Formulas of HCTL$^*_{\mathsf{pp}}$ are interpreted with respect to Kripke structures. A *Kripke structure* is a tuple $\mathcal{K} = \langle S, \to, L\rangle$ where S is a set of states, $\to \subseteq S \times S$ is a transition relation such that for every $s \in S$ there is a $t \in S$ with $s \to t$ and $L : S \to 2^{AP}$ is a labeling function.

A *path* π in $\mathcal{K}$ is an infinite sequence of pairs of states and the propositions that hold at them: $(s_0, L(s_0)), (s_1, L(s_1)), (s_2, L(s_2)), \ldots \in (S \times 2^{Prop})^\omega$ such that $s_i \to s_{i+1}$ for every $i = 0, 1, \ldots$. For a path π we write π^i to denote the i-th state of the path.

Hybrid branching-time formulas are interpreted over Kripke structures. State formulas are interpreted with respect to a state s of a Kripke structure $K = \langle S, \to, L\rangle$ and a variable mapping $\rho : \mathit{Var} \to S$ in order to give meaning to free variables in the inductive definition of the semantics as follows.

$$
\begin{aligned}
K, s, \rho &\models p && \text{iff } p \in L(s)\\
K, s, \rho &\models \neg\varphi && \text{iff } K, s, \rho \not\models \varphi\\
K, s, \rho &\models \varphi_1 \vee \varphi_2 && \text{iff } K, s, \rho \models \varphi_1 \text{ or } K, s, \rho \models \varphi_2\\
K, s, \rho &\models \mathsf{E}\psi && \text{iff there exists a path } \pi \text{ with } \pi^0 = s \text{ and } K, \pi, \rho \models \psi\\
K, s, \rho &\models x && \text{iff } \rho(x) = s\\
K, s, \rho &\models \downarrow x.\varphi && \text{iff } K, s, \rho[x \mapsto s] \models \varphi\\
K, s, \rho &\models @_x \varphi && \text{iff } K, \rho(x), \rho \models \varphi.
\end{aligned}
$$

Path formulas are interpreted on a path. To give meaning to them properly we need a function $\sigma : \mathit{Var} \to \mathbb{N}$ that stores the position of a bound variable on the current path under evaluation which helps to give meaning to the jump operator.[1] Thus, path formulas here are interpreted over a path π in K, a moment k on the path, a variable mapping ρ and a function σ storing the position on the path to which a variable is bound:

$$
\begin{aligned}
K, \pi, k, \rho, \sigma &\models \varphi && \text{iff } K, \pi^k, \rho \models \varphi\\
K, \pi, k, \rho, \sigma &\models \psi_1 \vee \psi_2 && \text{iff } K, \pi, k, \rho, \sigma \models \psi_1 \text{ or } K, \pi, k, \rho, \sigma \models \psi_2\\
K, \pi, k, \rho, \sigma &\models \mathsf{X}\psi && \text{iff } K, \pi, k+1, \rho, \sigma \models \psi\\
K, \pi, k, \rho, \sigma &\models \psi_1 \mathsf{U} \psi_2 && \text{iff there exists } j \in \mathbb{N} \text{ with } j \geq k \text{ such that } K, \pi, j, \rho, \sigma \models \psi_2\\
& && \quad \text{and for all } k \leq i < j\text{: } K, \pi, i, \rho, \sigma \models \psi_1\\
K, \pi, k, \rho, \sigma &\models \downarrow x.\psi && \text{iff } K, \pi, k, \rho[x \to \pi^k], \sigma[x \to k] \models \psi\\
K, \pi, k, \rho, \sigma &\models @_x \psi && \text{iff } K, \pi, \sigma(x), \rho, \sigma \models \psi.
\end{aligned}
$$

Fragments. The index $\cdot_{\mathsf{pp}}$ in the logic's name stands for $\mathsf{path}-\mathsf{path}$ and indicates that binders as well as jumps can range over path formulas (and therefore also state formulas). We obtain (syntactically) weaker fragments imposing stronger restrictions here.

First, if we restrict path formulas to

$$\psi := \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi\mathsf{U}\psi \mid \downarrow x.\psi,$$

i.e. disallow jumps on path formulas or, equivalently, require them to operate on genuine state formulas only, then we get the $\mathsf{path}-\mathsf{state}$ fragment $\mathrm{HCTL}^*_{\mathsf{ps}}$. If we also disallow $\downarrow x.\psi$ on path formulas we obtain the $\mathsf{state}-\mathsf{state}$ fragment $\mathrm{HCTL}^*_{\mathsf{ss}}$ in which hybrid-operators can only occur as state-formulas.

[1] The syntactic restriction of $\mathrm{HCTL}^*_{\mathsf{pp}}$ formulas about the non-occurrence of path quantifiers between binders and corresponding jumps ensures that variables which are referenced while evaluating a path formula are actually bound on the same path. This makes the semantics be well-defined.

We can also consider hybrid variants of weaker branching-time temporal logics as they are known in the literature: if we restrict the temporal operators further to not allow nesting of path formulas with the exception of fairness constraints, i.e. path formulas are generated by the grammar

$$\psi := \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi \mid \mathsf{GF}\varphi$$

we obtain the logic HFCTL$^+$. Restricting the grammar even further to

$$\psi := \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi$$

we get HCTL$^+$. And finally, also disallowing boolean combinations of path formulas,

$$\psi := \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi$$

we get HCTL.

Lastly, we also need linear temporal logic for some technical details. We obtain hybrid LTL by restricting the grammar of HCTL$^*_{\mathsf{pp}}$ to

$$\varphi := \mathsf{E}\psi$$
$$\psi := p \mid x \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi\mathsf{U}\psi \mid \downarrow x.\psi \mid @_x\,\psi.$$

Thus, hybrid LTL formulas basically consist only of a single path formula. Furthermore, hybrid LTL formulas are only interpreted over linear Kripke structures, i.e. Kripke structures $\mathcal{K} = \langle S, \to, L\rangle$ with $S = \mathbb{N}$ and $s \to t$ if and only if $t = s + 1$.

The usual non-hybrid temporal logics CTL, CTL$^+$, FCTL$^+$, CTL* and LTL are obtained by completely restricting the use of hybrid operators in their respective hybrid-variants. Note that the possibility to nest path formulas arbitrarily is vital for the distinction between the three fragments of $\mathsf{path}-\mathsf{path}$ etc. formulas. Hence, for hybrid variants of branching-time logics "smaller" than CTL* we do not make these distinctions anymore.

3 Ehrenfeucht-Fraïssé Game for HCTL

We start by defining Ehrenfeucht-Fraïssé (EF) games in order to capture the expressive power of HCTL. Such games often prove to be useful when comparing the expressive power of two logics because they condense all possibilities of how to distinguish two structures via a logical formula into a single framework of finding a winning strategy for one player. In this paper we will only use such games for HCTL. However, the general framework of these games can also be extended to similar games for the other hybrid logics above HCTL and may prove to be useful for future research into this topic.

Hybrid logics extend branching-time logics with certain first-order aspects. Thus, it is not surprising that these expressiveness games combine features from branching-time logics with aspects of EF games which are known from FO [6].

▶ **Definition 1.** Let $K_0 = \langle S_0, \to_0, L_0\rangle$ and $K_1 = \langle S_1, \to_1, L_1\rangle$ be two Kripke structures. The game $\mathcal{G}^m_{\mathrm{HCTL}}(K_0, s_0, K_1, s_1)$ is played between two players – Spoiler and Duplicator on K_0 and K_1.

The game is played for m rounds. At the beginning we only have one pebble in each structure placed at s_0 resp. s_1. In each round a new pair of pebbles gets placed on K_0 and K_1 according to the following rules. Spoiler first chooses one of the structures K_i, $i \in \{0, 1\}$, and a previously placed pebble p_i in this structure and then chooses one of the following moves:

- Choose a successor of p_i and place a new pebble on this successor. Duplicator then responds by choosing a pebble in K_{1-i} and also places a new pebble on one of its successors.
- Spoiler chooses a path π_i starting at p_i and a position l on this path. Then Duplicator chooses a path π_{1-i} on K_{1-i} starting at some previously placed pebble and some position l' on this path. Now Spoiler has two options. He can place a new pebble on π_i^l, forcing Duplicator to place a new pebble on $\pi_{1-i}^{l'}$. Or he can choose some $k' < l'$ and place a new pebble at $\pi_{1-i}^{k'}$ on K_{1-i}. Duplicator then has the possibility to choose some $k < l$ and place a new pebble at π_i^k on K_i.

It is Duplicator's task to maintain the following conditions after each round:

- For all pairs of pebbles placed in some round it holds that the states marked by those pebbles agree on all $p \in \mathit{Prop}$.
- For all pairs of pebbles (p_i, p_i') and (p_j, p_j') it holds that $p_i = p_j$ iff $p_i' = p_j'$

Spoiler wins if Duplicator cannot maintain these conditions after some round. Duplicator wins if Spoiler has not won after m rounds.

The following theorem can be proven by a standard induction on the number of rounds in this game; it is carried out in the appendix. Observe that the two types of moves cover the until- and next-operators. Hybrid operators are covered by placing pebbles and starting from some pebble in each round.

▶ **Theorem 2.** *If Duplicator wins $\mathcal{G}^m_{\mathrm{HCTL}}(K_0, s_0, K_1, s_1)$ then it holds for all formulas $\varphi \in$ HCTL with temporal nesting depth at most m that $K_0, s_0 \models \varphi$ if and only if $K_1, s_1 \models \varphi$.*

These games are essentially a combination of Ehrenfeucht-Fraissé games for first-order logic and expressiveness games for branching-time logics. The moves closely resemble the temporal operators in branching-time logic. However, in each step we also remember the new state similar to Ehrenfeucht-Fraissé games. As a consequence winning strategies also combine elements of both types of games.

4 The Expressive Power of Hybrid Branching-Time Logics

We begin by studying the connection between HCTL and HCTL$^+$ and will then continue to work our way up to the logics above HCTL$^+$.

4.1 HCTL$^+$ and HCTL

We first show that HCTL$^+$ $\equiv$ HCTL. Thus, as in the non-hybrid case adding boolean connectives to the path formulas does not increase the expressive power. The proof is very similar to the translation in the non-hybrid case [8] and has already been extended to HCTL$^+$ over tree-structures [11]. However, we will build upon the translation in Theorem 5. So we briefly review the key elements for the translation in the hybrid case here.

▶ **Theorem 3.** *For each* HCTL$^+$ *formula φ there is an* HCTL *formula φ' such that for all Kripke structures $K = \langle S, \to, L\rangle$, states $s \in S$ and variable assignments $\sigma : \mathit{Var} \to S$ it holds that $K, s, \sigma \models \varphi$ if and only if $K, s, \sigma \models \varphi'$.*

Proof sketch. Inspecting the grammar for HCTL$^+$, we see that the hybrid operators only occur as state formulas. Thus, path formulas are essentially non-hybrid in the sense that they are evaluated with respect to a fixed variable interpretation. Fixed variables however can simply be regarded as atomic propositions that happen to hold at exactly one state.

For this reason, we can simply utilise the techniques that are used in the non-hybrid case and follow [8]: we first transform path formulas into a normal form such that every path formula is a conjunction of Next-, Until- and Generally-operators and then guess the order in which all Until-formulas will be satisfied making sure that the Next- and Generally-formulas are also satisfied. ◀

A detailed proof of Theorem 3 can be found in the appendix.

4.2 HFCTL$^+$ and HCTL$^+$

In the non-hybrid case we know that CTL$^+$ is less expressive than FCTL$^+$. For example it is shown in [9] that the formula $\mathsf{EGF}p$, which states that there is a path along which p holds infinitely often, cannot be expressed by CTL$^+$.

A similar result was already shown for HCTL$^+$ interpreted over computation trees in [11]. This of course also gives us a separation result over general Kripke structures.

▶ **Theorem 4.** *There is no formula in* HCTL$^+$ *that is equivalent to the* HFCTL$^+$ *formula* $\mathsf{EGF}p$.

However, if we only consider finite structures the picture is different. Next we will see that Theorem 4 no longer holds on finite structures. This is because on finite structures we can characterise an infinite occurrence of p on a finite structure by a state that satisfies p and that lies on some (finite) loop in the structure. Thus, on finite structures we get that the HFCTL$^+$ formula $\mathsf{EGF}p$ is equivalent to the HCTL formula $\mathsf{EF}{\downarrow}x.\mathsf{EF}(p \wedge \mathsf{EXEF}x)$.

Extending the well-known translation from HCTL$^+$ to HCTL a bit we get an even stronger result:

▶ **Theorem 5.** *On finite structures, every* HFCTL$^+$ *formula is logically equivalent to an* HCTL *formula.*

Proof. We start with the same equivalences used in the proof for Theorem 3 and transform each path formula into one of the form

$$\mathsf{E}(\mathsf{X}\Lambda_1 \wedge \bigwedge_{i\in I_1} \varphi_i \mathsf{U} \psi_i \wedge \mathsf{G}\Lambda_2 \wedge (\bigwedge_{i\in I_2} \mathsf{GF}\chi_i)). \tag{1}$$

To transform this formula into an HCTL formula we first guess the order in which all Until-formulas are satisfied – ignoring the fairness constraints for the initial part – and then we guess a point from which there are cyclic paths along which all χ_i formulas will be satisfied. Thus, we get the following translation:

$$\Lambda_2 \wedge \bigvee_{J\subseteq I_1} (\bigwedge_{j\notin J} \psi_j) \wedge (\bigwedge_{j\notin J} \varphi_j) \wedge \mathsf{EX}\Big(\Lambda_1 \wedge \bigvee_{\pi\in \mathsf{Perm}(J)} \mathsf{E}((\Lambda_2 \wedge \bigwedge_{j\in J} \varphi_{\pi(j)})\mathsf{U}\Big(\psi_{\pi(1)} \wedge \\ \vdots \\ \mathsf{E}((\Lambda_2 \wedge \varphi_{\pi(|J|)})\mathsf{U}(\psi_{\pi(|J|)} \wedge \xi)\Big)\ldots\Big)$$

where $\mathsf{Perm}(J)$ denotes the set of all permutations over J and

$$\xi := \mathsf{E}(\Lambda_2 \mathsf{U}(\Lambda_2 \wedge {\downarrow}x. \bigwedge_{i\in I_2} \mathsf{E}(\Lambda_2 \mathsf{U}(\Lambda_2 \wedge \chi_i \wedge \mathsf{E}(\Lambda_2 \mathsf{U} x))))).$$

Suppose some state satisfies (1) in some finite structure. Then there is a path π satisfying all conjuncts of (1). We only argue correctness of the translation for the infinite part of the path after all Until-formulas were satisfied and also ignore that Λ_2 is satisfied on every state of the path. Correctness for the initial part follows along the same lines as the translation from HCTL$^+$ to HCTL in Theorem 3.

Since the structure is finite there has to be some point x occurring infinitely often along π. Furthermore, since every χ_i is satisfied infinitely often there has to be a part of the path such that χ_i is satisfied on some state between two occurrences of x. Thus, for every i there is a cycle starting at x in the structure along which χ_i is satisfied. Hence, ξ is satisfied. The converse direction follows by a piecewise reconstruction of the whole path with infinitely many occurrences of each cycle satisfying some χ_i. ◀

4.3 HCTL$^*_{\mathsf{ss}}$ and HFCTL$^+$

In the following we will prove that HCTL$^*_{\mathsf{ss}}$ is still more expressive than HFCTL$^+$. We will first prove that there are two classes of *finite structures* distinguishable by HCTL$^*_{\mathsf{ss}}$ such that no HCTL formula can distinguish these classes. Together with Theorem 5 this will also prove that HCTL$^*_{\mathsf{ss}}$ is more expressive than HFCTL$^+$.

To see this, we define the structure $\mathcal{A}$ as depicted in Figure 1a. Note that $\mathcal{A}$, despite being infinite as a whole, is essentially finite from each state because every path simply traverses the structure downwards ending either in t_1 or in t'_1. In the following we refer to the index of a state's name as the level of the structure and the letter of its name as the type of the state. Also note that each path that goes from level i to level $i-1$ goes either through s_{i-1} or s'_{i-1}.

▶ **Theorem 6.** *Let $n \in \mathbb{N}$ and $m = 2^{n+1} + n$. Duplicator wins $\mathcal{G}^n_{\mathrm{HCTL}}(\mathcal{A}, s_m, \mathcal{A}, s'_m)$.*

Proof. We describe a winning strategy for Duplicator. Suppose that i rounds have been played already. To win, Duplicator maintains the following invariant throughout the game:

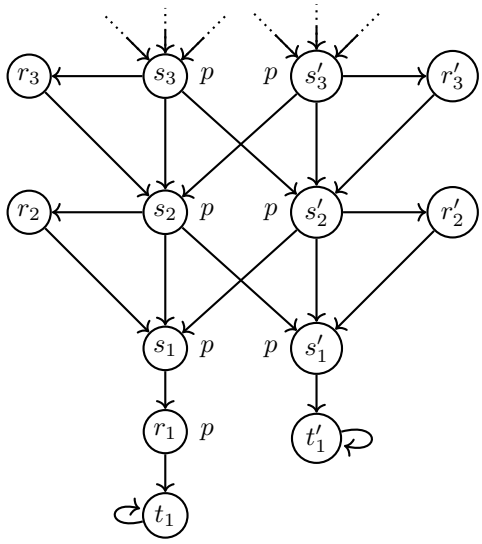
- The pebbles placed by Duplicator are always on the same level and of the same type as Spoiler's pebble in the same round.
- There is some $k \geq 2^{n-i}$ such that each pair of pebbles placed by Spoiler and Duplicator in the same round on level k or smaller mark exactly the same state. Furthermore, pairs of pebbles placed above level k are on opposing sides in the structure.
- The first pair of pebbles placed above level k is at least on level $k + 2^{n-i} + (n-i)$.

It is obvious that if Duplicator can maintain this invariant for n rounds then she wins. Furthermore, at the beginning of the game the invariant holds with $k = 2^n$.

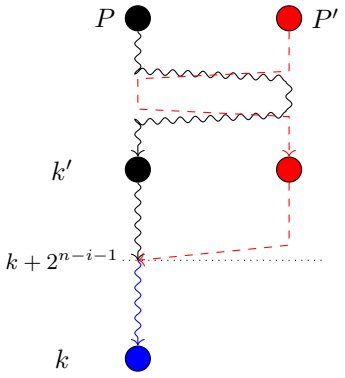
Suppose now that i rounds have been played in the game and Spoiler decides to move from some pebble P. Duplicator will always answer with the pebble P' on the same level. Spoiler has two types of moves available. First, he can simply choose a successor of P. If P, P' mark the same state then Duplicator simply copies Spoiler's pick, if not then P' is on the opposing side of the structure compared to P and Duplicator chooses the same type of successor on the other side of the structure. In both cases the invariant is maintained.

Suppose Spoiler chooses some path through the structure and some point on this path. It is obvious that to maintain the first part of the invariant Duplicator has to choose his endpoint of the path on the same level and at the same type of node as Spoiler. There are two possibilities for Spoiler's path:

First, P is at or below level k. In this case P, P' mark the same state. Thus, Duplicator simply copies Spoiler's path as well as his choice for the next pebble.



(a) Structure $\mathcal{A}$.



(b) Sketch of Duplicator's path choice. The black path is Spoiler's choice. Duplicator's path is depicted in red. The blue part means that both paths have joined.

Figure 1 Structure $\mathcal{A}$ and a sketch of Duplicator's path choice on $\mathcal{A}$.

Second, P is above level k. So P and P' are on opposing sides of the structure. Let k' be the level of the lowest pebbles above k. Then $k' \geq k + 2^{n-i} + (n-i)$. Suppose for the moment that Spoiler has chosen his endpoint of the path anywhere else than at the state $r_{k+2^{n-i-1}+1}$. Then Duplicator chooses his path as follows:

- Up to level $k + 2^{n-i-1} + 1$ he mimics Spoiler's path but on the opposing side of the structure. For example if Spoiler's path goes through s_j then Duplicator's will go through s'_j etc.
- At $s_{k+2^{n-i-1}+1}$ or $s'_{k+2^{n-i-1}+1}$ Duplicator's path changes sides to meet up with Spoiler's path at $s_{k+2^{n-i-1}}$ or $s'_{k+2^{n-i-1}}$ – even if Spoiler's path goes through $r_{k+2^{n-i-1}+1}$ or $r'_{k+2^{n-i-1}+1}$.
- From $s_{k+2^{n-i-1}}$ or $s'_{k+2^{n-i-1}}$ he simply copies Spoiler's path.

A rough illustration of Duplicator's path is depicted in Figure 1b. Spoiler now has only two options (the case that Spoiler chooses to play the endpoint of his path gets subsumed here since Duplicator will always play the same node on the same level). Either he places a new pebble above level $k + 2^{n-i-1}$ on Duplicator's path. Then Duplicator can answer with the same type of node on the same level in Spoiler's path on the opposing side of the structure. Or Spoiler places a pebble at or below level $k + 2^{n-i-1}$ on Duplicator's path. In this case Duplicator simply answers with the same state since both paths are the same there.

In both cases one can check that the invariant is maintained, possibly with a bigger k if Spoiler's choice is somewhere between level k and $k + 2^{n-i-1}$. In any case, the gap to k' is large enough to also maintain the third part of the invariant. This strategy only works if Spoiler cannot explicitly choose to go to $r_{k+2^{n-i-1}+1}$ or $r'_{k+2^{n-i-1}+1}$, i.e. if he does not choose either of those points as the endpoints of his path, since Duplicator's path changes the sides of the structure on this level and thus cannot go through the opposing r-node.

So, suppose Spoiler has chosen a path through $r_{k+2^{n-i-1}+1}$ or $r'_{k+2^{n-i-1}+1}$ and has chosen this point as his endpoint of the path. In this case Duplicator chooses almost the same path as before, however he switches sides to Spoiler's path one level earlier such that if Spoiler

decides to go to $r_{k+2^{n-i-1}+1}$ resp. $r'_{k+2^{n-i-1}+1}$ Duplicator can move to the same state. As one can check the invariant here is also maintained with a new k between the previous k and $k + 2^{n-i-1} + 1$ – at most one bigger than before.

Thus, by maintaining this invariant Duplicator wins $\mathcal{G}^n_{\mathrm{HCTL}}(\mathcal{A}, s_{2^{n+1}+n}, \mathcal{A}, s'_{2^{n+1}+n})$. ◀

▶ **Theorem 7.** *There is no* HFCTL$^+$ *formula that is logically equivalent to the* CTL* *formula* $\mathsf{AF}(p \wedge \mathsf{X}p)$.

Proof. Suppose there was such a formula. By Theorem 5 this formula would be equivalent to an HCTL formula on finite structures. Let n be the operator depth of this HCTL formula. Then by Theorem 2 and 6 this formula cannot distinguish between the states $s_{2^{n+1}+n}$ and $s'_{2^{n+1}+n}$ in $\mathcal{A}$. However, $\mathcal{A}, s_j \models \mathsf{AF}(p \wedge \mathsf{X}p)$ while $\mathcal{A}, s'_j \not\models \mathsf{AF}(p \wedge \mathsf{X}p)$ for all j. ◀

Thus, we get that CTL* and HFCTL$^+$ are incomparable and since HCTL$^*_{\mathsf{ss}}$ is an extension of CTL* and HFCTL$^+$:

▶ **Corollary 8.** *Already on finite structures* HCTL$^*_{\mathsf{ss}}$ *is more expressive than* HFCTL$^+$.

This result transfers to general Kripke structures. However, it is quite interesting to see that this proof does not simply carry over to the class of computation trees.

There, we can exploit that the path to some state s is unique. Using this, we can for example get that on tree structures the formula $\mathsf{AF}(p \wedge \mathsf{X}p)$ is equivalent to the HCTL formula $\downarrow s.\mathsf{AF}(p \wedge \downarrow x.\, @_s\, \mathsf{EF}(\mathsf{EX}x \wedge p))$. The latter formula states that on all paths we can finally find some state y satisfying p such that if we jump back to the root of the tree we can find a state that has a successor y and also satisfies p. Since predecessors on trees are unique this equivalence holds on all trees.

Since the latter formula is already in HCTL the proof showing that CTL* is not subsumed by HCTL does not carry over.

5 Hybrid Temporal Logics and the Hybrid μ-Calculus

The hybrid μ-calculus H$_\mu$ is an extension of the modal μ-calculus L$_\mu$ with hybrid operators. However, despite increasing the expressive power of the modal μ-calculus substantially it is known that H$_\mu$ – contrary to the non-hybrid case – does not subsume all hybrid extensions of CTL*. In [13] it was shown that formulas of H$_\mu$ using at most k first-order variables are invariant under hybrid k-bisimulations – a bisimulation notion that links $(k+1)$-tuples of states such that the i-th states of these tuples have matching atomic propositions, matching (hybrid) accessibility relations which includes jumping to the other k states and rebinding them and also match in regard to the other k states of the tuple. It is not too hard, though, to see that HCTL$^*_{\mathsf{ps}}$ contains formulas which are not invariant under hybrid k-bisimulations for any k. An example is $\mathsf{EG}(\downarrow x.\mathsf{XG}\neg x)$ stating that there is a loop-less infinite path. As an immediate consequence one obtains that HCTL$^*_{\mathsf{ps}}$ cannot be embedded into H$_\mu$.

We continue the study of the connection between hybrid extensions of CTL* and H$_\mu$ in this section. First, we briefly recall H$_\mu$. Secondly, we show that at least HCTL$^*_{\mathsf{ss}}$ is subsumed by H$_\mu$. As a byproduct of this translation we get that HCTL$^*_{\mathsf{ps}}$ is strictly more expressive than HCTL$^*_{\mathsf{ss}}$ and that the latter is also invariant under hybrid k-bisimulations. And finally, we lift the standard proof that the μ-calculus is more expressive than CTL* to the hybrid world, showing that HCTL$^*_{\mathsf{pp}}$ and H$_\mu$ are incomparable in terms of their expressive power.

5.1 The Hybrid μ-Calculus

Let $Var_2 = \{X, Y, \ldots\}$ be a countable set of second-order variables that is disjoint from *Var* and *Prop*. Formulas of the fully hybrid μ-calculus H_μ are given by the grammar

$$\varphi := p \mid x \mid X \mid \neg\varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid @_x\,\varphi \mid \downarrow x.\varphi \mid \mu X.\varphi(X)$$

where $p \in Prop$, $x \in Var$ and $X \in Var_2$. The modal μ-calculus L_μ is obtained by disallowing the occurrence of first-order variables. We make use of $\mathtt{tt}$, $\mathtt{ff}$, $\wedge$, $\Diamond$, $\nu X.\varphi$ as abbreviations in the usual way, and we assume the following standard sanity condition on formulas: every $X \in Var_2$ is bound at most once by a fixpoint quantifier μ or ν and can only occur under an even number of negations within its binding formula.

Formulas of H_μ are interpreted over Kripke structures $K = \langle S, \to, L\rangle$. Formally the semantics for H_μ with respect to a Kripke structure $K = \langle S, \to, L\rangle$ over *Prop* and an assignment $\rho : Var_2 \to 2^{S \times (Var \to S)}$ is the following:

$$\begin{aligned}
[\![p]\!]^K_\rho &= \{(s,\sigma) \mid p \in L(s)\},\\
[\![X]\!]^K_\rho &= \rho(X),\\
[\![x]\!]^K_\rho &= \{(s,\sigma) \mid s = \sigma(x)\},\\
[\![\neg\varphi]\!]^K_\rho &= \{(s,\sigma) \mid (s,\sigma) \notin [\![\varphi]\!]^K_\rho\},\\
[\![\varphi_1 \vee \varphi_2]\!]^K_\rho &= [\![\varphi_1]\!]^K_\rho \cup [\![\varphi_2]\!]^K_\rho,\\
[\![\Box\varphi]\!]^K_\rho &= \{(s,\sigma) \mid \forall t \in S\text{: if } s \to t, \text{ then } (t,\sigma) \in [\![\varphi]\!]^K_\rho\},\\
[\![@_x\,\varphi]\!]^K_\rho &= \{(s,\sigma) \mid (\sigma(x),\sigma) \in [\![\varphi]\!]^K_\rho\},\\
[\![\downarrow x.\varphi]\!]^K_\rho &= \{(s,\sigma) \mid (s,\sigma[x \mapsto s]) \in [\![\varphi]\!]^K_\rho\},\\
[\![\mu X.\varphi(X)]\!]^K_\rho &= \bigcap\{T \subseteq S \times (Var \to S) \mid [\![\varphi]\!]^K_{\rho[X \to T]} \subseteq T\}
\end{aligned}$$

with $p \in Prop$, $x \in Var$ and $X \in Var_2$. We write $K, s, \sigma, \rho \models \varphi$ if $(s,\sigma) \in [\![\varphi]\!]^K_\rho$. If there are no free second-order variables we also may drop ρ.

5.2 $\mathrm{HCTL}^*_{\mathsf{ss}}$ and H_μ

Our aim is to translate $\mathrm{HCTL}^*_{\mathsf{ss}}$ into H_μ. However, already for CTL^* the translation into the μ-calculus is nontrivial. A key part in the non-hybrid translation is that path formulas can be regarded as simple LTL formulas with embedded CTL^* state-formulas. Ignoring the state formulas for a moment, we can translate LTL-formulas into suitable Büchi automata on ω-words which accept a path if and only if it satisfies this formula. These automata can then be translated into a μ-calculus formula that basically simulates the automaton along *some* path. The embedded state formulas can be handled by a decomposition method as usual in CTL^*.

Something similar can be done in the hybrid case. Inspecting the grammar of $\mathrm{HCTL}^*_{\mathsf{ss}}$ again, we see that path formulas are basically also only simple LTL formulas with embedded state formulas. In particular path formulas are evaluated with respect to a fixed variable interpretation. The only difference is that these path formulas are not simply over the atomic propositions as vocabulary but they also encompass some variables that can be used. For example the path formula $\mathsf{F}x$ in $\downarrow x.\mathsf{EF}x$ features the variable x and thus a Büchi automaton that checks that somewhere along the path x holds, also needs to take care of these variables.

The idea now is the same as in the non-hybrid case: for path formulas we will construct a Büchi-automaton over an extended vocabulary that treats variables simply like atomic

propositions. This automaton, of-course, accepts more paths than in hybrid-logic intended. For example the restriction that a variable only holds at exactly one state will not be checked by this automaton.

However, since these automata are only an intermediate state in the translation this will not be a problem because we will then translate these automata into suitable H_μ formulas which will – by the semantics of H_μ– check that variables only occur at a single state.

To get started we need some definitions and notation:

▶ **Definition 9.** A path formula ψ is called *pure* if there are no occurrences of path quantifiers E, A or hybrid operators $\downarrow, @$ in ψ.

▶ **Definition 10.** Let $K = \langle S, \to, L\rangle$ be a Kripke structure over *Prop* and $\sigma : \mathit{Var} \to S$ a variable assignment. We define $K_\sigma = \langle S, \to, L'\rangle$ with $L'(s) := L(s) \cup \{x \in \mathit{Var} \mid \sigma(x) = s\}$.

Thus, K_σ extends K with new propositions for each variable. There is a 1-1 connection between paths from K and paths from K_σ. Thus, let $\mathsf{pr} : \mathsf{Paths}(K) \to \mathsf{Paths}(K_\sigma)$ be the unique function that maps a path from K to its copy in K_σ. Thus, for every path $\pi \in (S \times 2^{\mathit{Prop}})^\omega$ we get $\mathsf{pr}(\pi) \in (S \times 2^{\mathit{Prop} \cup \mathit{Var}})^\omega$.

▶ **Definition 11.** Let $K = \langle S, \to, L\rangle$ be a Kripke structure and $\sigma : \mathit{Var} \to S$ a variable assignment. A path $\pi \in (S \times 2^{\mathit{Prop} \cup \mathit{Var}})^\omega$ is called *consistent with* σ if for all positions $\pi^i = (s, M)$ on π it holds that $s = \sigma(x)$ if and only if $x \in M$ for all $x \in \mathit{Var}$.

The following lemma about consistent paths is easy to see:

▶ **Lemma 12.** *Let $K = \langle S, \to, L\rangle$ be a Kripke structure over Prop, $\sigma : \mathit{Var} \to S$ a variable assignment and π be a path in K. Then $\mathsf{pr}(\pi)$ is consistent with σ.*

The next lemma yields a connection between $\mathrm{HCTL}^*_{\mathsf{ss}}$ path formulas and LTL formulas. We use LTL as an index for the satisfaction relation to indicate that the (pure) path formula is interpreted as an LTL formula, i.e. $\pi \models_{\mathrm{LTL}} \psi$ means that the path π satisfies ψ interpreted as an LTL formula.

▶ **Lemma 13.** *Let $K = \langle S, \to, L\rangle$ be a Kripke structure, π a path in K and $\sigma : \mathit{Var} \to S$ a variable assignment. For every pure $\mathrm{HCTL}^*_{\mathsf{ss}}$ path formula ψ over atomic propositions Prop and variables $\{x_1, \ldots, x_k\}$ it holds that $K, \pi, \sigma \models \psi$ if and only if $K_\sigma, \mathsf{pr}(\pi) \models_{\mathrm{LTL}} \psi$.*

Proof. We will prove this by induction on ψ. Suppose first that $\psi = \varphi$ for some state-formula φ and suppose $K, \pi, \sigma \models \psi$. Since ψ is pure, there are no path quantifiers E, A, $\downarrow x.\varphi'$ nor $@_x \varphi'$ in ψ. Thus, φ is a boolean combination of propositions and variables and because $\mathsf{pr}(\pi)$ is consistent with σ and all states in K_σ agree with their respective states in K on atomic propositions we also get that $K_\sigma, \mathsf{pr}(\pi) \models_{\mathrm{LTL}} \psi$.

So, suppose $\psi = \mathsf{X}\psi'$. Then $K, \pi, 1, \sigma \models \psi$. With the induction hypothesis we get that $\mathsf{pr}(\pi), 1 \models_{\mathrm{LTL}} \psi'$ and also that $\mathsf{pr}(\pi) \models_{\mathrm{LTL}} \psi$.

For the last case, suppose that $\psi = \psi_1 \mathsf{U} \psi_2$ and $K, \pi, \sigma \models \psi$. Then there is some j such that $K, \pi, j, \sigma \models \psi_2$ and for all $i \leq j$ it holds that $K, \pi, i, \sigma \models \psi_1$. By the induction hypothesis we get that $\mathsf{pr}(\pi), j \models_{\mathrm{LTL}} \psi_2$ and $\mathsf{pr}(\pi), i \models_{\mathrm{LTL}} \psi_1$ for all $i \leq j$. Thus, we also get $\mathsf{pr}(\pi) \models_{\mathrm{LTL}} \psi$. This finishes the proof. ◀

The following theorems extend well-known results for LTL, Büchi automata and the μ-calculus to deal with the hybrid world.

▶ **Theorem 14.** *For each pure* $\mathrm{HCTL}^*_{\mathsf{ss}}$ *path formula* ψ *of size* n *over atomic propositions Prop and variables* $\{x_1, \ldots, x_k\}$ *there is a Büchi automaton* $\mathcal{A}_\psi$ *of size* $\mathcal{O}(n \cdot 2^n)$ *such that*
1. *a path* $\pi \in (S \times 2^{Prop \cup Var})^\omega$ *is accepted by* $\mathcal{A}_\psi$ *if and only if* $\pi \models_{\mathrm{LTL}} \psi$ *and*
2. *for all paths* $\pi' \in (S \times 2^{Prop})^\omega$, $\mathsf{pr}(\pi')$ *is accepted by* $\mathcal{A}_\psi$, *if and only if* $\pi', \sigma \models \psi$.

Proof. To construct $\mathcal{A}_\psi$ we first observe that ψ is a pure LTL formula with possibly added tests for variables. We treat variable test for the moment like usual atomic propositions. Furthermore, we know that for each LTL formula ψ there is a Büchi automaton $\mathcal{A}_\psi$ that accepts a path π iff this path satisfies ψ [16]. This immediately yields the first part of the theorem as well as the size estimation on the automaton. The second part follows from the first one in combination with Lemma 13. ◀

Observe that the constructed Büchi automaton only checks the sequence of propositions and thus, $\mathcal{A}_\psi$ accepts more paths than intended, for example paths in which the "proposition" x can occur on more than one state and thus cannot truly be an encoding of a hybrid variable.

To utilise these Büchi automata in the hybrid μ-calculus we also need to bridge the gap between L_μ and H_μ in some sense. Similar to the LTL case we write $K, s \models_{\mathrm{L}_\mu} \varphi$ to indicate that φ is interpreted as a purely modal μ-calculus formula.

▶ **Lemma 15.** *For each* H_μ*-formula* φ *without any occurrence of* $\downarrow x.\psi$ *or* $@_x\,\psi$ *in it, it holds that* $K, s, \sigma \models \varphi$ *if and only if* $K_\sigma, s \models_{L_\mu} \varphi$.

Proof sketch. To prove this by induction on φ we need to strengthen the hypothesis in order to deal with free second-order variables. Let $\varphi(X_1, \ldots, X_m)$ be a formula with free second-order variables $X_1, \ldots, X_m$ and $\rho : \{X_1, \ldots, X_m\} \to 2^{S \times (Var \to S)}$ be an interpretation for them. We define $\rho' : \{X_1, \ldots, X_m\} \to 2^S$ to be $\rho'(X) := \{s \mid (s, \sigma) \in \rho(x)\}$. We now show by induction on φ that $K, s, \sigma, \rho \models \varphi(X_1, \ldots, X_m)$ if and only if $K_\sigma, s, \rho' \models_{\mathrm{L}_\mu} \varphi(X_1, \ldots, X_m)$.

Suppose $\varphi = p$. Then the statement holds because K and K_σ agree everywhere on atomic propositions. The case $\varphi = x$ is by construction of K_σ because x as an atomic proposition in K_σ holds exactly at $\sigma(x)$. The case $\varphi = X$ follows by the construction of ρ'. The boolean cases as well as box and diamond operators follow by simple semantic arguments.

For the last case, suppose that $\varphi = \mu X.\psi(X)$. To show this case we can use the characterisation of a least fixpoint as the union of its approximations. We can then show that the statement holds for each approximation by a separate induction (and thus for the union of all of them). ◀

▶ **Theorem 16.** *For each Büchi automaton* $\mathcal{A}$ *over* $Prop \cup \{x_1, \ldots, x_k\}$ *of size* m *there is an* H_μ *formula* $\varphi_\mathcal{A}$ *of size at most* $O(m \cdot 2^m)$ *such that* $K, s, \sigma \models \varphi_\mathcal{A}$ *if and only if there is a path* π *in* K *starting at* s *such that* $\mathcal{A}$ *accepts* $\mathsf{pr}(\pi)$.

Proof. It is known that for each Büchi automaton $\mathcal{A}$ there is an L_μ formula $\varphi_\mathcal{A}$ such that $K_\sigma, s \models_{\mathrm{L}_\mu} \varphi_\mathcal{A}$ if and only if there exists a path π' starting at s such that $\mathcal{A}$ accepts π', c.f. [4, Chp. 10]. Since π' is a path in K_σ, there is a path π in K such that $\mathsf{pr}(\pi) = \pi'$. By Lemma 15 and the fact that $\varphi_\mathcal{A}$ does not have any occurrences of $\downarrow x$ or $@_x$ (in fact, $\varphi_\mathcal{A}$ is a pure modal μ-calculus formula extended by variable tests) we get that $K_\sigma, s \models_{\mathrm{L}_\mu} \varphi_\mathcal{A}$ if and only if $K, s, \sigma \models \varphi_\mathcal{A}$. ◀

We will use these theorems to show the following:

▶ **Theorem 17.** *For each formula* $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$ *there is a formula* $\varphi' \in H_\mu$ *such that* $K, s, \sigma \models \varphi$ *if and only if* $K, s, \sigma \models \varphi'$ *for all Kripke structures* K, *states* s *in* K *and variable assignments* σ.

Proof. We will first give a translation for $\text{HCTL}^*_{\text{ss}}$ formulas and then argue correctness of the translation. The cases up to path formulas are straightforward:

$$\begin{aligned} \tau(p) &:= p & \tau(\varphi \vee \chi) &:= \tau(\varphi) \vee \tau(\chi) \\ \tau(x) &:= x & \tau(\downarrow x.\varphi) &:= \downarrow x.\tau(\varphi) \\ \tau(\neg\varphi) &:= \neg\tau(\varphi) & \tau(@_x\,\varphi) &:= @_x\,\tau(\varphi) \end{aligned}$$

For the case of $\varphi = \mathsf{E}\psi$, let $\{\varphi_1, \ldots, \varphi_m\}$ be the maximal state-subformulas in ψ, i.e. subformulas that start with $\mathsf{E}, \mathsf{A}, \downarrow x., @_x$. We first replace those by fresh atomic propositions p_{φ_i}. The resulting formula is a pure $\text{HCTL}^*_{\text{ss}}$ path formula over the propositions $\mathit{Prop} \cup \{p_{\varphi_1}, \ldots, p_{\varphi_m}\}$ and variables $\{x_1, \ldots, x_k\}$. According to Theorem 14 we construct a Büchi-automaton $\mathcal{A}_\psi$. Furthermore, according to Theorem 16 there is a H_μ formula $\varphi_{\mathcal{A}_\psi}$ that simulates this Büchi-automaton and thus the formula ψ.

Finally, we translate the remaining maximal state subformulas $\{\varphi_1, \ldots, \varphi_m\}$ recursively. Let $\tau(\varphi_1), \ldots, \tau(\varphi_m)$ be their respective translations. We obtain the final translated formula by replacing the atomic propositions p_{φ_i} in $\varphi_{\mathcal{A}_\psi}$ by their respective translations. Thus:

$$\tau(\mathsf{E}\psi) := \varphi_{\mathcal{A}_\psi}\left[\tau(\varphi_1)/p_{\varphi_1}, \ldots, \tau(\varphi_m)/p_{\varphi_m}\right].$$

It remains to be shown that this translation is correct. For this, let $K = \langle S, \to, L\rangle$ be a Kripke structure, $s \in S$ and $\sigma : \mathit{Var} \to S$ a variable assignment. We prove that $K, s, \sigma \models \varphi$ if and only if $K, s, \sigma \models \tau(\varphi)$ by induction on φ.

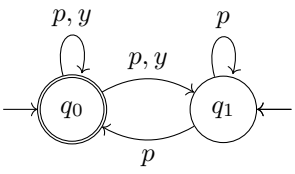
The only interesting case is $\varphi = \mathsf{E}\psi$. Suppose first, that ψ is pure and $K, s, \sigma \models \mathsf{E}\psi$. Then there is a path on K starting at s such that $K, \pi, \sigma \models \psi$. By the second part of Theorem 14 we get that $K, \pi, \sigma \models \psi$ if and only if $\mathsf{pr}(\pi)$ is accepted by $\mathcal{A}_\psi$ and by Theorem 16 we then get that $\mathsf{pr}(\pi)$ is accepted by $\mathcal{A}_\psi$ if and only if $K, s, \sigma \models \varphi_{\mathcal{A}_\psi}$.

For the case that ψ is not pure, we additionally need the fact that $K, s, \sigma \models \varphi\,[\chi/p] \Leftrightarrow K', s, \sigma \models \varphi$ where K' extends K with an atomic proposition p such that $p \in L(s) \Leftrightarrow K, s, \sigma \models \chi$. This can be shown by a straightforward induction on φ, both for $\text{HCTL}^*_{\text{ss}}$ and H_μ. ◀

To illustrate the translation we will give a short example:

▶ **Example 18.** Consider the formula $\chi := \downarrow y.\mathsf{EG}\,(\mathsf{F}y \wedge \downarrow x.\mathsf{EXF}x)$. The formula states that there is a path whose starting point is seen infinitely often along the path and at every point of the path there is another path that loops back to the current point.

We first begin by extracting the maximal state-subformula $\downarrow x.\mathsf{EXF}x$ leaving us with the formula $\downarrow y.\mathsf{EG}\,(\mathsf{F}y \wedge p)$. We first construct a Büchi automaton $\mathcal{A}$ for the LTL-formula $\mathsf{G}\,(\mathsf{F}y \wedge p)$ (ignoring that y is a variable test):



This Büchi-automaton can be translated into the following μ-calculus formula which is satisfied by a state s iff there is a path emerging from s that is accepted by $\mathcal{A}$.

$$\begin{aligned} \varphi := &[\nu Y.(p \wedge y \wedge \Diamond Y) \vee (p \wedge y \wedge \Diamond \mu Z.(p \wedge \Diamond Z) \vee (p \wedge \Diamond Y))] \vee \\ &\mu Z.(p \wedge \Diamond Z) \vee (p \wedge \Diamond \nu Y.(p \wedge y \wedge \Diamond Y) \vee (p \wedge y \wedge \Diamond \mu Z.(p \wedge \Diamond Z) \vee (p \wedge \Diamond Y)) \end{aligned}$$

Here the first line describes an accepting run starting at q_0 and the second line a run starting at q_1. The automaton accepts a path π if p holds everywhere and y is seen infinitely often along π. For the sake of presentation we will use that

$$\varphi \equiv \nu Y.\mu Z.p \wedge ((y \wedge \Diamond Y) \vee \Diamond Z)$$

and continue to use this shorter formula. Doing the same recursively we get that $\tau(\mathsf{EXF}x) = \Diamond\mu X.x \vee \Diamond X$. Putting them together we get $\tau(\chi) := \downarrow y.\nu Y.\mu Z.(\downarrow x.\Diamond\mu X.x \vee \Diamond X) \wedge ((y \wedge \Diamond Y) \vee \Diamond Z)$.

Knowing that $\mathrm{HCTL}^*_{\mathsf{ss}}$ is a fragment of H_μ also helps us to understand the connection between $\mathrm{HCTL}^*_{\mathsf{ss}}$ and $\mathrm{HCTL}^*_{\mathsf{ps}}$ given that the latter cannot be embedded into H_μ [13].

▶ **Corollary 19.** $\mathrm{HCTL}^*_{\mathsf{ps}}$ *is strictly more expressive than* $\mathrm{HCTL}^*_{\mathsf{ss}}$.

In particular, no $\mathrm{HCTL}^*_{\mathsf{ss}}$ formula is equivalent to the $\mathrm{HCTL}^*_{\mathsf{ps}}$ formula $\mathsf{EG}{\downarrow}x.\mathsf{XG}\neg x$.

Let $\mathrm{H}^k\mathrm{CTL}^*_{\mathsf{ss}}$ and H^k_μ be the fragments of $\mathrm{HCTL}^*_{\mathsf{ss}}$ and H_μ that use at most k first-order variables. We know from [13] that H^k_μ is hybrid k-bisimulation-invariant. Thus, since the translation from $\mathrm{HCTL}^*_{\mathsf{ss}}$ to H_μ basically leaves variables untouched we also get:

▶ **Corollary 20.** $\mathrm{H}^k\mathrm{CTL}^*_{\mathsf{ss}}$ *is hybrid k-bisimulation-invariant.*

5.3 $\mathrm{HCTL}^*_{\mathsf{ps}}/\mathrm{HCTL}^*_{\mathsf{pp}}$ and H_μ are incomparable

Finally, we are also interested in the connection between $\mathrm{HCTL}^*_{\mathsf{ps}}$ resp. $\mathrm{HCTL}^*_{\mathsf{pp}}$ and H_μ. We already know that one cannot be embedded into the other; in the remainder of this section we will show that this is also true of the other way: there are formulas in H_μ which cannot be expressed in $\mathrm{HCTL}^*_{\mathsf{pp}}$ and, hence, not on $\mathrm{HCTL}^*_{\mathsf{ps}}$ either. Hence, we will show that $\mathrm{HCTL}^*_{\mathsf{pp}}$ (resp. $\mathrm{HCTL}^*_{\mathsf{ps}}$) and H_μ are incomparable.

▶ **Theorem 21.** *Let K be a linear structure and $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$. Let φ' be the formula that is constructed by simply removing all path quantifiers in φ. Then $K, s, \sigma \models \varphi$ if and only if $K, s, \sigma \models \mathsf{E}\varphi'$.*

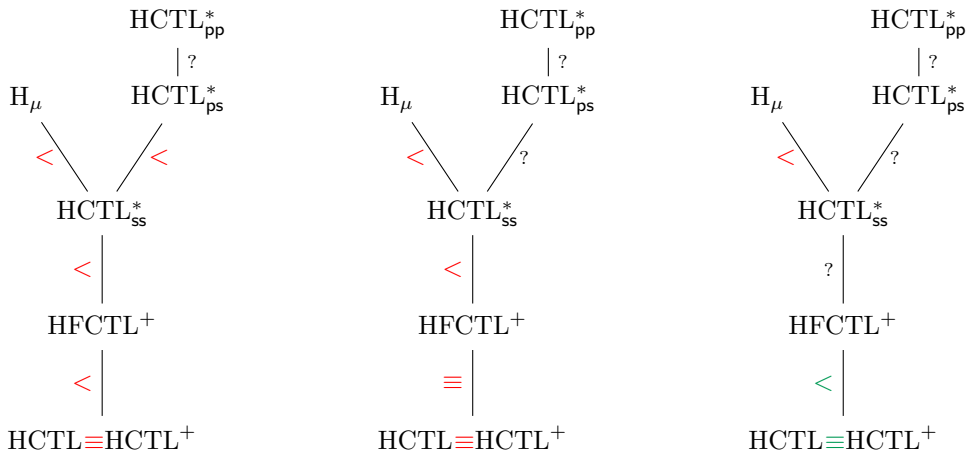
Proof. To prove this, we see that on linear structures there is no difference between E and A path quantifiers since from every point in the structure there is exactly one path. Consequently we can just drop the path quantifiers altogether. ◀

Thus, Theorem 21 essentially states that on linear structures $\mathrm{HCTL}^*_{\mathsf{pp}}$ (and $\mathrm{HCTL}^*_{\mathsf{ps}}$) is as expressive as hybrid LTL.

▶ **Theorem 22.** *There is no* $\mathrm{HCTL}^*_{\mathsf{pp}}$ *formula that can express the H_μ property $\mu X.\, p \vee \Diamond\Diamond X$.*

Proof. Suppose for the sake of contradiction that such a formula φ exists. Then by Theorem 21 we get that there is a hybrid LTL formula that characterises reachability in an even number of steps on word structures. However, hybrid LTL can be translated into first-order logic on word structures which, in turn, cannot express this property, c.f. [5]. Thus, such a formula cannot exist. ◀

▶ **Corollary 23.** $\mathrm{HCTL}^*_{\mathsf{ps}}/\mathrm{HCTL}^*_{\mathsf{pp}}$ *and H_μ are incomparable.*



(a) general structures. **(b)** finite structures. **(c)** tree structures.

Figure 2 The branching-time hierarchy on different classes of Kripke structures. Results colored in red are newly obtained in this paper while results colored in green have been previously shown. Still open questions are marked with a "?".

6 Conclusion & Further Work

To conclude, we have studied the expressive power of hybrid branching-time logics. The results are summarised in Figure 2.

For general structures the results, depicted in Figure 2a, are similar to their non-hybrid counterparts – at least for the part below $\mathrm{HCTL}^*_{\mathsf{ss}}$. We have proven that HCTL^+ and HCTL have the same expressive power. HFCTL^+ then can express more properties than HCTL^+ and $\mathrm{HCTL}^*_{\mathsf{ss}}$ can express even more. However, above $\mathrm{HCTL}^*_{\mathsf{ss}}$ the picture is a bit different: only $\mathrm{HCTL}^*_{\mathsf{ss}}$ can be translated into H_μ. Allowing dynamic naming of states as part of a path formula in CTL^* then drastically increases the expressive power in a way that is not even captured by H_μ. We do not yet know if allowing jumps as part of path formulas increases the expressive power of the resulting logic even further.

It is also interesting to see that these expressiveness results change depending on the classes of structures in reference. For general Kripke structures we have obtained an (almost) complete picture. However, if we restrict the attention to finite structures or tree structures we only have an incomplete and possibly quite different picture. For example, on finite structures we have shown that HFCTL^+ is equi-expressive to HCTL which helped us prove the expressiveness gap between HFCTL^+ and $\mathrm{HCTL}^*_{\mathsf{ss}}$. However, it is still unclear if this expressiveness gap also holds for trees. We have also summarised what we know about finite structures and trees in Figures 2b and 2c.

Future work will focus on studying the expressiveness over interesting classes of structures such as finite structures or trees and will aim to close the gaps in their respective hierarchies. We will also continue to study the expressiveness games defined in Section 3. Here, we have only proven one direction: if Duplicator wins, then there is no formula that can distinguish the structures in question. A proof for the reverse direction would in itself strengthen this framework but has turned out to be rather challenging.

References

1 C. Areces, P. Blackburn, and M. Marx. The computational complexity of hybrid temporal logics. *Logic J. of the IGPL*, 8(5):653–679, 2000.

2 C. Areces and B. ten Cate. Hybrid logics. In P. Blackburn, F. Wolter, and J. van Benthem, editors, *Handbook of Modal Logics*, pages 821–868. Elsevier, 2006.

3 P. Blackburn. Representation, reasoning, and relational structures: A hybrid logic manifesto. *Logic J. of the IGPL*, 8(3):339–365, 2000.

4 S. Demri, V. Goranko, and M. Lange. *Temporal Logics in Computer Science*, volume I – Finite State Systems of *Cambridge Tracts in Theor. Comp. Sc.* Cambridge Univ. Press, 2016.

5 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Math. Logic. Springer, Berlin, 1995.

6 A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961.

7 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.

8 E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. of Comp. and Sys. Sc.*, 30:1–24, 1985.

9 E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *J. of the ACM*, 33(1):151–178, 1986.

10 E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *SIAM J. on Computing*, 29(1):132–158, 2000.

11 A. Kara, V. Weber, M. Lange, and T. Schwentick. On the hybrid extension of CTL and CTL$^+$. In *Proc. 34th Int. Symp. on Mathematical Foundations of Computer Science, MFCS'09*, volume 5734 of *LNCS*, pages 427–438. Springer, 2009.

12 D. Kernberger and M. Lange. Model checking for the full hybrid computation tree logic. In *Proc. 23rd Int. Symp. on Temporal Representation and Reasoning, TIME'16*, pages 31–40. IEEE Computer Society, 2016.

13 D. Kernberger and M. Lange. The fully hybrid mu-calculus. In *Proc. 24th Int. Symp. on Temporal Representation and Reasoning, TIME'17*, volume 90 of *LIPIcs*, pages 17:1–17:16. Dagstuhl-Leibniz-Zentrum, 2017. URL: `http://www.dagstuhl.de/dagpub/978-3-95977-052-1`.

14 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.

15 U. Sattler and M. Y. Vardi. The hybrid μ-calculus. In *Proc. 1st Int. Joint Conf. on Automated Reasoning, IJCAR'01*, volume 2083 of *LNCS*, pages 76–91. Springer, 2001.

16 M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proc. 1st Symp. on Logic in Computer Science, LICS'86*, pages 332–344. IEEE, Washington, DC, 1986.

A Appendix

We need a technical definition for the proof of Theorem 2.

▶ **Definition 24.** For any $\varphi \in \text{HCTL}$ we define the *temporal nesting depth* of φ ($\mathsf{nd}(\varphi)$) as follows:

$$\begin{aligned} \mathsf{nd}(p) &:= 0, & \mathsf{nd}(x) &:= 0, \\ \mathsf{nd}(\neg\varphi) &:= \mathsf{nd}(\varphi), & \mathsf{nd}(\varphi_1 \vee \varphi_2) &:= \max\{\mathsf{nd}(\varphi_1), \mathsf{nd}(\varphi_2)\}, \\ \mathsf{nd}(\downarrow x.\varphi) &:= \mathsf{nd}(\varphi), & \mathsf{nd}(@_x\,\varphi) &:= \mathsf{nd}(\varphi), \\ \mathsf{nd}(\mathsf{EX}\varphi) &:= \mathsf{nd}(\varphi) + 1, & \mathsf{nd}(\mathsf{E}\varphi_1\mathsf{U}\varphi_2) &:= \max\{\mathsf{nd}(\varphi_1), \mathsf{nd}(\varphi_2)\} + 1. \end{aligned}$$

▶ **Theorem 2** (restated). *If Duplicator wins $\mathcal{G}^m_{\text{HCTL}}(K_0, s_0, K_1, t_0)$ then it holds for all formulas $\varphi \in \text{HCTL}$ with $\mathsf{nd}(\varphi) \leq m$ that $K_0, s_0 \models \varphi$ if and only if $K_1, t_0 \models \varphi$.*

Proof. We prove an extended statement for formulas with unbounded occurrences of variables. For this, let $\mathcal{G}^m_{\text{HCTL}}(K_0, s_0, \ldots, s_{n-1}, K_1, t_0, \ldots, t_{n-1})$ be the game with n pebbles already placed. We will show the following statement:

We show that if Duplicator wins $\mathcal{G}^m_{\text{HCTL}}(K_0, s_0, \ldots, s_{n-1}, K_1, t_0, \ldots, t_{n-1})$ then it holds for all formulas $\varphi \in \text{HCTL}$ with $\mathsf{nd}(\varphi) \leq m$ and all variable assignments $\sigma : \mathit{Var} \to S_0$, $\sigma' : \mathit{Var} \to S_1$ such that $\sigma(x) = s_i$ if and only if $\sigma'(x) = t_i$ that $K_0, s_i, \sigma \models \varphi$ if and only if $K_1, t_i, \sigma' \models \varphi$.

We show this by an induction on the number of rounds m and only show the cases where Duplicator moves on K_0. The other cases are completely analoguous.

Let $m = 0$ and suppose that Duplicator wins $\mathcal{G}^m_{\text{HCTL}}(K_0, s_0, \ldots, s_{n-1}, K_1, t_0, \ldots, t_{n-1})$. Then it holds for all atomic formulas $\varphi = p$ or $\varphi = x_j$ for some $p \in \mathit{Prop}$ and $0 \leq j \leq n-1$ that $K_0, s_i, \sigma \models \varphi$ if and only if $K_1, t_i, \sigma' \models \varphi$. This follows directly from the winning conditions of Duplicator. The cases for boolean connectives and hybrid operators can be shown by a straightforward induction.

So, let $m \geq 1$ and suppose that the statement already holds for $m-1$. Suppose again, that Duplicator wins $\mathcal{G}^m_{\text{HCTL}}(K_0, s_0, \ldots, s_{n-1}, K_1, t_0, \ldots, t_{n-1})$. We show that $K_0, s_i, \sigma \models \varphi$ if and only if $K_1, t_i, \sigma' \models \varphi$ with $\varphi, \sigma, \sigma', i$ as above by an induction over the structure of φ. Atomic formulas and boolean connectives can be shown in the same way as for $m = 0$.

- $\varphi = \mathsf{EX}\psi$. Suppose $K_0, s_i, \sigma \models \varphi$, then there is some successor s'_i of s_i such that $K_0, s'_i, \sigma \models \psi$. Suppose Spoiler chooses s'_i and moves there. Since Duplicator wins, she can choose a successor t'_i of t_i such that she wins $\mathcal{G}^{m-1}_{\text{HCTL}}(K_0, s_0, \ldots, s_{n-1}, s'_i, K_1, t_0, \ldots, t_{n-1}, t'_i)$. Since $\mathsf{nd}(\psi) < m$ we can use that the statement already holds for $m-1$ and get that $K_1, t'_i, \sigma' \models \psi$. This means also that $K_1, t_i, \sigma' \models \varphi$.
- $\varphi = \mathsf{E}\psi_1\mathsf{U}\psi_2$. Suppose $K_0, s_i, \sigma \models \varphi$. Then there is a path π starting at s_i and some l such that $K_0, \pi^l, \sigma_0 \models \psi_2$ and for all $j < l$, $K_0, \pi^j, \sigma \models \psi_1$. Suppose Spoiler chooses to play π on K_0 with l. Since Duplicator wins the game, she can answer with some path τ on K_1 starting at t_i and some l'. Spoiler now has two possibilities:
  First, he can choose to go to π^l. Duplicator then has to play $\tau^{l'}$ and since she plays a winning strategy wins the game $\mathcal{G}^{m-1}_{\text{HCTL}}(K_0, s_0, \ldots, s_{n-1}, \pi^l, K_1, t_0, \ldots, t_{n-1}, \tau^{l'})$. Since $K_0, \pi^l, \sigma_0 \models \psi_2$ and $\mathsf{nd}(\psi_2) < m$ we can deduce with the hypothesis for $m-1$, that $K_1, \tau^{l'}, \sigma' \models \psi_2$.
  Secondly, Spoiler can choose any $k' < l'$ and move to $\tau^{k'}$. Since Duplicator is winning, she can answer with some $k < l$, move to π^k and win from there. We know that $K_0, \pi^k, \sigma \models \psi_1$ and $\mathsf{nd}(\psi_1) < m$ so by the induction hypothesis we also get that $K_1, \tau^{k'}, \sigma' \models \psi_1$. This holds for any $k' < l'$ since it was Spoiler's choice.

Thus, we have that there is some l' on τ such that $K_1, \tau^{l'}, \sigma' \models \psi_2$ and for all $k' < l'$ it holds that $K_1, \tau^{k'}, \sigma' \models \psi_1$. Thus, since τ starts at t_i we get that $K_1, t_i, \sigma' \models \varphi$.

- $\varphi = \downarrow x.\psi$ for some $x \notin \{x_0, \dots, x_{n-1}\}$ (otherwise $x_0, \dots, x_{n-1}$ would not be free). Suppose $K_0, s_i, \sigma \models \varphi$. Then $K_0, s_i, \sigma[x \mapsto s_i] \models \psi$. Since ψ is smaller then φ and $\sigma[x \mapsto s_i]$, resp. $\sigma'[x \mapsto t_i]$ also satisfy the assumption for variable interpretations we can use the induction hypothesis for ψ and get that $K_1, t_i, \sigma'[x \mapsto t_i] \models \psi$ and thus also $K_1, t_i, \sigma' \models \varphi$.
- $\varphi = @_{x_j}\psi$ for some $0 \leq j \leq n-1$. Suppose $K_0, s_i, \sigma \models \varphi$. Then with the definition of σ we get that $K_0, s_j, \sigma \models \psi$. With the induction hypothesis for ψ we get that $K_0, t_j, \sigma' \models \psi$ and thus also $K_0, s_i, \sigma \models \varphi$.

This finishes the proof. ◀

▶ **Theorem 25.** *For each* HCTL$^+$ *formula* φ *there is an* HCTL *formula* φ' *such that for all Kripke structures* $K = \langle S, \to, L\rangle$*, states* $s \in S$ *and variable assignments* $\sigma : Var \to S$ *it holds that* $K, s, \sigma \models \varphi$ *if and only if* $K, s, \sigma \models \varphi'$*.*

Proof. We describe how to transform an HCTL$^+$ formula into an HCTL formula. This can be done by rewriting each path formula into an equivalent HCTL formula.

Using the equivalences $\mathsf{A}\psi \equiv \neg\mathsf{E}\neg\psi$, $\neg\mathsf{X}\psi \equiv \mathsf{X}\neg\psi$, $\neg(\psi_1\mathsf{U}\psi_2) \equiv \mathsf{G}\neg\psi_2 \vee (\neg\psi_2\mathsf{U}(\neg\psi_1 \wedge \neg\psi_2))$, $\mathsf{E}\psi_1 \vee \psi_2 \equiv \mathsf{E}\psi_1 \vee \mathsf{E}\psi_2$ we can push negations inward and assume conjunctions as the top-level operator in path formulas. We then use $\mathsf{X}\varphi_1 \wedge \mathsf{X}\varphi_2 \equiv \mathsf{X}\varphi_1 \wedge \varphi_2$ and $\mathsf{G}\varphi_1 \wedge \mathsf{G}\varphi_2 \equiv \mathsf{G}\varphi_1 \wedge \varphi_2$ to move Next- and Generally operators upwards and $\mathsf{E}(\varphi \wedge \psi) \equiv \varphi \wedge \mathsf{E}\psi$ for some state formula φ to remove state-formulas directly under a path quantifier. Thus, we can assume that each path formula has the form

$$\mathsf{E}(\mathsf{X}\Lambda_1 \wedge \bigwedge_{i \in I} \varphi_i \mathsf{U} \chi_i \wedge \mathsf{G}\Lambda_2)$$

with suitable state formulas $\Lambda_i, \varphi_i, \chi_i$.

We then obtain an HCTL formula by guessing the order in which the Until-formulas are satisfied along such a path. This is done by the following formula:

$$\Lambda_2 \wedge \bigvee_{J \subseteq I} \Big(\bigwedge_{j \notin J} \chi_j\Big) \wedge \Big(\bigwedge_{j \notin J} \varphi_j\Big) \wedge \mathsf{EX}\Big(\Lambda_1 \wedge \bigvee_{\pi \in \mathsf{Perm}(J)} \mathsf{E}\Big((\Lambda_2 \wedge \bigwedge_{j \in J} \varphi_{\pi(j)})\mathsf{U}\Big(\chi_{\pi(1)} \wedge \mathsf{E}\Big((\Lambda_2 \wedge \bigwedge_{j \in J, j \neq 1} \varphi_{\pi(j)})\mathsf{U}\Big(\chi_{\pi(2)} \wedge \vdots \mathsf{E}\big((\Lambda_2 \wedge \varphi_{\pi(|J|)})\mathsf{U}(\chi_{\pi(|J|)} \wedge \mathsf{EG}\Lambda_2)\big)\Big)\dots\Big)$$

where $\mathsf{Perm}(J)$ denotes the set of all permutations over J. ◀

A Temporal Logic for Modelling Activities of Daily Living

Malte S. Kließ
Interactive Intelligence, Delft University of Technology, Delft, The Netherlands
m.s.kliess@tudelft.nl
https://orcid.org/0000-0001-7219-309X

Catholijn M. Jonker
Interactive Intelligence, Delft University of Technology, Delft, The Netherlands and
Leiden Institute of Advanced Computer Science, Leiden University
C.M.Jonker@tudelft.nl and C.M.Jonker@liacs.leidenuniv.nl
https://orcid.org/0000-0003-4780-7461

M. Birna van Riemsdijk
Interactive Intelligence, Delft University of Technology, Delft, The Netherlands
M.B.vanRiemsdijk@tudelft.nl

Abstract

Behaviour support technology is aimed at assisting people in organizing their Activities of Daily Living (ADLs). Numerous frameworks have been developed for activity recognition and for generating specific types of support actions, such as reminders. The main goal of our research is to develop a *generic* formal framework for representing and reasoning about ADLs and their temporal relations. This framework should facilitate modelling and reasoning about 1) durative activities, 2) relations between higher-level activities and subactivities, 3) activity instances, and 4) activity duration. In this paper we present a temporal logic as an extension of the logic TPTL for specification of real-time systems. Our logic TPTL_{BIH} is defined over Behaviour Identification Hierarchies (BIHs) for representing ADL structure and typical activity duration. To model execution of ADLs, states of the temporal traces in TPTL_{BIH} comprise information about the start, stop and current execution of activities. We provide a number of constraints on these traces that we stipulate are desired for the accurate representation of ADL execution, and investigate corresponding validities in the logic. To evaluate the expressivity of the logic, we give a formal definition for the notion of Coherence for (complex) activities, by which we mean that an activity is done without interruption and in a timely fashion. We show that the definition is satisfiable in our framework. In this way the logic forms the basis for a generic monitoring and reasoning framework for ADLs.

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

Keywords and phrases Temporal Logic, Reasoning, Durative Activities

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.17

Funding This work is part of the research programme CoreSAEP, with project number 639.022.416, which is financed by the Netherlands Organisation for Scientific Research (NWO).

Acknowledgements We would like to thank the anonyomous referees for their insightful and constructive comments that helped improve the work presented in this paper.

1 Introduction

Behaviour support technology [15, 9, 12, 18] is aimed at helping people organize their Activities of Daily Living (ADLs) and change their habits. For example, smart home and wearable technology is being developed to recognize people's activities and determine whether it constitutes abnormal behavior [17, 11]. This technology is developed with the aim of allowing elderly and people with special needs to continue living independently by providing support in cases of deviations from expected behavior.

While numerous such frameworks have been developed, they typically focus on specific types of behavior – such as forgetting to perform certain actions, and corresponding support actions – such as smart reminders [9, 12, 18, 13]. Perhaps as a consequence, the specification of what is the desired user behaviour is typically left implicit in these frameworks, intertwined with activity recognition and supportive actions.

In our work we aim to develop a *generic* formal framework for representing and reasoning about ADLs, in which the representation of desired behavior is explicit and decoupled from recognition of actual user behavior and corresponding support actions. These expressions of desired behaviour can originate from users themselves or from others in their social context, such as caregivers. Our approach [8, 23, 16, 22] is inspired by research on normative multiagent systems [2] in which computational models are developed for explicitly representing and reasoning about norms that govern societies of artificial autonomous agents at run-time. An important advantage of our approach is that it can form the foundation for developing more *comprehensive* supportive technology that can support a wide variety of desired types of behavior. Moreover, such explicit representation of desired behavior allows *reasoning* about possible conflicting desires as well as underlying motivations and *values* (see also [16]). Finally, it allows development of *generic supportive strategies* based on the representations of desired behavior and values.

Considering the inherent temporal nature of ADLs, we take temporal logic as a basis for representing desired behavior [10]. Specifically, the framework should facilitate modelling and reasoning about 1) durative activities, 2) relations between higher-level activities and subactivities, 3) activity instances, and 4) normal activity duration.

The motivation for this is that we would not consider actions as being done instantaneosly; thus our logic should not be limited to *when* an activity is done, but also include *for how long* it is being done. A hierarchy of activities will allow for a more fine-grained support: it might be the case that while a user requests support for a particular, complex activity, support is only really needed for a particular part of the activity, e.g. a user requiring support for the activity of *cooking dinner* may only need support when preparing the ingredients, but not for the actual cooking activity. The logic should be able to handle instances of activities, i.e. it should be able to distinguish between the more abstract description given by a user or caregiver, and the activity instance actually carried out. This allows us to check for any deviation from the recorded description of an activity, whether there is a deviation in the manner in which the activity is carried out or whether it takes an unusually long or short time. There are several temporal logic frameworks that allow expressing durative properties, e.g. [14]. All of our requirements considered, we believe that choosing TPTL as the basis for our logic is the most suitable starting point. In particular, we need a way to encode the exact duration that an instance of an activity takes to be carried out, as well as at which time activities are started and completed, respectively. That is, we desire a logic which enables us to express various types of temporal goals. Reasoning about how to handle the interaction between these goals, handling conflicts etc. is not the focus of this paper. Concerning these questions see e.g. [20, 19].

Once we have an understanding of which notions of temporality we want and need to capture for this type of technology, we can analyze which of many existing frameworks for representing and reasoning about norms and time we can build on, e.g. [6, 5, 3, 4, 7, 22].

A key notion for supporting people in their daily lives is that of *normal behaviour* [17, 11]: a support system should be able to detect when a user's activities deviate from a norm, and render support to the user in such situations. However, this notion is sometimes considered in very strict bounds: changing the order of subactivities in a specific activity is already considered *abnormal*. While this certainly applies, e.g. when a user is going to bed before taking off their clothes, or starting to eat before taking certain medication, there are situations in which this ordering may be too strict. When preparing a meal, for instance, we may have a record of the usual order in which the ingredients are prepared. For example, the user usually cuts the tomatoes, then washes the salad leaves, and finally peels and cooks potatoes, deviations from this particular order would not necessarily be seen as abnormal. Especially in the cooking domain, where a lot of activities can be carried out by multiple people in parallel to prepare a meal, we would want to allow more flexibility. In order to achieve this, we define the notion of *Coherence*, by which we mean that an activity is carried out without interruption and in a timely fashion, but which does not account for the particular order in which subactivities are carried out. While this means that there are sometimes extra conditions needed to make sure that e.g., the potatoes are peeled before they are cooked, our notion allows this increased flexibility compared to the much stricter notions of normal / abnormal behaviour.

The paper is organized as follows: in section 2 we introduce the Behaviour Identification Hierarchy $\mathfrak{A}$, which will form the basis for our logic. We also introduce the example of making a pizza for dinner, which we will use throughout the paper as a motivational example as well as demonstrating how the features of our logic work. In section 3, the temporal logic $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ is defined and we discuss syntax and semantics as well as conditions on the semantic structures that correspond to properties we wish to hold in the logic. We will then introduce the notion of Coherence in section 4, and demonstrate that this is a notion that is satisfiable in our logic $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$.

2 A Basic Behaviour Identification Hierarchy

In this section we will introduce the Behaviour Identification Hierarchy (BIH) that forms the basis of our temporal logic. The BIH is intended to represent the user's recorded behaviour, desired or actual, that the agent will use for rendering support to the user. Before defining the BIH in detail, we will motivate our ideas by an example behaviour description. For this, we will follow the fictitious life of Pedro, who uses a supportive agent for assistance for certain daily tasks.

▶ **Example 1.** Pedro wants to have pizza for dinner and needs assistance for that. He describes his usual way of making a pizza as follows:

- He first prepares the dough and all ingredients he wants on the pizza. These include tomato sauce, cheese, and sometimes mushrooms and olives.
- For the preparation, he rolls out the dough, grates the cheese, washes and cuts the mushrooms, and drains and cuts the olives.
- He then spreads the tomato sauce on the rolled out dough, then decorates with mushrooms and olives, and finally puts the grated cheese on top.
- Finally, he puts the pizza in the oven for about 20 minutes.

Using this description, we obtain a basic abstraction structure for the activity of "*making pizza*":

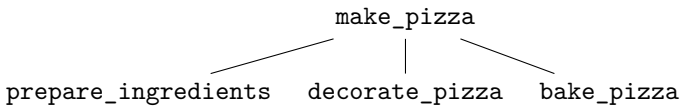


Figure 1

That is, we can refine the single activity of "*making pizza*" by separating it into three distinct subactivities, namely "*preparing ingredients*", "*decorating the pizza*" and "*baking the pizza*". Each of those can further be refined in turn (with the exception of *baking*, perhaps), with e.g. *washing* and *cutting mushrooms* part of *preparing ingredients*.

There is an implicit temporal structure given in the example: Pedro prepares the ingredients *before* he decorates the pizza with them, and he does so *before* baking the pizza. In the picture above one could read this as an ordering on the branches from left to right.

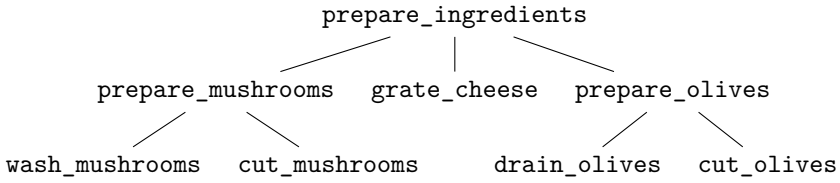


Figure 2

This implicit left-to-right temporal order does not always apply; in Figure 2 one could easily permute the ordering of the three children of `prepare_ingredients` on a temporal axis without changing the description of the activity: if all three child activities are done, the ingredients have been prepared. We even have some more flexibility: Pedro can do the activities described in the leaves of the tree in any order *provided* he washes the mushrooms before cutting them and drains the olives before cutting them. The ability to express temporal ordering will thus be a requirement for our logic. As we will opt for a logic based on TPTL, the *Until* operator will allow us to define the appropriate notion.

We will also need some attribute to record the length of activities; we already know that *baking* takes Pedro about 20 minutes, but it is unlikely that *washing and cutting mushrooms* is done instantaneously by Pedro. Even if Pedro does not care how long this is going to take him, an estimate should be given.

With this in mind we will define the Behaviour Identification Hierarchy as follows:

▶ **Definition 2** (Behaviour Identification Hierarchy)**.** An *activity name* is a string $\mathfrak{a}$. Let $\mathcal{A}$ be a (finite) collection of activity names and let T be a temporal domain. We define the Behaviour Identification Hierarchy $\mathfrak{A}$ as the quadruple $\langle \mathcal{A}, f_T, g_T, \mathit{passive} \rangle$, where $f_T, g_T : \mathcal{A} \to T$ are functions mapping each activity name to its (average) duration and (average) extra time, respectively. Let $\mathit{passive} : \mathcal{A} \to \{0, 1\}$ be an indicator function for passive activities, i.e. $\mathit{passive}(\mathfrak{a}) = 1$ if the activity $\mathfrak{a}$ is passive in the sense that it does not require active involvement by the user.[1]

[1] This can be a waiting "activity", e.g. baking a pizza does not require the user to attend to the oven during the whole time the activity takes place.

Furthermore we define a binary relation PartOf on $\mathcal{A}$, where $\mathfrak{a}$ PartOf $\mathfrak{b}$ iff activity $\mathfrak{a}$ is a part of activity $\mathfrak{b}$. PartOf is antisymmetric, irreflexive and such that for any activities $\mathfrak{a}$, $\mathfrak{b}$, $\mathfrak{c}$, if $\mathfrak{b}$ PartOf $\mathfrak{a}$ and $\mathfrak{b}$ PartOf $\mathfrak{c}$, then $\mathfrak{a} = \mathfrak{c}$.

The intuition behind this definition is as follows: The PartOf relation provides the tree structure: each activity has a unique parent activity in the tree, and cannot be a part of itself. The condition on unique parents has the effect that PartOf is not transitive. The main purpose is to ensure that we do not accidentally link two distinct subtrees:

Suppose we have an activity `prepare_mushrooms` in the tree as a sub-activity both for `make_pizza` and `make_pasta`. While the activity of preparing mushrooms would certainly look identical for an observer, we cannot use the same mushrooms for both pizza and pasta at the same time. Thus, any instance of preparing mushrooms will either be a sub-activity of an instance of making pizza, or a sub-activity of an instance of making pasta. We want the tree to reflect this.

Taking the transitive closure of PartOf will then enable us to state that `cut_mushrooms` as a sub-activity of `prepare_mushrooms` is also a sub-activity of `make_pizza` or `make_pasta`, respectively.

The functions f_T and g_T define the duration an activity usually takes the user. Here, f_T provides the average time, while g_T describes the time of a "grace period" a monitoring system should take into account before noticing that an activity takes too long. The reason for implementing this with a function, rather than a constant parameter, is that a user may have different "grace periods" for activities: for instance, mushroom cutting might be an easy activity that the user can do without hesitation, but switching the oven on and setting it to the correct temperature might be quite difficult for the user.

The definition above provides a *basic* BIH in the sense that we are not (yet) encoding any explicit temporal ordering. The links given by PartOf also do not distinguish between *optional* or *mandatory* parts. Furthermore, while the examples of such a BIH are all represented in the form of a tree structure, we did not in fact require a BIH to be a tree: the actions with the highest abstraction level, i.e. those that are not a part of another action, are not required to have a common root. Thus, a BIH can be seen as a forest of trees, each of which has as root a unique activity on this highest abstraction level.

3 A Temporal Logic

We will define a temporal logic for our notion of Behaviour Identification Hierarchy. Within this logic we can then express norms and norm compliance for the behaviours given in the BIH ([22]).

▶ **Definition 3.** We define the Temporal Logic $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ derived from TPTL (see e.g. [1]) as follows: Let $\mathfrak{A} = \langle \mathcal{A}, f_T, g_T, \mathit{passive} \rangle$ be a BIH. Let the temporal domain for $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ be $T = \mathbb{N}$.

- $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ has two sorts of variables: *temporal* variables, denoted with lower-case x, y, z, x_i, and *action* variables, denoted with upper-case X, Y, Z, X_i.
  Let V_T denote the set of temporal variables and V_A denote the set of action variables.
- For each $\mathfrak{a} \in \mathcal{A}$ there is a family of action constants $\gamma_1^{\mathfrak{a}}, \gamma_2^{\mathfrak{a}}, \gamma_3^{\mathfrak{a}}, \ldots$ in $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$. There is a 1-1 function $\mathcal{F}$, mapping each such family $\{\gamma_n^{\mathfrak{a}} \mid n \in \mathbb{N}\}$ of constant symbols to its corresponding action $\mathfrak{a} \in \mathcal{A}$. $\mathcal{F}$ induces a partial order $\text{PartOf}_{\mathcal{F}}$ on the constant symbols by defining

$$\gamma_n^{\mathfrak{b}} \text{ PartOf}_{\mathcal{F}} \gamma_m^{\mathfrak{a}} \Leftrightarrow n = m \wedge \mathfrak{b} \text{ PartOf } \mathfrak{a} \wedge \gamma_n^{\mathfrak{b}} \in \mathcal{F}^{-1}(\mathfrak{b}) \wedge \gamma_m^{\mathfrak{a}} \in \mathcal{F}^{-1}(\mathfrak{a}).$$

We will omit the subscript $\mathcal{F}$ in $\text{PartOf}_{\mathcal{F}}$ whenever the function $\mathcal{F}$ is understood from the context.

For each $t \in T$, there is a temporal constant c_t in $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$.

- $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ contains predicate symbols `Start`, `Stop`, `Doing`, `Wait`, `Started`, `Stopped` and function symbols `Duration`, `Extratime` for actions, and symbols $+$ (addition), $\leq$ and $\equiv_d$ (equality modulo d) for each $d \in T, d > 0$ for temporal variables and constants. We will use $=$ as short-hand for $\equiv_1$.[2]
- The temporal terms, denoted π, π_i, are defined via

$$\pi ::= x + \pi \,|\, c_t + \pi \,|\, x \,|\, c_t.$$

Action terms are given by $\Upsilon ::= X | \gamma_n^{\mathfrak{a}}$.

- We define the atomic formulae p of $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ by

$$\begin{aligned} p ::= & \texttt{Start}(\Upsilon) \,|\, \texttt{Stop}(\Upsilon) \,|\, \texttt{Doing}(\Upsilon) \,|\, \texttt{Wait}(\Upsilon) \,|\, \texttt{Duration}(\Upsilon) = \pi \,| \\ & \texttt{Extratime}(\Upsilon) = \pi \,|\, \texttt{Started}(\Upsilon, \pi) \,|\, \texttt{Stopped}(\Upsilon, \pi) \,|\, \Upsilon_1 \,\text{PartOf}_{\mathcal{F}}\, \Upsilon_2 \,|\, \pi_1 \leq \pi_2 \,| \\ & \pi_1 \equiv_d \pi_2. \end{aligned}$$

Formulae ϕ of $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ are then defined by

$$\phi ::= p \,|\, \phi \vee \psi \,|\, \neg\phi \,|\, \bigcirc \phi \,|\, \phi \mathcal{U} \psi \,|\, x.\phi \,|\, \forall X\, \phi,$$

where p is an atomic formula.

We obtain the usual logical operators, i.e. $\Diamond\phi$, $\Box\phi$ and $\phi \mathit{\ before\ } \psi$, as abbreviations for $\top \mathcal{U} \phi$, $\neg\Diamond\neg\phi$ and $\neg(\neg\phi \mathcal{U} \psi)$ (see e.g. [21]).

- Finally, ϕ is an $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$-*sentence* if any temporal variable x occurring in ϕ is bound by an occurrence of the fix quantifier $x.\psi$, and any action variable X is bound by an occurrence of the universal quantifier $\forall X \psi$.

The intuition behind $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ is the following: the activities of $\mathcal{A}$ should be viewed as the basis for propositions of the form $\texttt{Start}(\gamma_n^{\mathfrak{a}})$, $\texttt{Stop}(\gamma_n^{\mathfrak{a}})$, $\texttt{Doing}(\gamma_n^{\mathfrak{a}})$, etc. In particular, this allows us to treat the actions as *durative*: in the usual interpretation of LTL, actions would be treated as being done instantaneously between states, i.e. if $\mathfrak{a}$ is done in state σ_i, then it is completed in state σ_{i+1}. The attached times τ_i and τ_{i+1} would be able to record some notion of duration, but if other actions with shorter durations would be done simultaneously with $\mathfrak{a}$, we would need to have a number of consecutive states registering that $\mathfrak{a}$ is being done. The fact that $\mathfrak{a}$ then occurs in multiple states might be taken as multiple instances of $\mathfrak{a}$ being done, rather than just one action taking some longer time. Using `Doing`() here does not only allow for this interpretation to be avoided, it also allows the notion of "taking a break", i.e. the language is able to express that an action is interrupted for some time, and then continued.

The concrete language varies with different BIH $\mathfrak{A}$. Furthermore, our choice of taking countably many distinct instances for each activity is motivated by the idea that each instance of an activity can only be done once: if we baked pizza on Monday, 5th February 2018, then

[2] Note that both `Started` and `Stopped` are intended to work as predicates on *pairs* of the form $\langle \Upsilon, \pi \rangle$; while it certainly is permissible to have distinct action instances $\gamma_n^{\mathfrak{a}}$ associated with the same time c_t, conditions (C8) and (C9) in Definition 5 will ensure that each such instance will only be paired with precisely one such time. Thus both of these predicates could also be seen as functions. We opt to view them as predicates here to emphasize the rather static nature of record keeping for which they are intended.

we have an instance for the activity in any trace starting with or before this day, but we would certainly never bake the exact same pizza again in the future. Even if we chose to use the same recipe in the future to bake another pizza, we would still argue that it is not the exact same pizza that was baked in February 2018. To allow us to make this distinction one can see the index n of an activity constant $\gamma_n^{\mathfrak{a}}$ as some form of ID associated with a specific instance of an action, e.g. "the first time we bake this pizza", "the second time", etc.

While this means that the number of *distinct instances* of activities is infinite[3], the number of distinct *activities* is still finite as $\mathcal{A}$ is finite. We will now define trace semantics for $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$. The general idea is as follows: a model of a collection of $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ - sentences is a sequence of timed states. Each state consists of a number of actions holding, namely those actions that are (actively) being carried out at the given time. The "timed" nature of these states is accomplished by pairing each state σ_i with a time τ_i.

▶ **Definition 4** (Trace semantics). A $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$-structure $\mathfrak{M}$ is a pair $\mathfrak{M} = \langle \vec{\rho}, \mathcal{E} \rangle$, where

- $\vec{\rho}$ is a (countably infinite) sequence of pairs $\rho_i = \langle \sigma_i, \tau_i \rangle$, where

$$\sigma_i \in \mathcal{P}\left([\mathcal{A} \times \mathbb{N}] \cup \text{Start}(\mathcal{A} \times \mathbb{N}) \cup \text{Stop}(\mathcal{A} \times \mathbb{N}) \cup \text{Started}(\mathcal{A} \times \mathbb{N} \times T) \cup \text{Stopped}(\mathcal{A} \times \mathbb{N} \times T)\right)$$

  and $\tau_i \in T$, such that
  - $\text{Start}(\mathcal{A} \times \mathbb{N}) = \{\langle \text{Start}(\mathfrak{a}), i \rangle \mid \mathfrak{a} \in \mathcal{A}, i \in \mathbb{N}\}$,
  - $\text{Stop}(\mathcal{A} \times \mathbb{N}) = \{\langle \text{Stop}(\mathfrak{a}), i \rangle \mid \mathfrak{a} \in \mathcal{A}, i \in \mathbb{N}\}$,
  - $\text{Started}(\mathcal{A} \times \mathbb{N} \times T) = \{\langle \text{Started}(\mathfrak{a}), i, t \rangle \mid \mathfrak{a} \in \mathcal{A}, i \in \mathbb{N}, t \in T\}$,
  - $\text{Stopped}(\mathcal{A} \times \mathbb{N} \times T) = \{\langle \text{Stopped}(\mathfrak{a}), i, t \rangle \mid \mathfrak{a} \in \mathcal{A}, i \in \mathbb{N}, t \in T\}$,
  - the sequence $\vec{\tau} = \langle \tau_i \rangle$ is monotone, i.e. $i \leq j$ implies that $\tau_i \leq \tau_j$, and progressive, i.e. for each $t \in T$ there is some $j \in \mathbb{N}$ such that $\tau_j > t$,
- $\mathcal{E}_T : V_T \to T$ is an interpretation function for the temporal variables of $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$; we extend $\mathcal{E}_T$ to temporal constants by letting $\mathcal{E}_T(c_T) = c_t^{\mathfrak{M}} \in T$ for each temporal constant c_t of $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$.
- $\mathcal{E}_A : V_A \to \mathcal{A}$ is an interpretation function for the action variables of $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$; we extent $\mathcal{E}_A$ to action constants by letting $\mathcal{E}_A(\gamma_n^{\mathfrak{a}}) = \mathfrak{a}$ for each action constant $\gamma_n^{\mathfrak{a}}$.
- For readability, and since both the domains and ranges of $\mathcal{E}_T$ and $\mathcal{E}_A$ are distinct, we can define a common interpretation function $\mathcal{E} = \mathcal{E}_T \cup \mathcal{E}_A$.

Let $\mathfrak{M}$ be a $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$-structure and $i \in \mathbb{N}$. Then $\langle \mathfrak{M}, i \rangle \models \phi$ is defined as follows:

- $\langle \mathfrak{M}, i \rangle \models \texttt{Doing}(\gamma_n^{\mathfrak{a}})$ iff $\langle \mathfrak{a}, n \rangle \in \sigma_i$,
- $\langle \mathfrak{M}, i \rangle \models \texttt{Start}(\gamma_n^{\mathfrak{a}})$ iff $\langle \text{Start}(\mathfrak{a}), n \rangle \in \sigma_i$,
- $\langle \mathfrak{M}, i \rangle \models \texttt{Stop}(\gamma_n^{\mathfrak{a}})$ iff $\langle \text{Stop}(\mathfrak{a}), n \rangle \in \sigma_i$,
- $\langle \mathfrak{M}, i \rangle \models \texttt{Started}(\gamma_n^{\mathfrak{a}}, \pi)$ iff $\langle \text{Started}(\mathfrak{a}), n, t \rangle \in \sigma_i$ and $\mathcal{E}(\pi) = t$,
- $\langle \mathfrak{M}, i \rangle \models \texttt{Stopped}(\gamma_n^{\mathfrak{a}}, \pi)$ iff $\langle \text{Stopped}(\mathfrak{a}), n, t \rangle \in \sigma_i$ and $\mathcal{E}(\pi) = t$,
- $\langle \mathfrak{M}, i \rangle \models \texttt{Duration}(\gamma_n^{\mathfrak{a}}) = \pi$ iff $f_T(\mathfrak{a}) = t$ and $\mathcal{E}(\pi) = t$,
- $\langle \mathfrak{M}, i \rangle \models \texttt{Extratime}(\gamma_n^{\mathfrak{a}}) = \pi$ iff $g_T(\mathfrak{a}) = t$ and $\mathcal{E}(\pi) = t$,
- $\langle \mathfrak{M}, i \rangle \models \texttt{Wait}(\gamma_n^{\mathfrak{a}})$ iff $\mathit{passive}(\mathfrak{a}) = 1$,
- $\langle \mathfrak{M}, i \rangle \models \gamma_n^{\mathfrak{a}} \,\text{PartOf}_{\mathcal{F}}\, \gamma_m^{\mathfrak{b}}$ iff $\mathfrak{a} \,\text{PartOf}\, \mathfrak{b}$ and $n = m$,
- $\langle \mathfrak{M}, i \rangle \models \pi_1 \leq \pi_2$ iff $\mathcal{E}(\pi_1) \leq \mathcal{E}(\pi_2)$.
- $\langle \mathfrak{M}, i \rangle \models \pi_1 \equiv_d \pi_2$ iff $\mathcal{E}(\pi_1) \equiv_d \mathcal{E}(\pi_2)$.

[3] In all practicality, this number will be finite as well, as human life is finite. Limiting the number of instance for any activity right from the start, however, seems to be too strict a requirement here.

- $\langle \mathfrak{M}, i\rangle \models \phi \vee \psi$ iff $\langle \mathfrak{M}, i\rangle \models \phi$ or $\langle \mathfrak{M}, i\rangle \models \psi$,
- $\langle \mathfrak{M}, i\rangle \models \neg\phi$ iff $\langle \mathfrak{M}, i\rangle \not\models \phi$,
- $\langle \mathfrak{M}, i\rangle \models \bigcirc\phi$ iff $\langle \mathfrak{M}, i+1\rangle \models \phi$,
- $\langle \mathfrak{M}, i\rangle \models \phi \mathcal{U} \psi$ iff there is $k \geq i$ such that $\langle \mathfrak{M}, k\rangle \models \psi$ and for all $i \leq l < k$, $\langle \mathfrak{M}, l\rangle \models \phi$,
- $\langle \mathfrak{M}, i\rangle \models x.\phi$ iff $\langle \mathfrak{M}', i\rangle \models \phi$ for that $\mathfrak{M}' = \langle \vec{\rho}, \mathcal{E}'\rangle$ such that $\mathcal{E}'_T(y) = \mathcal{E}_T(y)$ for all $y \neq x$ and $\mathcal{E}'_T(x) = \tau_i$,
- $\langle \mathfrak{M}, i\rangle \models \forall X \phi$ iff $\langle \mathfrak{M}', i\rangle \models \phi(X)$ for any $\mathfrak{M}' = \langle \rho, \mathcal{E}'\rangle$ such that $\mathcal{E}_A$ agrees with $\mathcal{E}'_A$ on all variables except for X.

If the truth value of $\langle \mathfrak{M}, i\rangle \models \phi$ is independent of i, then we shall drop i and simply write $\mathfrak{M} \models \phi$.

We will now define a few criteria that one might want to hold on the traces. For this, we let $\mathfrak{M}$ be an $\mathrm{TPTL}_{\mathrm{BIH}}(\mathfrak{A})$-structure.

▶ **Definition 5.** We will define the following criteria for $\mathrm{TPTL}_{\mathrm{BIH}}(\mathfrak{A})$ structures.

- (C0), (Finiteness of states) For each $i \in \mathbb{N}$, σ_i is finite.
- (C1), (No Stop without Start) If $\langle \mathrm{Stop}(\mathfrak{a}), n\rangle \in \sigma_i$, then there exists $k < i$ such that $\langle \mathrm{Start}(\mathfrak{a}), n\rangle \in \sigma_k$.
- (C2), (No Doing before Start) $\langle \mathrm{Start}(\mathfrak{a}), n\rangle \in \sigma_i$ implies $\forall k < i\, \langle \mathfrak{a}, n\rangle \notin \sigma_k$.
- (C3), (No Doing after Stop) $\langle \mathrm{Stop}(\mathfrak{a}), n\rangle \in \sigma_i$ implies $\forall k > i\, \langle \mathfrak{a}, n\rangle \notin \sigma_k$.
- (C4), (Logging Start) If $\langle \mathrm{Start}(\mathfrak{a}), n\rangle \in \sigma_i$, then for all $k > i$, $\langle \mathrm{Started}(\mathfrak{a}), n, \tau_i\rangle \in \sigma_k$.
- (C5), (Logging Stop) If $\langle \mathrm{Stop}(\mathfrak{a}), n\rangle$, then for all $k > i$, $\langle \mathrm{Stopped}(\mathfrak{a}), n, \tau_i\rangle \in \sigma_k$.
- (C6), (Doing at Start) If $\langle \mathrm{Start}(\mathfrak{a}), n\rangle \in \sigma_i$, then $\langle \mathfrak{a}, n\rangle \in \sigma_i$.
- (C7), (Not Doing at Stop) If $\langle \mathrm{Stop}(\mathfrak{a}), n\rangle \in \sigma_i$, then $\langle \mathfrak{a}, n\rangle \notin \sigma_i$.
- (C8), (Unique Start) If $\langle \mathrm{Start}(\mathfrak{a}), n\rangle \in \sigma_i$ and $\langle \mathrm{Start}(\mathfrak{a}), n\rangle \in \sigma_k$, then $i = k$.
- (C9), (Unique Stop) If $\langle \mathrm{Stop}(\mathfrak{a}), n\rangle \in \sigma_i$ and $\langle \mathrm{Stop}(\mathfrak{a}), n\rangle \in \sigma_k$, then $i = k$.

The conditions above are intended to correspond to properties we wish to hold in our logic: for instance, as discussed above, the states of our logic will only be finite, hence we have (C0) in the list of desired criteria.

The criteria (C1)-(C5) state some simple properties for `Start`() and `Stop`(): we actually want that `Start`() cannot be preceded by an instance of doing an activity, since this would contradict the intuition that one can only start an activity that is not already being executed. Similarly, we do not want any instance of doing an activity to occur after a `Stop`() is placed in the trace. We also want to ensure that any stop signal for an activity is preceded by a start signal. Furthermore, (C4) and (C5) ensure that we are logging all occurrences of `Start`() and `Stop`().

One could argue here that `Start`() could also be defined via properties of the language like $\mathcal{U}$, e.g. $\texttt{Start}(\gamma_n^{\mathfrak{a}})$ could be defined via $\neg\texttt{Doing}(\gamma_n^{\mathfrak{a}})\,\mathcal{U}\,\texttt{Doing}(\gamma_n^{\mathfrak{a}})$. However, we want to allow that an activity can be paused, e.g. $\texttt{Doing}(\gamma_n^{\mathfrak{a}})$ is not required to always hold on the interval determined by $\texttt{Start}(\gamma_n^{\mathfrak{a}})$ and $\texttt{Stop}(\gamma_n^{\mathfrak{a}})$. Without $\texttt{Start}(\gamma_n^{\mathfrak{a}})$ as a primitive, we would need past time operators to determine whether a $\texttt{Doing}(\gamma_n^{\mathfrak{a}})$ is actually the first one occurring, or whether the action has just been paused for a long time, and thus $\neg\texttt{Doing}(\gamma_n^{\mathfrak{a}})\,\mathcal{U}\,\texttt{Doing}(\gamma_n^{\mathfrak{a}})$ holds at a point i just because the most recent instance of $\texttt{Doing}(\gamma_n^{\mathfrak{a}})$ has occurred at a point $k < i$.

With the criteria (C6) and (C7) we want to ensure that any `Start`() signal is indeed accompanied by actually doing an activity, while the `Stop`() does indeed mean that we are no longer seeing the activity being done.

The criteria (C8) and (C9) provide our interpretation of `Start`() and `Stop`(). We see the start of an activity as the initial point when it is begun, and the stop as the point of completion. Thus these signals are unique – we do not intend to use them in order to measure any breaks in the execution of an activity.

While it is an immediate consequence that (C8) follows from the conditions (C2) and (C6), this is not true for (C9); this condition does not follow from (C3) and (C7). The crucial point here is that the list of conditions does not guarantee that `Stop`() has to occur immediately after the last instance of doing an activity – therefore, having multiple `Stop`() signals occurring in the trace does not contradict any of the criteria (C0) - (C8).

The following proposition states that our list of conditions is indeed satisfyable.

▶ **Proposition 6.** *Given a BIH $\mathfrak{A}$, there exists a $TPTL_{\text{BIH}}(\mathfrak{A})$-structure $\mathfrak{M}$ satisfying the conditions (C0) - (C9).*

Sketch. Let $\mathfrak{A} = \langle \mathcal{A}, f_T, g_T, \mathit{passive} \rangle$ a BIH. We construct a structure as follows: Let $\vec{\tau}$ be an arbitrary, monotone and progressive sequence of T.

In a first step, add both $\langle \mathfrak{a}, 0 \rangle$ and $\langle \text{Start}(\mathfrak{a}), 0 \rangle$ to σ_0 for any $\mathfrak{a} \in \mathcal{A}$. Let σ_1 be the set containing the corresponding $\langle \text{Stop}(\mathfrak{a}), 0 \rangle$. Let all other σ_i be empty.

Now add, for any $\mathfrak{a} \in \mathcal{A}$, $\langle \text{Started}(\mathfrak{a}), 0, \tau_0 \rangle$ to all σ_i with $i > 0$, and similarly add $\langle \text{Stopped}(\mathfrak{a}), 0, \tau_1 \rangle$ to all σ_i with $i > 1$.

Note that since $\mathcal{A}$ is finite, each of the σ_i are finite.

It is easy to verify that the criteria (C1)-(C9) are satisfied as well. E.g., the fact that $\langle \text{Start}(\mathfrak{a}), 0 \rangle$ is not preceded by any occurrence of $\langle \mathfrak{a}, 0 \rangle$ is trivially satisfied. And since $\langle \mathfrak{a}, 0 \rangle$ is an element of σ_0 but not of σ_1, we have (C2), (C3) and (C6) holding. ◀

4 Coherence

In this section, we will define Coherence for an instance of an activity $\mathfrak{a}$. Informally, an activity is *coherently done* if it is done without distraction and without unexpected delays. In terms of Example 1, Pedro does not do any tasks not related to pizza making, and does not take too long breaks in between different subactivities, as well as during them. The only exception we may allow is the time when the pizza is baking in the oven: one could argue that since this is a *passive* activity, Pedro could use that time to attend to other activities, e.g. taking out the rubbish.

▶ **Definition 7** (Coherence)**.** We say that in a structure $\mathfrak{M}$, Coherence holds for an action instance $\gamma_n^{\mathfrak{a}}$ at point i, iff

$$\begin{aligned}
&\langle \mathfrak{M}, i \rangle \models \forall X \, \text{PartOf}_{\mathcal{F}} \, \gamma_n^{\mathfrak{a}} \, [\texttt{Start}(\gamma_n^{\mathfrak{a}}) \mathit{\ before\ } \texttt{Start}(X) \wedge \texttt{Stop}(X) \mathit{\ before\ } \texttt{Stop}(\gamma_n^{\mathfrak{a}})] \\
&\wedge \forall X \, [\neg X \, \text{PartOf}_{\mathcal{F}} \, \gamma_n^{\mathfrak{a}} \rightarrow (\neg \texttt{Start}(X) \vee \\
&\forall Y \, \text{PartOf}_{\mathcal{F}} \, \gamma_n^{\mathfrak{a}} \, [\texttt{Doing}(Y) \rightarrow (\texttt{Wait}(Y) \wedge \texttt{Stop}(X) \mathit{\ before\ } \texttt{Stop}(Y))])] \\
&\wedge \Diamond \, [\texttt{Started}(\gamma_n^{\mathfrak{a}}, \pi_1) \wedge \texttt{Stopped}(\gamma_n^{\mathfrak{a}}, \pi_2) \wedge \pi_2 \leq \pi_1 + \texttt{Duration}(\gamma_n^{\mathfrak{a}}) + \texttt{Extratime}(\gamma_n^{\mathfrak{a}})]
\end{aligned}$$

The intuition behind this definition is as follows: we consider an activity as being done coherently when any of its subactivities are performed only within the limits set by the activity itself; furthermore, unrelated activities are only performed if there is some time in which the user has to wait anyways; and finally, doing the activity does not take up more time than it should.

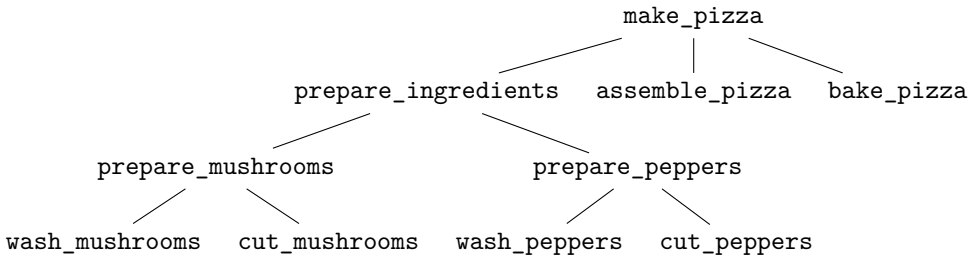


Figure 3

Notice that this definition means, Coherence holds only for activities that are *eventually completed*: we cannot refer to $\texttt{Stopped}(\gamma_n^a, \pi)$ before we actually see a $\texttt{Stop}(\gamma_n^{\mathfrak{a}})$ in the trace. The eventuality in the final conjunct thus implies that Coherence can only hold if the activity is eventually stopped. We would argue that this is indeed intended – the alternatives are that an activity is either abandoned or is carried on forever. Furthermore, this last conjunct implies that the collective duration of all subactivities of $\mathfrak{a}$ executed in this instance does not exceed the duration of $\mathfrak{a}$ in this instance. Since we want Coherence to have a chance of regularly being satisfied, we need the BIH $\mathfrak{A}$ to fulfill the further condition that for all $\mathfrak{a} \in \mathcal{A}$, $\sum_{\mathfrak{b}\,\text{PartOf}\,\mathfrak{a}} f_T(\mathfrak{b}) + g_T(\mathfrak{b}) \leq f_T(\mathfrak{a}) + g_T(\mathfrak{a})$. This is a rational assumption, as one would not expect that an activity can be regularly done in a shorter timespan than it takes to do all of its parts if these are done consecutively.

As an example for Coherence, consider the tree shown in Figure 3.

With respect to the remark above regarding the condition on the durations for subactivities, we also add the following average durations (in minutes):

action	average duration
`make_pizza`	45
`bake_pizza`	20
`assemble_pizza`	5
`prepare_mushrooms`	8
`prepare_peppers`	8

Summing the times for the different parts of `make_pizza`, we see that doing everything effectively, there is a buffer of 4 minutes for making pizza.

Consider the following untimed initial trace $\vec{\sigma}$. For readability, we will omit the notation $\gamma_n^{\mathfrak{a}}$ and just use $\mathfrak{a}$, with $n = 0$ in this case. E.g., `make_pizza` in the table below should be read as $\gamma_0^{\texttt{make\_pizza}}$.

σ_0	`Start(make_pizza)`	`Start(wash_mushrooms)`
σ_1		`Stop(wash_mushrooms)`
σ_2		`Start(wash_peppers)`
σ_3		`Stop(wash_peppers)`
σ_4		`Start(cut_mushrooms)`
σ_5		`Stop(cut_mushrooms)`
σ_6		`Start(cut_peppers)`
σ_7		`Stop(cut_peppers)`
σ_8		`Start(assemble_pizza)`
σ_9		`Stop(assemble_pizza)`
σ_{10}		`Start(bake_pizza)`
σ_{11}	`Stop(make_pizza)`	`Stop(bake_pizza)`

Furthermore, add the logging statement $\texttt{Started}(\texttt{make\_pizza}, \tau_0)$ to each σ_i with $i > 0$ and similarly add $\texttt{Stopped}(\texttt{make\_pizza}, \tau_{11})$ to each σ_i with $i > 12$. Proceed similarly for the logging events for each of the subactivities of `make_pizza`. We treat the τ_i here as placeholders for now, and replace them by the appropriate values once we have defined the concrete values below.

This initial segment can be extended to a trace $\vec{\sigma}$ arbitrarily to obtain a trace of states that satisfies a coherent run of making a pizza. To obtain an $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$-structure, we furthermore need some sequence of time points $\vec{\tau}$. The following initial sequence will then show that $\text{Coh}(\texttt{make\_pizza})$ is satisfiable:

$$\tau_0 = 0, \tau_1 = 3, \tau_2 = 3, \tau_3 = 6, \tau_4 = 6, \tau_5 = 11, \tau_6 = 11, \tau_7 = 16,$$
$$\tau_8 = 16, \tau_9 = 21, \tau_{10} = 21, \tau_{11} = 41$$

Interpreting each time point τ_i as the minutes, then any extension of this initial sequence of τ_i to a time trace $\vec{\tau}$ will lead to an $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$-structure witnessing the satisfiability. Note that we can make this an acceptable $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$-structure by just adding to the σ_i the corresponding logging entries. Furthermore, it is easy to see now, that this does not make Coherence a theorem of acceptable $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$-structures: the validity of $\text{Coh}(\texttt{make\_pizza})$ depends in this example on the values we pick for τ_i. Just picking different, larger values for the τ_i we can obtain an $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$-structure in which $\text{Coh}(\texttt{make\_pizza})$ is false, simply because the condition for starting time and stopping time of `make_pizza` does not hold.

Intuitively, it seems that there are only two ways in which Coherence for a complex activity can fail: one would not expect to see the $\texttt{Stop}(\texttt{make\_pizza})$ statement before e.g. $\texttt{Start}(\texttt{bake\_pizza})$ happens. Thus if the time sequence matches the condition of Coherence for some activity $\mathfrak{a}$, one would expect that the only other way to make it still fail is if one introduces an unrelated activity, e.g. if in the pizza making example we have $\texttt{Start}(\texttt{watch\_TV})$ in σ_0.

The strategy used above can be used to demonstrate that for any BIH $\mathfrak{A}$ satisfying the extra condition on the time functions, there exist $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ - structures $\mathfrak{M}$ satisfying Coherence for each $\mathfrak{a} \in \mathfrak{A}$:

▶ **Proposition 8.** *Let $\mathfrak{A}$ be a BIH satisfying the condition that for all $\mathfrak{a} \in \mathcal{A}$, $\sum_{\mathfrak{b}\ \text{PartOf}\ \mathfrak{a}} f_T(\mathfrak{b}) + g_T(\mathfrak{b}) \leq f_T(\mathfrak{a}) + g_T(\mathfrak{a})$. Then there exists a $TPTL_{\text{BIH}}(\mathfrak{A})$ - structure $\mathfrak{M}$ such that for any $\mathfrak{a} \in \mathfrak{A}$ and any instance $\gamma_n^{\mathfrak{a}}$ there is an index $i_{\mathfrak{a},n}$ such that*

$$\langle \mathfrak{M}, i_{\mathfrak{a},n} \rangle \models \text{Coh}(\gamma_n^{\mathfrak{a}}).$$

Sketch. Assume an enumeration $\mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_m$ of the top-level actions of $\mathcal{A}$, i.e. for each of those $\mathfrak{a}_i$, there is not $\mathfrak{b}$ such that $\mathfrak{a}_i\ \text{PartOf}\ \mathfrak{b}$. Starting with $\mathfrak{a}_1$, we construct a trace satisfying $\text{Coh}(\mathfrak{b})$ for any $\mathfrak{b} \in \mathcal{A}$ as follows.

Start off with any σ_i the empty set. Let $\tau_0 = 0$. Add to σ_0 the elements $\langle \text{Start}(\mathfrak{a}_1), 0 \rangle$ and $\langle \mathfrak{a}_1, 0 \rangle$. Identify each $\mathfrak{b}_j\ \text{PartOf}\ \mathfrak{a}_1$, and enumerate them as well, add $\langle \text{Start}(\mathfrak{b}_1), 0 \rangle$, $\langle \mathfrak{b}_1, 0 \rangle$ to σ_0 as well. Continue until reaching a level in the tree for which there is no child, i.e. reach $\mathfrak{c}_1\ \text{PartOf} \cdots \text{PartOf}\ \mathfrak{b}_1\ \text{PartOf}\ \mathfrak{a}_1$ such that there is no $\mathfrak{d}$ with $\mathfrak{d}\ \text{PartOf}\ \mathfrak{c}_1$. This exists, since $\mathcal{A}$ is finite, and PartOf is well-founded.

Now add to σ_1 the element $\langle \text{Stop}(\mathfrak{c}_1), 0 \rangle$, for any of the higher level activities $\mathfrak{d}$ that were started add the elements $\langle \mathfrak{d}, 0 \rangle$, and all the appropriate logging elements $\langle \text{Started}(\mathfrak{d}), 0, 0 \rangle$. Let $\tau_1 = f_T(\mathfrak{c}_1)$. At this point $\text{Coh}(\gamma_0^{\mathfrak{c}_1})$ holds, as it does not have any subactivities and the difference $\tau_1 - \tau_0$ has the correct value.

Traversing the BIH tree, continue like this until all the subactivities are recorded as being started and stopped in a finite initial segment of $\vec{\sigma}$, with the τ_i having the appropriate values. Having reached σ_k in this way, add $\langle \text{Stopped}(\mathfrak{a}_1), 0\rangle$ to this σ_k and observe that $\text{Coh}(\gamma_0^{\mathfrak{a}_1})$ holds at this point.

Proceed like this for all the other $\mathfrak{a}_i$. Having reached a finite point l such that at σ_l, all the activity instances with index 0 have Coherence satisfied, we can just copy this initial segment, increase the indices of the instances by one and add τ_l to each of the τ_i, obtaining $\langle \vec{\sigma}, \vec{\tau}\rangle$ up to index $2l$ satisfying coherence for the first two instances for each activity in $\mathfrak{A}$. Continuing in this way, we obtain an infinite trace $\vec{\rho} = \langle \vec{\sigma}, \vec{\tau}\rangle$ satisfying $\text{Coh}(\gamma_n^{\mathfrak{a}})$ for each $\mathfrak{a} \in \mathcal{A}$ and each $n \in \mathbb{N}$. ◀

While it is certainly conceivable that Coherence could be used in a recursive manner, this is not given by the definition. For instance, considering $\text{Coh}(\texttt{make\_pizza})$ in the example above, it is only required that `prepare_mushrooms` and `prepare_peppers` are executed within the activity of `make_pizza`, but not that coherence holds for both sub-activities. Depending on the user's need for support, one may add the requirement that $\text{Coh}(\texttt{make\_pizza}) \wedge \text{Coh}(\texttt{prepare\_mushrooms}) \wedge \text{Coh}(\texttt{prepare\_peppers})$ holds, as the user feels incapable of mixing the latter two tasks. Thus Coherence need not propagate recursively to sub-activities in the BIH.

However, the situation is different when considering two distinct, unrelated activities. Suppose that the user needs to take out the trash before 6 pm as well as prepare the pizza for dinner at 6 pm. Taking the trash out at some time during the pizza preparation would violate Coherence for `making_pizza`. However, we would argue that the role of our logic is precisely to recognize such conflicts; an agent utilizing the logic would thus be able to recognize such a conflict, and can act upon that, e.g. by suggesting to take the trash out later, when the pizza is in the oven, or by merely reminding the user that some ingredients are not prepared when the user has finished the activity of taking the trash out. How a conflict like this is handled should be decided on a different level, e.g. through providing priority levels for activities in the BIH or other means. That is, we do not intend our logic $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ to handle conflicts like these.

5 Conclusion/Discussion

In this paper, we have presented $\text{TPTL}_{\text{BIH}}(\mathfrak{A})$ as a logic capable of formulating statements about Activities of Daily Living. The language can act as a framework for reasoning about and monitoring user-reported behaviour.

We see this paper as an important step towards building supportive systems that take a focus on the needs and well-being of the people intended to use them. Our approach does not only allow modelling different levels of activities, and by that allow for a more fine-grained support, but also provides the ability to let the cared-for person have an impact on the way they are supported: our logic allows that desired norms and behaviours are either formulated as general statements applying to all activities alike, or being encoded into the logic by means of the Behaviour Identification Hierarchy.

In particular our notion of Coherence together with the BIH allows a supportive agent to be flexibly attending to the user's needs: while an agent following our logic is able to register when actions are done in a timely manner, it does allow for individual difficulties, e.g. Pedro might struggle a lot more with remembering to take the pizza out of the oven before it is burnt than cutting the mushrooms, and thus the "grace periods" given by the function g_T in Pedro's BIH can vary accordingly.

The notion of Coherence, as defined in this paper, is closely related to the notion of normal/abnormal behaviour as mentioned in e.g. [17]. However, the notion of "normal behaviour", and thus its counterpart 'abnormal behaviour' is rather strictly formulated in the literature. While those descriptions of normal behaviour would imply the notion of Coherence, our approach is much more flexible: subactivities can be carried out in any order, so as long as the end result is succesfully achieved in due time, the monitored user can act in any way they want without the need of the supportive agent to intervene.

While the basic logic TPTL is known to be decidable, it is currently unkown whether our modifications of the base language change this fact. In particular since several generalizations of TPTL, allowing for dense or real time, are undecidable, the question whether our modifications have similar effects on the decidability becomes important.

Furthermore, the definition of a BIH as used in this paper is far from being practical in real-life situations: we currently do not allow any partiality: if putting mushrooms on the pizza is at least sometimes desired by the user, our BIH model makes that a certainty. It is clear that adding a notion of "sometimes" or "often" needs to be added to the current framework in order to more closely model actual human behaviour. As part of future work in this direction experiments are planned within the CoreSAEP research project. These will involve the question whether a more complex version of our BIH trees will indeed be useful to describe human habits.

References

1. Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–203, 1994. doi:10.1145/174644.174651.
2. Giulia Andrighetto, Guido Governatori, Pablo Noriega, and Leendert van der Torre, editors. *Normative Multi-Agent Systems*, volume 4 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/DFU.Vol4.12111.i.
3. Jan Broersen, Frank Dignum, Virginia Dignum, and John-Jules Ch Meyer. Designing a deontic logic of deadlines. In *Proceedings Seventh International Workshop on Deontic Logic in Computer Science (DEON'04)*, volume 3065 of *LNCS*, pages 43–56. Springer-Verlag, 2004.
4. Stephen Cranefield. A rule language for modelling and monitoring social expectations in multi-agent systems. In *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems (ANIREM'05 and OOOP'05)*, volume 3913 of *LNCS*, pages 246–258, 2006.
5. F. Dignum and R. Kuiper. Specifying deadlines with dense time using deontic and temporal logic. *International Journal of Electronic Commerce*, 3(2):67–86, 1998.
6. Guido Governatori, Joris Hulstijn, Régis Riveret, and Antonino Rotolo. Characterising deadlines in temporal modal defeasible logic. In *Proceedings of the 20th Australian joint conference on Advances in artificial intelligence*, pages 486–496. Springer-Verlag, 2007.
7. Koen V. Hindriks and M. Birna van Riemsdijk. A real-time semantics for norms with deadlines. In *Proceedings of the twelfth international joint conference on autonomous agents and multiagent systems (AAMAS'13)*, pages 507–514. IFAAMAS, 2013.
8. Alex Kayal, Willem-Paul Brinkman, Hanna Zoon, Mark A. Neerincx, and M. Birna van Riemsdijk. A value-sensitive mobile social application for families and children. In *Posters, Demos, Late-breaking Results and Workshop Proceedings of the 22nd Conference on User Modeling, Adaptation, and Personalization (UMAP'14)*, volume 1181. CEUR, 2014.
9. Thomas Kleinberger, Andreas Jedlitschka, Holger Storf, Silke Steinbach-Nordmann, and Stephan Prueckner. An approach to and evaluations of assisted living systems using ambient intelligence for emergency monitoring and prevention. In Constantine Stephanidis, editor, *Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments*, pages 199–208, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

10 Malte S. Kließ and M. Birna van Riemsdijk. Requirements for a temporal logic of daily activities for supportive technology. In Matteo Baldoni, Cristina Baroglio, and Roberto Micalizio, editors, *Proceedings of the First Workshop on Computational Accountability and Responsibility in Multiagent Systems co-located with 20th International Conference on Principles and Practice of Multi-Agent Systems, CARe-MAS@PRIMA 2017, Nice, France, October 31st, 2017.*, volume 2051 of *CEUR Workshop Proceedings*, pages 43–51. CEUR-WS.org, 2017. URL: `http://ceur-ws.org/Vol-2051/paper4.pdf`.

11 Óscar D. Lara and Miguel A. Labrador. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys and Tutorials*, 15(3):1192–1209, 2013.

12 T. Magherini, A. Fantechi, C. D. Nugent, and E. Vicario. Using temporal logic and model checking in automated recognition of human activities for ambient-assisted living. *IEEE Transactions on Human-Machine Systems*, 43(6):509–521, Nov 2013. `doi:10.1109/TSMC.2013.2283661`.

13 Ehsan Nazerfard, Parisa Rashidi, and Diane J. Cook. Using association rule mining to discover temporal relations of daily activities. In Bessam Abdulrazak, Sylvain Giroux, Bruno Bouchard, Hélène Pigot, and Mounir Mokhtari, editors, *Toward Useful Services for Elderly and People with Disabilities*, pages 49–56, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

14 Isabel Nunes, Jose Luiz Fiadeiro, and Wladyslaw M. Turski. A modal logic of durative actions. In Howard Barringer, Michael Fisher, Dov Gabbay, and Graham Gough, editors, *Advances in Temporal Logic*, pages 299–317. Springer Netherlands, Dordrecht, 2000. `doi:10.1007/978-94-015-9586-5_15`.

15 Harri Oinas-Kukkonen. Behavior change support systems: A research model and agenda. In Thomas Ploug, Per Hasle, and Harri Oinas-Kukkonen, editors, *Persuasive Technology: 5th International Conference, PERSUASIVE 2010, Copenhagen, Denmark, June 7-10, 2010. Proceedings*, pages 4–14. Springer, 2010.

16 Pietro Pasotti, M. Birna van Riemsdijk, and Catholijn M. Jonker. Representing human habits: towards a habit support agent. In *Proceedings of the 10th International workshop on Normative Multiagent Systems (NorMAS'16)*, LNCS. Springer, 2016. To appear.

17 M. Ros, M.P. Cuéllar, M. Delgado, and A. Vila. Online recognition of human activities and adaptation to habit changes by means of learning automata and fuzzy temporal windows. *Information Sciences*, 220:86–101, 2013. Online Fuzzy Machine Learning and Data Mining. `doi:10.1016/j.ins.2011.10.005`.

18 Leila S. Shafti, Pablo Alfonso Haya, Manuel García-Herranz, and Xavier Alamán. Personal ambient intelligent reminder for people with cognitive disabilities. In José Bravo, Ramón Hervás, and Marcela Rodríguez, editors, *Ambient Assisted Living and Home Care*, pages 383–390, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

19 John Thangarajah, James Harland, David Morley, and Neil Yorke-Smith. Operational behaviour for executing, suspending, and aborting goals in bdi agent systems. In *Proceedings of the 8th International Conference on Declarative Agent Languages and Technologies VIII*, DALT'10, pages 1–21, Berlin, Heidelberg, 2011. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1996758.1996760`.

20 John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting & exploiting positive goal interaction in intelligent agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 401–408, New York, NY, USA, 2003. ACM. `doi:10.1145/860575.860640`.

21 M. Birna van Riemsdijk, Louise Dennis, Michael Fisher, and Koen V. Hindriks. Agent reasoning for norm compliance: a semantic approach. In *Proceedings of the twelfth international joint conference on autonomous agents and multiagent systems (AAMAS'13)*, pages 499–506. IFAAMAS, 2013.

22 M. Birna van Riemsdijk, Louise Dennis, Michael Fisher, and Koen V. Hindriks. A semantic framework for socially adaptive agents: Towards strong norm compliance. In *Proceedings of the fourteenth international joint conference on autonomous agents and multiagent systems (AAMAS'15)*. IFAAMAS, 2015.

23 M. Birna van Riemsdijk, Catholijn M. Jonker, and Victor Lesser. Creating socially adaptive electronic partners: Interaction, reasoning and ethical challenges. In *Proceedings of the fourteenth international joint conference on autonomous agents and multiagent systems (AAMAS'15)*, pages 1201–1206. IFAAMAS, 2015.

GSM+T: A Timed Artifact-Centric Process Model

Julius Köpke
Alpen-Adria-Universität, Klagenfurt, Austria
julius.koepke@aau.at
https://orcid.org/0000-0002-6678-5731

Johann Eder
Alpen-Adria-Universität, Klagenfurt, Austria
johann.eder@aau.at
https://orcid.org/0000-0001-6050-468X

Jianwen Su
Department of Computer Science, UC Santa Barbara, USA
su@cs.ucsb.edu
https://orcid.org/0000-0002-4637-1339

Abstract

We introduce an extension to the declarative and artifact-centric Guard Stage Milestone (GSM) process modeling language to represent temporal aspects (duration, deadlines, lower- and upper-bound constraints), define the correctness of executions of GSM processes with respect to temporal constraints, check controllability of processes, compute execution plans respecting temporal constraints, and provide a translation method allowing to execute controllable GSM+T processes on standard GSM Engines.

2012 ACM Subject Classification Applied computing → Business process modeling

Keywords and phrases Guard Stage Milestone, GSM, Time Constraints, Controllability, Case Handling, Business Process Modeling

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.18

1 Introduction

Expressing temporal constraints and checking whether a process specification can be executed without violating any constraint (controllability) has been studied for activity-centric processes models for quite some time with works such as [11, 7, 16]. Surprisingly, temporal constraints have not yet been addressed for artifact-centric processes models [15, 9, 1, 13].

We propose to enhance the well-known declarative and artifact-centric process model Guard Stage Milestone (GSM) [15, 9] with temporal constraints and develop techniques for checking their controllability and compute schedules for their correct execution. Since GSM has heavily influenced the new case handling standard CMMN[1] our results also build a foundation for future studies analyzing temporal constraints in CMMN models.

Different notions of correctness of time-constrained processes have been developed, in particular consistency and controllability [7, 8]. Consistency requires that for each possible run there is an execution plan that obeys all temporal constraints. This is generally considered

[1] `http://www.omg.org/spec/CMMN/`

too weak. Controllability requires an execution plan that obeys all temporal constraints for every potential duration of activities and for all possible constellations.

In this paper, we provide the following contributions:

(1) We extend the GSM model with time and temporal constraints (GSM+T).
(2) We formally define the semantics of theses extensions.
(3) We define controllability of GSM+T and provide complete and sound algorithms to check controllability and to compute schedules.
(4) We discuss approaches for the correct execution of GSM+T processes on available GSM engines.

To the best of our knowledge, this is the first approach to consider temporal constraints in artifact-centric process models. In activity centric systems controllability is either checked by constructing schedules along the control structures of process graphs (e.g. [11]) or by mapping timed process models to advanced temporal constraint networks (TCN) (e.g. [7, 17]). Both approaches are not easily applicable for GSM, as it neither has the process graph structure required for the first approach nor do current TCNs support the constructs of GSM.

Controllability is a strong condition for correct executability. The work presented here is also a stepping stone for studying more relaxed conditions like dynamic and history dependent controllability [8] for GSM+T.

2 GSM and GSM+T

2.1 Guard Stage Milestone (GSM)

We introduce the relevant aspects of Guard Stage Milestone (GSM) here and refer the reader to [9, 15] for all details. In GSM, a business process is modeled in form of potentially multiple interacting artifacts (or business objects). A process model defines artifact types with a data schema and a life-cycle schema. A life-cycle model consists of *guards*, *stages*, and *milestones*. Guards define conditions under which a stage can be opened. Milestones specify when a specific business goal is reached and the corresponding stage is then closed. Stages can be *atomic* or *composite*. An atomic stage contains a service (e.g. a task). Composite stages contain other composite or atomic stages and are used to group tasks semantically. Guards and milestones are attached to a specific stage and are defined in form of *sentries*. A sentry is an expression of the form "*On event if condition*" or "*On event*" or "*if condition*". An event can be an *internal* or *external* event. Internal events are opening or closing events of stages (denoted by $+stageName$ or $-stageName$) or achievement or invalidation events of milestones ($+milestoneName$ or $-milestoneName$). External events are arbitrary events received from the environment or completion events of services (called from atomic stages). The condition clause of a sentry is a boolean expression over the data-schema. The data schema contains boolean status attributes for stages and milestones and data relevant for the business object.

Execution Semantics. GSM models basically provide a skeleton for restricting rule-based systems. When an external event (e.g. arbitrary or a service completion event) arrives, the payload of the event is incorporated into the data schema of the artifact instance, and potential rules (translated from sentries plus GSM invariants) are triggered leading to the production of internal events and the invocation of external services. Internal events may trigger other rules (micro-steps) until no further internal events are generated. Then, the next external event is picked from an event queue. One such step of consuming an external

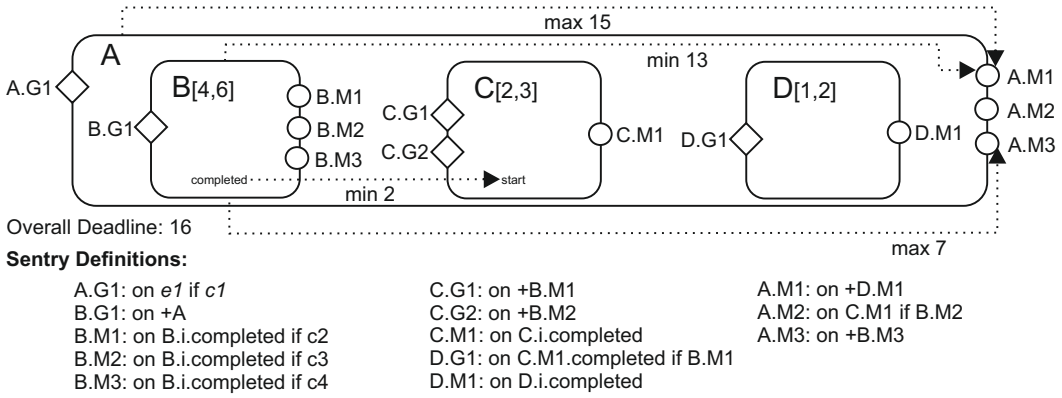


Figure 1 Example GSM Process and some temporal constraints (dotted lines).

event is called a B-Step. The state of the artifact before and after one B-Step is called a Snapshot. A run of a GSM system is a specific sequence of snapshots. All potential runs of a GSM system form a (in general infinite) stage transitions system. GSM prohibits internal events from forming an endless loop (toggle-once principle). During execution, guards and milestones are disjoint. Therefore, only one guard can trigger a single activation of a stage and only one milestone of a stage can be reached for a single invocation.

▶ **Example 1.** In Figure 1 an example GSM process is shown using the usual graphical notion, except from the newly introduced temporal constraints (dotted lines) and service duration intervals (square brackets). It consists of a composite stage A and three atomic stages B, C, D. Stages are depicted in rounded boxes, guards as diamonds, milestones as circles. According to the given sentry definitions (shown below the graphical notation), stage A is opened when the external event $e1$ arrives and the boolean condition $c1$ over the data schema holds. Stage B is opened when A is opened (on $+A$). B is an atomic stage holding the service $B.i$. When $B.i$ completes, one of the milestones $B.M1$ to $B.M3$ is activated. The actual milestone depends on the data conditions $c2$ to $c4$. Stage C is opened when either $B.M1$ or $B.M2$ are reached. C completes when its enclosed service completes. D is opened when $C.M1$ is reached and $B.M1$ was reached. Finally, the milestones of A are reached if either $D.M1$ gets active or $C.M1$ gets active and $B.M2$ is active or if $B.M3$ gets active. Therefore, the process permits the following traces of service executions: $<B>$, $<B, C>$, and $<B, C, D>$. The total number of GSM runs is much higher (potentially infinite) since they also cover all potential data values of the data schema.

Adding Temporal Constraints. In GSM, actual work is performed in atomic stages using external services. Therefore, we specify duration intervals indicating the minimum and maximum execution times of atomic stages. In the graphical representation in Figure 1, we denote the min and max execution time of services within atomic stages in square brackets after the stage name: B takes between 4 and 6 time units (TUs), C between 2 to 3, and D between 1 and 2 TUs. In contrast to activity-centric processes where temporal constraints are defined between activities (tasks), we have selected a more flexible approach for GSM+T: Temporal constraints may be defined between events (external events, opening of stages, service invocations or completions, reaching of milestones). Constraints between external events are typically interpreted as descriptive and constraints involving other events as prescriptive.

The process in Figure 1 includes two upper- and two lower-bound constraints. The time between opening A and reaching milestone $A.M1$ is restricted to max. 15 TUs. E.g.: After a test result has been received the treatment of a patient has to be finished in at most 15 TUs. The time to achieve $A.M3$ after opening B is limited to max. 7 TUs. The time between the completion of the service in stage B and starting the service of stage C must be at least 2 TUs. As an example: Within two days after receiving some medication B a patient must not ingest some medication C. The time between opening B and reaching $A.M1$ must be at least 13 TU. Finally, the example process has an overall deadline of 16 TUs.

2.2 Introducing GSM+T

We have already informally introduced temporal constraints in GSM+T models. In this section, we formalize the GSM+T model.

▶ **Definition 2** (GSM+T Process Model). $(A, X, S, M, G, Y, I, C, B^u, B^l, \Omega)$ is a *GSM+T* process model, if A is a set of artifact types, X is a set of events, S is a set of stages, M is a set of milestones, G is a set of guards, Y is a set of sentries, I is a set of external service invocations, and C is a set of service completions. B^u is a set of upper-bound constraints of the form (s, d, δ), B^l is a set of lower-bound constraints (s, d, δ), where $s, d \in X$ and $\delta \in \mathbb{N}$, and the deadline $\Omega \in \mathbb{N}$.

Events. The set of events X includes a distinct event for each milestone, each stage opening, service invocation and completion of each external service and some external events X^E. $S \cup M \cup I \cup C \cup X^E = X$. Each external event x in X^E has the attribute $x.d_{from}$, $x.d_{to}$ indicating the time interval in which the event may occur after starting the process.

Artifact Types. An artifact type is a tuple (Att, S'), where $Att = Att_{status} \cup Att_{data}$, $S' \subseteq S$. Att_{status} is a set of boolean status attributes for all stages in S' and all milestones of stages in S' indicating currently opened stages and reached milestones of artifact instances. Att_{data} is a set of arbitrary data attributes.

Stages. A stage s is a tuple $s = (G, M, sub, sup, I, C, d_{min}, d_{max})$, where $s.G$ is a non-empty set of guards ($s.G \subseteq G$ and $s.G \neq \emptyset$), $s.M$ is a non-empty set of milestones ($s.M \subseteq M$ and $s.M \neq \emptyset$). The attribute $s.sub$ holds the sub-stages of s ($s.sub \subseteq S$). The attribute $s.sup$ refers to the super stage of s or to s for root stages. Either $s.sub$ or $s.I$ are empty, but never both. $s.I$ refers to an external service in I, $s.C$ refers to the completion event of the external service. Invocation and completion events have an attribute $.S$ referring to their atomic stage. $s.I.d_{min}, s.I.d_{max} \in \mathbb{N}$ are the minimum and maximum duration of the external service, with $0 \leq s.I.d_{min} \leq s.I.d_{max}$.

Sentries of Guards and Milestones. Each milestone $m \in M$ has one sentry $m.se \in Y$. Each guard $g \in G$ has one sentry $g.se \in Y$. Each sentry belongs to exactly one artifact type (Att, S'). A sentry is an expression of the form "on event if condition" where "condition" is optional. A sentry se has two components: $se.trig$ is the triggering event $\in X$ and $se.cond$ is a condition. $se.cond$ has the form $p_s \wedge p_d$, where p_s is a conjunction over positive atoms $\in Att_{status}$ that defines the necessary conditions for the sentry to evaluate to $true$, and p_d is an arbitrary query (over elements in Att_{data}) and negative atoms of p_s. $se.ref$ is the set of all (internal) events $\in X$ that are referenced from p_s via their status attributes. As a short-hand, we use $m.trig$ for $m.se.trig$ and $m.ref$ for $m.se.ref$ for milestones $m \in M$ and $g.trig$ for $g.se.trig$ and $g.ref$ for $g.se.ref$ for guards.

Constraints on GSM+T Models. Milestone belong to exactly one stage: $\forall m \in M\ \exists_1 s \in S : m \in s.M$;
Guards belong to exactly one stage: $\forall g \in G\ \exists_1 s \in S : g \in s.G$;

Invocations and completions belong to exactly one stage: $\forall i \in I\ \exists_1 s \in S : i = s.I$; $\forall c \in C\ \exists_1 s \in S : c = s.C$; Completions belong to the same stage as their invocation: $\forall c \in C\ \forall i \in I\ \exists_1 s \in S : i = s.I \wedge c = s.c$; Stages belong to exactly one artifact type: $\forall s \in S\ \exists_1 a \in A : s \in a.S$; There is exactly one root stage per artifact type: $\forall a \in A\ \exists_1 r \in a.S' : r.sup = r$;
All (recursive) sub-stages of an artifact type are closed under the stages of the artifact type: $\forall a \in A\ \forall s \in a.S' : s.sup \in a.S' \wedge s.sub \subseteq a.S'$.

The remainder of this paper focuses on defining the semantics of the GSM+T and on checking the controllability of GSM+T processes with the following restrictions. In contrast to standard GSM, we restrict sentries to expressions containing triggering events and we require that the status condition p_s is a conjunction of positive atoms while p_d can be any arbitrary predicate, the evaluation of which we can only observe. Requiring conjunctions does not limit the generality of our approach since non-conjunctive terms can always be represented by duplicating the corresponding sentries (guards and milestones) to achieve DNF. Sentries with no triggering events can be replaced by sets of sentries for all possible events. Additionally, we require a somewhat more strict *toggle-once principle* requiring that no guard can trigger the re-activation of a stage after it is executed or a milestone is reached. Active milestones remain active. Therefore, we do not support loops of stages. There is currently no approach for temporal constraints of process models with full support of loops in the literature [5]. Loops are either not supported at all or treated as abstract blocks.

2.3 Semantics of GSM+T

We define the semantics of GSM+T as an extension of the semantics of GSM by (1) timestamping all events of a run and (2) define which conditions these timestamps have to adhere to consider a run as temporally correct. We use a finite abstraction of GSM+T to define time constraints, check controllability and calculating schedules. It is inspired by the polarized dependency graph [15] which is used for defining well-formedness of GSM and it is an extension of timed workflow graphs [11] which are used for defining temporal constrains in activity-centric processes. Basically, the nodes of the graph are the events and guards of the process and edges are defined by dependencies between nodes.

▶ **Definition 3** (Temporal Dependency Graph (TDG)). Let $p = (A, X, S, M, G, Y, I, C, B^u, B^l, \Omega)$ be a GSM+T process. (N, E, B^u, B^l, Ω) is a temporal dependency graph of p, iff: $N = X \cup G$,

$$
\begin{aligned}
E = \quad & \{(n,g) | n \in g.ref \cup g.trig, g \in G \cup M\} && \text{// Referenced events to guards and milestones}\\
& \cup \{(s,i) | s \in S, i = s.I\} && \text{// Stages to service invocations}\\
& \cup \{(i,c) | \exists s \in S : i = s.I \wedge c = s.C\} && \text{// Service invocation to service completion}\\
& \cup \{(s,m) | s \in S, m \in s.M\} && \text{// Stages to their milestones}\\
& \cup \{(g,s) | g \in s.G, s \in S\} && \text{// Guards to their stages}\\
& \cup \{(s,g) | s \in S, g \in s.sub.G\} && \text{// Stages to guards of sub-stages}
\end{aligned}
$$

▶ **Example 4.** Figure 2 shows the TDG of the process in Figure 1. The upper- and lower-bound constraints in B^u and B^l are shown as dotted lines (they are not edges of the TDG). Duration intervals of services are denoted in square brackets.

A scenario is a complete run of a process with timestamps for all elements. We now specify the semantics of the temporal constraints by defining which possible execution scenarios are correct. Then we define schedules as a definition of admissible temporal intervals for the nodes in the TDG.

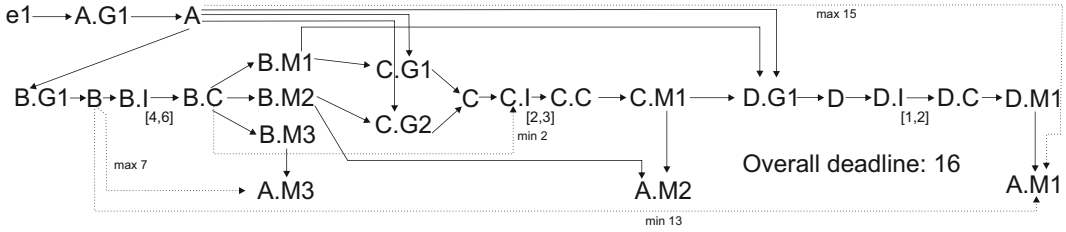


Figure 2 Temporal Dependency Graph of example process in Figure 1.

▶ **Definition 5** (Scenario). A scenario $\bar{S}$ of a GSM+T process assigns timestamps to all elements of a process that occurred during one complete GSM+T run. The TDG ST of a scenario only contains the nodes that occurred during that run. Each node $n \in ST$ is associated with a timestamp $n.t$, the time point of this event in a process instance.

▶ **Definition 6** (Valid Scenario). A scenario $\bar{S}$ of a GSM+T Process (A, X, S, M, G, Y, I, C, B^u, B^l, Ω) with the TDG (N, E, B^u, B^l, Ω), is valid, iff the following constraints hold: $\forall n \in N$, $\forall m \in N$

1. $n \in X^E \Rightarrow n.d_{min} \leq n.t \leq n.d_{max}$
2. $n \in N \wedge (n, m) \in E \Rightarrow n.t \leq m.t$
3. $n \in S \Rightarrow \exists g \in n.G : g.t = n.t$
4. $n = m.trig \Rightarrow m.t = n.t$
5. $n \in S \wedge n.sub = \emptyset \Rightarrow n.I.t + n.I.d_{min} \leq n.C.t \leq n.I.t + n.I.d_{max}$
6. $(s, d, \delta) \in B^u \Rightarrow d.t \leq s.t + \delta$.
7. $(s, d, \delta) \in B^l \Rightarrow s.t + \delta \leq d.t$.

(1) External events occur in their defined intervals. (2) All nodes have equivalent or smaller timestamps than their successors. (3) The guard of a stage has the same time stamp as the stage (In a scenario, there is exactly one guard for each stage.). (4) Nodes with triggering events have the same time point as their triggering event. (5) Service completions of atomic stages occur in their defined intervals. (6-7) upper- and lower-bound constraints are satisfied.

▶ **Example 7.** A fragment (ignoring nodes with equivalent time-stamps) of a valid scenario for the process in Figure 1 is the following: $B.I.t = 0$, $B.I.C.t = 6$, $C.I.t = 8$, $C.I.C.t = 11$, $D.I.t = 12$, $D.I.C.t = 13$. Therefore, $B.I$ directly starts and $B.I.C$ arrives at time 6. $C.I$ is delayed by 2 to obey $lbc(B.I.C, D.I, 2)$ resulting in $C.I.t = 8$. Then $C.I.C$ arrives at time-point 11. $D.I$ is started at time 12 to obey $lbc(B, A.M1, 13)$ and returns at time 13. The scenario complies with all constraints.

2.4 Controllability

Based on these definitions we define the property *controllability* as a notion of the correctness of a process definition with temporal constraints. Controllability reflects the ability of a GSM+T engine to control the execution of GSM+T processes in a way to eventually satisfy all temporal constraints. To define controllability, we have to distinguish four types of events: (1) invocation of a service (task), (2) completion of a service (task), (3) opening/closing of a stage, and (4) achieving a milestone. Only (1) is controlled by the engine, as the engine could delay the invocation of services. (2) depends on external services and (3) and (4) are controlled by conditions and thus automatic. For example, Barcelona [15] is an eager GSM engine, where services are invoked as soon as their enclosing atomic stages are opened. So the only control option for GSM+T execution to satisfy all temporal constraints is the delay

of service invocations. Why can such delays be necessary? For an example, if two concurrent stages have to reach their milestones at the same time, then this can only be ensured, if the invocation of the service with shorter duration is delayed.

For simplicity, we assume that each process instance starts at time 0 and all timestamps are relative to the instance start time. The time point of a stage is the time point of opening a stage, thus the time-point of its (activated) guard. The time-point of a guard or milestone is the time-point of the triggering event of its sentry. This is in-line with the GSM semantics [9], where B-Steps/Snapshots have timestamps and micro-steps (triggered by internal events) are assumed to happen at the time of processing the external event triggering the B-Step.

▶ **Definition 8** (Controllability). A GSM+T Process is controllable, iff there exists a mapping T (schedule) which assigns each service invocation $i \in I$ with an execution time $t_i \in \mathbb{N}$ such that for all possible timestamps for external events and for all possible service durations and for all possible guard validations the resulting scenario is valid.

Controllability [21] is a strong condition. More relaxed notions of correctness (e.g. dynamic controllability [8, 16, 21]) would, when adopted to GSM+T, allow the service invocation decisions to depend on already completed snapshots.

3 Checking Controllability and Calculating Schedules

3.1 Schedule Frames and Controllability

In the following we present a procedure for checking the controllability of GSM+T processes by constructing a schedule. Controllability requires that *any* correct schedule exists. We construct schedules which take all the worst cases of durations into account. Faster execution of the process might be possible, as we discuss in Section 4. For computing schedules, we first define a schedule frame of a process as an extension of a TDG:

▶ **Definition 9** (Schedule Frame). A schedule frame associates each $n \in N$ of a TDG (N, E, B^u, B^l, Ω) of a GSM+T process (A, X, S, M, G, Y, I, C, B^u, B^l, Ω) with intervals for the occurrence of nodes: E_b, E_w, L. $n.E_b$ expresses the earliest best time in which n can happen, $n.E_w$ expresses the earliest worst time in which n can happen, $n.L$ is the latest time where n may occur. A schedule frame is correct, iff $\forall n \in N, \forall m \in N$:

1. $n \in X^E \Rightarrow n.E_b = n.d_{min}, n.E_w = n.d_{max}$
2. $n.E_b \leq n.E_w \leq n.L$,
3. $(n,m) \in E \Rightarrow n.E_b \leq m.E_b, n.E_w \leq m.E_w, n.L \leq m.L$
4. $n \in S \wedge n.sub = \emptyset \Rightarrow n.I.E_w + n.I.d_{max} \leq n.C.E_w$
5. $n \in S \wedge n.sub = \emptyset \Rightarrow n.I.E_b + n.I.d_{min} \leq n.C.E_b$
6. $n \in S \wedge n.sub = \emptyset \Rightarrow n.I.L + n.I.d_{max} \leq n.C.L$
7. $n \in S \wedge n.sub = \emptyset \Rightarrow n.E_w \leq n.I.E_b$
8. $n \in G \cup M \Rightarrow n.E_b = n.trig.E_b, n.E_w = n.trig.E_w, n.L = n.trig.L$
9. $\forall (m, m', \delta) \in P.B^u : m'.E_w \leq m.E_b + \delta$,
10. $\forall (m, m', \delta) \in P.B^u : m'.L \leq m.L + \delta$
11. $\forall (m, m', \delta) \in P.B^l : m.E_b + \delta \leq m'.E_w$,
12. $\forall (m, m', \delta) \in P.B^l : m.L + \delta \leq m'.L$
13. $n \in S \Rightarrow n.E_b = min(\{g.E_b | g \in n.G\})$,
    $n \in S \Rightarrow n.E_w = max(\{g.E_w | g \in n.G\})$

(1) The time interval of external events is their defined time interval. (2) Invariant: Earliest best smaller or equal earliest worst smaller or equal latest. (3) Nodes have equivalent or smaller E_b, E_w and L values than their successors. (4-6) The time difference between invocation and completion is defined by the duration interval of the service. (7) The time of an atomic stage is smaller or equal to the time of its service invocation. (8) Guards and milestones have the E_b, E_w, and L values of their triggering events. (9-12) Upper- and lower-bound constraints are satisfied. (13) The earliest best opening time of a stage is the earliest time of all their guards. The earliest worst opening time is the earliest worst time of all their guards.

Example of a schedule frame: The essential parts of a schedule frame - the ones controllable by the engine - for the GSM+T process in Figure 1 in the format (E_b, E_w, L) are the following: *B.I: (0,0,3), C.I: (8,8,11), D.I: (12,12,14).* The complete schedule frame is shown in the *Backward* 2 column of Table 1.

▶ **Theorem 10.** A GSM+T process P is controllable, iff it has a correct schedule frame.

Proof Sketch. (1) soundness: Let SF be a correct schedule frame for P, let $T(i) = i.E_b$ be a schedule. Following from Def. 9 and Definitions 6, 8 we can show that T fulfills the requirements of controllability.
(2) completeness: Let T be a schedule such that P is controllable. We construct a schedule frame as follows:

1. $\forall i \in I : i.E_b = i.E_w = T(i)$.
2. $\forall s \in S : s.C.E_b = s.I.E_b + d_{min}, s.C.E_w = s.I.E_w + d_{max}$
3. $\forall x \in X^E : x.E_b = x.d_{min}, x.E_w = x.d_{max}$
4. $\forall n \in N - I - C - X^E :$
   $n.E_b = max\{m.E_b | (n,m) \in E\}$,
   $n.E_w = max\{m.E_w | (n,m) \in E\}$
5. $\forall n \in N : n.L = n.E_w$

With some calculations, it is easy to see that this schedule frame is correct. ◀

▶ **Lemma 11** (Cyclic Dependencies)**.** *If the schedule frame graph contains a cycle the process is not controllable.*

Proof. We distinguish 2 cases: (a) there is an (s, i) edge involved in the cycle. Let S_1 and S_2 be stages contained in a cycle in the dependency graph. This means that both sequences $< S_1, S_2 >$ and $< S_2, S_1 >$ are possible. Hence it is not possible to assign start times to the stages such that both sequences are admissible unless the stages have duration 0.
(b) If there is no (s, i) edge in the cycle, then the process violates the toggle once principle of well-formed GSM. ◀

3.2 Computing Correct Schedule Frames:

We compute a correct schedule frame for an acyclic schedule frame graph of a process P as follows: We initialize the schedule frame as follows: For external event nodes we set the E_b and E_w values to the time intervals of the corresponding events: $(x \in X^E) \rightarrow x.E_b = x.d_{min}, x.E_w = x.d_{max}$. The E-values of all other nodes are set to 0. The L-values are set to the deadline of the whole process ($P.\Omega$). (Computing a schedule is also possible if no overall deadline is given, but the algorithm is considerably longer and out of scope of this paper.)

The events may take place during the interval defined by (E_b, L) of its node. In the course of Algorithm 1 these intervals are now reduced iteratively through incorporating the implicit (an event can only take place after its predecessors) and explicit temporal constraints.

Algorithm 1 CSF(sf,P) Compute correct schedule frame sf for process P.

```
1: {inout: (N, E, B^u, B^l, d) schedule frame, in: (A, X, S, M, G, Y, I, C, B^u, B^l, Ω) Process}
2: {(N,E) is acyclic}
3: {Output: correct schedule frame or ok is false}
4: ok := false
5: while ¬ok do
6:    ok := true
7:    if !Forward((N, E, B^u, B^l, d), (A, X, S, M, G, Y, I, C, B^u, B^l, Ω), ok) then
8:       return false
9:    else if !Backward((N, E, B^u, B^l, d), (A, X, S, M, G, Y, I, C, B^u, B^l, Ω), ok)) then
10:      return false
11:   else
12:      for all (s, d, δ) ∈ B^u do
13:         {incorporation of upper-bound constraints}
14:         if δ < (d.E_w − s.E_b) then
15:            ok := false
16:            s.E_b := max(s.E_b, d.E_w − δ)
17:            s.E_w := max(s.E_w, s.E_b)
18:         end if
19:         if δ < (d.L − s.L) then
20:            ok := false
21:            d.L := min(d.L, s.L + δ)
22:         end if
23:      end for
24:      for all (s, d, δ) ∈ B^u do
25:         {incorporation of lower-bound constraints}
26:         if δ > (d.E_b − s.E_w) then
27:            ok := false
28:            d.E_b := max(d.E_b, s.E_w + δ)
29:            d.E_w := max(d.E_w, d.E_b)
30:         end if
31:         if δ > (d.L − s.L) then
32:            ok := false
33:            d.L := min(d.L, s.L + δ)
34:         end if
35:      end for
36:   end if
37: end while
38: return true
```

If a constraint is violated, the E_b and E_w values of nodes are increased and the L values are decreased to the lowest resp. highest value satisfying the constraint. The procedure is repeated until either all constraints are satisfied or the invariant $E_w \leq L$ is violated for any node.

Forward Calculation. Algorithm 2 visits all nodes n in a topological sort order and computes the values $n.E_b$ and $n.E_w$ form the corresponding values of its predecessors according to the definition of correct schedule frames. The calculation depends on the type of the current node:

For completion event nodes, there exists exactly one predecessor, the corresponding service invocation node. The E_b and E_w values are defined by the maximum of the current E_b and E_w values and the E_b and E_w of the service invocation node plus the min/max duration times of the service.

Algorithm 2 Forward(sf,P,ok) Forward calculation.

```
1: {inout: (N, E, B^u, B^l, Ω) schedule frame, in: (A, X, S, M, G, Y, I, C, B^u, B^l, Ω) Process, inout:
   ok Boolean}
2: {(N,E) is acyclic}
3: for all n ∈ N − X^E in a topological order do
4:    {forward calculation}
5:    if n ∈ C then
6:       n.E_b := max({n.E_b, n.I.E_b + n.I.d_min})
7:       n.E_w := max({n.E_w, n.I.E_w + n.I.d_max})
8:    else if n ∈ S then
9:       n.E_b := max(n.E_b, {m.E_b|(m, n) ∈ E, m ∉ G}, min{m.E_b|(m, n) ∈ E, m ∈ G})
10:      n.E_w := max(n.E_w, {m.E_w|(m, n) ∈ E})
11:   else if n ∈ I then
12:      n.E_b := max(n.E_b, {m.E_w|(m, n) ∈ E}
13:      n.E_w := max(n.E_w, {m.E_w|(m, n) ∈ E}
14:   else
15:      n.E_b := max(n.E_b, {m.E_b|(m, n) ∈ E})
16:      n.E_w := max(n.E_w, {m.E_w|(m, n) ∈ E})
17:   end if
18:   if n ∈ I then
19:      n.E_b := n.E_w
20:   end if
21:   if n ∈ G ∪ M ∧ (n.L ≠ n.trig.L ∨ n.L ≠ n.trig.L) then
22:      ok := false
23:      n.L := n.trig.L
24:      if n.L < n.E_w ∨ n ∈ X^E ∧ (n.E_b ≠ n.d_min ∨ n.E_w ≠ d_max then
25:         return false
26:      end if
27:   end if
28: end for
29: return true
```

Stage nodes may have multiple predecessors in form of multiple guards and optionally a parent stage. The E_b values of a stage is set to the max of its current E_b value and the E_b value of its parent stage and the minimum E_b value of all guards. I.e. a stage is opened when at least one guard triggers and the parent stage is open. The E_w value is set to the max of the current E_w value and the max E_w of all predecessor nodes. Invocation nodes have exactly one predecessor (a stage). Since we only need to find a controllability schedule, the E_b and E_w value of invocation nodes are set to the maximum of the current value and the E_w value of the stage. The E_b and E_w values of all other node types are defined by the maximum of the current values and the max corresponding values of predecessor nodes.

Algorithm 2 also calculates the value $n.L$ for triggered events, as they must be the same as that of the preceding triggering event. If an update of an L value is required the boolean variable *ok* is set to false which triggers another iteration of forward and backward calculations in Algorithm 1. Algorithm 2 also checks for violations of the $n.E_w \leq n.L$ invariant and returns false if it is violated. A return value of false means that no correct schedule frame exists.

Backward Calculation. Algorithm 3 visits all nodes n in a reverse topological sort order and computes the values $n.L$ from the corresponding values of its successor nodes according to the definition of correct schedule frames. It also calculates the value $n.E_b$, $n.E_w$ for

Algorithm 3 Backward(sf,P,ok) Backward calculation.

1: {inout: (N, E, B^u, B^l, Ω) schedule frame, in: $(A, X, S, M, G, Y, I, C, B^u, B^l, \Omega)$ Process, inout: ok Boolean}
2: **for all** $n \in N$ in a reverse topological order **do**
3: {backward calculation}
4: **if** $n \in N - I$ **then**
5: $n.L := min(\{n.L\} \cup \{s.L | (n, s) \in E\} \cup \{m.L - \delta | (n, m, \delta) \in B^l\}$
6: **else**
7: {invocation node}
8: $n.L := min(\{n.L\} \cup \{s.L - s.I.d_{max} | (n, s) \in E\} \cup \{m.L - \delta | (n, m, \delta) \in B^l\})$
9: **end if**
10: **if** $n \in G \cup M \wedge (n.E_b > n.trig.E_b \vee n.E_w > n.trig.E_w)$ **then**
11: {Move wait times along triggering edges}
12: $ok := false$
13: $n.trig.E_b := n.E_b$
14: $n.trig.E_w := n.E_w$
15: **else if** $n \in I \wedge (n.E_b < n.S.C.E_b - n.d_{min} \vee n.E_w < n.S.C.E_w - n.d_{max})$ **then**
16: {Move wait times from completion to invocation}
17: $ok := false$
18: $n.E_b := n.S.C.E_b - n.d_{min}$
19: $n.E_w := max(n.E_b, n.E_w)$
20: **else if** $n \in S$ **then**
21: {Move wait times from stage to guards}
22: **for all** $e \in \{(g, n) \in E | g \in G \wedge g.E_b < n.E_b)\}$ **do**
23: $ok := false$
24: $d = e.g.E_w - e.g.E_b$
25: $e.g.E_b := n.E_b$
26: $e.g.E_w := n.E_b + d$
27: **end for**
28: **end if**
29: **if** $n.L < n.E_w \vee n \in X^E \wedge (n.E_b \neq n.d_{min} \vee n.E_w \neq d_{max})$ **then**
30: **return** false
31: **end if**
32: **end for**
33: **return** true

triggering events as they have to be the same as that of the triggered events. Following the same principle, it increases the $n.E_b$, $n.E_w$ values for invocation nodes and of guards, when required. This basically shifts delays to the corresponding invocations as they are the only times that can be controlled. If any shifting was required the status variable *ok* is set to false. This will later trigger another round of forward and backward calculations in Algorithm 1. During backward calculation, Algorithm 3 also checks for violations of the $E_w \leq L$ invariant and whether the admissible interval for external events had been reduced. It returns false, if any of these invariants is violated. In this case, no correct schedule frame exists and Algorithm 1 also terminates.

▶ **Example 12.** The complete calculation of the schedule frame for the example process in Figure 1 based on the TDG in Figure 2 is shown in Table 1. The result is obtained after two rounds of forward and backward calculations. All data is given in the format (E_b, E_w, L). After the first forward and backward cycle (columns Forward 1, Backward 1), the ubc and lbc constraints are enforced. The ubc $(B, A.M3, 7)$ is violated by $B = (0, 0, 3)$ and $A.M3 = (4, 6, 16)$. In particular, $A.M3.L > B.L + 7$. Therefore, $A.L$ is set to 10.

Table 1 Computing schedule frame for Figure 2.

	Forward 1				Backward 1		
Node	E_b	E_w	L		E_b	E_w	L
e1	0	0	16		0	0	3
A.G1	0	0	16		0	0	3
A	0	0	16		0	0	3
B.G1	0	0	16		0	0	3
B	0	0	16		0	0	3
B.I	0	0	16		0	0	3
B.C	4	6	16		4	6	9
B.M1	4	6	16		4	6	11
B.M2	4	6	16		4	6	11
B.M3	4	6	16		4	6	11
A.M3	4	6	16		4	6	16
C.G1	4	6	16		4	6	11
C.G2	4	6	16		4	6	11
C	4	6	16		4	6	11
C.I	6	6	16		6	6	11
C.C	8	9	16		8	9	14
C.M1	8	9	16		8	9	14
A.M2	8	9	16		8	9	16
D.G1	8	9	16		8	9	14
D	8	9	16		8	9	14
D.I	9	9	16		9	9	14
D.C	10	11	16		10	11	16
D.M1	10	11	16		10	11	16
A.M1	10	11	16		10	11	16

	Forward 2				Backward 2		
Node	E_b	E_w	L		E_b	E_w	L
e1	0	0	3		0	0	3
A.G1	0	0	3		0	0	3
A	0	0	3		0	0	3
B.G1	0	0	3		0	0	3
B	0	0	3		0	0	3
B.I	0	0	3		**0**	**0**	3
B.C	4	6	9		4	6	9
B.M1	4	6	9		4	6	9
B.M2	4	6	9		4	6	9
B.M3	4	6	9		4	6	9
A.M3	4	6	9		4	6	9
C.G1	4	6	9		4	6	9
C.G2	4	6	9		4	6	9
C	4	6	11		4	6	11
C.I	8	8	11		**8**	**8**	11
C.C	10	11	14		10	11	14
C.M1	10	11	14		10	11	14
A.M2	10	11	14		10	11	14
D.G1	10	11	14		10	11	14
D	10	11	14		10	11	14
D.I	11	11	14		**12**	**12**	14
D.C	12	13	16		13	13	16
D.M1	12	13	16		13	13	16
A.M1	13	13	16		13	13	16

The ubc $(A, M1, 15)$ is not violated. The *lbc* constraints $(B, A.M1, 13)$ and $(B.C, C.I)$ are violated. Therefore, the values for $C.I$ and $A.M1$ are updated to $(8, 8, 11)$ and *(13, 13, 16)*, respectively. The change of $A.M1$ is populated to $D.I$ in the following backward calculation phase. One additional iteration of forward and backward calculations is executed without changing values. No additional violations exists and the algorithm terminates. The final results are shown in the column *Backward* 2. A schedule is obtained by only using the E_b values of invocation nodes: *B.I.t: 0, C.I.t = 8, D.I.t = 12.*

3.3 Correctness, Completeness, Complexity, and Feasibility

▶ **Theorem 13.** *A GSM+T process P is controllable iff P is acyclic and Algorithm 1 computes a correct schedule frame.*

Proof. The algorithm terminates always due to the monotonicity of increasing E_b and E_w and decreasing L. Following Theorem 10 we have to show that it computes a correct schedule frame, if there exists a correct schedule frame.
Soundness: All conditions for a correct schedule frame are either explicitly checked, or are immediate result of the calculations and assignments.
Completeness: For all $n \in N$ the values for E_b, E_w, L have to be in the interval $[0, \Omega]$, the variables are initialized accordingly. In the course of the algorithm E_b and E_w can only be increased, L can only be decreased. A change of the value of one of these variables is only done, when a constraint, a condition of a correct schedule frame is violated. Then the variables are set to the lowest resp. highest value which satisfies this constraint. With this monotonicity, we ensure that either a correct set of values is computed or the $E_w \leq L$ invariant is violated or the interval of external events is decreased. ◀

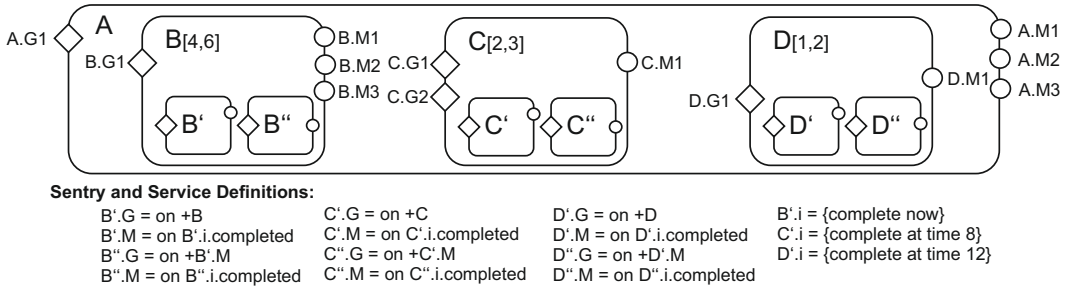


Figure 3 Standard GSM translation of GSM+T example in Figure 1.

The worst case complexity of the algorithm is $O(n^2)$ where n is the number of nodes in the temporal dependency graph plus the number of explicit constraints. One iteration has n steps and the number of iterations is at least limited by n times the deadline. Preliminary data of explorative experiments with implementations of the basic algorithms, have shown that for reasonably sized models the run-time of the algorithm is within seconds - feasible for design time checks.

4 Execution of GSM+T Processes

In this paper we focus on checking controllability and not on computing optimal schedules nor on the run-time of GSM+T processes. Nevertheless, we discuss in this section how controllable GSM+T processes can be correctly executed to show the feasibility of the GSM+T approach. There are basically two options which rewrite GSM+T processes to be able to run the process on any GSM execution engine.

(1) Fixed execution. Fixed execution has the advantage that there is no need for calculations of time values during the run-time of the process and all invocation times of services are fixed at the process start time. We replace every atomic stage ds by a composite stage ds containing a sequence of two sub-stages ds', which implements a parameterized wait and ds which invokes the requested service. The guards and milestones of ds remain the same but we nest an atomic stage ds' into ds such that ds' has a single guard with the sentry *on* $+ds$. Therefore, it is triggered when ds is opened. The nested stage ds' contains a service call to a special waiting service. The service completion event of $waitDS$ is delayed until the specified time (relative to process start) has passed. Stage ds' has one single milestone $ds'.M$ with the sentry on *ds'.i.completed*. We nest an additional stage ds'' into ds with the guard *on* $+ds'.M$. The service call of ds'' is the service call of the original stage ($ds.impl$). ds'' has a single milestone with the sentry: *on ds".impl.completed*. The Milestones of ds are not touched. Therefore, they are still connected to the service completion of ds''.

The mapping of the GSM+T model in Figure 1 to a standard GSM Model is shown in Figure 3. According to the schedule (Example 12), the stages B, C, D are replaced by composite stages with wait parameters 0,8, and 12.

(2) Flexible execution. Flexible execution may use the same rewritten process as fixed execution. However, the (possible) delay is computed at run-time by recomputing the schedule frame using the actual values of already executed B-steps and the information on which goals evaluated to true and which milestones were reached. Flexible execution allows for faster execution of the process, but requires significantly more effort at run-time.

5 Related Work

To the best of our knowledge, this is the first approach to consider temporal constraints in artifact-centric process models. For activity-centric process models there is already quite some body of research results in the area of temporal aspects of workflows and business processes. We refer to [12, 4] for overviews. Early approaches for checking process definitions with temporal constraints of activity-centric processes were proposed in works such as [19, 2, 11]. The approaches are based on different techniques such as network analysis, scheduling, or temporal constraint networks [10]. The stream of research based on temporal constraint networks now reached the necessary expressiveness for analyzing business process models: checking the controllability and dynamic controllability of temporal networks, as discussed in [8]. In particular, [16, 6] present sound-and-complete algorithms for checking the dynamic consistency resp. controllability of conditional simple temporal networks with uncertainty.

Works addressing time constrains for interorganizational processes and for service composition were presented in [3, 14]. A quite different approach was proposed in [20], where process mining is used to asses temporal qualities of process models from process logs, but cannot check temporal qualities of process models at build time. Recently the work of Lanz et al. [18] contributed to the consolidation of expressing temporal constraints and defining the semantics of temporal constraints. However, all these works address activity-centric process models and provide no solution for time aware GSM process models.

Our approach is inspired from the algorithms for computing schedules for time constrained workflows presented in [11], but is based on different much more complex type of graph defined to represent the temporal dependencies of GSM+T events and considerably extended to capture the uncertainty of the duration of contingent service calls.

6 Conclusions

Current artifact-centric modeling approaches lack support for temporal constraints. We have extended GSM with deadlines, upper- and lower-bound constraints and intervals defining the potential durations of services. In contrast to activity-centric models, time constraints cannot only be defined between start- and end-times of services but also between (composite) stages, guards and milestones. We have defined controllability of GSM+T models and have provided sound and complete algorithms for checking controllability and calculating schedules. We have provided a method to rewrite controllable GSM+T models to standard GSM models that obey all time constraints when executed on a standard GSM engine (e.g. Barcelona). We see this work as an important foundation for future works including investigating dynamic controllability, where the engine can make different scheduling decisions based on past snapshots and for investigating processes with time dependent sentries.

References

1 Serge Abiteboul, Omar Benjelloun, and Tova Milo. The active xml project: An overview. *The VLDB Journal*, 17(5):1019–1040, 2008. doi:10.1007/s00778-007-0049-y.

2 Claudio Bettini, X Sean Wang, and Sushil Jajodia. Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 11(3):269–306, 2002.

3 J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *J. of Web Semantics*, 1(3):281–308, 2004.

4 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.

5 Margareta Ciglic. Time management in workflows with loops. In *Proc. of OTM 2015 Workshops*, pages 5–9, 2015. doi:10.1007/978-3-319-26138-6_2.

6 Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Dynamic controllability via timed game automata. *Acta Informatica*, pages 1–42, 2016.

7 C. Combi, M. Gozzi, R. Posenato, and G. Pozzi. Conceptual modeling of flexible temporal workflows. *ACM TAAS*, 7(2), 2012. doi:10.1145/2240166.2240169.

8 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.

9 E. Damaggio, R. Hull, and R. Vaculin. On the equivalence of incremental and fixpoint semantics for business artifacts with Guard-Stage-Milestone lifecycles. *Information Systems*, 38:561–584, 2013.

10 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3), 1991.

11 Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In *Advanced information systems engineering*. Springer, 1999.

12 Johann Eder, Euthimios Panagos, and Michael Rabinovich. Workflow time management revisited. In *Seminal Contributions to Information Systems Engineering*. Springer, 2013.

13 Cagdas E. Gerede and Jianwen Su. Specification and verification of artifact behaviors in business process models. In Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors, *ICSOC 2007*, pages 181–192. Springer, 2007.

14 N. Guermouche and C. Godart. Timed model checking based approach for web services analysis. In *ICWS 2009. IEEE*, pages 213–221. IEEE, 2009.

15 R. Hull, E. Damaggio, R. De Masellis, et al. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proc. of DEBS 2011*, pages 51–62. ACM, 2011.

16 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 *Temporal Representation and Reasoning (TIME)*. IEEE, 2015.

17 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*. Springer, 2013.

18 Andreas Lanz, Manfred Reichert, and Barbara Weber. Process time patterns: A formal foundation. *Information Systems*, 57, 2016.

19 O. Marjanovic and M.E. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge & Inform. Syst.*, 1(2):157–192, 1999.

20 W.M.P. van der Aalst, M.H. Schonenberg, and M. Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011.

21 T. Vidal and H. Fargier. Contingent durations in temporal csps: From consistency to controllabilities. In *Proc. TIME '97*. IEEE Computer Society, 1997.

Learning Qualitative Constraint Networks

Malek Mouhoub[1]
University of Regina 3737 Wascana Parkway, Regina SK S4V 0A2, Canada
mouhoubm@uregina.ca
https://orcid.org/0000-0001-7381-1064

Hamad Al Marri
University of Regina 3737 Wascana Parkway, Regina SK S4V 0A2, Canada
almarrih@uregina.ca

Eisa Alanazi
Um Al Qura University Aif Road, 21955, Mecca, Makkah Province, Saudi Arabia
eaanazi@uqu.edu.sa

Abstract

Temporal and spatial reasoning is a fundamental task in artificial intelligence and its related areas including scheduling, planning and Geographic Information Systems (GIS). In these applications, we often deal with incomplete and qualitative information. In this regard, the symbolic representation of time and space using Qualitative Constraint Networks (QCNs) is therefore substantial.

We propose a new algorithm for learning a QCN from a non expert. The learning process includes different cases where querying the user is an essential task. Here, membership queries are asked in order to elicit temporal or spatial relationships between pairs of temporal or spatial entities. During this acquisition process, constraint propagation through Path Consistency (PC) is performed in order to reduce the number of membership queries needed to reach the target QCN. We use the learning theory machinery to prove some limits on learning path consistent QCNs from queries. The time performances of our algorithm have been experimentally evaluated using different scenarios.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning

Keywords and phrases Temporal Reasoning, Qualitative Constraint Network (QCN), Constraint Learning, Path Consistency, Constraint Propagation

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.19

1 Introduction

Reasoning with time and space is a fundamental task in artificial intelligence and its related areas including scheduling, planning, Geographic Information Systems (GIS) and computational linguistics. Handling the qualitative aspects of time and space is an important matter when designing such systems especially when dealing with incomplete information. Several research works have therefore been proposed in order to represent and reason on symbolic temporal and spatial information. One of the most known approaches is the Allen Algebra [1], based on the notion of time intervals and binary relations on them. A time interval I is an ordered pair (I^-, I^+) such that $I^- < I^+$, where I^- and I^+ are points on the time line. There are thirteen basic relations (Allen primitives) that can hold between intervals. A binary relation between two intervals is represented by the disjunction of some

[1] Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN-2016-05673

Allen primitives and expresses the incompleteness of the temporal information. Any problem under temporal constraints can be converted into an Interval Algebra (IA) network (also called qualitative temporal network) where nodes correspond to intervals and each arc represents the binary relation between the corresponding intervals. Note that an IA network is a particular case of a Qualitative Constraint Networks (QCN) which is a general network for representing problems under qualitative temporal or spatial constraints. Given an IA network, the main reasoning task is to decide its consistency and return one or more solutions (consistent scenarios satisfying all the temporal constraints) if it is the case. These tasks can be achieved with a backtrack search algorithm enhanced with constraint propagation techniques.

One of the challenges when dealing with problems under temporal or spatial constraints is the modelling task. This latter requires some expertise in temporal and spatial representation and reasoning, such as a good knowledge of the Allen Algebra. In this regard, we propose a new algorithm for learning a QCN network from a non expert with a focus on the IA network. The learning process includes different cases where querying the user is an essential task. Here, membership queries are asked in order to elicit the temporal or spatial relationships between pairs of temporal or spatial entities. During this acquisition process, constraint propagation is performed in order to reduce the number of membership queries needed to reach the target QCN. The time performance of our algorithm has been experimentally evaluated using different randomly generated scenarios. Note that, while the focus of this paper is on temporal constraints, our proposed algorithm can be easily generalized to any QCN.

2 Qualitative Constraint Networks (QCNs)

A Qualitative Constraint Network (QCN) is a pair (V, C) in which V is a finite set of variables representing temporal or spacial entities and C is a finite set of constraints on these variables. Each constraint c_i is expressed as a disjunction of binary relations between the involved variables. Each of these relations is defined on a language set $\mathcal{B} = \{b_1, b_2, \ldots, b_p\}$ where $p > 0$. A particular case of such networks is the IA network [1] where V is a set of temporal events, each representing an assertion over a time interval; and $\mathcal{B}$ is the set of Allen primitives depicted in Table 1 [2]. For instance, let us consider two temporal events, E_1 and E_2. The following constraint expresses the fact that both events are mutually exclusive: E_1 $(B \vee Bi)$ E_2. Note that a universal relation, corresponding to the disjunction of all relations within $\mathcal{B}$ (I, representing the 13 Allen primitives in the case of the Interval Algebra), is used to express the fact that there is no constraint between the involved entities (the knowledge on such constraint is completely unknown).

Given a QCN, the main reasoning task that can be performed consists of checking the consistency of the network and, if it is the case, returns one or more consistent scenarios satisfying all the constraints of the related problem. This task can be done using a backtrack search algorithm. Constraint propagation, before and during search, can be enforced in order to prevent late failure earlier. The most known constraint propagation technique used in QCN networks and especially in IA networks is Path Consistency (PC) [8, 14, 15, 20]. This technique (also called transitive closure) applies local consistency on every subset of three variables which results in removing some inconsistent relations from the network. This

[2] Note that, six of the seven primitives listed have inverse relations. This will bring the total of number of possible relations to 13.

Table 1 Allen's primitives.

Relation	Symbol	Inverse	Meaning
X precedes Y	B	Bi	XXXX YYYY
X equals Y	E	E	XXXX YYYY
X meets Y	M	Mi	XXXXYYYY
X overlaps Y	O	Oi	XXXX YYYY
X during Y	D	Di	XXXX YYYYYYYY
X starts Y	S	Si	XXXX YYYYYYYY
X finishes Y	F	Fi	XXXXX YYYYYYYYY

will lead to the reduction on the search space. Assuming X, Y and Z are three temporal events, if X precedes Y and Y meets Z, then it must be that X precedes Z as well. Any other relation belonging to the constraint between X and Z should be removed. Local consistency is enforced using a $|\mathcal{B}|^2$ composition table (13 x 13 composition table between Allen's primitives in the case of IA networks). Note that PC can also infer new relations by refining universal relations into a more specific one. For instance, if we take the previous example and assume that there is no constraint between Y and Z then PC will infer a new relation (precedes) between these two events.

Partial Path Consistency (PPC) is an extension of PC based on chordal graphs. A chordal graph is a graph such that, for every connected four vertexes and higher there must be one chordal edge [8]. A chordal graph is also called triangulated graph since the graph can be printed in a map in which the vertices are connected in triangle shapes. Partial Path Consistency assures that the chordal graph is consistent such as when applying regular PC [8]. Partial Path Consistency is much efficient than PC when working with chordal graphs. It is useful in case there are universal edges in the graph that can be removed to form a chordal graph.

3 Constraint Acquisition

Constraint acquisition is the process of inducing, from a set of examples, a constraint network representing a given problem. The main goal of constraint acquisition algorithms is to minimize the number of examples needed to converge to a target constraint network. In this regard, several strategies have been explored and a passive learning algorithm (CONACQ.1 or the passive version of the CONACQ acquisition system [7]) using unit propagation to speed up the convergence test has been proposed in [5]. In [6], a proposed active constraint acquisition technique, called QUACQ, works by asking the user to classify partial queries (those queries not involving all variables) in addition to membership queries (where the user is asked to answer "yes" or "no" depending on whether the provided assignment is a solution or not). Partial queries are asked when given a negative example and allow QUACQ to converge in a number of queries that is logarithmic in the number of variables. An extension to this latter algorithm has been proposed in [3] where multiple constraints are learned (and not only one as in QUACQ) using Minimal Unsatisfiability Subsets (MUS).

In [7], the active version of the CONACQ architecture, called CONACQ.2, is proposed as shown in Algorithm 1. CONACQ.2 starts with a bias B (similar to our language set $\mathcal{B}$) and its corresponding background language K. K is a set of declarative rules (definite

Algorithm 1 CONACQ.2 [7].

1: **procedure** CONACQ.2
2: **Input:** a bias B, a background knowledge K, a strategy $Strategy$
3: **Output:** a clausal theory T encoding the target network
4: $T \leftarrow \emptyset$; $converged \leftarrow false$; $N \leftarrow \emptyset$
5: **while** $\neg converged$ **do**
6: $q \leftarrow QueryGeneration(B, T, K, N, Strategy)$
7: **if** $q = nil$ **then**
8: $converged \leftarrow true$
9: **else**
10: **if** $Ask(q) = no$ **then**
11: $T \leftarrow T \wedge (\bigvee_{c \in K(q)} a(c))$
12: **else**
13: $T \leftarrow T \wedge \bigwedge_{c \in K(q)} \neg a(c)$
14: **end if**
15: **end if**
16: **end while**
return T
17: **end procedure**

Horn clauses) expressing some properties (such as the transitivity property) that can be used to enforce local consistency between constraints. In the case of QCNs, the set K basically corresponds to the composition table we mentioned in the previous section. Given these two inputs B and K, in addition to a given strategy to follow for generating the queries, the algorithm iterates by asking the user a new generated query at each time and expands the theory set T (initially set to the empty set) according to the answer provided. More precisely, if the user answers "no" to a given query q then the algorithm removes all the concepts that support this query (as shown in line 11 of the Algorithm). In case the answer is "yes" then all the concepts rejecting q must be removed (as per line 13 of the Algorithm). The algorithm terminates when there is no query to generate. The target network, in the form of the clausal theory T is then returned.

4 Proposed Learning Algorithm for QCNs

In this section, we present an algorithm for learning QCNs in the particular case of AI networks. Note that our algorithm can be easily generalized to any QCN. We consider the following three cases, each with a different target QCN.

1. Learning consistent and complete scenarios: the target QCN is a complete graph where each edge (constraint) contains exactly 1 relation (Allen primitive in the case of IA networks).
2. Learning consistent but incomplete scenarios: same as case 1 but in the case of incomplete graphs (some constraints do not exist between pairs of variables).
3. Learning QCNs: the target QCN is a graph (can be complete) where each edge contains 1 or more primitives (each constraint has one or more relations). Note that, in this particular case we are learning a QCN problem rather than a consistent scenario (solution) as it is in cases 1 and 2.

Following CONACQ.2 [7], our algorithm learns through membership queries and uses PC to reduce the number of queries. Note that when using PC for case 3, our learning algorithm will return a path consistent QCN problem.

4.1 Proposed Learning Algorithm

The algorithm starts with all the constraints completely unknown (corresponding to the universal relation I in the case of IA networks). The user is then asked a membership query for each relation within each constraint. If the answer is "yes" for a given query, the corresponding constraint will be replaced by the relation that has been confirmed (all the other relations will be eliminated). Otherwise (if the answer is "no"), the related relation will be removed from the constraint. After every query, PC is applied on the graph to remove path inconsistent relations due to this recent update. This will reduce the number of relations per constraint which will reduce the number of subsequent queries and helps getting the target QCN scenario (solution) sooner.

The pseudo-code of our method is listed in Algorithm 2. G_t is initially set to a complete constraint graph with universal relations. $\mathcal{B}$ and CT are respectively the set of possible relations and composition table of the QCN to learn. *QueryGeneration* is a function that generates a membership query each time it is called. We use two implementations of this function. In the first one, the constraint and its related relation (to confirm) are picked randomly. In the second implementation, the relations to confirm are picked according to the first fail principle used when solving general Constraint Satisfaction Problems (CSPs) [9]. In this regard we use the ordering heuristics proposed in [20], namely "weight" and "cardinality". The idea behind the "weight" heuristic is to quantify the restriction imposed by a relation, when assigned to a given edge, on the temporal constraints of the other edges. The "cardinality" heuristic is a special case of the "weight" heuristic when considering that all relations have the same weight (equal to 1). The algorithm iterates by processing a query at each time, until a solution (target QCN) is found or a path inconsistency is detected.

Given that the constraint graph is not necessarily complete, our algorithm for case 2 differs from the previous one, as follows. If the answer is "yes" for a given query, the relation is confirmed but we do not remove the other relations as the corresponding constraint can be the universal relation, I. Instead, we ask the user a second query for the same constraint in order to check if this latter is I. If the answer is "yes" then the constraint is confirmed to be universal, otherwise (the answer is "no") we replace the constraint by the relation confirmed with the first query. The rationale is that, in case 2 we either have a single relation or a universal relation for each constraint.

In case 3, given that the number of relations per constraint varies from 1 to $|\mathcal{B}|$ (13 in the case of IA networks) then we will have to ask the user subsequent queries for the same constraint regardless of the answer. More precisely, if the answer is "yes" then we need to ask the user for the remaining relations as we can have more than one relation per constraint.

4.2 Dealing with Inconsistent Answers

As stated in Algorithm 2, our learning method collapses (fails to return the target QCN) if PC detects an inconsistency after one of the constraints becomes empty (has no relations). In this case, we need to identify the query that has been answered incorrectly. We address this task using a backtrack search algorithm to go backward and let the user confirm previous answers until we reach the state where the user changes the answer of a query. Given that we can have more than one incorrect answer, every time a response to a query is changed, we

Algorithm 2 Learning QCN for Case 1.

```
1: procedure LearningQCN
2: Input: a language set B, a composition table CT
3: Output: a target QCN G_t
4:     G_t ← complete graph with universal relations
5:     q ← QueryGeneration(G_t)
6:     while q ≠ nil do
7:         r ← Relation(q)
8:         if Ask(q) = "yes" then
9:             ConfirmRelation(G_t, r)
10:        else
11:            RemoveRelation(G_t, r)
12:        end if
13:        status ← PC(G_t, CT)
14:        if status = "inconsistent" then
15:            return "collapse"
16:        end if
17:        q ← QueryGeneration(G_t)
18:    end while
19:    return G_t
20: end procedure
```

resume the normal querying starting back from the state where the user fixed the incorrect answer. For example, assume that after a query q_j is answered, an inconsistency is detected. Our algorithm will then backtrack until it identifies the query q_m causing this inconsistency. The user will then change the answer and our learning algorithm will resume from query q_{m+1}. The pseudo code of our method is listed in Algorithm 3. This algorithm is very similar to Algorithm 2 except that we save the query and the current G_t at each time in the stack s.

5 Theoretical Limits on Learning QCNs

In this section we use the learning theory machinery to prove some limits on learning Path Consistent QCNs from queries. In particular, we are interested in the number of queries required by the best possible algorithm. The problem addressed in this work can be viewed as a concept learning problem [2]. In concept learning, a concept is a subset of a given universe $\mathcal{X}$ and a concept class $\mathcal{C}$ is a set of concepts. We assume the user has a hidden target $c^* \in \mathcal{C}$ and we try to exactly identify it with the minimum number of queries. There are different types of queries and we are interested in what is known as membership queries [2, 12]. In learning with membership queries, the learning algorithm, having access to the set of concepts $\mathcal{C}$, picks an instance $x \in \mathcal{X}$ and asks the user *"Is $x \in c^*$?"*. The answer is either "yes" or "no". This process continues until the algorithm exactly identifies c^*. The main challenge here is to have a sequence of instances $x, x', \ldots$ of minimum size that exactly identifies any hidden concept. Let $\mathcal{I} = \{1, 2, \ldots, n\}$ be the set of entities and $\mathcal{B} = \{b_1, b_2, \ldots, b_p\}$ be the set of $p > 0$ relations. In this work, we are interested in the learnability of the three cases of QCNs presented in the previous Section and defined over $\mathcal{I}$ and $\mathcal{B}$ as follows:

1. The set of complete scenarios $\mathcal{Q}_{cmp}$.
2. The set of incomplete scenarios $\mathcal{Q}_{inc}$.
3. The set of all QCNs $\mathcal{Q}_{all}$.

Algorithm 3 Learning with Mistakes.

```
1: procedure LearningWithMistakesQCN
2: Input: B: Language set. CT: Composition Table
3: Output: G_t : target QCN
4:     G_t ← complete graph with universal relations
5:     s ← ∅
6:     q ← QueryGeneration(G_t)
7:     stackpush(s, < G_t, q >)
8:     while q ≠ nil do
9:         r ← Relation(q)
10:        if Ask(q) = "yes" then
11:            ConfirmRelation(G_t, r)
12:        else
13:            RemoveRelation(G_t, r)
14:        end if
15:        status ← PC(G_t, CT)
16:        while status = "inconsistent" do
17:            stackpop(s, < G_t, q >)
18:            if Ask(q) = "yes" then
19:                ConfirmRelation(G_t, r)
20:            else
21:                RemoveRelation(G_t, r)
22:            end if
23:            status ← PC(G_t, CT)
24:        end while
25:        q ← QueryGeneration(G_t)
26:        stackpush(s, < G_t, q >)
27:    end while
28:    return G_t
29: end procedure
```

Clearly, $\mathcal{Q}_{cmp} \subsetneq \mathcal{Q}_{inc} \subsetneq \mathcal{Q}_{all}$. Let $\mathcal{S}$ be the set of two-element subsets of $\mathcal{I}$. An instance x is a pair (s, b) where $s \in \mathcal{S}$ and $b \in \mathcal{B}$. For simplicity, we write the triple (i, j, b) to denote the instance (s, b) where $s = \{i, j\}$. The instance space $\mathcal{X}$ is then defined simply as $\mathcal{X} = \mathcal{S} \times \mathcal{B}$ and clearly $|\mathcal{X}| = \binom{n}{2} \times p$. A concept c is a subset of $\mathcal{X}$ or equivalently it is a mapping from $\mathcal{X}$ to $\{0, 1\}$ where $c(x) = 1$ iff $x \in c$ for any $x \in \mathcal{X}$. For a set $\mathcal{Q}$ of QCNs, c is representable by $\mathcal{Q}$ if there exists a QCN $N \in \mathcal{Q}$ where $c(x) = 1$ if and only if x holds in N. The concept class $\mathcal{C}$ is then defined as the set of all concepts that are representable by $\mathcal{Q}$. We consider the three concept classes $\mathcal{C}_{cmp}, \mathcal{C}_{inc}$ and $\mathcal{C}_{all}$ that represent respectively the set of concepts that are representable by the set of QCNs in $\mathcal{Q}_{cmp}, \mathcal{Q}_{inc}$ and $\mathcal{Q}_{all}$. One of the important parameters in learning theory is the teaching dimension [11]. The teaching dimension of a concept c w.r.t. a class $\mathcal{C}$, $\mathrm{TD}(c, \mathcal{C})$, is the smallest number of examples (or instances) that distinguishes c from every other concept in the class. The teaching dimension of a class $\mathcal{C}$ (denoted as $\mathrm{TD}(\mathcal{C})$) is the teaching dimension of the hardest concept to teach, i.e., $\mathrm{TD}(\mathcal{C}) = \max_{c \in \mathcal{C}} \mathrm{TD}(c, \mathcal{C})$ [11]. It is known that the number of membership queries required by the best possible algorithm is lower bounded by TD [11]. Therefore, any learning algorithm would need at least TD queries in the worst case scenario. In the following, we show the teaching dimension of the three classes of QCNs. We believe such results would give some insights into the learnability of QCNs in general.

▶ **Proposition 1.** $\mathrm{TD}(\mathcal{C}_{cmp}) = \binom{n}{2}$.

Proof. For the upper bound, every concept $c \in \mathcal{C}_{cmp}$ represents a complete scenario with $\binom{n}{2}$ edges and we can teach any edge (i, j) in c by at least one instance (i, j, b_k) where $b_k \in \mathcal{B}$ is the singleton relation that holds between i and j. To see why $\mathrm{TD}(\mathcal{C}_{cmp})$ is at least $\binom{n}{2}$, consider the concept c where, for any two edges (i, j) and (j, r) in c, we cannot infer any useful information on the relation between (i, r) from the transitive closure table. Thus, there exists no edges in c where we can infer their relation from the other two edges and one instance per edge is required. ◀

Note that the fact that $\mathcal{C}_{cmp} \subset \mathcal{C}_{inc}$ does not necessarily mean $\mathrm{TD}(\mathcal{C}_{cmp}) \leq \mathrm{TD}(\mathcal{C}_{inc})$ i.e., TD is not monotonic. There could be concepts that are hard to teach on small classes but their teaching becomes easy on large classes. Therefore, the above result gives no clue over the TD of the set of incomplete scenarios.

▶ **Proposition 2.** $\mathrm{TD}(\mathcal{C}_{inc}) = n(n-1) - 1$.

Proof. For the upper bound, at least two examples $x = (i, j, b_k)$ and $x' = (i, j, b_{k'})$ for $b_k \neq b_{k'}$ suffice to teach the relation between any pair (i, j) of entities. In particular, if there exists an edge (i, j) with relation b_k, then $c(x) = 1$ and $c(x') = 0$ otherwise if there exists no edge between i and j then $c(x) = 1$ and $c(x') = 1$ for any concept $c \in \mathcal{C}_{inc}$. As any concept in $\mathcal{C}_{inc}$ has at least one edge in its graph, it follows that we need at least two examples per pair and for at least one pair (i, j) we need exactly one instance (since the QCN where all the edges are universal is not included in this class). Therefore, $\mathrm{TD}(\mathcal{C}_{inc}) \leq n(n-1) - 1$. For the lower bound, consider the concept $c^{\emptyset}$ that represents a QCN with empty edge set in its graph. It is easy to see that we cannot teach such concept with fewer than $n(n-1)$ examples as we need to confirm that every pair (i, j) holds a universal relation. ◀

▶ **Proposition 3.** $\mathrm{TD}(\mathcal{C}_{all}) = |\mathcal{X}|$.

Proof. The instance space size $|\mathcal{X}|$ is a trivial upper bound on the teaching dimension of any class. For the lower bound, consider the concept $c^{\emptyset}$ again. Our argument is that $\mathrm{TD}(c^{\emptyset}, \mathcal{C}_{all}) > |\mathcal{X}| - 1$. To see this, consider any set T of instances where $|T| = |\mathcal{X}| - 1$ and w.l.o.g. assume $\mathcal{X} = T \cup \{(i, j, b_k)\}$ for an arbitrary instance (i, j, b_k). We can always have another concept $c' \neq c^{\emptyset}$ where it represents a QCN with only one edge (i, j) and its relation equals to $\mathcal{B} \backslash \{b_k\}$. Thus, T cannot distinguish $c^{\emptyset}$ from c' and $\mathrm{TD}(c^{\emptyset}, \mathcal{C}_{all}) > |\mathcal{X}| - 1$. ◀

6 Experimentation

We report on the experiments conducted in order to assess the effect of PC on the number of queries needed to learn an IA network. In this regard, we compare our proposed learning algorithm with and without PC or PPC (we call the method without PC, the "Naive" method) for each of the three cases listed in Section 4 as well as the case of incorrect answers. In the case where PC is used, we consider 2 situations: the case where the *QueryGeneration* function generates the queries randomly as well as using an ordering heuristic as described in Section 4. Moreover, for case 2 we consider PPC in addition to PC, given that we have incomplete graphs in this case. All the experiments are conducted on a Dell XPS 8900, i7-6700K, 32 GB RAM, running Linux. All the algorithms are coded in Java. For each of these cases, we first generate a random consistent temporal scenario (that we call G_t). We then add primitives randomly to get an initial graph $G_{problem}$ with the goal of achieving the following, at the end of the learning process: $G_{problem} = G_t$. Each query is generated with a probability P_y of getting a "yes" answer. The general approach is similar for all the cases we consider in the following, and differs in terms of the target G_t).

In case 1, we use the $\mathbf{S}(n, p)$ model [20] that starts by generating n numeric random intervals (pairs of natural values) assigned to each temporal variable in the graph. Allen primitives are then deduced from pairs of these numeric intervals. For example, let us assume the following assignments to two temporal events X and Y: $X = (3, 11)$ and $Y = (7, 18)$. The corresponding Allen primitive is then: (XOY) i.e. X overlaps Y. The model $\mathbf{S}(n, p)$ ensures a consistent solution since all variables have numeric intervals. A random consistent scenario is then build and considered as our target graph G_t. Our algorithm then starts from a complete graph $G_{problem}$ where each edge contains the universal relation I (the disjunction of all the 13 primitives).

Case 2 is similar to case 1 with the following difference when generating the target network. Some edges in G_t will be removed based on a parameter P_u corresponding to the percentage of universal relations in G_t. These relations (edges to be removed) will be in a set E_u that is used in the query generation. $Q_{solution}$ is generated by traversing all the edges in G_t, and by adding all the universal relations in E_u. In case 3, we first generate G_t as in case 1 and then randomly add more primitives to each constraint (X, Y). G_t is therefore a complete graph. $Q_{solution}$ contains queries for the primitives within the constraint (X, Y). The query generation for inconsistent scenarios is produced as follows. Mistakes are picked randomly with a percentage ratio P_m. Query generators are similar to the query generators in case 1, case 2, and case 3. However, queries are labelled with $isMistake$ as a boolean indicator which is checked when answering each query. If $isMistake$ is true, then the answer would be incorrect, otherwise, the answer must be correct. Figures 1, 2, and 3 show the comparative results for the 3 cases when 100 variables are considered. It is clear from the chart corresponding to case 1 that PC has a significant effect on reducing the number of queries especially when there is a small percentage of "yes" answers. For instance, in the extreme case where there is no "yes" answers, the Naive method requires about 60000 queries while the learning method using PC only needs about 2500 queries to reach the target IA scenario. By increasing the percentage of "yes" answers, the number of queries starts to drop down to 4950 queries for 100% of "yes" answers using the Naive method, which is very close to the 3731 queries that are asked using PC. The "cardinality descending" heuristic is the most effective one as we can easily see from the chart, while the other two do not seem to have an effect when compared to PC without heuristics.

The situation in case 2 is similar to the one in case 1. All the methods using path consistency are better than the naive method. These methods have however similar performance.

For case 3, we can easily see that there is a significant difference between the 3 methods considered. PC with "cardinality" ordering heuristic is the winner in this situation and is followed by the PC method without heuristic. These results are justified by the fact that we are dealing with complete graphs with much more relations per constraint than cases 1 and 2.

Tables 2, 3 and 4 show the results for inconsistent scenarios due to mistakes for each of the 3 cases when 100 variables are considered. As we can notice, in all cases the number of queries, mistakes and related backtracks is considerably reduced when path consistency is used.

In Table 2, we can easily see that there is a significant difference between the Naive method and PC in terms of number of queries, number of mistakes detected and running time. There are 14 times more mistakes detected by the Naive method than the PC method. This is explained by the fact that PC removes inconsistent relations at each time leaving less chances for the user to choosing them. Also, the ordering heuristic does not seem to do a good job in this case.

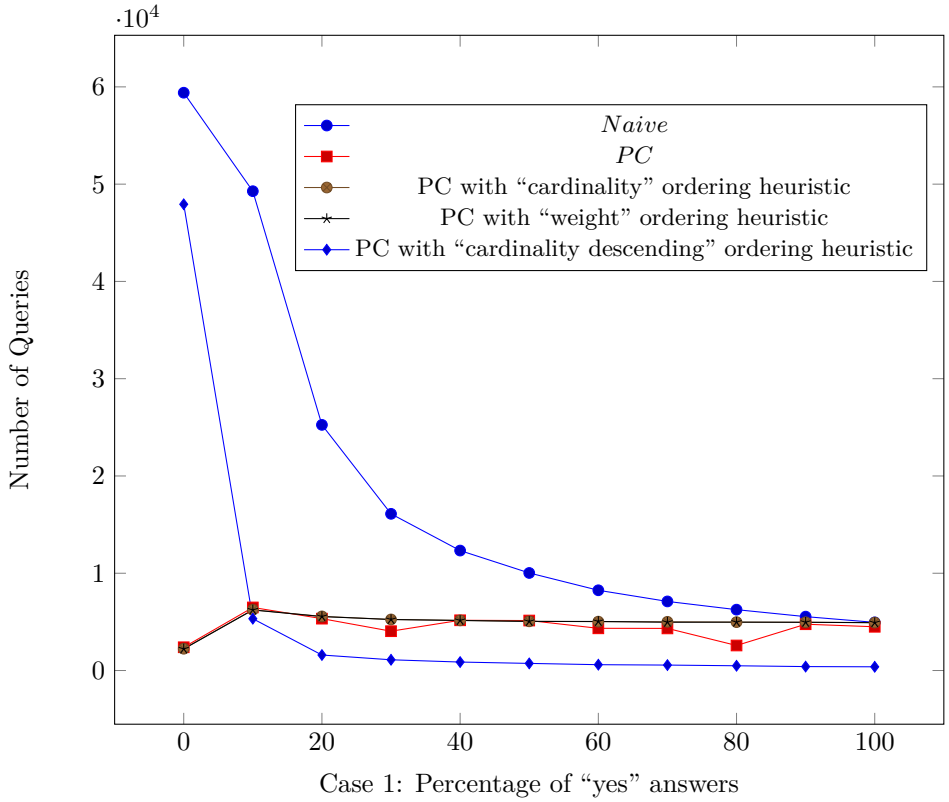


Figure 1 Test results for case 1.

In Table 3, we notice that the performance PPC is better than the one of PC and this is due to the fact that we are dealing with incomplete graphs. Also, the "weight" ordering heuristic is effective in this case for both PC and PPC. This is again explained by the nature of problems we are dealing with in case 2.

Table 4 clearly shows that both the "cardinality" and "weight" ordering heuristics are effective given that the objective is to produce a path consistent QCN.

7 Conclusion

The symbolic representation of time and space is very relevant especially when dealing with incomplete information. We have proposed a new algorithm for learning QCNs by queries. The proposed algorithm is enhanced with path consistency to reduce the number of queries needed to reach a target QCN. In order to assess the effect of PC and ordering heuristics on reducing this number, in practice, we have conducted several experiments on randomly generated IA networks, considering several scenarios. The results of these tests are very promising and encouraging. In this regard, we plan to pursue this work by considering other QCNs such as the Region Connection Calculus (RCC) [13], the rectangle algebra [4] and the n-intersections [10]. We will as well consider temporal constraint networks involving both quantitative and qualitative information [17, 18]. Another future direction, we will consider, is to extend our algorithm to learning preferences as these often co-exist with constraints in many real world applications [16, 19].

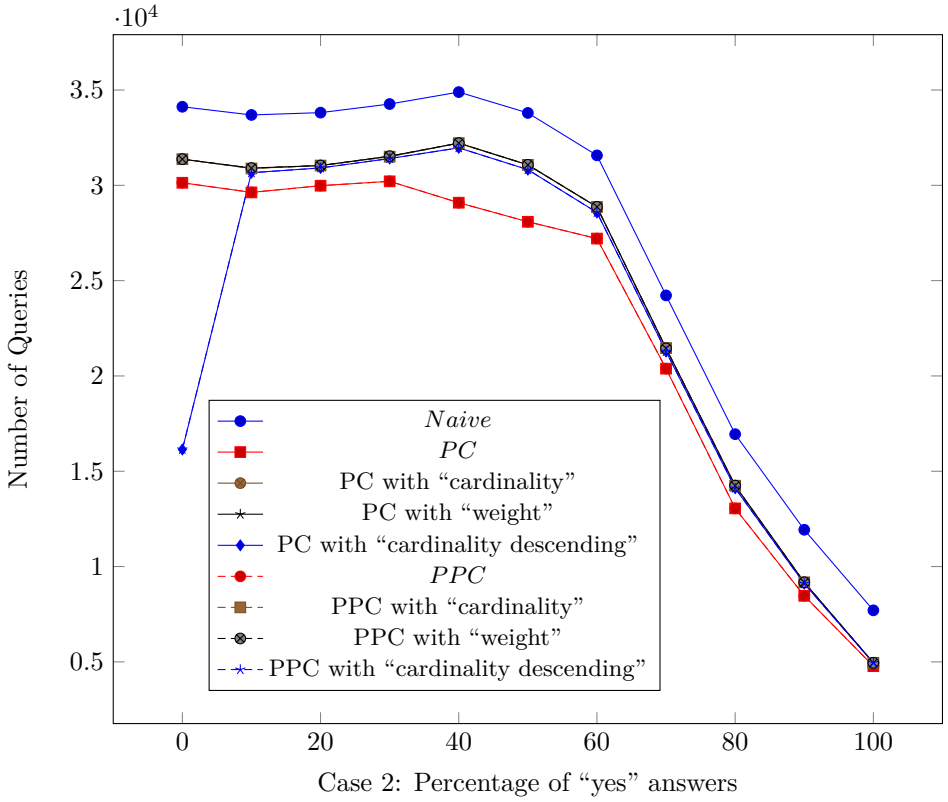


Figure 2 Test results for case 2.

Table 2 Results for inconsistent scenarios for case 1.

Method	# of Queries	Time(s)	Mistakes	Backtracks
Naive	13005708	802.175	399	398
PC	4444	55.061	28	28
PC with "cardinality"	6879	82.741	43	43
PC with "weight"	7819	122.455	36	36
PC with "cardinality descending"	38367	524.589	75	75

Table 3 Results for inconsistent scenarios for case 2.

Method	# of Queries	Time(s)	Mistakes	Backtracks
Naive	63206804	3430.845	6676	6666
PC	36662	458.491	4	5
PPC	10598	114.350	3	2
PC with "cardinality"	28906	327.168	5	0
PPC with "cardinality"	88593	986.297	14	14
PC with "weight"	8786	198.325	1	1
PPC with "weight"	8655	109.046	2	2
PC with "cardinality descending"	132225	2470.913	19	20
PPC with "cardinality descending"	67066	706.155	8	9

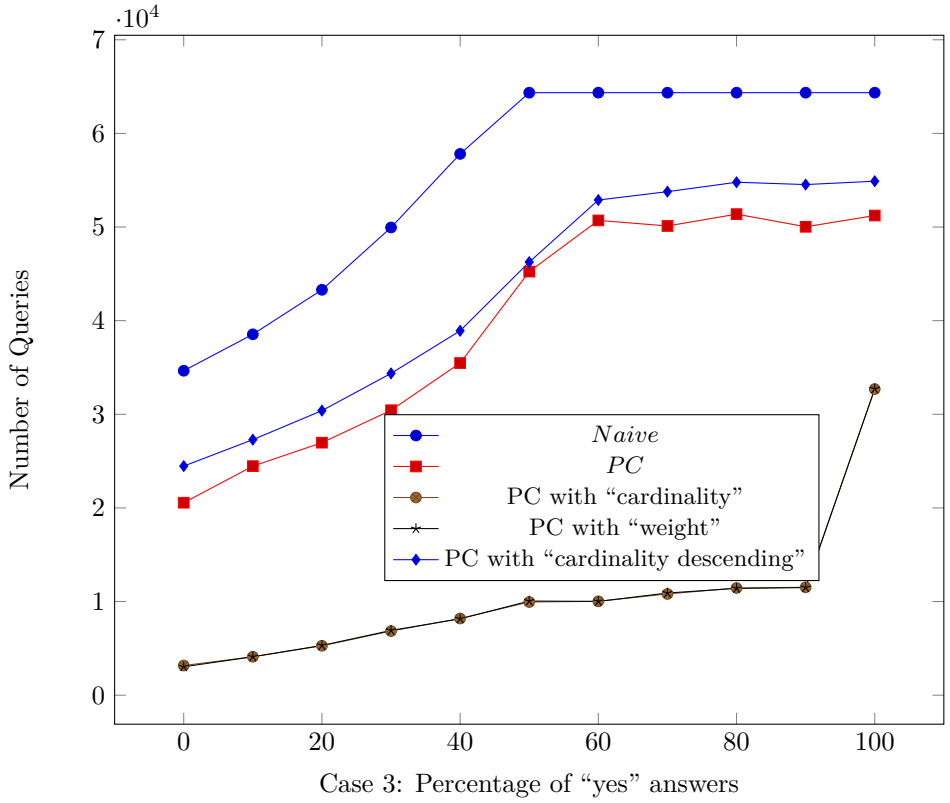


Figure 3 Test results for case 3.

Table 4 Results for inconsistent scenarios for case 3.

Method	# of Queries	Time(s)	Mistakes	Backtracks
Naive	2384321	123.455	66	66
PC	40917	542.344	41	1
PC with "cardinality"	11514	46.772	10	6
PC with "weight"	11280	54.366	11	2
PC with "cardinality descending"	330215	5323.526	69	61

References

1 James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

3 R. Arcangioli and N. Lazaar. Multiple Constraint Acquisition. In *Workshop on Constraints and Preferences for Configuration and Recommendation, the Twenty-Forth International Joint Conference on Artificial Intelligence*, 2015.

4 Philippe Balbiani, Jean-François Condotta, and Luis Farinas del Cerro. A new tractable subclass of the rectangle algebra. In *Proceedings of the 16th international joint conference on Artifical intelligence - Volume 1*, pages 442–447, 1999.

5 C. Bessiere, R. Coletta, B. O'Sullivan, and M. Paulin. Query-Driven Constraint Acquisition. In *Proceedings of the Twentiest International Joint Conference on Artificial Intelligence*, pages 50–55, 2007.

6 Christian Bessiere, Remi Coletta, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Constraint acquisition via partial queries. In *IJCAI'2013: 23rd International Joint Conference on Artificial Intelligence*, page 7, 2013.

7 Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. Constraint acquisition. *Artificial Intelligence*, 244:315–342, 2017. doi:10.1016/j.artint.2015.08.001.

8 Christian Bliek and Djamila Sam-Haroud. Path consistency on triangulated constraint graphs. In *IJCAI*, volume 99, pages 456–461. Citeseer, 1999.

9 Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

10 Max J. Egenhofer and Robert D. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

11 Sally A. Goldman and Michael J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50:20–31, 1995.

12 Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994.

13 Sanjiang Li and Mingsheng Ying. Region connection calculus: Its models and composition table. *Artificial Intelligence*, 145(1):121–146, 2003.

14 Zhiguo Long, Michael Sioutis, and Sanjiang Li. Efficient path consistency algorithm for large qualitative constraint networks. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1202–1208. AAAI Press, 2016.

15 Roger Mohr and Thomas C Henderson. Arc and path consistency revisited. *Artificial intelligence*, 28(2):225–233, 1986.

16 Malek Mouhoub and Amrudee Sukpan. Managing temporal constraints with preferences. *Spatial Cognition & Computation*, 8(1-2):131–149, 2008.

17 Malek Mouhoub and Amrudee Sukpan. Conditional and composite temporal csps. *Applied Intelligence*, 36(1):90–107, 2012.

18 Peter Revesz. Tightened transitive closure of integer addition constraints. In *Eighth Symposium on Abstraction, Reformulation, and Approximation*, pages 136–142. AAAI, 2009.

19 Samira Sadaoui and Shubhashis Kumar Shil. A multi-attribute auction mechanism based on conditional constraints and conditional qualitative preferences. *JTAER*, 11(1):1–25, 2016.

20 Peter Van Beek and Dennis W Manchak. The design and experimental analysis of algorithms for temporal reasoning. *Journal of Artificial Intelligence Research*, 4(1):1–18, 1996.

A Stream Reasoning System for Maritime Monitoring

Georgios M. Santipantakis
University of Piraeus, Karaoli and Dimitriou 80, 18534 Piraeus, Greece
gsant@unipi.gr

Akrivi Vlachou
University of Piraeus, Karaoli and Dimitriou 80, 18534 Piraeus, Greece
avlachou@unipi.gr

Christos Doulkeridis
University of Piraeus, Karaoli and Dimitriou 80, 18534 Piraeus, Greece
cdoulk@unipi.gr

Alexander Artikis
Department of Maritime Studies, University of Piraeus, Greece
Institute of Informatics & Telecommunications, NCSR "Demokritos", Ag. Paraskevi, Greece
a.artikis@unipi.gr

Ioannis Kontopoulos
Institute of Informatics & Telecommunications, NCSR "Demokritos", Ag. Paraskevi, Greece
ikon@iit.demokritos.gr

George A. Vouros
University of Piraeus, Karaoli and Dimitriou 80, 18534 Piraeus, Greece
georgev@unipi.gr

Abstract

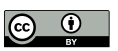
We present a stream reasoning system for monitoring vessel activity in large geographical areas. The system ingests a compressed vessel position stream, and performs online spatio-temporal link discovery to calculate proximity relations between vessels, and topological relations between vessel and static areas. Capitalizing on the discovered relations, a complex activity recognition engine, based on the Event Calculus, performs continuous pattern matching to detect various types of dangerous, suspicious and potentially illegal vessel activity. We evaluate the performance of the system by means of real datasets including kinematic messages from vessels, and demonstrate the effects of the highly efficient spatio-temporal link discovery on performance.

2012 ACM Subject Classification Computing methodologies → Activity recognition and understanding

Keywords and phrases event pattern matching, Event Calculus

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.20

Acknowledgements This work is supported by the datAcron project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687591 (`http://datacron-project.eu`).



25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek; Article No. 20; pp. 20:1–20:17

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

1 Introduction

Nowadays, maritime surveillance systems that keep track of the daily operation of vessel fleets constitute an essential tool for large shipping companies, coast guards, as well as governmental agencies, due to their immense effect on economy and environment [10]. Advances in navigation technology enable the real-time provision of vessel positional information for the benefit of surveillance systems. The Automatic Identification System (AIS)[1], for example, is a tracking system for identifying and locating vessels at sea through data exchange. The acquisition of positional data is achieved either by AIS base stations along coastlines, or even by satellites when out of range of terrestrial networks. As this data is streamed in a maritime surveillance system, several operations need to be performed in real-time, including data integration and complex maritime activity recognition.

We present a monitoring system that exploits spatio-temporal relations between vessels, or vessels and areas of interest (e.g., protected areas), which are calculated online by a dedicated component for spatio-temporal link discovery (stLD). These relations are provided as input to a complex activity recognition component, which is based on the "Event Calculus for Run-Time reasoning" (RTEC) [3]. This is an Event Calculus [12] implementation with various optimization techniques for continuous narrative assimilation on data streams.

We address the problem of online spatio-temporal link discovery over streaming and archival data. This link discovery (LD) problem is challenging because of (a) the streaming nature of data, and (b) the complexity of evaluating spatio-temporal similarity functions to determine the links between spatio-temporal entities. Typically, LD tasks are solved by *filter-and-refine* algorithms that identify a set of candidate entities (for linking) in the *filtering step*, which need to be verified in the *refinement step*. The refinement step is the principal cost factor that determines the overall performance of LD, since it requires the evaluation of distance/similarity functions on complex geometrical entities. For the case of stream to static LD, we propose a filtering technique that improves the efficiency of the filtering step by eliminating entities that cannot be linked, thereby reducing the number of entities that have to be considered during refinement. For the case of streaming data only, we provide an efficient method for streaming LD, identifying proximity relations between moving vessels.

In earlier work, we presented a maritime monitoring system which employed RTEC for complex activity recognition [20]. In that work, RTEC performed spatial calculations to determine whether a vessel is close to a port, or within an area of interest. Furthermore, it approximated crudely the *nearby* relation between vessels by checking whether a pair of vessels is located within the same cell of a grid. In this work, we placed emphasis on the detection of spatial relations and developed a separate component for highly efficient spatio-temporal link discovery. Moreover, RTEC has at its disposal additional spatial relations – whether a vessel is *nearby* some area – and a much more accurate account of proximity between vessels.

To summarise, this paper makes the following contributions:

- We present a stream reasoning system integrating a component for spatio-temporal link discovery (stLD), and a component for recognizing complex activities by means of temporal pattern matching.
- We propose a technique for stream-based LD, which improves the efficiency of filtering, by eliminating entities that cannot be linked, thereby reducing the number of entities that have to be considered during refinement.
- We evaluate the performance of the system by means of real datasets including AIS kinematic messages from vessels.

[1] http://www.imo.org/en/OurWork/Safety/Navigation/Pages/AIS.aspx

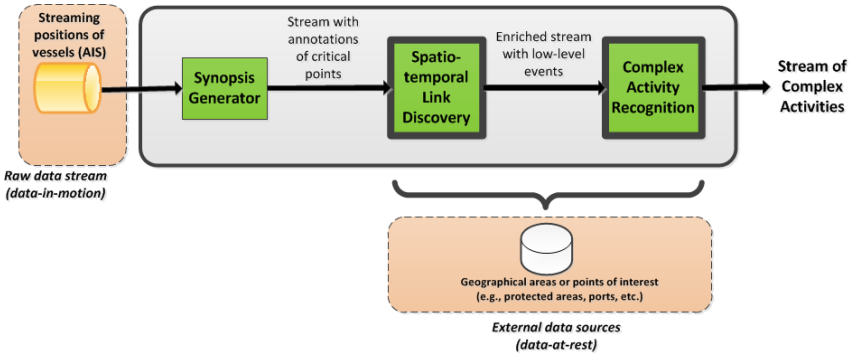


Figure 1 The datAcron prototype architecture.

2 System Architecture

Our work is performed in the context of the datAcron research project[2], which aims at advancing the management of voluminous and heterogeneous data-at-rest (archival data) and data-in-motion (streaming data) sources, so as to promote the safety and effectiveness of critical operations of moving objects in large geographical areas. The architecture of the datAcron prototype for complex activity recognition is depicted in Figure 1.

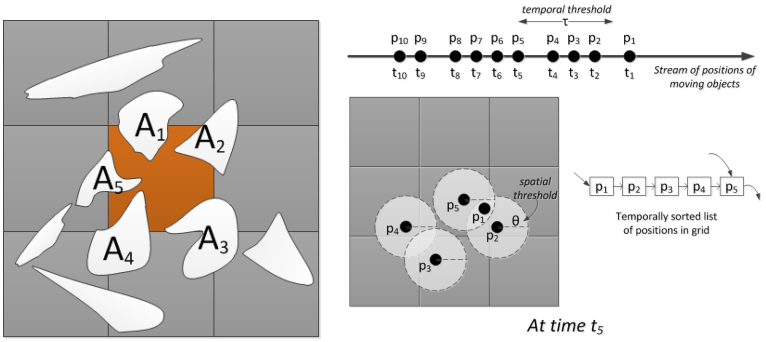
The main input is streaming positions of vessels, in the form of AIS messages. As this streaming data flows in the system, trajectory compression takes place, by annotating a subset of the original vessel positions as *critical points/events*. The *Synopses Generator* component (see Figure 1) provides algorithms for trajectory reconstruction and compression, by cleaning erroneous data and eliminating vessel positions that do not significantly affect the quality of trajectory representation. Then, critical points are linked with archival data online, namely marine areas or points of interest, such as protected areas (Natura2000) or ports (World Port Index). Link discovery is performed at the spatio-temporal level, thus identifying those areas (or points) that are related to a given position of a vessel. Relations may be topological (e.g., a vessel is located *within* an area) or proximity-based (e.g., a vessel is *nearby* a port). In addition, vessel positions are linked to each other, by processing the streaming data only; this results in discovering proximity relations between moving vessels (e.g., a vessel is *nearby* another vessel) in an online fashion. Subsequently, a complex activity recognition module consumes the stream of spatial relations and critical points to recognize various types of vessel activity. The *Complex Activity Recognition* component is based on the "Event Calculus for Run-Time reasoning" (RTEC) [3].

The innovative feature of the proposed architecture is the integration of an optimized component for online discovery of spatial relations with an activity recognition engine. In the following, we delve into the technical details of spatio-temporal link discovery (Section 3) and complex activity recognition (Section 4) for maritime surveillance.

3 Spatio-temporal Link Discovery

We begin by providing the definitions and problem setting for spatio-temporal link discovery. Then, we focus on: (a) topological and proximity relations between the positions of moving entities and static geographical areas of interest (Section 3.1), and (b) proximity relations

[2] `http://datacron-project.eu/`



(a) Illustration of the mask of a grid cell that overlaps five areas $\{A_1, \ldots, A_5\}$.

(b) Example of grid and bookkeeping structure for "nearby" LD between moving vessels.

Figure 2 Link discovery examples.

between the positions of moving entities (Section 3.2). The former is a case of streaming to static link discovery, while the latter is a case of streaming to streaming link discovery.

Let $p = (p.x, p.y, p.t)$ denote a spatio-temporal point corresponding to a vessel's V position $(p.x, p.y)$ at a given time $p.t$, and $\mathcal{A}$ a dataset that consists of geographical areas represented as polygons. A polygon $A \in \mathcal{A}$ is represented as a set of points a_i, i.e., $A = \{a_1, a_2, \ldots, a_n\}$, and we write $a_i \in A$ to denote that a_i is included in the representation of A.

Further, let $d(p, p')$ denote the distance between the spatial positions $(p.x, p.y)$ and $(p'.x, p'.y)$ of two vessels, whereas $t(p, p')$ stands for their temporal difference. Without loss of generality, in this paper, we employ the Haversine distance function to quantify the spatial distance of two points, whereas $t(p, p') = |p.t - p'.t|$. However, our approach is readily applicable to other domains, where other distance functions (e.g., Euclidean) are more appropriate. We abuse notation slightly by denoting $d(p, A)$ the distance between a point p and an area A (polygon). This is also known as MINDIST, and is defined as the distance of p to the closest point of A (one point of the perimeter), if p is outside of A. Otherwise, $d(p, A) = 0$. Below are the formal descriptions of the spatial relations discovered by the stLD component.

▶ **Definition 1.** *withinArea*(V, A): Given a spatio-temporal position p of a vessel V and an area $A \in \mathcal{A}$, *withinArea*(V, A) is true, if p is enclosed in A.

▶ **Definition 2.** *nearbyArea*(V, A, θ): Given a spatio-temporal point p of a vessel V, an area $A \in \mathcal{A}$, and a distance threshold θ, *nearbyArea*(V, A, θ) is true, if $d(p, A) \leq \theta$.

▶ **Definition 3.** *nearby*(V_1, V_2, θ, τ): Given two spatio-temporal points p and p' of vessels V_1 and V_2 respectively, a distance threshold θ, and a temporal threshold τ, *nearby*(V_1, V_2, θ, τ) is true, if $d(p, p') \leq \theta$ and $t(p, p') \leq \tau$.

3.1 Stream to Static LD

Discovery of Topological Relations: Within. Given a *target* dataset T, a *source* dataset S, and a relation r, the goal of link discovery is to detect the pairs $(\sigma, \tau) \subseteq S \times T$, where $\sigma \in S$ and $\tau \in T$, s.t. (σ, τ) satisfies r. Consider the case of relation "within" between a moving entity p, whose current spatio-temporal position is streamed into our system, and a set of static geographical areas of interest $\mathcal{A} = \{A_1, \ldots, A_n\}$. The aim of link discovery is to identify all areas $A_i \in \mathcal{A}$ which enclose the spatial position $(p.x, p.y)$ of p.

Algorithm 1 Spatio-temporal LD algorithm for relation *"within"* using mask.

Input: Grid cells $C = \{c_1, \ldots, c_m\}$, Areas $\mathcal{A} = \{A_1, \ldots, A_n\}$, position p $(p.x, p.y)$
Output: Subset of areas $\mathcal{A}^w \subseteq \mathcal{A}$ that enclose the position $(p.x, p.y)$ of p
Requires: Grid has been constructed and areas have been assigned to overlapping cells
$\mathcal{A}^w \leftarrow \emptyset$
locate cell c_i that encloses p
if within(p, $mask(c_i)$) **then**
  return $\mathcal{A}^w$
else
  for each $A_j \in c_i$ **do**
    if within(p, A_j) **then**
      $\mathcal{A}^w \leftarrow \mathcal{A}^w \cup A_j$
return $\mathcal{A}^w$

A brute force link discovery algorithm would have to perform the geometrical test between p and all areas in $\mathcal{A}$, thereby performing $O(n)$ comparisons, where $n = |\mathcal{A}|$. To avoid this prohibitively expensive cost, the state-of-the-art LD methods (e.g., [16, 22]) employ a *space tiling* approach, which essentially partitions the space in cells and assigns each area to its overlapping cells. Then, to compute the relation "within", the position of a vessel is compared only against those (say c) areas overlapping the cell, thus resulting in $O(c)$ comparisons, and typically $c \ll n$. Practically, this method belongs to the filter-and-refine paradigm, where in the filtering step only a small set of c (out of n) candidate areas are identified, and in the refinement step the candidates are examined one-by-one to discover the subset of areas actually enclosing the vessel. Since the refinement step is the most costly processing part, any efficient link discovery algorithm should minimize the number of candidates.

In several applications of spatial link discovery, such as the maritime domain, grid cells contain a significant amount of "empty space", namely the space of the cell that overlaps with no areas. Our observation is that positions of moving vessels located in the empty space of a cell induce high processing cost, as they must be compared to all areas in the cell in vain, since no "within" links can be produced. Motivated by this observation, we propose a technique to explicitly represent the empty space within cells as yet another area. Thus, for each grid cell, we construct an artificial area called *mask*, which is defined as the difference between the cell and the union of areas overlapping with the cell, i.e. $mask = c - (c \cap \bigcup(A)_i)$. Notice that the intersection with c before computing the set difference is only used to cover the case where areas are larger the cells. Figure 2a shows an example of the mask of a cell; the middle cell overlaps with areas $\{A_1, \ldots, A_5\}$, and the mask of the cell is the area represented in orange color.

Having the mask of a cell as yet another area, we can devise an efficient algorithm for link discovery that eagerly avoids comparisons to areas for positions located in the empty space. In practice, after we identify the enclosing cell of a position of a vessel, we first compare it to the mask of the cell, to check if it is enclosed in the empty space. If this single comparison returns true, we stop processing this position, thereby saving c comparisons (c denotes the average number of areas in a cell). For the typical case where a cell contains several areas, this technique can save significant computational cost, as will be demonstrated in the empirical analysis presented in Section 5.

Algorithm 1 presents the pseudo-code for discovering a "within" link between the position p of a vessel and the areas that enclose it. As a prerequisite, the grid has already been constructed and the static areas in the dataset R have been assigned to cells. This is a

pre-processing step. In the first step, the cell c_i that encloses p is determined (line 5). This operation is performed in constant time $O(1)$ in the case of equi-grids. Then, we check if p is contained in the mask $mask(c_i)$ of cell c_i (line 6). If it is contained, no further processing is required, and the algorithm terminates returning the empty set. If it is not contained, then we check for containment against all areas A_j in cell c_i (line 9). For those areas A_j that contain p, we add them to the result set $\mathcal{A}^w$ (line 11) and eventually return them as result.

The lines 9–11 of Algorithm 1 are processed in parallel, i.e., each iteration in the for-loop is carried out by a different thread/"worker". The number of concurrent workers is usually a predefined constant to allow uninterruptible system operation (in our experiments we employ 8 workers). We have enabled multi-thread processing using a *pool of tasks*, populated with the refinement tasks of $within(p, A_j)$. As soon as a worker is available and the pool contains tasks, the next task is selected and assigned to the worker for processing. Also, the algorithm is amenable to parallelization beyond the scope of a single machine, by simply partitioning the stream of positions of vessels to the available processing nodes, each of which runs an instance of Algorithm 1 on an off-line constructed grid.

Discovery of Proximity Relations: Nearby. The above technique is applicable also for link discovery of a proximity relation, such as the "nearby" relation, between vessel position p and a set of static areas $\mathcal{A}$. The "nearby" relation is defined using a spatial threshold θ, and retrieves the subset of areas in $\mathcal{A}$ that are located at most at distance θ from p.

The technique of using the mask needs to be slightly adjusted to work in the case of relation "nearby". The main adjustment concerns the way the mask of each cell is computed. We expand each area A_i by θ and then the cell's mask is computed as previously, only using the expanded areas (A_i^θ) instead of the actual areas A_i. To differentiate this mask of a cell c_i from the one used in the previous algorithm, we denote it by $mask^\theta(c_i)$.

The LD algorithm for relation "nearby" operates as follows. The grid is constructed exactly as before, using the original areas $\{A_i\}$. First, the cell c_i that encloses p is located. If $mask^\theta(c_i)$ contains p, then we can safely stop processing, since no area is nearby p. This is because $mask^\theta(c_i)$ has been constructed based on expanded areas. If this pruning is not successful, we need to examine all cells $c_i \in C'$ that overlap with a circle centered at p with radius θ, since they may contain results. We examine each area A_j in c_i, and if the distance of p to A_j is lower or equal to the threshold θ, then A_j is added to the result $\mathcal{A}^n$. To avoid computing the distance of p to an area A_j multiple times (due to A_j assignment to multiple cells), we maintain the already examined areas in a set ($\mathcal{P}$). In summary, the algorithm avoids processing points that would safely produce no results, due to the use of the mask technique.

3.2 Stream to Stream LD

Streaming link discovery of "nearby" relations between positions of moving vessels requires meticulous use of the available memory, since it is not feasible to store the complete data stream in memory. Our solution relies on the use of a grid data structure on the spatial domain, similarly to the case of Section 3.1. The grid is used to maintain the positions arriving in the stream. When a new vessel position p arrives in the stream, in the filtering step, we examine the cells that intersect with a circle centered at $(p.x, p.y)$ with radius θ, and retrieve all vessel positions p'_i that satisfy the spatial constraint, i.e., $d(p, p'_i) \leq \theta$. Then, in the refinement step, we identify the subset of vessel positions $\{p'_i\}$ that in addition satisfy the temporal constraint, i.e., $t(p, p'_i) \leq \tau$. This solution avoids the exhaustive comparison of p to all other positions that have arrived before p. Algorithm 2 is invoked for each incoming p and describes this solution.

Algorithm 2 Spatio-temporal LD algorithm for relation *"nearby"* between vessels.

```
1: Input: Grid cells C = {c_1, ..., c_m}, vessel position p (p.x, p.y), thresholds θ, τ
2: Output: Set R of pairs of vessel positions (p, p') satisfying Definition 3
3: R ← ∅
4: locate cells C that overlap with circle at p with radius θ
5: for each c_j ∈ C do
6:    for each p'_i ∈ c_j do
7:       if d(p, p'_i) ≤ θ then
8:          if t(p, p'_i) ≤ τ then
9:             R ← R ∪ (p, p'_i)
10: add p to grid C
11: return R
```

However, in order to manage the available memory effectively, we need to find a suitable way to clean up the grid, since vessel positions that have arrived before more than τ time units will never produce a "nearby" link to a new vessel position. A second reason to perform this cleaning operation is efficiency. If no cleaning were performed, then too many (old) vessel positions would be retrieved that satisfy the spatial constraint, but would be eliminated due to the temporal constraint, leading to wasteful processing.

A naive approach for cleaning would be to scan all grid cells (set at time t_{now}) and delete vessel positions p whose timestamp $p.t$ is more than τ units ago, i.e., $t_{now} - p.t > \tau$. However, this operation has linear complexity wrt. the number of positions in the grid, and incurs non-negligible overhead, as it must be performed during stream processing. To address this problem, we introduce an auxiliary, bookkeeping data structure that efficiently detects the vessel positions that need to be deleted. For this purpose, we maintain a list of pointers to the vessel positions in the grid. The list is in temporal order, since vessel positions are inserted in the list in the order in which they arrive in the stream. Then, cleaning can be performed efficiently, by traversing the list and deleting vessel positions (from the grid and list) until a position p is found with timestamp $t_{now} - p.t \leq \tau$. The maintenance cost of this list is small; insertions have $O(1)$ cost, whereas deletions have linear cost to the number of vessel positions that need to be deleted.

Figure 2b shows an example of the bookkeeping structure. Assuming that $t_{now} = t_5$, then the grid and the list contain the five vessel positions $p_1, \ldots, p_5$, as depicted. In case no cleaning is performed, then p_1 would be identified as candidate for "nearby" to p_5, which would then be rejected due to the temporal threshold, since $p_5.t - p_1.t > \tau$. In case cleaning has already been performed, p_1 would have already been deleted from the grid, thus we would have avoided the cost of examining p_1.

An interesting issue that deserves further discussion is the frequency of performing the cleaning operation. An *eager* strategy could invoke the cleaning operation prior to processing every new vessel position that arrives in the stream. This would guarantee that all vessel positions satisfying the spatial distance threshold would be results, without the need to check the temporal threshold. However, the eager strategy would result in paying the overhead of grid cleanup for every new position. Another approach is to follow a *lazy* strategy, where the cleaning operation is invoked periodically, thus limiting the induced overhead at the expense of retrieving some candidate vessel positions that may be rejected by the temporal threshold. We evaluate the frequency of performing the grid cleanup in Section 5 presenting the empirical evaluation.

Table 1 Complex Activity Recognition: Input events are presented above the two horizontal lines, while the output stream is presented below these lines. The input events above the single horizontal line are detected by stLD, while the remaining input events are emitted by the trajectory compression component. All input events, except $nearBy(V_1, V_2)$, are instantaneous, while all output activities are durative.

Event/Activity	Description
$withinArea(V, A)$	A vessel V is within some area A of interest
$nearbyArea(V, A)$	A vessel V is close to some area A of interest
$nearPorts(V)$	A vessel V is close to one or more ports
$nearby(V_1, V_2)$	Two vessels V_1 and V_2 are close to each other
$gapStart(V)$	A vessel V stopped sending position signals
$gapEnd(V)$	A vessel V resumed sending position signals
$slowMotionStart(V)$	A vessel started moving at a low speed
$slowMotionEnd(V)$	A vessel stopped moving at a low speed
$stopStart(V)$	A vessel started being idle
$stopEnd(V)$	A vessel stopped being idle
$changeInSpeedStart(V)$	A vessel started changing its speed
$changeInSpeedEnd(V)$	A vessel stopped changing its speed
$changeInHeading(V)$	A vessel changed its heading
$gap(V)$	A vessel V has a communication gap in the open sea
$stopped(V)$	A vessel V is stopped in the open sea
$slowMotion(V)$	A vessel V moves slowly in the open sea
$illegalFishing(V)$	A vessel V is potentially engaged in illegal fishing
$loitering(V)$	A vessel V is suspiciously idle
$rendezVous(V_1, V_2)$	Two vessels V_1 and V_2 are having a rendez-vous
$highSpeedIn(V, AreaType)$	A vessel V exceeds the speed limit of an area of $AreaType$

4 Complex Activity Recognition

The complex activity recognition (CAR) component of the datAcron prototype consumes the stream of spatial relations detected by stLD, as well as the critical points expressing compressed trajectories, to detect various types of vessel activity (see Section 2). The upper part (above the two horizontal lines) of Table 1 presents the input to the CAR component. The output of this component, i.e. the list of recognized activities, specified in collaboration with the domain experts of datAcron, is presented in the lower part of Table 1.

4.1 Run-Time Event Calculus

The CAR component of datAcron is based on the "Event Calculus for Run-Time reasoning" (RTEC) [3]. RTEC is a Prolog implementation of the Event Calculus [12], designed to compute continuous narrative assimilation queries for pattern matching on data streams. RTEC has a formal, declarative semantics – complex (vessel) activity formalisations are (locally) stratified logic programs [21]. Moreover, RTEC includes various optimisation techniques for efficient pattern matching, such as "windowing", whereby all input events that took place prior to the current window are discarded/"forgotten". Details about the reasoning algorithms of RTEC, including a complexity analysis, may be found in [3]. In what follows, we illustrate the use of RTEC for specifying maritime activities, focusing on the effects of stLD on CAR.

Table 2 Main predicates of RTEC.

Predicate	Meaning
happensAt(E,T)	Event E occurs at time T
holdsAt$(F=V,T)$	The value of fluent F is V at time T
holdsFor$(F=V,I)$	I is the list of the maximal intervals for which $F=V$ holds continuously
initiatedAt$(F=V,T)$	At time T a period of time for which $F=V$ is initiated
terminatedAt$(F=V,T)$	At time T a period of time for which $F=V$ is terminated
union_all(L,I)	I is the list of maximal intervals produced by the union of the lists of maximal intervals of list L
intersect_all(L,I)	I is the list of maximal intervals produced by the intersection of the lists of maximal intervals of list L
relative_complement_all (I',L,I)	I is the list of maximal intervals produced by the relative complement of the list of maximal intervals I' with respect to every list of maximal intervals of list L

The time model in RTEC is linear and includes integer time-points. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. Where F is a *fluent* – a property that is allowed to have different values at different points in time – the term $F=V$ denotes that fluent F has value V. An *event description* in RTEC includes rules that define the event instances with the use of the happensAt predicate, the effects of events on fluents with the use of the initiatedAt and terminatedAt predicates, and the values of the fluents with the use of the holdsAt and holdsFor predicates. Table 2 summarizes the main predicates of RTEC. Complex activities are typically durative (see the lower part of Table 2), thus in CAR the task generally is to compute the maximal intervals for which a fluent expressing a complex activity has a particular value continuously. Below, we discuss the representation of fluents/complex maritime activities, and briefly present the way we compute their maximal intervals.

4.2 Maritime Pattern Representation

For a fluent F, $F=V$ holds at a particular time-point T if $F=V$ has been initiated by an event at some time-point earlier than T, and has not been terminated at some other time-point in the meantime. This is an implementation of the law of *inertia*. The time-points at which $F=V$ is initiated (terminated) are computed with the use of initiatedAt (terminatedAt) rules. Consider the following example:

$$\begin{aligned}&\text{initiatedAt}(gap(V)=\text{true},\ T)\leftarrow\\&\quad\text{happensAt}(gapStart(V),\ T),\ \text{not happensAt}(nearPorts(V),\ T)\\&\text{terminatedAt}(gap(V)=\text{true},\ T)\leftarrow\\&\quad\text{happensAt}(gapEnd(V),\ T)\end{aligned}\tag{1}$$

$gap(V)$ is a Boolean fluent denoting a communication gap for some vessel V, i.e. V stops transmitting AIS messages in the open sea. In some cases, the absence of AIS messages is suspicious and thus we need to record it. $gapStart(V)$ and $gapEnd(V)$ are instantaneous critical events indicating, respectively, the time-points in which a vessel V stops and resumes sending AIS messages (see Table 1). "not" is negation by failure. $nearPorts(V)$ is an event emitted by stLD when vessel V is close to a port. Thus, rule-set (1) states that $gap(V)=\text{true}$ is initiated if the trajectory compression component reports a *gapStart* event for V, and

stLD does not report that V is close to a port. Furthermore, $gap(V) = \mathsf{true}$ is terminated when V resumes communications. Given rule-set (1), RTEC computes the maximal intervals I for which $gap(V) = \mathsf{true}$ holds continuously, i.e. $\mathsf{holdsFor}(gap(V) = \mathsf{true},\ I)$, by finding all time-points T_s at which $gap(V) = \mathsf{true}$ is initiated, and then, for each T_s, computing the first time-point T_e after T_s at which $gap(V) = \mathsf{true}$ is terminated.

Most of the maritime patterns (defined in collaboration with domain experts) concern vessel activity close to, or within, some area of interest, such a Natura area. To simplify the structure of such patterns, we defined the auxiliary fluent *inArea* as follows:

$$\begin{aligned}
&\mathsf{initiatedAt}(inArea(V, protected) = \mathsf{true},\ T) \leftarrow \\
&\quad \mathsf{happensAt}(withinArea(V, A),\ T),\ protected(A) \\
&\mathsf{terminatedAt}(inArea(V, protected) = \mathsf{true},\ T) \leftarrow \\
&\quad \mathsf{happensAt}(nearbyArea(V, A),\ T),\ protected(A) \\
&\mathsf{terminatedAt}(inArea(V, protected) = \mathsf{true},\ T) \leftarrow \\
&\quad \mathsf{happensAt}(\mathsf{start}(gap(V) = \mathsf{true}),\ T)
\end{aligned} \tag{2}$$

inArea$(V, AreaType)$ records the maximal intervals in which a vessel V is in an area of some type. *withinArea*(V, A) and *nearbyArea*(V, A) are spatial events emitted by stLD, stating, respectively, that V is within, or close to area A. *protected*(A) is an atemporal predicate that stores protected areas. $\mathsf{start}(F = V)$ (respectively $\mathsf{end}(F = V)$) is a built-in RTEC event taking place at each starting (ending) point of each maximal interval for which $F = V$ holds continuously. According to rule-set (2), $inArea(V, protected) = \mathsf{true}$ is initiated when stLD detects a *withinArea*(V, A) event for vessel V and protected area A. Furthermore, $inArea(V, protected) = \mathsf{true}$ is terminated when there is information that V has exited the area, i.e. when a *nearbyArea*(V, A) event is emitted for the protected area A, or when V stops transmitting position signals (in this case we do not know the location of V).

Similar rule-sets define *inArea* for other types of area. Moreover, *inArea* is represented as a Boolean fluent since areas of different type may overlap, and thus a vessel may be within various types of area at the same time.

In previous work, RTEC performed both temporal and atemporal reasoning for CAR [20]. For example, RTEC performed spatial calculations to determine whether a vessel is within some area of interest or close to a port. In this work, all spatial relations necessary for CAR are computed by stLD. Thus, the evaluation of the initiatedAt rule of rule-set (2), for example, is reduced to checking for (*withinArea*) facts (in the input stream). This way, RTEC is used only for temporal reasoning for which it is optimized.

The absence of the *nearbyArea* relation in earlier work [20] did not allow us to compute the *intervals* during which a vessel is in some area. These intervals allow us to develop more accurate patterns of maritime activity – consider e.g. illegal fishing:

$$\begin{aligned}
&\mathsf{holdsFor}(illegalFishing(V) = \mathsf{true},\ I) \leftarrow \\
&\quad fishing(V),\ \mathsf{holdsFor}(slowMotion(V) = \mathsf{true},\ I_1), \\
&\quad \mathsf{holdsFor}(inArea(V, protected) = \mathsf{true},\ I_2),\ \mathsf{intersect_all}([I_1, I_2],\ I)
\end{aligned} \tag{3}$$

In addition to the domain-independent definition of holdsFor, an event description may include domain-specific holdsFor rules, such as rule (3), used to define the values of a fluent F, e.g. *illegalFishing*, in terms of a Boolean function on the values of other fluents. Domain-specific holdsFor rules make use of interval manipulation constructs: union_all, intersect_all, and relative_complement_all. These are presented in Table 2. *fishing* is an atemporal predicate recording fishing vessels. According to rule (3), the list I of maximal intervals during which a fishing vessel V is said to be engaged in illegal fishing is computed by determining the

list I_1 of maximal intervals during which V is moving in slow motion in the open sea (these intervals are computed given the instantaneous *slowMotionStart* and *slowMotionEnd* critical events), the list I_2 of maximal intervals during which V is in a protected area, and then calculating the list I representing the intersections of the maximal intervals in I_1 and I_2.

The interval manipulation constructs of RTEC support the following type of definition: for all time-points T, $F = V$ holds at T if and only if some Boolean combination of fluent-value pairs holds at T. For a wide range of fluents, this is a much more concise definition than the traditional style of Event Calculus representation, i.e. identifying the various conditions under which the fluent is initiated and terminated so that maximal intervals can then be computed using the domain-independent holdsFor. For instance, the representation of *illegalFishing* by means of initiatedAt and terminatedAt predicates requires four rules.

In addition to patterns for individual vessels, the knowledge base of the CAR component of datAcron includes formalizations of activities for pairs of vessels. Consider *rendezVous*:

$$\begin{aligned}&\mathsf{holdsFor}(\mathit{rendezVous}(V_1, V_2) = \mathsf{true},\ I) \leftarrow \\ &\quad \mathsf{holdsFor}(\mathit{nearby}(V_1, V_2) = \mathsf{true},\ I_1),\ \ \mathsf{holdsFor}(\mathit{slowMotion}(V_1) = \mathsf{true},\ I_2), \\ &\quad \mathsf{holdsFor}(\mathit{stopped}(V_1) = \mathsf{true},\ I_3),\ \ \mathsf{union\_all}([I_2, I_3],\ I_4), \\ &\quad \mathsf{holdsFor}(\mathit{slowMotion}(V_2) = \mathsf{true},\ I_5),\ \ \mathsf{holdsFor}(\mathit{stopped}(V_2) = \mathsf{true},\ I_6), \\ &\quad \mathsf{union\_all}([I_5, I_6],\ I_7),\ \ \mathsf{intersect\_all}([I_1, I_4, I_7],\ I)\end{aligned} \tag{4}$$

$\mathit{nearby}(V_1, V_2) = \mathsf{true}$ is a fluent denoting whether two vessels V_1 and V_2 are close to each other. This relation is computed by stLD. $\mathit{stopped}(V) = \mathsf{true}$ expresses that vessel V has stopped in the open sea. The intervals of this fluent-value pair are computed with the use of the instantaneous *stopStart* and *stopEnd* critical events (see Table 1). According to rule (4), vessels V_1 and V_2 are said to have a rendez-vous when they are close to each other, and each of them is stopped or moving in slow motion in the open sea. In previous work, we approximated very crudely the $\mathit{nearby}(V_1, V_2)$ relation by checking whether V_1 and V_2 are located within the same cell of a grid. This approximation produced several false negatives (vessels located close to each other but in different cells) as well as false positives (vessels located within the same cell but not close to each other). In this work, we can avoid these issues, due to the efficient spatial relation detection of stLD.

5 Experimental Evaluation

We present the results of our experimental study using real maritime datasets.

5.1 Experimental Setup

Platform. All experiments presented below were performed on a computer with 8 cores (Intel(R) Core(TM) i7-7700 CPU @ 3.6GHz) and 16 GB of RAM, running Ubuntu 16.04 LTS 64-bit with Linux Kernel 4.8.0-53-generic, Java 8, and SWI Prolog 7.2.3 for RTEC.

Datasets. Our main dataset contains AIS kinematic messages from vessels sailing in the Atlantic Ocean around Brest, France, spanning between 1 October 2015 to 31 March 2016 (`https://zenodo.org/record/1167595#.WwlWjp99LJ9`). AIS messages from 5,050 vessels are compressed by the trajectory synopsis module (see Figure 1), keeping only critical points (see Table 1). Each critical point is accompanied by mobility information such as speed and heading. The dataset contains 4,765,647 critical points. These are consumed by stLD which produces 3,204,206 spatial events that are additionally provided to CAR as input. Spatial events determine whether a vessel is in or near an area of interest, and if two vessels are close to each other in space and time.

Table 3 Distribution of areas from target dataset for each grid configuration.

Granularity	Average Number of Areas in Cell	Cells Constructed
0.5°	5.74	2,852
1°	13.54	858
2°	36.09	272
3°	64.14	147

We also employ a set of spatial datasets describing areas and points of interest, as follows:

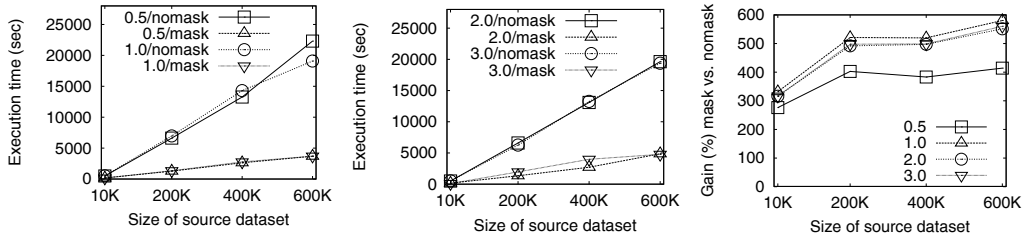
- Natura2000 regions: Natura 2000 is an index of protected regions for biodiversity in the European Union. It is set up to ensure the survival of Europe's most valuable species and habitats, indexing 3,522 polygons.
- Fishing areas: This dataset includes 5,076 polygons generated from raster images depicting the fishing intensity in European waters (as reported by European Union).
- The World Port Index (WPI) including a listing of 3,685 ports throughout the world, describing their location, characteristics, known facilities, and available services. Ports in this dataset are represented as points, and WPI is used for discovering proximity relations between vessel positions and ports.

Metrics. Our main metric is the achieved throughput of the stream reasoning system, assessed by delving into the individual performance of its two constituent components, stLD and CER. Other auxiliary metrics are additionally presented, in order to explain performance, such as processing time and recognition time for stLD and complex activity recognition respectively. We also measure the number of *unrewarding comparisons* of each LD configuration, i.e., the number of comparisons that did not result in a relation.

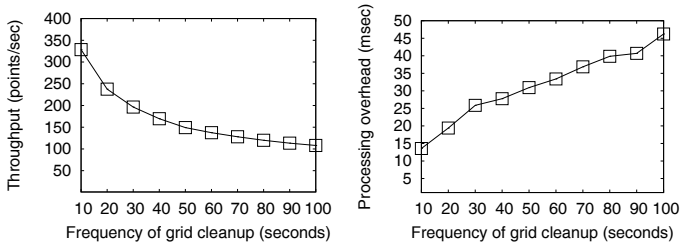
Parameters. In the experiments evaluating the performance of link discovery, we vary the input size and granularity of the grid (i.e., cell size). In the case of stream to static LD (i.e., "within" and "nearby" relations between moving objects and regions), we setup multiple equi-grid configurations varying in granularity {0.5°, 1°, 2°, 3°} (degrees in latitude/longitude), using the datasets of Natura2000 and fishing regions, totaling 8,599 regions. An equi-grid of 0.5° granularity means that the size of a cell is $0.5° \times 0.5°$. Notice that we construct the grid over the complete static datasets, as typically done in link discovery tasks. We evaluate the performance and scalability of stLD, using subsets of critical points datasets, i.e., {10K, 200K, 400K, 600K} entries. Also, we evaluate the gain obtained by using the mask technique (for the stream to static link discovery method of Section 3.1), and using the bookkeeping technique (for the stream to stream link discovery method of Section 3.2). All topological relations (such as within, but also those required for computing the mask) are provided by the Java Topology Suite (JTS) v.1.13. Finally, we evaluate the effect of varying the number of cores on CAR, as well as the effect of increasing the window size.

5.2 Link Discovery

Stream to Static LD. We refer to the areas organized in a grid as *target* dataset, whereas *source* dataset refers to the dataset of streaming vessel positions. We use as *target* dataset the regions of Natura2000 and fishing areas, totaling 8,599 areas. This static data is organized off-line in main-memory using grid configurations of varying granularity for the complete geographical space covered by the areas. For the streaming *source* dataset, we employ subsets



(a) Total execution time. **(b)** Total execution time. **(c)** Gain (%) of using mask.



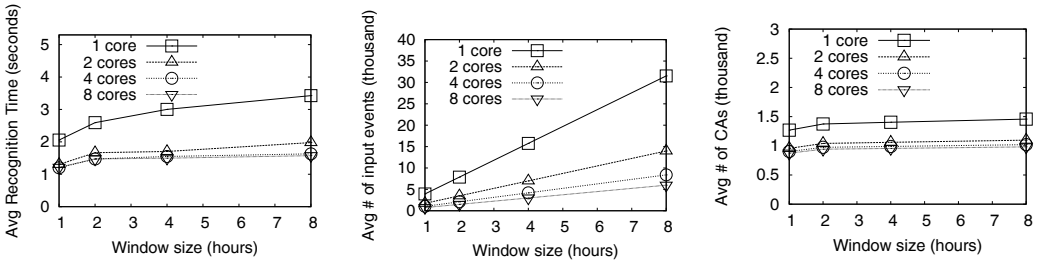
(d) Throughput. **(e)** Overhead of grid cleanup.

Figure 3 Spatio-temporal LD. Stream to static: **(a)**–**(c)**; Stream to stream: **(d)** and **(e)**.

of the critical points dataset of different sizes, in order to evaluate its effect. The distribution of areas for each grid configuration is shown in Table 3. The second column shows the average number of areas in a cell, while the third column indicates the number of non-empty cells that were constructed (i.e., hold at least one of the given areas).

Figures 3a–3c show the benefits of using the proposed technique with mask. Figure 3a depicts the total execution time required for processing a subset of critical points, whose size is indicated in the x-axis, using two grids with granularity 0.5° and 1.0° respectively. We observe that the mask technique achieves better execution time for all grid configurations and input sizes. Figure 3b shows both grid configurations of granularity 2.0° and 3.0°. In terms of throughput, the use of the mask achieves to increase the throughput up to a factor of 5, compared to not using the mask. Figure 3c quantifies this gain, as the ratio of unrewarding comparisons, i.e., the number of unrewarding comparisons without mask, divided by the number of unrewarding comparisons with mask. For instance, the ratio of unrewarding comparisons for input size 200K and granularity 0.5° is 402.7%. This result clearly demonstrates the advantage of using the mask technique.

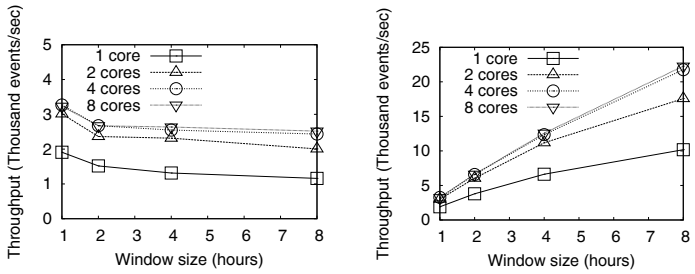
Stream to Stream LD. We evaluate the effect of the grid cleaning technique on throughput, using a temporal threshold of 30 seconds. Figure 3d shows how throughput is affected when varying the frequency of grid cleanup from 10–100 sec. The chart shows that the more frequent cleanup is performed, the higher the achieved throughput. When the cleanup is delayed, the processing time spent on comparisons is increased, affecting the total execution time, and decreasing the average throughput. The next question to be answered concerns the overhead imposed by frequent grid cleanup. Figure 3e shows the average time spent for grid cleanup in milliseconds. We observe that the overhead is very small, and increases when the cleaning is not performed regularly. Recall that the bookkeeping structure is a list of spatio-temporal positions of vessels ordered by their temporal values, and the disposal of part of the list on each call allows us to avoid searching every cell of the grid for "expired" entries. In summary, the cleanup operation has minimal overhead, which makes it applicable with high frequency, thereby increasing the achieved throughput.



(a) Average recognition time.

(b) Average number of input events.

(c) Average number of recognised activities.



(d) Throughput for slide step $\beta = 1$ hour.

(e) Throughput for slide step $\beta = \omega$.

Figure 4 Complex activity recognition.

5.3 Complex Activity Recognition

RTEC performs continuous query processing, computing the maximal intervals of fluents representing complex maritime activities. At each query time Q_i, the input events that fall within a specified sliding window ω are taken into consideration. All input events that took place before or at $Q_i - \omega$ are discarded/"forgotten". This way, the cost of reasoning is independent of the data stream size. At Q_i, the complex activity intervals computed by RTEC are those that can be derived from input events that occurred in the interval $(Q_i - \omega, Q_i]$, as recorded at time Q_i. When the range ω is longer than the slide step β, it is possible that an input event occurs in the interval $(Q_i - \omega, Q_{i-1}]$ but arrives at RTEC only after Q_{i-1}; its effects are taken into account at query time Q_i. Window parameters for ω and slide step β are defined by the domain experts along with the maritime patterns, since they reflect the time horizon over which interesting phenomena may be detected. Usually, ω spans many minutes or even hours for capturing meaningful events across a vessel's route.

Figure 4 presents the experimental results for increasing window sizes ω: 1 hour to 8 hours. Figure 4a presents the average recognition time per window ω, while Figures 4b and 4c show, respectively, the average number of input events and recognized activities per window. Recall that Table 1 lists the types of input event and recognized activity. The slide step β is set to 1 hour. The empirical analysis shows that RTEC is capable of real-time maritime activity recognition – e.g. RTEC recognizes ($\approx$ 1,500) maritime activities given a window of 8 hours ($\approx$ 31,000 input events) in less than 3.5 sec, even when operating on a single processing core of the desktop computer. (The use of multiple cores will be discussed shortly.) Figure 4d presents the throughput. As the window size ω increases, throughput decreases since the average recognition time per window increases (see Figure 4a). Figure 4e presents the throughput when the slide step is the same as the window size ($\beta = \omega$), i.e. windows are non-overlapping. In this case, the average recognition time per window increases, but only

very slightly (due to the higher cost of "forgetting" more past events), and thus not presented here. (The average numbers of input events and recognized activities per window are also similar to the case of $\beta = 1$ hour). Figure 4e shows that, as the window size ω increases, throughput increases. When ω increases, the number of windows/query-times decreases (we have significantly less windows than the case of $\beta = 1$ hour), while the average recognition time per window increases only slightly.

We also run RTEC in parallel, by launching different instances of the engine, each operating on a different processing core. Each RTEC instance was responsible for activity recognition in a separate sub-area of the dataset, receiving input events only from vessels in that sub-area. To avoid false negatives on the borders of the sub-areas, we allowed for some overlap. Figure 4 presents the results when 2, 4 and 8 processing cores are used. Figure 4a presents the average recognition of the worst-performing RTEC instance, while Figures 4b and 4c show the average input events and recognised activities for that instance. Figures 4d and 4e present the throughput. As shown in Figure 4, the benefits of parallelization can be significant. A more refined segmentation of the dataset into sub-areas, optimizing load allocation, would have had more profound effects on performance.

6 Discussion

We presented a stream reasoning system for maritime monitoring, which supports complex activity recognition assisted by online spatio-temporal discovery of relations between vessels and areas of interest. Compared to earlier work, the proposed system optimizes the computation of spatial relations, leading to improved system performance. Our experimental evaluation on real datasets demonstrated the efficiency of the proposed prototype.

Even though the topic of link discovery has attracted much interest lately (see [15] for a recent survey), there is not much work on spatio-temporal link discovery nor on link discovery over streaming data. Our work tackles explicitly these topics. Generic LD frameworks include LIMES [17] and SILK [11]. LIMES [17] is an LD framework for metric spaces that uses the triangular inequality in order to avoid processing all pairs of objects. LIMES employs the concept of exemplars, which are used to represent areas in the multidimensional space, and prunes entire areas (and the respective enclosed entities) during link discovery. SILK [11] is an LD framework proposing a novel blocking method called MultiBlock, which uses a multidimensional index. The spatial LD methods [16, 22] apply grid partitioning (a.k.a. space tiling) on sources A and B, in order to perform efficiently the *filtering step*. Then, in the *refinement step*, different optimizations are employed in order to minimize the number of computations necessary to produce the correct result set. However, all these works focus on topological relations, and it is not straightforward to extend them for proximity relations.

Various languages and systems have been proposed in the literature for complex event/activity recognition [7, 1]. RTEC has a formal, declarative semantics – activity patterns in RTEC are (locally) stratified logic programs. In contrast, most complex event processing languages, event query languages, data stream processing languages and commercial production rule systems, rely on an informal and/or procedural semantics [8]. Furthermore, RTEC explicitly represents complex activity intervals (unlike e.g. [9, 8, 4]) and thus avoids the related logical problems [18]. Maritime activities form hierarchies, in the sense that the formulation of one activity is used to define other, higher-level activities. We defined e.g. potentially illegal fishing in terms of slow motion (recall rule (3)). In contrast to state-of-the-art recognition systems, such as the Esper engine (`http://www.espertech.com/esper/`) and SASE (`http://sase.cs.umass.edu/`), RTEC can naturally express hierarchical knowledge

by means of well-structured specifications, and consequently employ caching techniques to avoid unnecessary re-computations [3]. Concerning the Event Calculus literature, a key feature of RTEC is that it includes a windowing technique. No other Event Calculus system (including [6, 5, 13, 19, 2, 14]) "forgets" or represents concisely the data stream history.

References

1 E. Alevizos, A. Skarlatidis, A. Artikis, and G. Paliouras. Probabilistic complex event recognition: A survey. *ACM Comput. Surv.*, 50(5):71:1–71:31, 2017. doi:10.1145/3117809.

2 Alexander Artikis and Marek J. Sergot. Executable specification of open multi-agent systems. *Logic Journal of the IGPL*, 18(1):31–65, 2010.

3 Alexander Artikis, Marek J. Sergot, and Georgios Paliouras. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.*, 27(4):895–908, 2015.

4 H. Beck, M. Dao-Tran, and T. Eiter. LARS: A Logic-Based Framework for Analytic Reasoning over Streams. Technical Report INFSYS RR-1843-17-03, Institute of Information Systems, TU Vienna, 2017.

5 Iliano Cervesato and Angelo Montanari. A calculus of macro-events: Progress report. In *Proceedings of TIME*, pages 47–58, 2000.

6 L. Chittaro and A. Montanari. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence*, 12(3):359–382, 1996.

7 G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44(3):15, 2012.

8 Gianpaolo Cugola and Alessandro Margara. TESLA: a formally defined event specification language. In *Proceedings of DEBS*, pages 50–61, 2010.

9 C. Dousson and P. Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchisation. In *Proceedings of IJCAI*, pages 324–329, 2007.

10 Bilal Idiri and Aldo Napoli. The automatic identification system of maritime accident risk using rule-based reasoning. In *Proceedings of SoSE*, pages 125–130, 2012.

11 Robert Isele, Anja Jentzsch, and Christian Bizer. Efficient multidimensional blocking for link discovery without losing recall. In *Proceedings of WebDB*, 2011.

12 Robert A. Kowalski and Marek J. Sergot. A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95, 1986.

13 R. Miller and M. Shanahan. Some alternative formulations of the event calculus. In *Computational Logic: Logic Programming and Beyond*, LNAI 2408, pages 452–490. Springer, 2002.

14 M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. van der Aalst. Monitoring business constraints with the event calculus. *ACM TIST*, 5(1):17:1–17:30, 2013.

15 Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. A survey of current link discovery frameworks. *Semantic Web*, 8(3):419–436, 2017.

16 Axel-Cyrille Ngonga Ngomo. ORCHID - reduction-ratio-optimal computation of geo-spatial distances for link discovery. In *Proceedings of ISWC*, pages 395–410, 2013.

17 Axel-Cyrille Ngonga Ngomo and Sören Auer. LIMES - A time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of IJCAI*, pages 2312–2317, 2011.

18 A. Paschke. ECA-RuleML: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics. Technical Report 11, Technische Universität München, 2005.

19 Adrian Paschke and Martin Bichler. Knowledge representation concepts for automated SLA management. *Decision Support Systems*, 46(1), 2008.

20 K. Patroumpas, E. Alevizos, A. Artikis, M. Vodas, N. Pelekis, and Y. Theodoridis. Online event recognition from moving vessel trajectories. *GeoInformatica*, 21(2):389–427, 2017.

21 T. Przymusinski. On the declarative semantics of stratified deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*. Morgan, 1987.

22 Mohamed Ahmed Sherif, Kevin Dreßler, Panayiotis Smeros, and Axel-Cyrille Ngonga Ngomo. Radon - rapid discovery of topological relations. In *Proceedings of AAAI*, pages 175–181, 2017.

An Empirical Study on Bidirectional Recurrent Neural Networks for Human Motion Recognition

Pattreeya Tanisaro
Institute of Cognitive Science, University of Osnabrück, Germany
pattanisaro@uni-osnabrueck.de

Gunther Heidemann
Institute of Cognitive Science, University of Osnabrück, Germany
gheidema@uni-osnabrueck.de

Abstract

The deep recurrent neural networks (RNNs) and their associated gated neurons, such as Long Short–Term Memory (LSTM) have demonstrated a continued and growing success rates with researches in various sequential data processing applications, especially when applied to speech recognition and language modeling. Despite this, amongst current researches, there are limited studies on the deep RNNs architectures and their effects being applied to other application domains. In this paper, we evaluated the different strategies available to construct bidirectional recurrent neural networks (BRNNs) applying Gated Recurrent Units (GRUs), as well as investigating a reservoir computing RNNs, i.e., Echo state networks (ESN) and a few other conventional machine learning techniques for skeleton-based human motion recognition. The evaluation of tasks focuses on the generalization of different approaches by employing arbitrary untrained viewpoints, combined together with previously unseen subjects. Moreover, we extended the test by lowering the subsampling frame rates to examine the robustness of the algorithms being employed against the varying of movement speed.

2012 ACM Subject Classification Mathematics of computing → Time series analysis

Keywords and phrases Recurrent Neural Networks, Human Motion Classification, Echo State Networks, Motion Capture, Bidirectional Recurrent Neural Networks

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.21

1 Introduction

The recurrent neural networks, whose structures are similar to those of multilayer perceptrons (MLPs) have been widely used for sequential data processing. Nonetheless, they are different from the MLPs by allowing connections among hidden units, therefore the networks can retain information of past inputs as a vector of activation for each time step which makes RNNs exceedingly deep. Their depth, however, makes them difficult to train because the update of the weight matrices with a gradient-based approach such as Backpropagation Through Time (BPTT) leads to exploding and vanishing gradient problems [1]. Many techniques have been introduced in order to solve these two issues, especially for the vanishing gradient problem, but training RNNs was still a very difficult task and the applications were limited.

Since the emergence of special architecture for gradient-based methods called LSTM [17], training RNNs has become easier and more successful in numerous tasks such as speech recognition [12], acoustic modeling [24], sequence labeling in speech recognition [13], handwriting recognition [15], language modeling and machine translation [30, 31, 4], prediction of successful shooting in basketball [28], analyzing motion patterns in autonomous driving [11], image caption [36] and learning of video representation [29]. The LSTM is

designed to solve the vanishing gradient problem whereas truncating the gradient is harmless to the networks because an LSTM can enforce a constant error flow within special units to bridge time lags. Unlike the traditional recurrent unit which calculates the weighted sum of inputs and directly applies the activation function, the LSTM unit contains a memory cell. Furthermore, there are several studies such as [26] and [18] that reported their achievement of an improvement of the output performance by introducing the depth to the RNNs. Typically, any RNNs when unfolded in time might be considered deep themselves, because the input to output in a given time span has crossed several nonlinear layers as computational paths [26]. Nonetheless, the *depth* of neural networks is usually defined by the number of feedforward neural layers. Most studies in deep RNNs concentrate on sequence-to-sequence modeling, particularly for language modeling for instance [30, 16, 18, 26, 6].

In our study, we focused on solving a classification problem using a special type of deep RNNs, called bidirectional RNNs (BRNNs) which were first introduced by [27] in the late 1990s. Nevertheless, they started to attract attention many years later after a groundbreaking achievement of sequence labeling in speech recognition by [13]. The BRNN is an RNN which contains a separation of a forward and backward pass for positive and negative time direction. Therefore, it is able to store the past and future context, whereas a conventional RNN can only partially achieve this by delaying the output by certain time steps. Our study is set up by employing more than 300 configurations for deep BRNNs after the preliminary tests, of which we inspect how the classification performance is affected by changing the ***width*** and the ***depth*** of the hidden layers. The designed networks are set up in a generic sense by simply stacking multilayer RNNs to have the required depth. The evaluation is based on three Motion Capture datasets for comparing BRNNs with ESNs [19, 20] and traditional machine learning techniques. Motion Capture (MoCap) is a marker-based system which by its high-dimensional nature, nonlinearities and long-range dependencies make it ideal for studying the limitations of time series models [35]. Although the focus of our study is on this particular domain, the design of BRNNs is not just solely specific for MoCap datasets. We are convinced that the study is also applicable to other high dimensional time series data.

2 Related Work

Many studies have demonstrated a superior functionality of applying deep RNNs when compared to shallow networks, for instance, [30] introduced a new architecture called multiplicative RNNs by using multiplicative connections to allow the current input character for the character-level in language modeling to determine the hidden-to-hidden weights. However, this model was trained with Hessian-Free optimizer (HF) instead of gradient descent. The work of [16] focused on a hierarchy of RNNs for character-level language modeling using stochastic gradient descent. It proposed two alternative architectures which are deep MLPs with three hidden layers stacked from one layer on top of each other with temporal feedback loops. One architecture uses feedback loops from output but with the last hidden layer contributing to the output layer, while another architecture allows all the connections from each hidden layer contributing to the output layer. Four other different models to construct deep RNNs have been proposed by [26] for three language models. Quite recently, a hierarchical multiscale RNN model has been presented by [6]. It shows that the proposed network architecture can learn the latent hierarchical multiscale structure from temporal data for character-level language modeling. A similar study to our work which employs bidirectional RNNs for classification tasks has been discussed in [14]. It used five hidden layers of BRNNs with LSTM to classify 61 phoneme outputs in which each layer

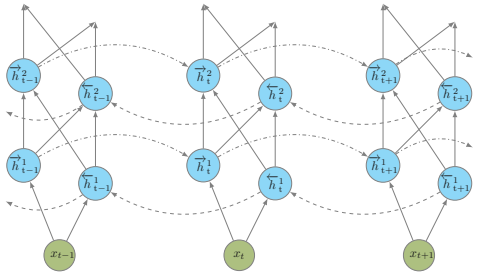


Figure 1 A deep BRNN with two hidden layers following [14]. The *dashed-dotted lines* indicate the **forward** direction depicted by $\overrightarrow{h}_t$ and the *dashed lines* indicate the **backward** direction $\overleftarrow{h}_t$.

consists of $2 \cdot 250$ cells. A comprehensive comparative study of deep RNNs has been revealed in [12]. It demonstrated the results of using from one to five hidden layers while fixing the number of neurons for all hidden layers. The results from this experiment exhibited that: i) LSTM works better than the typical *tanh* neuron, ii) bidirectional RNNs with LSTM also give better output performances than typical unidirectional RNNs with LSTM units, and iii) the depth size is more important than the width size. In addition, by fixing the number of neurons of each hidden layer, the networks with three hidden layers work as well as those with five layers, while the number of weights of five hidden layers is almost twice their number for three layers. Furthermore, the evaluation in [18] also confirms that shallow BRNNs outperform shallow unidirectional RNNs on extracting sentence-level opinion expression. It concludes that, for a large network, three hidden layers provide the best output performance for their tasks. In case of a small network, two, three and four hidden layers show equally good performance for certain sizes. By adding more layers, its performance decreases. Further, the study suggests that in conventional stacked deep learners, every hidden layer conceptually lies in a different representation space, and establishes a more abstract and higher-level representation of the input. By taking these findings as our guidelines, we then hypothesized that the activities at each layer could represent some forms of the action descriptors.

3 Classification Approaches

3.1 Deep Bidirectional RNNs

Generally, the BRNN has been applied for sequence-to-sequence learning, particularly for Natural-Language Processing (NLP) tasks. The structure of the BRNN consists of two RNNs, one to compute the forward hidden sequences $\vec{h}_t$ and the second is to compute backward hidden sequences $\overleftarrow{h}_t$. Figure 1 shows an architecture of the BRNN for two hidden layers. Let $x = (x_1, .., x_T)$ be an input sequence for T time steps. The final output y_t is accumulated across the T frames at the last layer and is classified by the probability of human action classes using the Softmax function. We computed the output sequence at time t of y according to [14] as:

$$y_t = g(W_{\overrightarrow{h}y}\vec{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y) \tag{1}$$

$g(\cdot)$ is an activation function, $W_{\overrightarrow{h}y}$ and $W_{\overleftarrow{h}y}$ are weight matrices at the output layer and b_y is an output bias. The $\overleftarrow{h}_t$ can be interpreted as a summary of the past from $t = T$ to 1, whereas $\vec{h}_t$ is the summary of the future from $t = 1$ to T. The activities in forward and

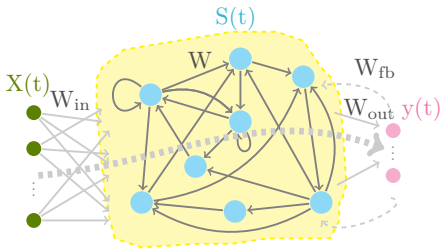


Figure 2 Architecture of an ESN. The *dashed lines* denote the connections which are not compulsory.

backward direction can be written by:

$$\vec{h}_t^n = f(W_{\vec{h}^{n-1}\vec{h}^n}\vec{h}_t^{n-1} + W_{\vec{h}^n\vec{h}^n}\vec{h}_{t-1}^n + b_{\vec{h}}^n) \tag{2}$$

$$\overleftarrow{h}_t^n = f(W_{\overleftarrow{h}^{n-1}\overleftarrow{h}^n}\overleftarrow{h}_t^{n-1} + W_{\overleftarrow{h}^n\overleftarrow{h}^n}\overleftarrow{h}_{t+1}^n + b_{\overleftarrow{h}}^n) \tag{3}$$

where h^0 is the input sequence.

3.2 Echo State Networks

An ESN shown in figure 2 is a type of RNN, whose design does not depend on updating weights by gradient computation, but, instead it creates a random *dynamical reservoir RNN*. The reservoir is then driven by the training data and leaves the weights untrained. The output weights are computed at the readout connection using a linear regression of $y(t)$. The internal unit activities $\vec{S}$ in figure 2 can be updated by:

$$\vec{S}(t) = f(W_{in}\vec{x}(t) + W\vec{S}(t-1) + W_{fb}\vec{y}(t)) \tag{4}$$

$f(\cdot)$ is an activation function of the neurons, a common choice is $tanh(\cdot)$ applied element-wise. By employing the time warping of the input signals, the *leaky integration rate* [21] $\alpha \in (0, 1]$ is adopted to determine the speed of the reservoir update dynamics. The update rule for the internal units is extended to:

$$\vec{S}_{leaky}(t) = (1-\alpha)\vec{S}(t-1) + \alpha\vec{S}(t). \tag{5}$$

Applying a simple linear regression at the readout layer leads to output $\vec{y}(t)$:

$$\vec{y}(t) = W_{out}[\vec{x}(t); \vec{S}(t)] \tag{6}$$

3.3 Traditional Machine Learning Methods

Most work on time series classification for example, the well-known UCR archive, of which all datasets are of one dimensional feature, focuses on an adaptation of distance measures using 1-Nearest Neighbor (1-NN) with Euclidean Distance (ED) and Dynamic Time Warping (DTW). Both techniques have proven to perform very well on the UCR archive, especially DTW. Nevertheless, DTW has limited its applications only on the fixed length data (that is one must extrapolate the shorter sequence in order to have an equal length between two sequences) and gives rise to some issues for the case of multi-dimensional data e.g., the computational complexity and the selection of the dependent or independent warping distance function [32].

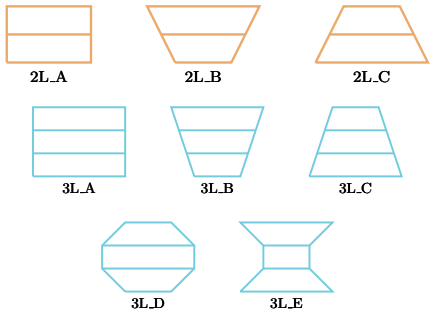


Figure 3 Different geometries representing eight models used in our experiment with varying width of two and three hidden layers.

We adopted two transformation approaches to extract the feature vectors in our experiments. They are **i) a naïve method** by stacking each frame on top of one another, and using majority voting to decide the best course of action, and **ii) a dimensionality reduction technique** of the zero-mean skeleton configuration for feature vectors demonstrated in [34, 23]. The two best classifiers for MoCap classification, cited in [34], k-NN and Random Forest (RF), were chosen for both transformation approaches.

4 Configurations

4.1 BRNN architectures

The RNNs with one hidden layer (**1L**) represent shallow networks, while those with more than one hidden layers represent the deep RNNs. According to [14] and [18], three hidden layers are sufficient to achieve the best performance, while adding more hidden layers worsens the output performance. On this account, in our experiment, we concentrated on evaluating geometries of the networks with two and three hidden layers as illustrated in figure 3. RNNs with three stacked hidden layers numbered from bottom to top labeled as ($\mathbf{L_1} \cdot \mathbf{L_2} \cdot \mathbf{L_3}$) will be referred to throughout the experiment indicating the number of units in **one direction**. (We use the term cell or unit instead of neuron to emphasize the use of gated units in RNNs.) Model *2L_A*, as seen in the figure 3 is depicted for the two hidden layers in which the number of cells of the two hidden layers is almost equal. The model *2L_B* is for two hidden layers, of which the number of neurons of L_2 (top) is at least double the size of L_1 (bottom), and vice versa for model *2L_C*. Note that the first hidden layer ($\boldsymbol{L_1}$ connects to the input nodes laying at the bottom. For models with three hidden layers, we extended the geometries to five architectures as illustrated in figure 3. Here, we combined cell numbers in $\{50, 100, 150, 200, 250, 300, 350, 400\}$ to form these geometries which have a limited total amount of cells from $2 \cdot 200$ up to $2 \cdot 600$ units.

4.2 BRNN configurations and hyperparameters

In order to verify the impact of the width and the depth on the network as well as to simplify the experiment, several parameters in the experiment had been previously investigated. They are: **i) Cell type**. In our experiment, instead of a well-known LSTM, we replaced each cell unit at h with a gated recurrent unit called GRU [4]. GRU is a variation of a gating mechanism and is comparable to LSTM and has been primarily used in machine translation as encoder-decoder models. It is similar to LSTM in that the gating units modulate the

flow of information inside the unit, but without memory cells. Comparative studies of using traditional recurrent unit *tanh*, LSTM and GRU, found in [7] and [22] have shown that the gated units, both the LSTM and GRU outperform the conventional recurrent unit. To achieve certainty, we had examined these three cells in BRNNs in our preliminary tests. The networks with GRU cells significantly outperformed the other two cells by more than 5% in all experiments. Therefore, our classification results were from applying the GRU units to the networks. **ii) Optimizer**. In contrast to the optimizer benchmark for RNNs in Penn TreeBank language modeling [8] which mentions that Adam and RMSProp do not work well with RNNs, we found that in our case, both of them converged very quickly even with a very small learning rate and gave good output performance. The primary test was carried out for a shallow BRNN. The selected learning rates for each optimizer here were the recommended values in the papers based on MNIST dataset. For the rest of the experiments, we chose RMSProp as our default optimizer which outperformed all other optimizers. **iii) Regularization**. Because of the limited amount of data, we cannot take out some data for validation. Nonetheless, we added a regularization term L^2 to the objective function with a fixed regulation $\lambda = 0.02$. It is interesting to note that increasing the regularization parameter from 0.01 to 0.02 increases the recognition rate by about 3-5% in most models. We applied the norm clipping with a maximum gradient norm limited to one, and no *dropout* was applied.

4.3 ESN configurations

The ESN configurations in the experiments follow the guidelines suggested in [33] which demonstrated the influence of the ESN settings on various datasets on the UCR archive. Several key parameters are: **i) Sparsity of the reservoir**. In corresponding with BRNN networks which have the networks size in $2 \cdot \{200, .., 600\}$, we set the reservoir size using only half of BRNN with connectivity of 10, 30, 50 and 70% respectively. **ii) Spectral radius** which is considered to be big for the tasks that require an extensive history of an input, while one is served as a reference point. We picked 5.0 from [33]. **iii) Leaky rate** which can be regarded as time warping of the input signals was fixed at 0.1 for all configurations. **iv) Input scaling**: was set to 2.0 similar to [33] and **v) regularization coefficient** was fixed at 0.1. Moreover, the network weight was set to have a uniform distribution in the range of [-0.5,0.5] and no feedback connection was considered here.

4.4 Traditional machine learning configurations

For the direct naïve method, we employed two popular classifiers for classification, 1-NN and RF with 75 trees, which yielded best output performance in [34]. To prove the case of a combined manifold learning with classification, we chose the PCA with two and three components associated with RF and 1-NN.

5 Datasets and Experimental Setups

5.1 Datasets

We evaluated the proposed techniques as described in section 3 and 4 using three MoCap datasets, MHAD-27, MHAD-10 and HDM05.

MHAD-27[1] [2] consists of 27 different actions performed by eight subjects. Each subject repeated the same action four times. The dataset contains a total of 861 data sequences, where three sequences were corrupted and removed from the dataset on the official website.

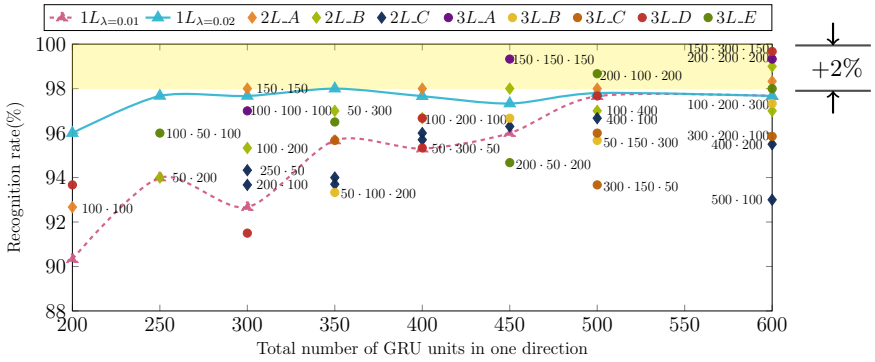


Figure 4 The recognition rates of MHAD-10 from designated network architectures using five-fold cross validation. The setup does not condition on the separation of test subjects from the training set.

The training was performed on six subjects, and two subjects were left out for the test. The recognition rate was reported on an average of 28 combinations. This dataset was recorded using 20 markers. Therefore, we have the feature vector which is captured in 3D space for BRNN of size $(3 \cdot 20) \times 120$, where 120 is the amount that is close to the maximal sequence length of this dataset. Zeros were appended to the shorter sequences in BRNNs. It is important to note that ESN can handle different lengths of the data sequences, so the data is trained with the original length.

MHAD-10 [2] is a 3D MoCap dataset of six subjects performing 10 different hand gestures tracked with 25 markers. The four additional markers comparison with MHAD-27 were put on the left and right hand and a thumb, and one additional marker was put on a spine. Each subject repeats an action 5 times (trials). Therefore, we have a total of 300 videos with various sequence lengths. For BRNN, we chose to fix the number of sequences to 150 which is close to the maximal length of the sequence in the dataset. This makes a feature vector size of $(3 \cdot 25) \times 150$. The training is performed on five subjects from all trials and an unseen subject is left out for testing. Hence the recognition rate is an average of six-fold cross-validation.

HDM05 [25] was originally made up of 130 classes consisting of five subjects performing actions with and without repeating the same cycles separately. This created a total of 2343 sequences. We followed [5, 9, 38] in grouping non-repetitive and repetitive motions together yielding 65 actions. There were about 20 actions which have samples less than 20 i.e., *throwBasketball*, *throwFarR* and *jumpDown* having only 14 trials each, while actions such as *walk*, *elbowToKnee* and *runOnPlaceStart* have 94, 80, 74 trials, respectively. This leads to an unbalancing of data and causes a huge bias towards a particular action. Nonetheless, since we focused on the action recognition of unseen subjects, therefore four subjects were used in the training set and one subject was for the test. We reduced the original number of markers to 19, where some nearby sensors e.g., on the spine as used in MHAD were merged. The average sequence length is 261 with the maximum of 901. With the limitation of our computational capacity, we set the data using the fixed length of 400 for BRNNs. Therefore, the feature vectors of BRNNs have a size of $(3 \cdot 19) \times 400$.

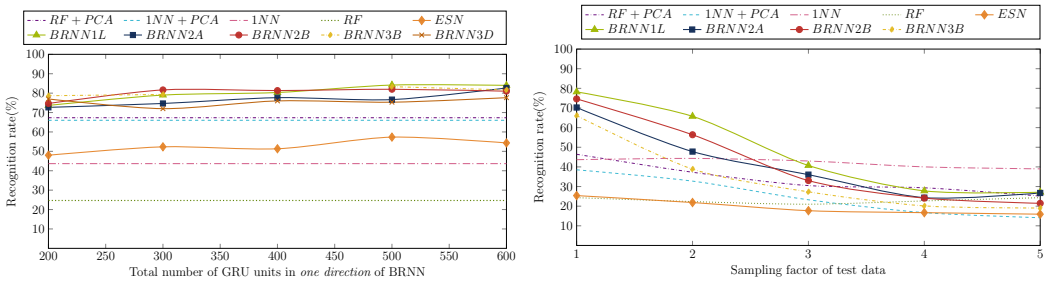


Figure 5 The recognition rates of MHAD-10 by excluding test subjects from the training set. **Left)** Changing the total number of GRU units in one direction of BRNN or the reservoir size of ESN. **Right)** Extending the test on the left by changing subsampling factors of test data of untrained viewpoints.

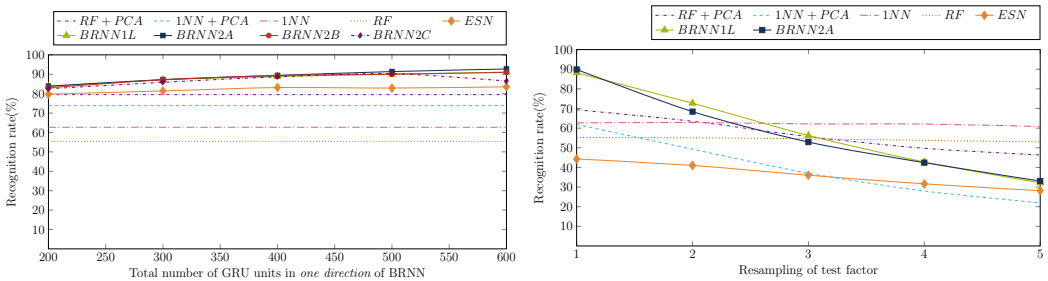


Figure 6 The recognition rates of MHAD-27 by excluding test subjects from the training set. **Left)** Changing the total number of GRU units in one direction of BRNN or the reservoir size of ESN. **Right)** Extending the test on the left by changing subsampling factors of test data of untrained viewpoints.

5.2 Experimental Setups

We normalized each sequence with respect to its in-frame reference of that dataset, where the reference is the joint that laid at the center of the skeletal torso. The evaluations were composed of three experiments: **i) Having some insights of deep networks strategies by varying the width and height of the network.** In this experiment, we did not impose conditions on separation of test subjects from the training set. **ii) Finding a few good models by varying the number of cells in the networks using unknown subjects**. The experiment was set up in a way to find a few good models of each dataset by varying the number of cells in the networks in 200-600 units (one direction in BRNN and the total units in a reservoir for ESN), where the test subjects were excluded from the training set. More than 300 configurations for BRNN were constructed to obtain the best output and created some insights of construction strategies for deep networks. **iii) Extending the test using untrained viewpoints with the variations of speed.** We enhanced the experiment by extending the training set to have five camera angles $\{-90, 45, 0, 45, 90\}$, whereas test angles are in $\{-80, -70, ..., 70, 80\}$. Moreover, we subsampled the original test data using a subsampling factor of 1, 2, 3, 4 and 5, while the training data still remained the same. The subsampling factor of 2 means that every 2^{nd} frame of the data will be taken instead of each single frame (factor of 1).

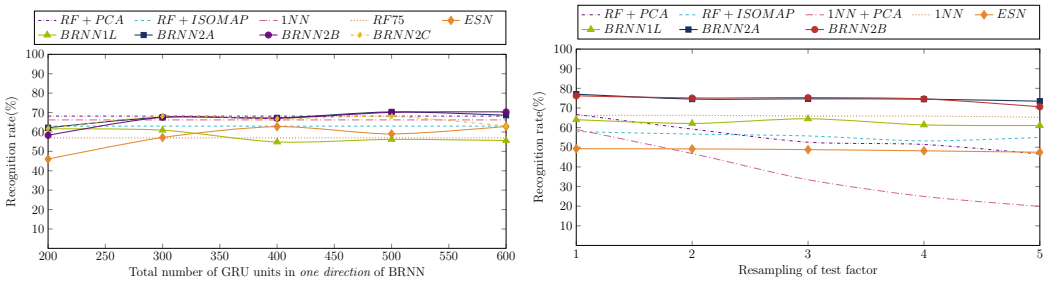


Figure 7 The recognition rates of HDM05 by excluding test subjects from the training set. **Left)** Changing the total number of GRU units in one direction of BRNN or the reservoir size of ESN. **Right)** Extending the test on the left by changing subsampling factors of test data of untrained viewpoints.

6 Experimental Results

6.1 Discussion of results

Firstly, we investigated the effects of geometries of BRNNs following the **construction strategies** depicted in Figure 3. The recognition rates of MHAD-10 performing on average of five-fold cross-validation using shallow and deep BRNNs are shown in Figure 4. Moreover, the recognition rates of each strategy are the average of two runs with the standard deviation of $\pm 3\%$. These test results gain very high recognition rates because they are not based on the separation of test subjects from the training set. It is obvious that the recognition rates which are better than 98% (the yellow shaded area in Figure 4) can only be achieved when the networks are relatively large i.e., the number of cells is greater than $2 \cdot 400$. Furthermore, the models which have any layer containing 50 cells yield output worse than others. This might be because the input feature of MHAD-10 has a size of 75 and any form of dimensionality reduction at any hidden layer in RNNs by shrinking the network's width is not suitable. Hence, by the experimental results, we conclude that the width of a hidden layer next to the input layer in one direction should be larger than the size of the input features.

We enhanced the experiment by **excluding test subjects from the training set to examine the generalization of the models**. The recognition rates of three datasets for two experimental setups of MHAD-10, MHAD-27 and HDM05 can be found in Figure 5, 6 and 7 on the left and right of the figures respectively. In these figures, only a few best models of BRNN were chosen from various configurations based on five geometries in Figure 3. Furthermore, the results also demonstrate the output of the shallow networks, the output of ESNs in section 3.2 and the output of machine learning approaches from section 3.3.

The results of the first experiment displayed on the left of Figure 5, 6 and 7 show that the shallow and deep BRNNs significantly outperform other methods in the MHAD-10 and MHAD-27 dataset. The deep BRNNs are slightly better than the other methods on large networks for HDM05. In addition, there are some considerable differences in recognition rates among both datasets and algorithms, especially for MHAD-10. Even though MHAD-10 consists of only 10 actions, nine out of these actions are the movements of solely the right hand. On the contrary, even though MHAD-27 consists of more actions than MHAD-10, the actions also involve a variety of movements of hands and legs which leads to overall better recognition rates of all methods. The dimensionality reduction techniques in combination with RF outperform other machine learning techniques and are close to the winner of HDM05 using deep networks, *BRNN2C* using the total of $2 \cdot 600$ cells. The confusion matrix of

HDM05 using *BRNN2B* model on Figure 7-left is depicted in Figure 8. Actions with one hundred percent recognition rate (recall) from all runs (filled with dark brown in Figure 9) are for instance, *clapAboveHead*, *jogLeftCircle*, *rotateArmsBothBackward*, *standUpKneelToStand* and so forth. The most common misclassified actions in all classifiers are between *deposit* and *grab* as the trajectories of actions drawn in Figure 9. It is difficult to track the trajectories in 3D of a stationary image by eye; hence, we projected a 3D image onto a 2D plane and as we can see, the trajectories from these two groups cannot be distinguished. Therefore, when we allow classification using top three correctness, the recognition rates increase by 12-15% in all methods. For HDM05, there are no significant differences among different classification methods. When applying subsampling factors to **simulate the changes of movement speed** for the test data of MHAD-10 and MHAD-27 using untrained viewpoints, then increasing the subsampling factor decreases the recognition rate. Interestingly, however, this effect does not apply to HDM05. This might be because only HDM05 consists of non-repetitive and repetitive sequences in one action which allows the networks to easily capture the changes of patterns of the action as varying of movement speeds, while MHAD-10 and MHAD-27 only consist of one periodic movement in each action. Besides, by increasing the number of viewpoints in training deep BRNNs, the recognition rates of HDM05 have been increasing by approximately 10% on arbitrary untrained viewpoints.

The results also reveal that deep BRNNs using two layers for the total number of cells greater than $2 \cdot 500$ units such as *BRNN2A* and *BRNN2B* surpass all other models for all three datatsets, including three layers of BRNNs. The shallow BRNNs work equally well or even better than the deep BRNN for MHAD-10 and MHAD-27 but not for HDM05. It is important to note that models with three hidden layers do not perform better than models with two hidden layers, while training such gradient-based approaches requires a large amount of computation time on GPU. The computation time and cost of training and testing BRNNs is much greater than training ESNs, which the training is only performed for the output weights at the readout where there is no cyclic dependency. Training and testing using dimensionality reduction methods demand the least time and computational power. Considering the time complexity for the gradient-based learning by BPTT, it must be analyzed in terms of space for the number of values stored and the time complexity in terms of the number of arithmetic operations required [37]. Therefore, measuring architecture complexity of RNNs is not a trivial task. Nonetheless, for our configurations when the network is fully connected and all weights are adaptable, if the shallow network requires time $\mathcal{T}$ to complete the task, the deep network can be expected to complete the task in about $\mathcal{L} \cdot \mathcal{T}$, where $\mathcal{L}$ is the number of hidden layers.

A Comparison. Other studies which resemble our first experiment use only one default view and do not exclude test subjects from training data. Furthermore, some approaches apply some prior filters before passing data to the networks, for instance, [3] proposed a hybrid MLPs which reported the recognition rate with an accuracy of 95% on HDM05 on 10-fold evaluation. Next, [9] introduced deep BRNN by stacking BRNNs using LSTM units on each skeleton part yielding the best result of 96.92% for HDM05. Followed by [38] which use deep BRNNs with LSTM units on body parts similar to [9] but added a so-called co-occurrence matrix and dropout to a three-LSTM layer with two feedback layers. It accounted for the recognition rate of 97.25%.

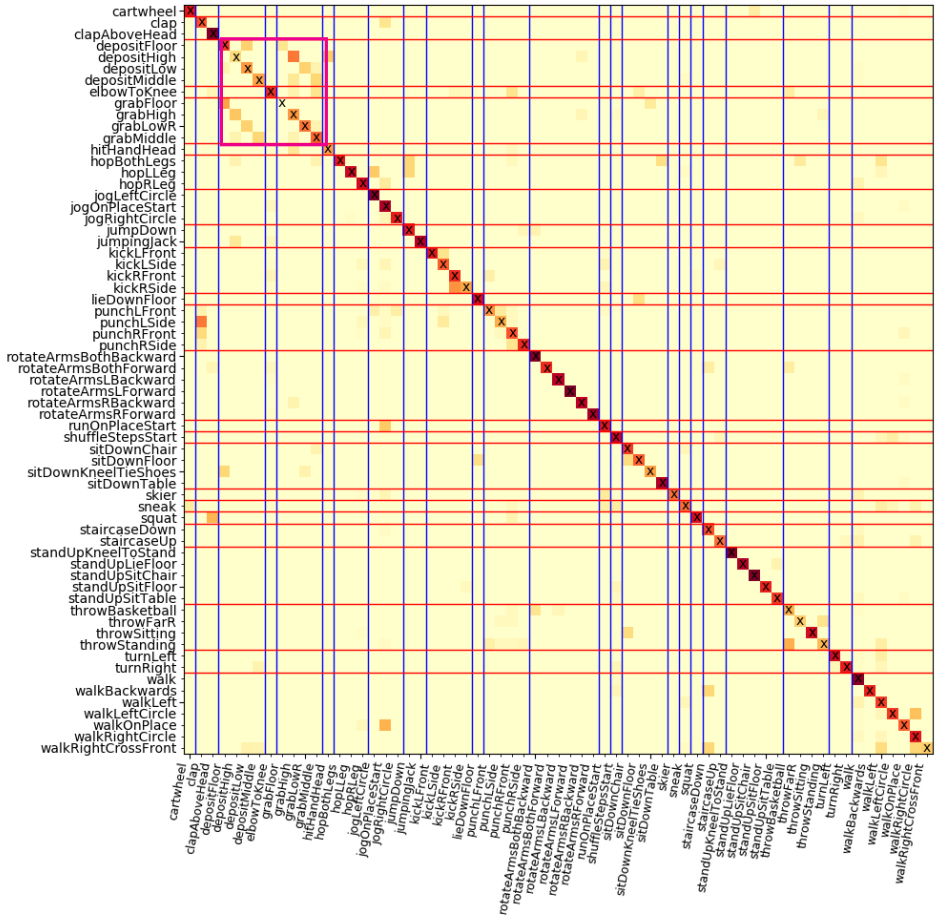


Figure 8 Confusion matrix of HDM05 with 65 actions on average of 2-folds using best model of *BRNN2B* ($2 \cdot [100 \cdot 400]$). The weighted colors are computed from the percentage of the total number of that action. The thick pink rectangle at the left corner shows a group of actions which significantly misclassified in all methods. The red horizontal and vertical blue lines are drawn to highlight groups of the actions.

6.2 Visualization of BRNN

Last hidden layer. For the BRNNs, only half of the output activities of the last hidden layer contribute to the output layer as can be seen in Figure 10. This figure could explain why the number of cells in one direction at the last hidden layer needs to be much greater than the number of the output classes which are required by the Softmax function at the output.

Visualization of input and other hidden layers. Multidimensional time-series data cannot be directly visualized, therefore investigating its behavior is very difficult. One common approach that is normally used in order to get some insight into high dimensional time-series data is by examining their distance matrix. One benefit of using distance matrices, such as Euclidean distance is that we can further analyze the matrix using recurrence plots (RPs) [10] by applying a threshold distance and the Heaviside function. The RPs can tell when the phase space trajectory of the dynamic system re-occur roughly in the same area in the phase

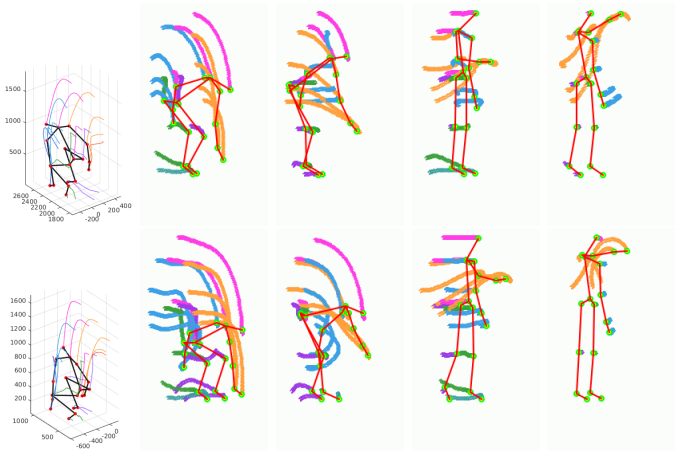


Figure 9 Most common misclassified patterns are the confusion between *"deposit"* and *"grab"*. Top) From left to right: 3D projection of *depositFloor* of default view, and the rotated 2D projections of *depositFloor*, *depositLow*, *depositMiddle* and *depositHigh*, Bottom) *grabFloor* in 3D and 2D projections of *grabFloor*, *grabLowR*, *grabMiddle* and *grabHigh*.

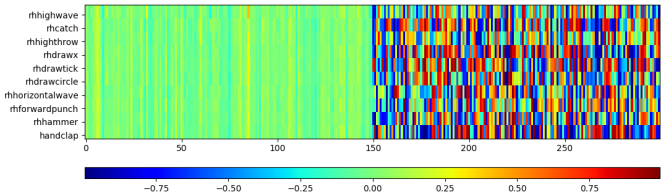


Figure 10 The activities from the last hidden layer with $2 \cdot 150$ cells of MHAD-10 dataset. The left side shows the activities from forward and the right side from the backward direction.

space. Figure 11 shows the Euclidean distance matrices of a subject in HDM05 performing an *elbowToKnee*. The left-most of the figure shows the distance matrix of the input which reveals a few harmonic oscillations that can be observed by the checkerboard structures. The next three figures are the activities from the first hidden layer L_1 for the combination of both directions, for forward and backward direction, respectively. We can infer from the changes of one state of learning to another that the networks opt to differentiate their output activities at each layer. At the upper layer, the scale of the differentiation is larger. The very dark blue corresponds to distance zero and red to the maximum distance between the features in this time span.

7 Conclusion

During the course of conducting our research, we have demonstrated the various influences of various geometries of the deep BRNN's upon human motion recognition. It is crucial to have some empirical insights to amend and influence both the width and depth of the model to suit the research requirements and objectives. The evaluations of the classifications were performed by focusing on the generalization and the robustness of the models by testing on unseen subjects with the variation of movement speeds. The results showed that BRNNs outperformed ESNs and other conventional classification techniques. Correspondingly, we

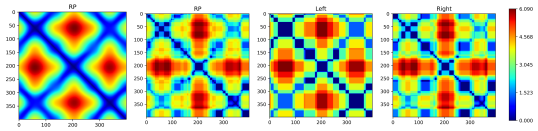


Figure 11 The Euclidean distance matrices of a subject performing *elbowToKnee*. From left to right: i) input ii) the retrieved activities from the first hidden layer of combined directions, iii) forward and iv) backward direction.

discovered that any form of dimensionality reduction, caused by reducing the width of the hidden layers to less than the number of input features or reducing the width of last hidden layer in one direction less than the output units is unsatisfactory. The shallow networks should be included and examined in the experiment as they may not only demonstrate good performance for some datasets, but also provide some insights into the impact of hyper parameters. Nonetheless, to achieve the best outcomes, based on our research, we strongly recommend that deep RNN's as the method of choice for researchers to employ.

References

1 Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, mar 1994. doi:10.1109/72.279181.

2 Chen Chen, Roozbeh Jafari, and Nasser Kehtarnavaz. *UTD-MHAD: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor*, volume 2015-December, pages 168–172. IEEE Computer Society, 12 2015. doi:10.1109/ICIP.2015.7350781.

3 Kyunghyun Cho and Xi Chen. Classifying and visualizing motion capture sequences using deep neural networks. *CoRR*, abs/1306.3874, 2013. URL: http://arxiv.org/abs/1306.3874.

4 Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111, 2014. URL: http://aclweb.org/anthology/W/W14/W14-4012.pdf.

5 Kyunghyun Cho, Bart van Merriënboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics, oct 2014.

6 Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *ICLR 2017 conference*, 2017. URL: https://arxiv.org/abs/1609.01704.

7 Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

8 Timothy Dozat. Incorporating nesterov momentum into adam, 2015.

9 Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

10 J. P. Eckmann, Oliffson S. Kamphorst, and D. Ruelle. Recurrence plots of dynamical systems. *Europhysics Letters*, 4, nov 1987.

11 Mona Fathollahi and Rangachar Kasturi. Autonomous driving challenge: To infer the property of a dynamic object based on its motion pattern using recurrent neural network. *CoRR*, abs/1609.00361, 2016.

12 A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013. doi:10.1109/ICASSP.2013.6638947.

13 Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM. doi:10.1145/1143844.1143891.

14 Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with Deep Bidirectional LSTM. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 273–278. IEEE, 2013. doi:10.1109/asru.2013.6707742.

15 Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(5):855–868, 2009. doi:10.1109/TPAMI.2008.137.

16 Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 190–198. Curran Associates, Inc., 2013. URL: http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf.

17 S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer and Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

18 Ozan İrsoy and Claire Cardie. Opinion mining with deep recurrent neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 720–728, 2014. URL: http://aclweb.org/anthology/D14-1080.

19 Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in Neural Information Processing Systems 15, NIPS 2002*, pages 593–600, 2002.

20 Herbert Jaeger and Harald Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. *Science*, 304(5667):78–80, 2004.

21 Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352, 2007.

22 Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2342–2350, Lille, France, 07–09 Jul 2015. PMLR. URL: http://proceedings.mlr.press/v37/jozefowicz15.html.

23 Marco Körner and Joachim Denzler. Analyzing the subspaces obtained by dimensionality reduction for human action recognition from 3d data. In *IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, pages 130–135, 2012.

24 Yajie Miao and Florian Metze. On speaker adaptation of long short-term memory recurrent neural networks. In *in Sixteenth Annual Conference of the International Speech Communication Association (INTERSPEECH) (To Appear). ISCA*, 2015.

25 M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation mocap database hdm05. Technical Report CG-2007-2, Universität Bonn, June 2007.

26 Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*, 2014.

27 M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, nov 1997. doi:10.1109/78.650093.

28 Rajiv Shah and Rob Romijnders. Applying deep learning to basketball trajectories. *KDD 2016, Large Scale Sports Analytic Workshop*, 2016.

29 Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In David Blei and Francis Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 843–852. JMLR Workshop and Conference Proceedings, 2015. URL: http://jmlr.org/proceedings/papers/v37/srivastava15.pdf.

30 Ilya Sutskever, James Martens, and Geoffrey E. Hinton. Generating text with recurrent neural networks. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, New York, NY, USA, 2011. ACM. URL: http://www.icml-2011.org/papers/524_icmlpaper.pdf.

31 Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press. URL: http://dl.acm.org/citation.cfm?id=2969033.2969173.

32 Pattreeya Tanisaro and Gunther Heidemann. Time series classification using time warping invariant echo state networks. In *15th IEEE International Conference on Machine Learning and Applications, (ICMLA)*, 2016.

33 Pattreeya Tanisaro, Constantin Lehman, Leon Sütfeld, Gordon Pripa, and Gunther Heidemann. Classifying bio-inspired model in point-light human motion using echo state network. In *The 26th International Conference on Artificial Neural Networks (ICANN), 2017*, Lecture Notes in Computer Science. Springer, 2017.

34 Pattreeya Tanisaro, Florian Mahner, and Gunther Heidemann. Quasi view-independent human motion recognition in subspaces. In *Proceedings of 9th International Conference on Machine Learning and Computing (ICMLC)*, ICMLC 2017, pages 278–283. ACM, 2017. doi:10.1145/3055635.3056577.

35 Graham W. Taylor, Geoffrey E. Hinton, and Sam T. Roweis. Two distributed-state models for generating high-dimensional time series. *J. Mach. Learn. Res.*, 12:1025–1068, 2011. URL: http://dl.acm.org/citation.cfm?id=1953048.2021035.

36 Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3156–3164, 2015.

37 Ronald J. Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity, 1995.

38 Wentao Zhu, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, and Xiaohui Xie. Co-occurrence feature learning for skeleton based action recognition using regularized deep lstm networks. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 3697–3703, 2016.

A Lists of Complete Actions

Table 1 Ten hand gestures in MHAD-10 from default view at 0°.

Action	#Trials	minLength	maxLength	meanLength
RHHighWave	30	55	161	85
RHCatch	30	38	75	54
RHHighThrow	30	44	78	56
RHDrawX	30	55	81	64
RHDrawTick	30	43	72	57
RHDrawCircle	30	54	89	67
RHHorizontalWave	30	52	139	70
RHForwardPunch	30	42	71	55
RHHammer	30	51	85	65
HandClap	30	47	68	57

B Confusion Matrices from Various Models of Unseen Subjects with Untrained Viewpoints

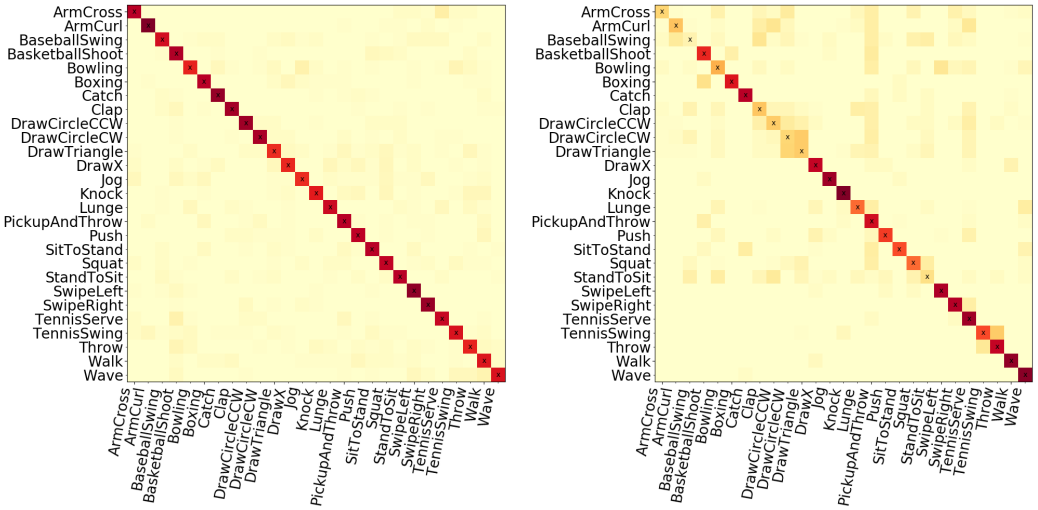


Figure 12 Confusion Matrices of MHAD-27 of the first testing fold from two classification methods. Left) Subspaces employing RF with PCA. Right) Majority vote using 1-NN.

Table 2 27 actions in MHAD-27 from default view at 0°.

Action	#Trials	minLength	maxLength	meanLength
ArmCross	32	50	86	64
ArmCurl,	32	42	86	59
BaseballSwing	32	63	90	74
BasketballShoot	32	46	81	60
Bowling	32	64	101	76
Boxing	32	51	92	68
Catch	32	41	74	59
Clap	32	51	78	61
DrawCircleCCW	32	64	98	74
DrawCircleCW	32	66	95	75
DrawTriangle	32	61	106	77
DrawX	31	55	81	66
Jog	32	51	82	67
Knock	32	53	95	67
Lunge	32	63	103	80
PickupAndThrow	32	68	125	87
Push	32	47	80	62
SitToStand	32	47	69	54
Squat	31	50	116	82
StandToSit	32	46	71	57
SwipeLeft	32	48	76	61
SwipeRight	32	47	75	59
TennisServe	32	52	94	67
TennisSwing	32	44	87	64
Throw	32	44	70	58
Walk	31	60	104	76
Wave	32	49	81	65

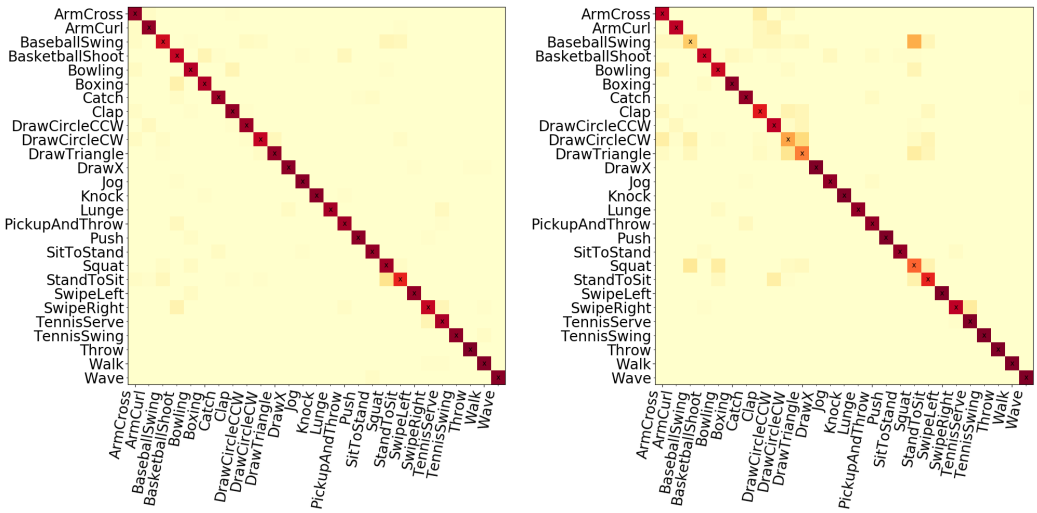


Figure 13 Confusion Matrices of MHAD-27 of 27 folds. Left) BRNN of $2 \cdot [300 \cdot 300]$ cells. Right) ESN with network size of 600 neurons.

Table 3 65 actions in HDM05 from default view at 0°.

Action	#Trials	minLength	maxLength	meanLength
cartwheel	28	281	701	450
clap	31	32	211	109
clapAboveHead	31	59	657	248
depositFloor	32	181	641	363
depositHigh	28	129	509	245
depositLow	28	196	481	277
depositMiddle	29	178	662	270
elbowToKnee	80	93	574	243
grabFloor	16	186	401	268
grabHigh	29	170	460	258
grabLowR	29	191	544	294
grabMiddle	28	112	352	210
hitHandHead	13	141	281	226
hopBothLegs	55	56	432	151
hopLLeg	64	62	254	119
hopRLeg	65	58	246	116
jogLeftCircle	32	197	400	292
jogOnPlaceStart	70	80	241	147
jogRightCircle	33	190	441	288
jumpDown	13	177	381	288
jumpingJack	65	116	484	201
kickLFront	43	129	841	294
kickLSide	39	131	721	315
kickRFront	45	121	668	296
kickRSide	44	127	740	310
lieDownFloor	20	301	901	655
punchLFront	45	119	761	263
punchLSide	45	90	721	235
punchRFront	45	138	761	286
punchRSide	42	97	662	242
rotateArmsBothBackward	32	62	649	214
rotateArmsBothForward	32	62	739	230
rotateArmsLBackward	32	57	708	215
rotateArmsLForward	32	55	739	215
rotateArmsRBackward	32	54	649	210
rotateArmsRForward	32	54	733	213
runOnPlaceStart	74	58	182	100
shuffleStepsStart	51	161	540	319
sitDownChair	20	154	441	318
sitDownFloor	20	224	601	407
sitDownKneelTieShoes	17	425	825	645
sitDownTable	20	162	401	270
skier	40	123	459	202
sneak	63	164	751	372
squat	65	136	823	271
staircaseDown	15	139	319	222
staircaseUp	27	164	444	292
standUpKneelToStand	17	100	301	182
standUpLieFloor	20	279	703	525
standUpSitChair	20	176	441	295
standUpSitFloor	20	167	641	403
standUpSitTable	20	121	454	250
throwBasketball	14	281	721	407
throwFarR	14	361	600	524
throwSitting	28	188	404	282
throwStanding	28	242	541	353
turnLeft	30	119	281	196
turnRight	30	135	260	196
walk	94	122	369	214
walkBackwards	30	158	433	299
walkLeft	32	277	659	411
walkLeftCircle	37	261	560	397
walkOnPlace	60	121	400	233
walkRightCircle	27	246	542	381
walkRightCrossFront	29	195	701	434

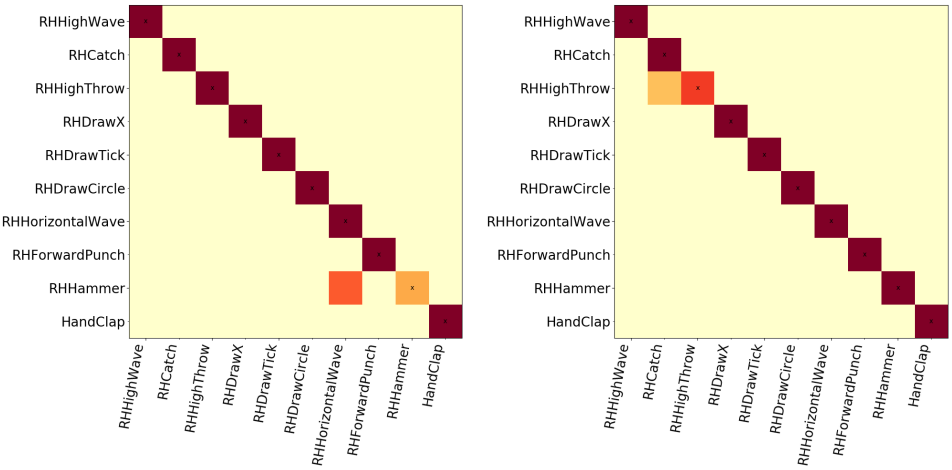


Figure 14 Confusion Matrices of MHAD-10 with the same configurations of BRNN employing $2 \cdot [50 \cdot 150 \cdot 250]$ cells, but testing on different subjects shown on the left and the right figure.

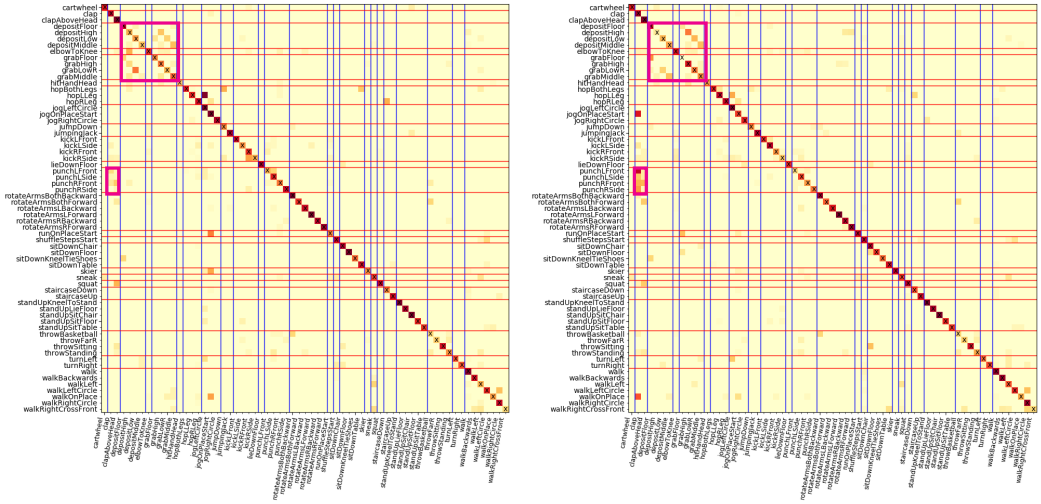


Figure 15 Confusion Matrices of HDM05 of all folds in each configuration. Two common misclassified groups of actions are highlighted in the pink color. Left) BRNN of $2 \cdot [200 \cdot 400]$ cells. Right) BRNN of $2 \cdot [250 \cdot 250]$ cells.

Population Based Methods for Optimising Infinite Behaviours of Timed Automata

Lewis Tolonen
The University of Western Australia

Tim French
The University of Western Australia
tim.french@uwa.edu.au

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

Abstract

Timed automata are powerful models for the analysis of real time systems. The optimal infinite scheduling problem for double-priced timed automata is concerned with finding infinite runs of a system whose long term cost to reward ratio is minimal. Due to the state-space explosion occurring when discretising a timed automaton, exact computation of the optimal infinite ratio is infeasible. This paper describes the implementation and evaluation of ant colony optimisation for approximating the optimal schedule for a given double-priced timed automaton. The application of ant colony optimisation to the corner-point abstraction of the automaton proved generally less effective than a random method. The best found optimisation method was obtained by formulating the choice of time delays in a cycle of the automaton as a linear program and utilizing ant colony optimisation in order to determine a sequence of profitable discrete transitions comprising an infinite behaviour.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Timed Automata, Heuristic Search, Ant Colony Optimisation

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.22

1 Introduction

Many systems exhibit behaviour that is dependent not only on the ordering of a sequence of events, but also the precise time at which these events occur. These are known as *real time systems*, and include such things as industrial plants which may carry out a set of sub-processes each of which requires different resources for a specific duration of time, or communication protocols where messages are sent and received with the restriction that at any given time the communication medium can only be used by one of these actions. The analysis of real time systems is important for ensuring the behaviour of these systems meets the required specifications. In particular, for cases such as industrial plants, it is also desirable that the long term behaviour of these systems is profitable, that is the cost to reward ratio of the system behaviour is minimised. Timed automata are powerful models for performing such analysis of these systems. A timed automaton models a real time system as a set of states with transitions between them, alongside a set of real valued clock variables that count up uniformly in real time. The taking of transitions is dependent on the values of the clocks, and can reset some of the clocks when taken. The model has been extended to be able to describe other quantitative aspects of systems behaviours, such as costs and rewards.

The optimal infinite scheduling problem for timed automata is concerned with finding the behaviour of a system that minimises the long term ratio of the operating cost to the obtained rewards. Solving this problem is particularly useful when modelling an industrial process, as the profitable operation of such a system is typically a key goal in its design. However, due to the state-space explosion that can occur when modelling all potential behaviours of a real time system, finding the exact solution for the optimal infinite scheduling problem generally proves infeasible. Instead, this paper focuses on developing methods for finding approximate solutions to the problem. In particular, the use of population based metaheuristics such as ant colony optimisation as a means of efficiently exploring the candidate behaviours of a real time system. By expressing the timed component of the optimal infinite scheduling problem as a linear program that can be exactly solved relatively efficiently, these techniques give an effective method for approximating the optimal behaviour of a given system.

2 Preliminaries

In this section we will present the preliminary definitions required for this paper.

2.1 Timed Automata

Clocks, Valuations, and Constraints. Before formally defining a timed automaton it is necessary to introduce the *clock variables* on which their behaviour depends. A set of clocks $\mathcal{X}$ is a finite set of variables $\{x_0, \ldots, x_n\}$ taking values in the non-negative real numbers $\mathbb{R}_{\geq 0}$. A clock *valuation* is a function $v : \mathcal{X} \to \mathbb{R}_{\geq 0}$ assigning to each clock its current value. $\mathbf{0}$ denotes the valuation that assigns 0 to all clocks. The *reset operation* on a clock valuation v by a subset X of $\mathcal{X}$, denoted $v[X := 0]$, gives the valuation v' where the clocks in X are set to 0, while the other clocks remain unchanged. For a real number δ, the *delay* of v by δ, denoted $v + \delta$ gives the valuation where δ has been added to the value of each clock, i.e. $(v + \delta)(x) = v(x) + \delta$. $\mathcal{C}(\mathcal{X})$ denotes the set of *clock constraints* over $\mathcal{X}$. These are conjunctions of atomic constraints of the form $x \bowtie c$ or $x - y \bowtie c$ where $x, y \in \mathcal{X}$, $c \in \mathbb{Z}$ and $\bowtie \in \{<, >, \leq, \geq\}$. A clock valuation v satisfies a clock constraint g if the value of each clock under v satisfies each atomic constraint comprising g, if so we write $v \vDash g$. A clock constraint is said to be *diagonal free* if it is the conjunction of atomic constraints that only place upper and lower bounds on individual clocks, and does not directly restrict the difference between any pair of clocks.

▶ **Definition 1** (Timed Automaton). A timed automaton $\mathcal{A}$ is defined to be a tuple $(Q, q_0, \mathcal{X}, \Sigma, E, I)$, where

- Q is a finite set of discrete locations.
- $q_0 \in Q$ is the starting location.
- $\mathcal{X}$ is a set of clocks.
- Σ is a finite alphabet of actions.
- $E \subset Q \times \Sigma \times \mathcal{C}(\mathcal{X}) \times 2^{\mathcal{X}} \times Q$ is a relation defining the jumps between locations. $e = (q, \alpha, G, X, q')$ represents a jump from the location q to q' by taking the action α. G is the guard on e: a diagonal free clock constraint that must be satisfied by the current clock valuation in order for the jump to be taken. X is the subset of clocks to be reset to 0 when the jump is taken.
- $I : Q \to \mathcal{C}(\mathcal{X})$ assigns to each location an invariant condition that must be satisfied at all times while in that location. Like the guards on the jumps, each location invariant must be diagonal free.

A *state* of a timed automaton $\mathcal{A}$ is a pair (q, v) with $q \in Q$ and v a valuation over $\mathcal{X}$. The execution of $\mathcal{A}$ gives a transition system over the infinite space of possible states with two types of transitions:

- A *delay transition* from a state (q, v) has the form $(q, v) \xrightarrow{\delta} (q, v + \delta)$. This transition represents the automaton idling in location q for a duration of δ, provided the state continues to satisfy the invariant condition of the location, that is $v + t$ satisfies $I(q)$ for all $0 \leq t \leq \delta$.
- A *discrete transition* with respect to a jump $e = (q, \alpha, G, X, q') \in E$ from a state (q, v) has the form $(q, v) \xrightarrow{e} (q', v')$, where $v' = v[X := 0]$, v satisfies G and v' satisfies $I(q')$. This represents the automaton taking action α in q resulting in the jump to q' provided the guard on e is satisfied by v and the invariant at q' is satisfied by v'.

A finite *run* of $\mathcal{A}$ is a finite sequence $\rho = (q_0, \mathbf{0}) \xrightarrow{\delta_1} \xrightarrow{e_1} s_1 \xrightarrow{\delta_2} \xrightarrow{e_2} \ldots \xrightarrow{\delta_n} \xrightarrow{e_n} s_n$ where each $s_{i-1} \xrightarrow{\delta_i} \xrightarrow{e_i} s_i$ is a delay-transition (possibly of 0 duration) followed by a discrete-transition. This definition can be extended to infinite runs of $\mathcal{A}$ by extending ρ to be an infinite sequence.

2.1.1 Double-Priced Timed Automata and Optimal Infinite Scheduling

The model of timed automata has been extended in various ways to model different aspects of real time systems. A common extension is to associate a price with each location and jump; a cost is accrued for each unit of time spent in a location, as well as for each jump taken [3]. The accumulated cost of runs of the automaton can then be analysed, for example the minimum cost reachability of certain states [2]. In 2008, Bouyer, Brinksma and Larsen proposed the model of *double-priced timed automata*: a timed automaton extended with two price functions: costs and rewards [4]. This model was designed for the purpose of analysing the infinite behaviours of systems via their long term cost to reward ratio. The minimisation of this ratio for long term behaviours is called the *optimal infinite scheduling problem*. The solutions to this problem are of particular interest when considering systems that need to perform a series of operations indefinitely, such as industrial processes where the operator would want the long term rewards to exceed the costs. We now formally define a double-priced timed automaton:

▶ **Definition 2** (Double-Priced Timed Automaton)**.** A double-priced timed automaton (DPTA) $\mathcal{A}$ is a tuple $(Q, q_0, \mathcal{X}, \Sigma, E, I, \texttt{Cost}, \texttt{Reward})$ where

- $(Q, q_0, \mathcal{X}, \Sigma, E, I)$ defines a timed automaton as defined in definition 1.
- $\texttt{Cost} : (\texttt{Q} \cup \texttt{E}) \to \mathbb{N}$ is a function assigning to each jump a cost and to each location a cost rate.
- $\texttt{Reward} : (\texttt{Q} \cup \texttt{E}) \to \mathbb{N}$ is a function assigning to each jump a reward and to each location a reward rate.

The transition system of a DPTA is identical to that of a timed automaton. However each transition now has an associated cost and reward. A delay transition $(q, v) \xrightarrow{\delta} (q, v + \delta)$ has cost $\texttt{Cost}(q) \cdot \delta$ and reward $\texttt{Reward}(q) \cdot \delta$, that is each time unit spent in location q contributes an amount equal to its cost or reward rate. A discrete transition $(q, v) \xrightarrow{e} (q', v')$ simply has the cost and reward $\texttt{Cost}(e)$ and $\texttt{Reward}(e)$ respectively. Given a finite run of a DPTA $\rho = (q_0, \mathbf{0}) \xrightarrow{\delta_1} \xrightarrow{e_1} (q_1, v_1) \xrightarrow{\delta_2} \xrightarrow{e_2} \ldots \xrightarrow{\delta_n} \xrightarrow{e_n} (q_n, v_n)$ we define the cost of the run as the sum of the costs of each transition: $\texttt{Cost}(\rho) = \sum_{i=1}^{n} (Cost(e_i) + Cost(q_{i-1} \cdot \delta_i)$. The reward of a finite a run is defined similarly. The ratio of a run is simply the cost divided by the reward: $\texttt{Ratio}(\gamma) = \texttt{Cost}(\gamma)/\texttt{Reward}(\gamma)$. For infinite runs, we consider the limit of

the ratio as the run progresses. For an infinite run $\rho = (q_0, \mathbf{0}) \xrightarrow{\delta_1} \xrightarrow{e_1} (q_1, v_1) \xrightarrow{\delta_2} \xrightarrow{e_2} \ldots$, let ρ_n be the prefix of ρ consisting of the first n transitions. The ratio of the run is defined as $\texttt{Ratio}(\rho) := \lim_{n\to\infty} \texttt{Ratio}(\rho_n)$. To ensure that the ratios of all infinite runs of the DPTA exist, we require that the automaton be *strongly reward diverging*: for every infinite run ρ, we have $\lim_{n\to\infty} \texttt{Reward}(\rho_n) = \infty$, i.e an infinite number of actions leads to an infinite reward. This effectively means that there are no cycles with zero reward.

We now define the optimal infinite scheduling problem for DPTA. Given a strongly reward diverging DPTA $\mathcal{A} = (Q, q_0, \mathcal{X}, \Sigma, E, I, \texttt{Cost}, \texttt{Reward})$, let Γ be the set of all infinite runs of $\mathcal{A}$. The optimal infinite scheduling problem is to determine the lower bound on achievable cost to reward ratios of infinite runs of the automaton: $\texttt{Ratio}^* = \inf\{\texttt{Ratio}(\rho) \mid \rho \in \Gamma\}$. Note that there may not be a run of $\mathcal{A}$ with this ratio, but instead a family of runs such that their ratio becomes arbitrarily close to $\texttt{Ratio}^*$. Bouyer, Brinksma and Larsen have shown that the optimal infinite scheduling problem is computable and is PSPACE-complete [4].

While the region abstraction defined by Alur and Dill preserves reachability properties [1], it does not preserve the ratios of infinite runs as the duration of timed transitions is abstracted away. In their proof of the computability of the optimal infinite scheduling problem, Bouyer, Brinksma and Larsen introduce an extension of the region abstraction that does preserve this property called the *corner point abstraction* [4]. Let $\mathcal{A} = (Q, q_0, \mathcal{X}, \Sigma, E, I, \texttt{Cost}, \texttt{Reward})$ be a DPTA with $|\mathcal{X}| = k$. We also require that the reachable clock-space of $\mathcal{A}$ is bounded, that is there exists a constant M such that for all reachable clock-valuations v, $v(x) \leq M$ for all clocks x in $\mathcal{X}$. A *corner point* is an element $\mathbf{a} = (a_1, \ldots, a_k)$ of $\mathbb{N}^k$, which can be thought of as a clock valuation. If (q, R) is a region of $\mathcal{A}$, a corner point $\mathbf{a}$ is associated with (q, R) if $\mathbf{a}$ lies in the closure of R. Interpreting R as a polyhedron in the clock space, then the corner points associated with R are the vertices of the closure of that polyhedron. The corner point abstraction of $\mathcal{A}$ is a double-weighted graph with vertices of the form $(q, R, \mathbf{a})$ where (q, R) is a region and $\mathbf{a}$ is a corner point associated with (q, R). The edges of the graph represent either discrete or delay transitions of the automaton. The edges have two weights - a cost and a reward corresponding to the cost and reward of the transition in the original automaton. For each jump $e = (q_{\texttt{src}}, \alpha, G, X, q_{\texttt{dest}})$ of $\mathcal{A}$, for every vertex $(q, R, \mathbf{a})$ such that $q = q_{\texttt{src}}$ and R satisfies the guard G, then there is an edge from this vertex to the unique vertex $(q', R', \mathbf{a}')$ such that $q' = q_{\texttt{dest}}$, $R' = R[X := 0]$ and $\mathbf{a}' = \mathbf{a}[X := 0]$. This edge has weights $\texttt{Cost}(e)$ and $\texttt{Reward}(e)$. There are two types of edges corresponding to delay transitions in the corner point abstraction:

- Given two vertices $(q, R, \mathbf{a})$ and $(q, R, \mathbf{a}+1)$ (i.e $\mathbf{a}+1$ is an immediate time successor of $\mathbf{a}$), there is an edge between them with weights $\texttt{Cost}(q)$ and $\texttt{Reward}(q)$ as this corresponds to a delay of 1 time unit in location q.
- Given two vertices $(q, R, \mathbf{a})$ and $(q, R', \mathbf{a})$ with the region R' being a time successor of R, there is an edge between the vertices with cost and reward equal to 0, as this is a delay of zero duration.

A path through the corner point abstraction corresponds to a run of the automaton, with corresponding accumulated cost and reward. However, the only delay transitions represented in the corner-point abstraction are those of duration $z + \epsilon$ where z is a non-negative integer and ϵ is some real number with $|\epsilon| << 1$. Bouyer, Brinksma and Larsen [4] prove that these transitions are sufficient, and that the corner point abstraction is sound and complete with respect to the optimal infinite scheduling problem for double priced timed automata, i.e. that the optimal ratio of infinite paths in the corner point abstraction is equal to $\texttt{Ratio}^*$ in the corresponding DPTA. The optimal infinite scheduling problem can therefore be reduced to finding the minimum cost to reward ratio cycle in the corner point abstraction. If the

corner point abstraction has V vertices and E edges, then the best known algorithm for this problem - Burn's Algorithm has time complexity $O(V^2E)$ [7]. As the number of vertices in the corner point abstraction is proportional to the number of regions of the automaton, which has an upper bound of $|Q| \cdot |\mathcal{X}|! \cdot 2^{|\mathcal{X}|} \cdot \prod_{x \in \mathcal{X}}(2c_x + 2)$ as seen above, applying this method does not give a feasible algorithm for most automata.

A possible approach to effectively solving the optimal infinite scheduling problem is to use a zone-based abstraction that preserves the optimal ratio of runs, rather than a region-based one. David et al. have constructed such a method specifically for the case of timed automata with only one clock [8]. Their method is based on *strong time abstracting bisimulations* (STABs)- a zone based method of partitioning the state space of an automaton into zones that have equivalent behaviour [16]. In the one clock case zones are simply intervals of non-negative real numbers. David et al. [8] define an abstraction based on the end points of these intervals and prove that it preserves the optimal infinite ratio. The procedure of constructing this abstraction and finding the optimal ratio has complexity $O(|Q| \cdot (|Q| + |E|)^6)$, which is much more feasible than using the corner point abstraction. However, most real time systems require the use of several clocks to effectively model their behaviour. This work has not currently been extended to the case with more than one clock, and so there is a lack of effective algorithms for solving the optimal infinite scheduling problem for timed automata with two or more clocks. Additionally, in the worst case the size of a STAB of a timed automaton is as large as that of the region abstraction, so even given a STAB based method, the computation of the optimal infinite ratio will still be infeasible.

2.2 Population Based Metaheuristics

For many optimisation problems, especially those of high complexity, an exact solution is not required. Instead an approximate solution is often sufficient. In this case, we can make use of heuristic algorithms to rapidly explore the state-space of the problem to find a solution that is good enough, but not provably optimal. As demonstrated in the previous section, the complexity of the optimal infinite scheduling problem for double-priced timed automata is typically too high to admit an exhaustive search of the state-space, and so the use of heuristics is a suitable method for compromising precision for computational efficiency in this domain.

2.2.1 Ant Colony Optimisation

Ant colony optimisation is a population based metaheuristic first developed by Marco Dorigo in 1996 as a method of determining approximate solutions to the travelling salesman problem [9]. The method is inspired by the way ants utilize pheromone in order to communicate the best path between their nest and a food source. Ant colony optimisation is typically used for combinatorial optimisation problems consisting of a finite set of discrete decisions, such as finding traversals of graphs, but it has been extended to problems in continuous domains [12]. While these problems are typically NP, Ant Colony optimisation has been applied to some PSPACE problems such as the Canadian Traveller Problem [13].

Ant colony optimisation operates by simulating a population of agents representing artificial ants. To illustrate we will suppose we have a combinatorial optimisation problem given as finding a traversal of a graph $G = \{V, E\}$ with V being the set of vertices and E the set of edges. We have some objective function over the traversals of the graph that we are trying to optimize. As each possible solution is a traversal, the discrete decisions are the choice of which edge to take at each point. The artificial ants communicate and make

Algorithm 1: High Level Ant Colony Optimisation.

```
repeat
    place ants;
    repeat
        foreach ant do
            if path not complete then
                follow state transition rule to choose next edge;
                take edge and apply local pheromone update;
            end
        end
    until all paths complete;
    apply global pheromone update;
until terminated;
return best path;
```

decisions based on a *pheromone function*: $\texttt{Pheromone} : \texttt{E} \to \mathbb{R}_{\geq 0}$, a function assigning to each edge a non-negative real number which represents the concentration of pheromone on that edge. A high pheromone concentration means that traversals including that edge evaluated well under the objective function. At a decision point a traversal is more likely to take an edge with a higher concentration of pheromone.

The overall algorithm operates over a series of iterations. Within each iteration each ant in the population builds a solution by first being placed at a starting vertex, and then taking edges one by one according to a *state transition rule*. After taking an edge, the ant updates that edge's pheromone concentration according to a *local pheromone updating rule*. The ant continues to take edges until a complete solution is formed. Once all ants have complete solutions, the pheromone along all edges is updated. This is called the *global pheromone update*. After a number of iterations the process completes, and the overall best found traversal is returned as the solution to the optimisation problem.

State Transition Rule. The state transition rule is how ants determine what edge to add to their solution. If an ant is at a vertex v, with outgoing edges $\texttt{Out}(\texttt{v})$, the simplest used state transition rule is to choose a particular edge with probability corresponding to its relative pheromone concentration. For an edge e in $\texttt{Out}(v)$ the probability of taking this edge, $P(e)$ is given by:

$$P(e) = \frac{\texttt{Pheromone}(e)}{\sum_{e' \in \texttt{Out}(v)} \texttt{Pheromone}(e')}$$

This means that an ant is more likely to take an edge known to give results, but the probabilistic factor means that alternate solutions will still be explored. It is also common to apply a heuristic factor, i.e. an approximation of that edge's quality, to the probability of taking an edge. The effect of the pheromone relative to the effect of the heuristic can then be adjusted using real-valued weighting parameters [9].

Pheromone Updates. In combination with the state transition rule, pheromone updates act as a mechanism for encouraging ants in future iterations to explore in the neighbourhood of previously found good solutions. The global pheromone update occurs after each ant has

built its solution. First, some amount of the pheromone along all edges *evaporates*, that is it is reduced by a linear factor $\lambda \in [0, 1]$ called the *evaporation rate*. For each edge e in E this is characterized by the update:

$$\texttt{Pheromone}(e) \leftarrow (1 - \lambda) \cdot \texttt{Pheromone}(e)$$

This is a mechanism in place to make ants weight more recent information more heavily than older information. Then, each ant applies to each of the edges involved in its solution an amount of pheromone proportional to that solution's quality under the objective function. If f is the objective function, and P_k is the solution found by the kth ant, the update for an edge e is:

$$\texttt{Pheromone}(e) \leftarrow \texttt{Pheromone}(e) + \sum_{k \in Ants} \begin{cases} f(P_k) & e \in P_k \\ 0 & e \notin P_k \end{cases}$$

In his revised version of the ant colony optimisation [10], Dorigo found that using an *elitist* global update gave better performance in general. This is a modification where only the best n ants of an iteration are allowed to place pheromone along their path. In the same revision, Dorigo introduced the local pheromone update, where upon adding an edge to its partial solution an ant will slightly reduce the taken edge's pheromone concentration. This discourages ants within the same iteration from taking the same set of edges, ensuring that a diverse set of solutions is tested in each iteration. Later work by Stüzle and Hoos found that performance is also improved if pheromone concentration is capped between a minimum and maximum level [15].

3 Approximate Optimisation

In the following section we detail the application of population based metaheuristic methods to the optimal infinite scheduling problem for a double priced time automata $\mathcal{A} = (Q, q_{start}, \mathcal{X}, \Sigma, E, I, \texttt{Cost}, \texttt{Reward})$. There are several assumptions we will make regarding $\mathcal{A}$:

1. $\mathcal{A}$ is strongly reward diverging - This is to ensure that every infinite run of $\mathcal{A}$ has a defined cost to reward ratio.
2. The invariants I of $\mathcal{A}$ in each location bound the values of every clock. This ensures that the reachable clock space is bounded and hence the corner point abstraction [4] can be used.
3. $\mathcal{A}$ is *deadlock free*. That is for all reachable states $s = (q, v)$ of $\mathcal{A}$ there exists some positive delay δ such that the state $(q, v + \delta)$ satisfies the location invariant at q and the guard condition of at least one jump from q. Intuitively this means that at all states, by letting time pass the automaton is guaranteed to be able to take a discrete transition.

When considering the corner point abstraction, conditions 2 and 3 together ensure that starting from a particular vertex of the corner point an infinite execution will eventually cycle and repeat a corner point without reaching any 'dead ends'. This fact is useful for generating cyclic executions of the automaton on the fly, without the need for backtracking.

Our approach will consist of applying metaheuristic techniques in order to find cycles in the corner point abstraction that have a ratio that is as small as possible. The goal is to explore the state-space as efficiently as possible to give a good solution, although the solutions that are found will not be provably optimal.

In order to generate these cycles, we use a simple representation for the vertices of corner point abstraction, which we will refer to as a *CP-state*. For an automaton with k clocks, a CP-state $\underline{s} = (q, \mathbf{z}, \mathbf{d})$ is a location q, alongside a pair of integer-valued vectors $\mathbf{z}$ and $\mathbf{d}$ with $\mathbf{z} = (z_1, \ldots, z_k)$ and $\mathbf{d} = (d_1, \ldots, d_k)$. The ith component of each vector is related to the ith clock of the automaton. The vector $\mathbf{z}$ gives the corner-point with which the CP-state is associated, while $\mathbf{d}$ defines the relative ordering of the fractional components of each clock. If ϵ is a very small positive real number, the CP-state $(\mathbf{z}, \mathbf{d})$ corresponds to the vertex of the corner point abstraction associated with the corner point $\mathbf{a} = \mathbf{z}$, and with the region to which the point $\mathbf{z} + \epsilon \cdot \mathbf{d}$ belongs.

3.1 Ant Colony Optimisation

As the corner point abstraction is a graph, we could theoretically apply ant colony optimisation directly in order to find a good cycle. However for this approach, as part of the pheromone function representation, we would need to store a value for each individual edge in the corner point abstraction. The upper bound on the number of vertices is of the order $|Q| \cdot |\mathcal{X}|! \cdot 2^{|\mathcal{X}|} \cdot \prod_{x \in \mathcal{X}} (2c_x + 2)$ [1], with the number of edges being even larger, so memory-wise this is not feasible. Instead we will approach the problem as a multi-step problem with two components. The first component is the problem of determining what sequence of discrete transitions of the automaton will compose the cycle. The second component is the problem of determining what delays to take between the discrete transitions.

3.1.1 Simple Ant Colony Optimisation with Random Delays

The simplest approach under this interpretation is to have the pheromone function assign values to the jumps of the automaton. Given a DPTA $\mathcal{A} = (Q, q_{start}, \mathcal{X}, \Sigma, E, I, \texttt{Cost}, \texttt{Reward})$, we have a pheromone function $\texttt{pheromone} : E \to \mathbb{R}_{\geq 0}$. Initially, before performing any exploration we set $\texttt{pheromone}(e) = 1$ for all e in E. In each iteration of the algorithm, each ant will be placed at a random CP-state of $\mathcal{A}$. The ant will then build a solution by sequentially taking a delay transition (randomly) and then a discrete transition (based on pheromone), until it reaches a CP-state it has already visited. By our assumptions listed at the start of the chapter, this is guaranteed to occur and as such we will have a cycle from the repeated CP-state to itself. As we are only interested in the cycle itself, the path leading up to the cycle can be discarded.

To choose a delay from a particular CP-state, the ant considers the set of delays that lead to a subsequent CP-state where at least one discrete-transition is enabled, as the next step requires choosing a discrete transition from that state. Due to the assumption of the automaton being deadlock-free this set of delays is guaranteed to be non-empty. A delay from this set is chosen with uniform probability.

For this algorithm the state-transition rule will determine which jump of the automaton an ant will take from a given CP-state $\underline{s}$. The probability of taking a jump e depends on both the pheromone concentration as well as the cost/reward ratio contributed by e, with a smaller ratio leading to a higher probability. Let $\texttt{Out}(\underline{s})$ be the set of jumps that can be taken from $\underline{s}$, the probability of taking a jump e in $\texttt{Out}(\underline{s})$, $P(e)$ is given by:

$$P(e) = \frac{\texttt{pheromone}(e)^\alpha \cdot (\frac{\texttt{Reward}(e)}{\texttt{Cost}(e)})^\beta}{\sum_{e' \in \texttt{Out}(\underline{s})} \texttt{pheromone}(e')^\alpha \cdot (\frac{\texttt{Reward}(e')}{\texttt{Cost}(e')})^\beta}$$

α and β are real valued parameters that allow for weighting the effect of the pheromone and the jump's ratio when determining the probability.

As per the standard implementation of ant colony optimisation, we have local and global pheromone updates. As the local update, when an ant takes a jump e, the pheromone concentration of e is scaled towards the initial value 1 by a linear *decay factor* μ:

$$\texttt{Pheromone}(e) \leftarrow (1-\mu) \cdot \texttt{Pheromone}(e) + \mu \cdot 1$$

The global update first has the pheromone along all edges reduced according to the evaporation factor λ:

$$\texttt{Pheromone}(e) \leftarrow (1-\lambda) \cdot \texttt{Pheromone}(e)$$

Then we perform an elitist update based on the paths found. Let ρ_i be the ith best cycle found in this iteration, and n be the elitism factor, i.e. the number of top ants we are considering. We update by:

$$\texttt{Pheromone}(e) \leftarrow \texttt{Pheromone}(e) + \sum_{i=1}^{n} \begin{cases} (\texttt{Ratio}(\rho_i))^{-1} & e \in \rho_i \\ 0 & e \notin \rho_i \end{cases}$$

where $(\texttt{Ratio}(\rho))^{-1}$ is the reciprocal of the ratio of ρ, so a lower cost to reward yields a higher pheromone concentration. We also cap all pheromone values to be below a bound $\texttt{Pheromone}_{\texttt{max}}$. If we ever evaluate a reward/cost fraction where the cost is 0, we interpret this value as $\texttt{Pheromone}_{\texttt{max}}$ to ensure all values are well defined.

Using these steps as part of a standard ant colony optimisation algorithm we get a process that attempts to optimise the discrete transitions of runs in our DPTA. However, our pheromone function stores no information about the delay transitions being used, so these cannot be effectively optimised by this method. The next section introduces an approach to optimise both the discrete and delay transitions.

3.1.2 Hybrid Ant Colony Optimsation

Research has shown that with a modification to pheromone function representation, ant colony optimisation can be applied to optimisation in continuous domains [12]. With further modification we can use this approach to solve hybrid discrete-optimisation problems. The optimal infinite scheduling problem can be constructed in this way. Choosing the jumps that form the cycle is the discrete problem, and choosing the delays is the continuous problem. This section will extend the previous method to utilize this hybrid approach. We are still considering paths within the corner point abstraction, but instead of choosing a delay and then a jump we choose a *joint action* consisting of both these parts: (δ, e) where δ is the positive delay, and e is the jump.

When considering what joint action to take, while the delay itself determines the cost and reward contributed by the current location, the state that we delay to and take the jump from has more of an impact over the automaton's potential future behaviour. Therefore, our new pheromone function should act as an estimator of the utility of taking a jump from a particular state. As a jump can only be taken from its source location, we only need to consider the valuation of the state and not its location. So for a k clock automaton, we have $\texttt{Pheromone} : \mathbb{R}^k_{\geq 0} \times E \to \mathbb{R}_{\geq 0}$. A challenge here is the effective representation of the pheromone function, as the domain is partially continuous. We would also like the placement pheromone to affect not only the exact state the jump was taken from, but also

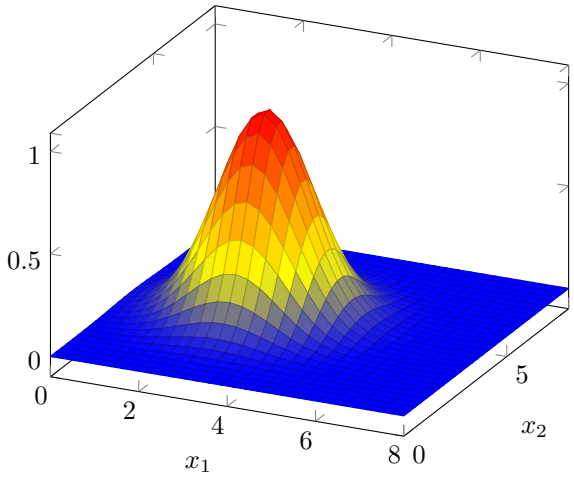


Figure 1 2D Gaussian kernel $G_{\mathbf{v}}(\mathbf{x})$ with $\mathbf{v} = (3, 4)$ and $\sigma = 1$.

the neighbourhood around that state so as to inform later ants that similar states to this might also be good candidates. As explored by Socha and Dorigo [12], a solution is to have the pheromone function be a sum of k-dimensional Gaussian kernels. Given a center-point $\mathbf{v}$ the Gaussian kernel $G_{\mathbf{v}} : \mathbb{R}^k \to \mathbb{R}$ is of the form:

$$G_{\mathbf{v}}(\mathbf{x}) = \exp\left(-\frac{|\mathbf{v} - \mathbf{x}|^2}{2\sigma^2}\right)$$

where $|\mathbf{v} - \mathbf{x}|$ is the Euclidean distance between $\mathbf{v}$ and $\mathbf{x}$ and σ is a constant defining the width of the curve. As seen in Figure 1, the function gives a maximum of 1 at $\mathbf{x} = \mathbf{v}$ and approaches 0 as $\mathbf{x}$ moves further away.

We can then define our pheromone function as a linear combination of these curves centered at the points pheromone has been applied. For a particular jump e, let $\mathbf{v}_1, \ldots, \mathbf{v}_m$ be the points at which pheromone has been applied, and $\tau_1, \ldots, \tau_m$ be the corresponding concentration that was applied at each point. Each value τ_i is the maximum height of the corresponding curve, We also add the constant function 1 so that the initial pheromone concentration at each point is 1 like in the previous method. Our pheromone function is then given by:

$$\texttt{Pheromone}(\mathbf{x}, e) = 1 + \tau_1 \cdot G_{\mathbf{v}_1}(\mathbf{x}) + \ldots + \tau_m \cdot G_{\mathbf{v}_m}(\mathbf{x})$$

Figure 2 shows an example of a pheromone function at a particular jump in a 1 clock automaton, where varying concentrations of pheromone have been placed at $x = 2$, $x = 6$ and $x = 12$.

This method is easily represented by a list of centre points and concentrations for each jump of the automaton. Evaluation of the function simply requires taking the value 1 and adding the corresponding Gaussian functions at the particular point, restricting the value if it exceeds our $\texttt{Pheromone}_{max}$ value. As the computation time for this is linear in the number of curves present, we restrict the number of curves allowed to a number n, keeping only the most recent n curves for each jump.

The rest of the algorithm operates mostly the same as the previous with a few modifications. As our pheromone function has changed, the state-transition rule is slightly modified. At a given CP-state $\underline{s}$ the ant considers the set of all legal joint actions $\texttt{Actions}(\underline{s})$ which is

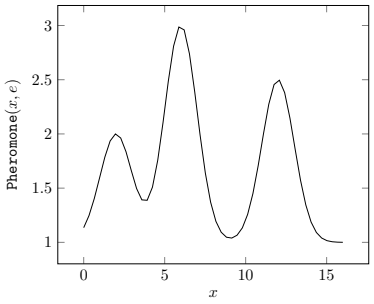


Figure 2 Example 1D pheromone function for a particular jump.

the set of actions (δ, e) such that $\underline{s} + \delta$ satisfies both the current location invariant and the guard of e. The probability of taking an action (δ, e) depends on the pheromone value at the state the discrete transition is taken from: $\texttt{Pheromone}(\underline{s} + \delta, e)$, and the reward and cost that this action contributes. We define the reward-cost ratio of an action (δ, e) from a CP-state $\underline{s} = (q, \mathbf{z}, \mathbf{d})$ as:

$$\texttt{Ratio}(\delta, e, \underline{s})^{-1} := \frac{\texttt{Reward}(q) \cdot \delta + \texttt{Reward(e)}}{\texttt{Cost}(q) \cdot \delta + \texttt{Cost(e)}}$$

The state -transition rule then gives the probability of taking an action (δ, e) in $\texttt{Actions}(\underline{s})$:

$$P(\delta, e) = \frac{\texttt{Pheromone}(\underline{s} + \delta, e)^{\alpha} \cdot \texttt{Ratio}(\delta, e, \underline{s})^{-\beta}}{\sum_{(\delta', e') \in \texttt{Actions}(\underline{s})} \texttt{Pheromone}(\underline{s} + \delta', e')^{\alpha} \cdot \texttt{Ratio}(\delta', e', \underline{s})^{-\beta}}$$

The parameters α and β are weightings of each component as before. Local and global pheromone updates act as before, with the change that whenever pheromone is scaled, instead the height parameter of each component Gaussian curve is scaled, and whenever a value is added, a Gaussian curve with that height is added instead. To keep the number of Gaussian curves low, we set a small real valued constant $\texttt{height}_{min}$. Whenever the scaling of a Gaussian function results in its height being less than $\texttt{height}_{min}$, the curve is removed from that pheromone function.

The remaining parts of the algorithm: iteration, elitist updating and returning of the best solution are identical to that of the simple ant colony optimisation method. The modified pheromone function and state transition rule gives a method for approximating the optimal infinite ratio of the automaton, that takes into account more information than the previous method, but has more significant time and memory requirements due to the pheromone function representation.

3.2 Cycle Optimisation

In the previous methods the delays for a given cycle of the automaton were determined approximately. In this section we introduce a method that, given a set of jumps forming a cycle in the automaton, will exactly determine the delays that give the best possible cost to reward ratio for that cycle. With this method, the optimal infinite scheduling problem only requires the trial of different combinations of discrete transitions.

Let $\mathcal{A} = (Q, q_{start}, \mathcal{X}, \Sigma, E, I, \texttt{Cost}, \texttt{Reward})$ be a double-priced timed automaton, and let $(e_i)_{i=1}^{n} = e_1, e_2, \ldots, e_n$ be a sequence of jumps such that $e_i = (q_{i-1}, \alpha_i, G_i, X_i, q_i) \in E$ with $q_0 = q_n$, that is the edges of $(e_i)_{i=1}^{n}$ form a cycle in $\mathcal{A}$. Given this sequence we

wish to find a cyclic execution of $\mathcal{A}$, $\rho = (q_0, v_{0,0}) \xrightarrow{\delta_0} (q_0, v_{0,1}) \xrightarrow{e_1} (q_1, v_{1,0}) \xrightarrow{\delta_1} \dots \xrightarrow{e_{n-1}} (q_{n-1}, v_{n-1,0}) \xrightarrow{\delta_n - 1} (q_{n-1}, v_{n-1,1}) \xrightarrow{e_n} (q_0, v_{0,0})$ with $\texttt{Ratio}(\rho)$ being minimal. As the jumps involved are fixed, this problem consists of finding optimal delays δ_i such that ρ is a valid execution. The valuations $v_{i,j}$ can then be derived from the delays. Note that unless all delays are 0, then for a solution to exist each clock must be reset by at least one jump, as such we will only consider sequences in which all clocks are reset at least once. To begin with, we will also assume that all invariant and guard conditions are closed, that is they only make use of the weak inequalities $\geq$ and $\leq$.

To solve this problem we will first derive a set of constraints that will ensure that ρ is a valid execution. We then redefine the objective function and constraints with respect to a new set of variables such that the constraints define a bounded polyhedron and the objective function is a quotient of affine functions. This will give a linear-fractional program that we can transform into a linear program to then solve. The constraints, objective function and solution are described in the appendix (Appendix A). This method gives a computable function $\texttt{CycleOpt} : \{e_n\} \to \mathbb{R}$ that maps to each sequence of jumps in the automaton forming a cycle, a real number that is the minimum achievable cost to reward ratio for that particular cycle. If a sequence $e_1, \dots, e_n$ is infeasible, that is no valid execution of the sequence of jumps exists, we set $\texttt{CycleOpt}(e_1, \dots, e_m) = \infty$.

The cycle optimisation can be extended to handle strict inequalities in guards, as described in the appendix (Appendix A.1).

4 Ant Colony Optimisation with Cycle Optimisation

The cycle optimisation algorithm can be applied in conjunction with ant colony optimisation to the optimal infinite scheduling problem. The ants are responsible for testing cycles consisting of different jumps of the automaton, while the cycle optimisation algorithm returns the optimal delays as well as the best achievable cost reward ratio for a cycle composed of those jumps.

The implementation of this algorithm operates largely the same as the simple ant colony optimisation algorithm discussed earlier. Ants traverse the CP-states of the automaton until they reach a cycle, informed by the pheromone function over the set of jumps. The difference occurs around the time of the global pheromone update. In the orginal algorithm each of the best k ants would place pheromone corresponding to the cycle it built. Instead, for each of these ants, the sequence of discrete-transition used in its solution $e_1, \dots, e_n$ is passed into the cycle optimisation method. $\texttt{CycleOpt}(e_1, \dots, e_n)$ is calculated, with the result being used to update the concentration of pheromone along each jump in the sequence. This reduces the drawback of using random delays during exploration, while still having the benefits of the simple pheromone function this enables. During the exploration ants are essentially finding *feasible* sequences of jumps, while the `CycleOpt` function finds the best ratio over a subset of these feasible cycles. This is beneficial as the `CycleOpt` function, the most computationally expensive part of the iteration, only needs to be executed for a small number of the candidate solutions. A drawback that remains in this method is that the pheromone function only informs what set of jumps to use, while the optimal achievable ratio depends on the exact sequence of jumps. The ratio varies on what order the jumps appear in as well as on how many times a jump is taken.

Table 1 Rates of finding the overall minimal ratio on each of the 175 timed automata test cases.

Method	Rate
Ant Colony with CO	61.5%
Simple Ant Colony	44.6%
Random	43.5%
Hybrid Ant Colony	38.2%

Table 2 Normalized average ratios across all test cases. This is given by dividing the ratio obtained in each trial by the best ratio found for that automaton.

Method	Normalized Avg.
Ant Colony with CO	1.131
Random	1.161
Simple Ant Colony	1.202
Hybrid Ant Colony	1.297

5 Results

This section details the execution and results of several experiments designed to test the effectiveness of each of the above algorithms for solving the optimal infinite scheduling. Of interest are the ratios obtained, the time taken to do so, and the types of cycle structures each algorithm is best suited finding. Each algorithm was implemented in Java 8.

As a baseline for comparison with the other algorithms, a method that simply explores the CP-states randomly was implemented. This method generates some number of random cycles in the corner-point abstraction, with the best found cycle being returned as the solution. At each decision point, which delay or jump to take is determined using a uniform distribution. The parameter tuning is described in the appendix (Appendix B).

The primary experiment for evaluation consisted of running each algorithm on 175 randomly generated double priced timed automata. Each method was performed 10 times per automaton with the best found ratio, the average ratio and the average time taken to terminate being recorded. The test case generation (Appendix C) and optimal ratio calculation (Appendix D) are described in the appendix. Even on these small instances it was not feasible to run the deterministic algorithm [4] to confirm the optimal ratio.

5.1 Results and Discussion

For each automaton, the best ratio that either the optimal ratio search or any of the five methods returned was recorded and used as a point of comparison. How frequently each method returned the best ratio for a given automaton was recorded. This is shown in Table 1.

The algorithms making use of cycle optimisation performed the best by this measure, being the only methods to find the optimum in over 50% of the trials. Notably, the hybrid ant colony algorithm found the optimum less frequently than even the random method.

The normalized average ratio found by each method was also recorded. This value is given by taking the ratio found in a particular trial and dividing by the best ratio found across all trials for the same automaton. A normalized average of 1.0 would suggest that the method found the optimum in every trial, with higher values suggesting worst performance. The normalised ratio gives a measure of how close to the optimum the ratios returned by each method were. These results are shown in Table 2. Ant colony optimisation with cycle

Table 3 Rates of taking the optimal loop.

Method	Rate
Ant Colony with CO	52.4%
Simple Ant Colony	50.4%
Hybrid Ant Colony	15.4%
Random	10.4%

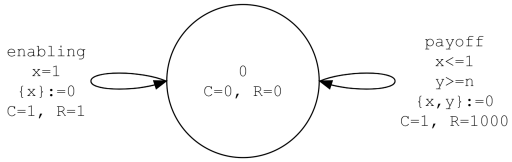


Figure 3 An example loop automaton. The optimal behaviour is given by taking `enabling` $n-1$ times and then taking `payoff`.

optimisation gave the best performance by this measure. Interestingly the random method had the second best performance. However, this performance comes at a price, taking on average 9 seconds per trial, compared with 1 second per trial for the simple ant colony methods. The hybrid and random average 1.5 seconds per trial.

Based on the results of this experiment, ant colony optimisation with cycle optimisation appear to be the best method for finding a profitable cyclic behaviour of a timed automaton. Although this method has a greater time requirement than the others, the improvement to the behaviours found is likely to make this time increase worthwhile. Overall the hybrid ant colony optimisation algorithm was the worst at optimising the behaviour of the automaton, performing even worse than the baseline random method. It is possible that the pheromone representation more frequently led the algorithm astray rather than allowing it to improve upon its solutions in each iteration.

6 Additional Experiments

In addition to the primary experiment, the methods were tested on automata with a particular structure designed to stress the algorithms and find potential weaknesses.

6.1 Jump Repetition

This experiment tested how frequently each method found the optimal cycle when that cycle consisted of taking a particular jump n times followed by a high reward jump. Each test case had 2 clocks and a single location with 0 cost and reward. The jumps consisted of an enabling jump which allows the difference between the two clock values to increase by at most 1 each time it is taken, a payoff jump which can only be taken when the difference between the clocks is at least n and a set of k "noise" jumps. All jumps have cost 1, with the enabling jump having reward 1, the payoff jump having reward 1000 and the noise jumps having rewards in the range 1 to 10. The payoff jumps reward dominates that of all others and hence the optimal ratio is obtained by the cycle where the enabling jump is taken n times and then the payoff jump is taken. Figure 3 shows the structure of the automata.

There were 50 test cases with n ranging from 1 to 10 and k ranging from 1 to 5. In each test case, each method was run 10 times. The percentage of these runs that found the

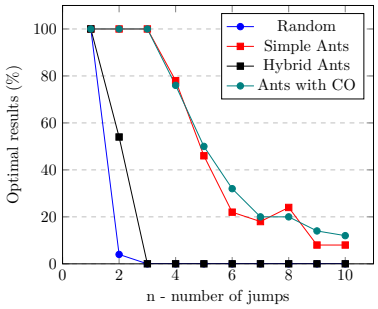


Figure 4 Rate of taking the optimal loop, with respect to number of jumps.

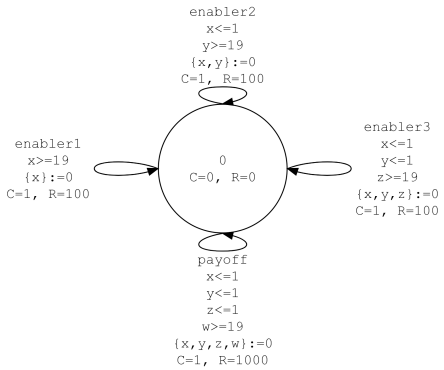


Figure 5 An example sequence automaton of length 4. The optimal behaviour is given by `enabler1` → `enabler2` → `enabler3` → `payoff`.

optimal ratio were recorded. These results are shown in Table 3. Figure 4 shows how the number of times the repeated jump was required to be taken affects how frequently each method found the optimal behaviour.

The two best performing methods were the simple ant colony optimisation, and ant colony optimisation with cycle optimisation. Notably these two methods use a pheromone function associated only with individual jumps, so a high pheromone concentration on the enabling jumps would cause these algorithms to take it more frequently.

6.2 Sequence Length

This experiment consisted of test cases with one location where the optimal cycle consisted of taking a specific sequence of jumps of length n, again in the presence of k "noise" jumps. All jumps had cost 1. Noise jumps had rewards between 1 and 10, the first $n-1$ jumps in the sequence had reward 100 and the final jump has reward 1000. The test case has n clocks, with the ith jump resetting the first i clocks and requiring that the first $i-1$ clocks, but not the ith have been reset, making it so the final jump is only possible to take if the first $n-1$ jumps have been taken in the correct order with no noise jumps (which reset all clocks) in between. The final jump's high reward again dominates, making the set sequence the optimal cycle for the automaton. Figure 5 shows an example with an optimal sequence of length 4.

There were 50 test cases with n ranging from 1 to 10 and k ranging from 1 to 5. In each test case, each method was run 10 times. The percentage of these 10 that found the optimal

Table 4 Rates of taking the optimal sequence.

Method	Rate
Hybrid Ant Colony	100%
Ant Colony with CO	76.6%
Simple Ant Colony	73.6%
Random	27.2%

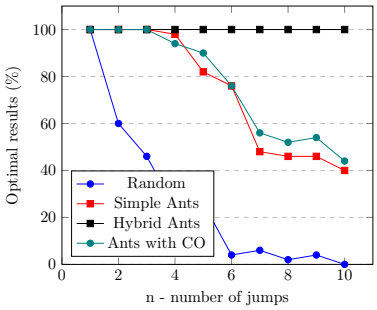


Figure 6 Rate of taking the optimal sequence, with respect to number of jumps.

ratio were recorded. These results are shown in Table 4. Figure 6 shows how the length of the optimal sequence affects how frequently each method found the optimal behaviour.

In this experiment the ant colony optimisation based methods greatly outperformed the other methods. Notably the hybrid ant colony optimisation algorithm found the optimal sequence in every single trial, despite performing very poorly in the other experiments. As the hybrid ant colony optimisation pheromone function varies over where in the clock space a jump is taken from. As what jumps can currently be taken depends on the resets of the jumps taken previously, the function is able to express some heuristic information about the impact of the order of the jumps. This result suggests that while in the general case the hybrid ant colony is less efficient at finding a good behaviour, this pheromone representation is well suited to the cases where a very specific sequence of actions is required.

7 Conclusion

This investigation shows that in many cases it is possible to achieve optimal or near optimal strategies for double priced timed automata using variations of ant colony optimisation, and linear programming for cycle optimisation. However, it also shows that it is possible to engineer test cases with specific solutions that are difficult for some heuristic search methods to find. Future work will work towards finding robust variations that are able to consistently produce near optimal strategies.

References

1 R Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

2 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control*, HSCC '01, pages 147–161, 2001.

3 Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Priced timed automata: Algorithms and applications. In *in International Symposium Formal Methods for Components and Objects (FMCO*, pages 162–182, 2005.

4 Patricia Bouyer, Ed Brinksma, and Kim G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):3–23, 2008.

5 A. Charnes and W. W. Cooper. Programming with linear fractional functionals. *Naval Research Logistics Quarterly*, 9(3-4):181–186, 1962. doi:10.1002/nav.3800090303.

6 George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysi of production and allocation*, 1951.

7 Ali Dasdan, Sandy S. Irani, and Rajesh K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, DAC '99, pages 37–42, 1999.

8 Alexandre David, Daniel Ejsing-Duun, Lisa Fontani, Kim G. Larsen, Vasile Popescu, and Jacob Haubach Smedegård. Optimal infinite runs in one-clock priced timed automata. Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS), 2011.

9 M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *Trans. Sys. Man Cyber. Part B*, 26(1):29–41, 1996.

10 Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

11 Peter Niebert, Stavros Tripakis, and Sergio Yovine. Minimum-time reachability for timed automata. In *IEEE Mediteranean Control Conference*, 2000.

12 Krzysztof Socha and Marco Dorigo. Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155–1173, 2008. doi:10.1016/j.ejor.2006.06.046.

13 P Soustek, R Matousek, J Dvorak, and J Bednar. Canadian traveller problem: A solution using antcolony optimization. In *Proceedings of 19th International Conference on Soft Computing – MENDEL 2013*, page 439–444, 2013.

14 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004. doi:10.1145/990308.990310.

15 Thomas Stützle and Holger H. Hoos. Max–min ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000. doi:10.1016/S0167-739X(00)00043-1.

16 Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Form. Methods Syst. Des.*, 18(1):25–68, 2001.

A Cycle Optimisation

Constraints. We first derive a set of constraints over the clock valuations $v_{i,j}$ that limit the choice of delays to only those that result in a valid execution:

1. $v_{i,1}$ satisfies $G_{(i+1) \pmod n}$ as the jump $e_{(i+1) \pmod n}$ is taken from $v_{i,1}$.
2. If $x \in X_i$ then, $v_{i,0}(x) = 0$ as jump e_i resets clock x.
3. $v_{i,j}$ satisfies $I(q_i)$, as all valuations in the run must satisfy the corresponding invariant condition. As the location invariant defines a convex region in the clock-space, all valuations between $v_{i,0}$ and $v_{i,1}$ will also satisfy the invariant.
4. $v_{i,1}$ is a direct time successor of $v_{i,0}$. That is, for each i there is a non-negative constant δ (corresponding to the delay taken from $v_{i,0}$) such that for x in $\mathcal{X}$, $v_{i,1}(x) = v_{i,0}(x) + \delta$.

Note that while the first three conditions can easily be expressed in terms of clock difference constraints between the valuations, the final constraint cannot as the constant δ depends on the delay chosen for that step of the run. To get around this issue, we need to reframe the constraints with respect to the delays, rather than the valuations. To achieve this, we define a vector of variables $\mathbf{t} = (t_0, \ldots, t_n)$ with $t_i := \sum_{j=0}^{i-1} \delta_j$. t_i can be thought of the total elapsed time between the beginning of the execution and taking jump e_i. We then define each valuation with respect to these variables. In order to do so we first define a function $\texttt{lastReset} : (0, \ldots, n) \times \mathcal{X} \to 0, \ldots, n$, which given an index and a clock gives the index of the most recent jump that reset that clock:

$$\texttt{lastReset}(i, x) := \begin{cases} max(j \leq i \,|\, x \in X_j) & \text{if such a } j \text{ exists} \\ max(j > i \,|\, x \in X_j) & \text{otherwise} \end{cases}$$

As we are considering only sequences of jumps in which all clocks are reset, $\texttt{lastReset}(i, x)$ is defined for all $i = 0, \ldots, n$ and all $x \in \mathcal{X}$. The value of a clock at a given point of our sequence is simply equal to the total time elapsed since that clock was last reset, so we can use this function to define each valuation in terms of $\mathbf{t}$, being careful to take into account the cyclical nature of the sequence:

$$v_{i,0}(x) := \begin{cases} t_i - t_{\texttt{lastReset}(i,x)} & \texttt{lastReset}(i, x) \leq i \\ t_i - t_{\texttt{lastReset}(i,x)} + t_n & \texttt{lastReset}(i, x) > i \end{cases}$$

$$v_{i,1}(x) := \begin{cases} t_{i+1} - t_{\texttt{lastReset}(i,x)} & \texttt{lastReset}(i, x) \leq i \\ t_{i+1} - t_{\texttt{lastReset}(i,x)} + t_n & \texttt{lastReset}(i, x) > i \end{cases}$$

This definition of the valuations conveniently ensures that a run given in terms of $\mathbf{t}$ will satisfy several of the above constraints. Constraint 2 is satisfied as if $x \in X_i$, then $\texttt{lastReset}(i, x) = i$ which implies $v_{i,0}(x) = t_i - t_i = 0$. Constraint 4 is almost satisfied, as for all x in $\mathcal{X}$, $v_{i,1}(x) - v_{i,0}(x) = t_{i+1} - t_i$, which does not depend on x, however we need to ensure that the delay this corresponds to is non-negative. To satisfy the rest of constraint 4, as well as constraints 1 and 3 we build a set of constraints $\mathcal{C}$ on $\mathbf{t}$. To ensure the non-negativity of delays we add the constraints that $t_{i+1} - t_i \geq 0$ as $\delta_i = t_{i+1} - t_i$. Each clock value of a valuation $v_{i,j}$ is expressed as a linear combination of the elements of $\mathbf{t}$, and each location invariant and jump guard is the conjunction of upper or lower bounds on these clock values, so adding these conditions expressed in terms of $\mathbf{t}$ to $\mathcal{C}$ ensures that all runs in our system will be valid. All the conditions in $\mathcal{C}$ are bounds on linear combinations of elements in $\mathbf{t}$, and as the invariants bound the clock values, each t_i is also bounded, hence $\mathcal{C}$ defines a bounded convex polyhedron over $\mathbf{t}$.

Objective Function. The function we are trying to minimize is $\texttt{Ratio}(\rho) = \texttt{Cost}(\rho)/\texttt{Reward}(\rho)$, which we wish to express in terms of $\mathbf{t}$. The cost and reward functions are:

$$\texttt{Cost}(\gamma) = \sum_{i=0}^{n-1} \texttt{Cost}(q_i) \cdot \delta_i + \sum_{i=1}^{n} \texttt{Cost}(e_i)$$

$$\texttt{Reward}(\gamma) = \sum_{i=0}^{n-1} \texttt{Reward}(q_i) \cdot \delta_i + \sum_{i=1}^{n} \texttt{Reward}(e_i)$$

To get our objective function $\texttt{Ratio}(\mathbf{t})$ we substitute $\delta_i = t_{i+1} - t_i$, giving:

$$\texttt{Ratio}(\mathbf{t}) = \frac{\sum_{i=0}^{n-1} \texttt{Cost}(q_i) \cdot (t_{i+1} - t_i) + \sum_{i=1}^{n} \texttt{Cost}(e_i)}{\sum_{i=0}^{n-1} \texttt{Reward}(q_i) \cdot (t_{i+1} - t_i) + \sum_{i=1}^{n} \texttt{Reward}(e_i)}$$

Solving the System. The set of polyhedral constraints $\mathcal{C}$ can be represented by the inequality $A\mathbf{t} \leq \mathbf{b}$, where A is a matrix with a column for each clock. Each row of A corresponds to one of the constrains in C, with the matching entry in $\mathbf{b}$ being the bound on that linear combination of members of $\mathbf{t}$. As the objective function $\texttt{Ratio}(\mathbf{t})$ is a quotient of affine functions of $\mathbf{t}$, this function in conjunction with our constraint matrix A, as well as the implicit constraint that each t_i be non-negative gives a *linear-fractional program*. To solve the system we can first apply the Charnes-Cooper transformation [5] to give an equivalent linear program. If we rewrite our objective function as

$$\texttt{Ratio}(\mathbf{t}) = \frac{\mathbf{c} \cdot \mathbf{t} + \alpha}{\mathbf{d} \cdot \mathbf{t} + \beta}$$

Our transformed system has variables $\mathbf{y}$ and λ with:

$$\mathbf{y} = \frac{1}{\mathbf{d} \cdot \mathbf{t} + \beta} \cdot \mathbf{t};\ \lambda = \frac{1}{\mathbf{d} \cdot \mathbf{t} + \beta}$$

The equivalent linear program is then given by:

$$\begin{array}{ll} \text{minimize} & \mathbf{c} \cdot \mathbf{y} + \alpha\lambda \\ \text{subject to constraints} & A\mathbf{y} \leq \alpha\lambda \\ & \mathbf{d} \cdot \mathbf{y} + \beta\lambda = 1 \end{array}$$

If $\mathbf{y}^*, \lambda^*$ is the solution to the linear program, the solution to the linear fractional program is $\mathbf{t} = \frac{1}{\lambda^*}\mathbf{y}^*$. From this we can extract the optimal delays as $\delta_i = t_{i+1} - t_i$.

The linear program can be solved with a method such as the simplex algorithm [6]. In the worst case this has time complexity $O(2^n)$, but it has been shown that on average the simplex algorithm terminates in polynomial time [14].

A.1 Strict Inequalities in Guards

As described, the cycle optimisation algorithm only works for DPTA whose guards and invariants contain weak inequalities ($\leq$ and $\geq$). This is because linear programs require the feasible region to be closed for the precise optimum to exist. However we are interested in the minimum cost to reward ratio that infinite runs of the automaton can approach arbitrarily close to, even if this limit itself is not obtained from an actual run. This situation can arise when strict inequalities ($<$ and $>$) are present in the guards on invariants of the DPTA. To solve the system in this case we replace all strict inequalities by weak inequalities and construct and solve the linear program as before, obtaining the solution vertex $\mathbf{t}^*$. We then reintroduce the strict inequalities and check if their exists a feasible solution within an arbitrarily small neighbourhood around $\mathbf{t}^*$. If such a solution exists, runs of the automaton can approach the ratio obtained at $\mathbf{t}^*$ by taking delays arbitrarily close to those given by this solution that are inside the feasible region. If not, this means that the feasible region is empty and that this sequence of jumps does not give rise to any valid cycles.

Let $\texttt{active}_{\leq}$ be the set of weak constraints in $\mathcal{C}$ that are active at $\mathbf{t}^*$, that is the constraints $\mathbf{a}_i \cdot \mathbf{t} \leq b_i$ such that $\mathbf{a}_i \cdot \mathbf{t}^* = b_i$. Let $\texttt{active}_{<}$ be the be the equivalent set of strong constraints $\mathbf{a}_i \cdot \mathbf{t} < b_i$. If this set is non-empty, the current solution is not feasible. Each active constraint defines a hyperplane in $\mathbb{R}^{|\mathcal{X}|}$ on which $\mathbf{t}^*$ lies. To check if a neighbouring feasible solution exists, we determine if there exists a direction $\mathbf{v}$ that we can move in from $\mathbf{t}^*$ that moves off of all of the hyperplanes in $\texttt{active}_{<}$ without moving onto the wrong side of any other hyperplane. Such a direction requires that for all constraints $\mathbf{a}_i \cdot \mathbf{t} < b_i$ in $\texttt{active}_{<}$, $\mathbf{v} \cdot \mathbf{a}_i > 0$ and for all constraints $\mathbf{a}_i \cdot \mathbf{t} < b_i$ in $\texttt{active}_{\leq}$, $\mathbf{v} \cdot \mathbf{a}_i \geq 0$. Such a $\mathbf{v}$ does not exist

if for some active strict constraint the other active inequalities can be combined to derive a contradictory constraint. If $\mathbf{a}$ is the normal vector of a constraint in $\texttt{active}_<$, the other constraints are contradictory if there exists a positive linear combination of normal vectors from $\texttt{active}_\leq \cup \texttt{active}_<$ that gives $-\mathbf{a}$, as this implies that both $\mathbf{a} \cdot \mathbf{t} < b$ and $\mathbf{a} \cdot \mathbf{t} \geq b$ are active constraints.

Let $\mathbf{n} \cdot \mathbf{t} < b$ be an active strict constraint. Let $\mathbf{a}_i = (a_{i,1}, \ldots, a_{i,k})$ be the ith active constraint at $\mathbf{t}^*$. We want to check if there exists a positive linear combination of these vectors $\alpha_1 \cdot \mathbf{a}_1 + \ldots + \alpha_m \cdot \mathbf{a}_m$, $\alpha_i \geq 0$ equal to $-\mathbf{n}$. Let A be the matrix whose ith row is $\mathbf{a}_i$. We construct a linear program over $(\alpha_1, \ldots, \alpha_m)$:

$$\begin{aligned} &\text{minimize} && \alpha_1 + \ldots + \alpha_m \\ &\text{subject to constraints} && A^T \cdot \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix} = \mathbf{n} \\ & && \alpha_1, \ldots, \alpha_m \geq 0 \end{aligned}$$

If this program has a feasible solution then the constraint $\mathbf{n} \cdot \mathbf{t} < b$ is contradicted by the other active constraints, meaning that this sequence of jumps does not have a valid execution. This is checked for each constraint in $\texttt{active}_<$.

A.2 Ensuring Reachability

While the execution found by the cycle optimisation algorithm satisfies all the guards and location invariants, the states involved are not necessarily reachable from the starting state of the automaton. To restrict the result to only reachable cycles, additional computation is required. Through the use of zones and the discrete-successor operation, we can construct the *simulation graph* - a graph whose nodes are locations paired with a zone representing a set of reachable states of the automaton and whose edges correspond to jumps of the automaton [11]. The simulation graph is constructed by performing a depth-first search beginning from a zone containing the starting state and its time successors. By taking all the nodes of the simulation graph associated with a given location we get a union of convex zones representing all the reachable valuations within that location.

Given a sequence of jumps $e_1, e_2, \ldots, e_n$ to be optimised, let q be the location from which the jump e_1 is taken from, and let $\texttt{Reach}(q) = \{Z_1, \ldots, Z_k\}$ be the set of reachable zones at q obtained from the simulation graph. To restrict to reachable cycles, the cycle optimisation algorithm needs to be executed for each zone Z_i in $\texttt{Reach}(q)$. For each execution, the constraints defining Z_i are added to the linear fractional constraint in the same way as the location invariant at q. This ensures that the states in q that the feasible cycles visit are contained in Z_i and are therefore reachable. By definition, if one of the states of the cycle is reachable, the entire cycle is reachable. The algorithm is run for each zone in $\texttt{Reach}(q)$, with the cycle giving the best ratio returned as the result.

B Parameter Tuning

For comparison between the different methods, the total number of solutions evaluated was kept fixed at 3000. Several experiments were carried out to determine what combination of population size and number of iterations gave each algorithm the best average performance, measured in terms of the normalised average ratio the method returned over 10 trials with each set of parameters. The best performing parameters were then used for the main

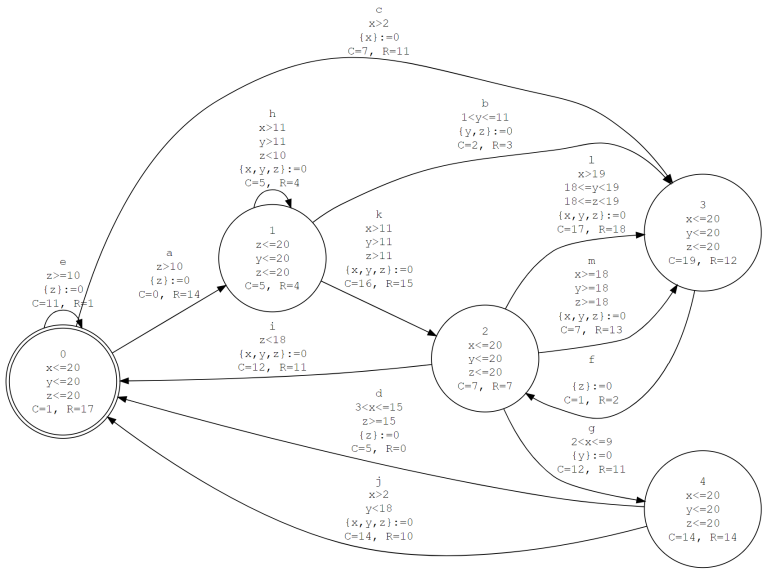


Figure 7 A Generated DPTA with $C = 3, L = 5, J = 5$. Guards and invariants are denoted by inequalities on the clock variables x, y, and z. $X := 0$ denotes a clock reset. C and R denote costs and rewards respectively.

experiments. The ant colony optimisation algorithms perform best with population of 30 ants over 100 iterations. For probability calculations the weighting for pheromone was $\alpha = 1$ and the weighting based on jump and location ratios was $\beta = 1.5$. The evaporation rate used is 0.05 and the decay rate in the local pheromone updates is 0.1. Elitist updates were used with the top 10 ants contributing in the global pheromone update.

C Test Case Generation

Test cases were procedurally generated, ensuring that each automaton met the assumptions of being strongly-reward diverging, deadlock free and bounded. The procedure for generating an automaton takes three inputs: the number of clocks C, the number of locations L, and the minimum number of jumps J. One test case was generated for each assignment of these variables with $1 \leq C \leq 5$, $L \in \{5, 10, 15, 20, 25\}$ and J taking values between L and 45 inclusive at increments of 5. Generation of an automaton begins with a single location with no jumps.

The automaton is built up by either adding a jump from a location to itself, or splitting a location into two. Splitting a location q consists of adding a location q' with a jump from q to q'. Either all of q's inbound jumps are changed to have their destinations be q' or all of q's outbound jumps are changed to have their source be q'. These processes are repeated until the required location and jump counts are reached. All locations have invariants restricting all clocks to be less than or equal to 20. Costs and rewards of locations and jumps take random integer values between 0 and 20, with the exception of jumps added as part of cycles, which have a minimum reward of 1 to ensure reward divergence. Guards and resets are also determined randomly.

Zone based reachability analysis is used to check that all locations are reachable and all jumps can be taken from at least one state. Jump guards are randomly weakened until this is satisfied. Finally, the automaton is checked for deadlocks, again using zone based analysis.

When a deadlock is detected an additional random jump is added that can eventually be taken from all states within that deadlock zone. This process repeats until the automaton is deadlock free. An example of one of the smaller automatons generated via this method is shown in Figure 7.

D Optimal Ratio Calculation

To evaluate the effectiveness of the heuristic methods, it is desirable to know the actual optimal ratio for each generated timed automaton. As a feasible algorithm for calculating this ratio exactly does not currently exist, we instead find the optimal ratio up to a certain cycle length N. To do this we perform a depth first search to generate each discrete cycle of lengths from 1 to N and pass these into the `CycleOpt` function. If the branching factor of the timed automaton (the maximum number of jumps coming out of a single state) is b, then this procedure has worst case time complexity $O(2^N \cdot b^N)$. To balance the amount of time required and the degree of accuracy a maximum cycle length of 8 was used to process each of the 175 test cases.

Computational Complexity of a Core Fragment of Halpern-Shoham Logic

Przemysław Andrzej Wałęga
University of Oxford, United Kingdom
University of Warsaw, Poland
p.a.walega@gmail.com

Abstract

Halpern-Shoham logic (HS) is a highly expressive interval temporal logic but the satisfiability problem of its formulas is undecidable. The main goal in the research area is to introduce fragments of the logic which are of low computational complexity and of expressive power high enough for practical applications. Recently introduced syntactical restrictions imposed on formulas and semantical constraints put on models gave rise to tractable HS fragments for which prototypical real-world applications have already been proposed. One of such fragments is obtained by forbidding diamond modal operators and limiting formulas to the core form, i.e., the Horn form with at most one literal in the antecedent. The fragment was known to be NL-hard and in P but no tight results were known. In the paper we prove its P-completeness in the case where punctual intervals are allowed and the timeline is dense.

Importantly, the fragment is not referential, i.e., it does not allow us to express nominals (which label intervals) and satisfaction operators (which enables us to refer to intervals by their labels). We show that by adding nominals and satisfaction operators to the fragment we reach NP-completeness whenever the timeline is dense or the interpretation of modal operators is weakened (excluding the case when punctual intervals are disallowed and the timeline is discrete). Moreover, we prove that in the case of language containing nominals but not satisfaction operators, the fragment is still NP-complete over dense timelines.

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

Keywords and phrases Temporal Logic, Interval Logic, Computational Complexity, Hybrid Logic

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.23

Funding This work was supported by the NCN grant 2016/23/N/HS1/02168.

1 Introduction

Halpern-Shoham logic (HS) is an elegant and highly expressive interval modal logic with a number of potential applications in the area of temporal *Knowledge Representation and Reasoning* [14, 15, 8]. The logic is known as the logic of Allen's relations, since the modal operators of HS express the well-known Allen's binary relations between intervals, namely *begins* (B), *during* (D), *ends* (E), *overlaps* (O), *adjacent to* (A), *later than* (L), and their converses, which are denoted by $\overline{\mathsf{B}}$, $\overline{\mathsf{D}}$, $\overline{\mathsf{E}}$, $\overline{\mathsf{O}}$, $\overline{\mathsf{A}}$, and $\overline{\mathsf{L}}$, respectively [1] (for a precise definition of the Allen's relations see Table 2). The language of HS contains *diamond* and *box* modal operators corresponding to all Allen relation, which gives 24 modal operators in total (it is known that 4 of them are enough to express the remaining ones [19]), for example $\langle \mathsf{B} \rangle \varphi$ means that "there is an interval beginning the current interval, in which φ holds", and $[\mathsf{B}]\varphi$ that "in all intervals beginning the current interval φ holds."

25th International Symposium on Temporal Representation and Reasoning (TIME 2018).
Editors: Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek; Article No. 23; pp. 23:1–23:18

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

Importantly, the logic allows us to model a continuous flow of time and to interpret formulas over time intervals (rather than time-points), which is essential for various applications in qualitative physics, process planing, and modeling natural language sentences with temporal operators. For instance, HS enables us to represent such sentences as "if you open the tap then, unless someone punctures the canteen, the canteen will eventually be filled" [15], namely with the following formula:

$$OpenTap \rightarrow \langle \mathsf{A} \rangle([\mathsf{D}]\neg Puncture \rightarrow [[\mathsf{EP}]]Filled),$$

whose explicit reading is "whenever the propositional variable *OpenTap* is true, then *there is an adjacent interval* ($\langle \mathsf{A} \rangle$) such that: if the proposition *Puncture* is not true in *any of its subintervals* ($[\mathsf{D}]$), then *in the ending point of this interval* ($[[\mathsf{EP}]]$) the proposition *Filled* is true. The operator $[[\mathsf{EP}]]$ allowing to access the ending time-point of the current intervals does not belong to the language of HS but may be easily expressed in this language [15].

Due to high expressive power, reasoning in HS – which we identify in this paper with the problem of checking whether a given formula of the logic is satisfiable – is undecidable [15]. The result holds for most of the interesting time structures, for example for any class of structures in which the ordering of time-points contains an infinite ascending chain (e.g., natural numbers, integers, and rational numbers). The task of restricting HS to obtain fragments with a good trade-off between expressive power and computational complexity became the main goal for researchers working in this area [13].

The main method of restricting HS – which was already proposed by Halpern and Shoham in their seminal paper – is to limit the number of modal operators occurring in the language [13, 11]. A systematic study of all possible combinations of modal operators resulted in a nearly complete classification, with the easiest fragments being NP-complete, and other PSpace-complete, NExpTime-complete, ExpSpace-complete, or undecidable [5, 7]. Other fragments of HS were obtained by limiting the nesting degree of modal operators, which resulted in decidable, and in particular NP-complete fragments [6].

In this paper we will study the recently introduced methods of restricting HS, which is as follows [8, 10, 16, 20]. First, the diamond modal operators are disallowed and the formulas are in Horn form with at most one literal in antecedent (which we denote by $\mathsf{HS}^{\Box}_{core}$-formulas). We introduce three classifications for time structures:

- Adopting the original, i.e., *irreflexive* definition of Allen's relations, denoted by ($<$), or *weakening* them (see Table 2), which we denote by ($\leq$);
- Allowing punctual intervals, i.e., adopting the *non-strict* definition of an interval, denoted by (Non-S), or disallowing punctual intervals, which results in the *strict* definition of an interval (S);
- Imposing additional conditions on the ordering of time-points, namely their discreteness (Dis) or density (Den).

Combinations of these 3 lines of division give rise to 8 classes of frames. Each combination will be denoted by a sequence of symbols abbreviating chosen types of a time structure, for example irreflexive, non-strict, and discrete structures will be denoted by $(<, \mathsf{Non\text{-}S}, \mathsf{Dis})$. If one of the elements in the tuple is missing, it means that it is not specified, for example $(<, \mathsf{Dis})$ denotes irreflexive and discrete structures, which can be non-strict or strict.

Since $\mathsf{HS}^{\Box}_{core}$-formulas are not referential [21] – in a sense that we cannot use them to label intervals and then to refer to these intervals by labels – we will study their referential counterparts. We will consider $\mathsf{HS}^{\Box,i}_{core}$-formulas obtained by extending $\mathsf{HS}^{\Box}_{core}$-formulas with *nominals*, which are the second sort of atoms that are satisfied in exactly one interval, and

Table 1 Complexity of core HS fragments depending on the structure of time. Contributions of this paper are written in bold and on a gray background, where "undec", "h", and "co" stand for undecidable, hard, and complete, respectively.

Frames:	Irreflexive (<)				Reflexive (≤)			
	Non-Strict (Non-S)		Strict (S)		Non-Strict (Non-S)		Strict (S)	
	Dis	Den	Dis	Den	Dis	Den	Dis	Den
HS	undec	undec	undec	undec	undec	undec	undec	undec
$\mathsf{HS}^{\Box,i,@}_{horn}$	undec	NP-co	undec	NP-co	NP-co	NP-co	PSPACE-h	NP-co
$\mathsf{HS}^{\Box,i}_{horn}$	undec	NP-co	undec	NP-co	NP-co	NP-co	PSPACE-h	NP-co
$\mathsf{HS}^{\Box}_{horn}$	undec	P-co	undec	P-co	P-co	P-co	PSPACE-h	P-co
$\mathsf{HS}^{\Box,i,@}_{core}$	PSPACE-h	**NP-co**	PSPACE-h	**NP-co**	**NP-co**	**NP-co**	**NP-h**	**NP-co**
$\mathsf{HS}^{\Box,i}_{core}$	PSPACE-h	**NP-co**	PSPACE-h	**NP-co**	NL-h	NL-h	NL-h	NL-h
$\mathsf{HS}^{\Box}_{core}$	PSPACE-h	**P-co**	PSPACE-h	NL-h	NL-h	NL-h	NL-h	NL-h

$\mathsf{HS}^{\Box,i,@}_{core}$-formulas obtained by further extending formulas with *satisfaction operators* indexed with nominals, which allow us to refer to the interval in which the particular nominal is satisfied.

In the paper we study the computational complexity of the above mentioned fragments of HS. We show the following results (see also Table 2):

1. $\mathsf{HS}^{\Box}_{core}$-satisfiability is P-complete over (<,Non-S,Den);
2. $\mathsf{HS}^{\Box,i,@}_{core}$-satisfiability is NP-complete over (<,Den), (≤,Non-S), and (≤,S,Den);
3. $\mathsf{HS}^{\Box,i}_{core}$-satisfiability is NP-complete over (<,Den).

The first result partially solves the open problem of determining the computational complexity of $\mathsf{HS}^{\Box}_{core}$-satisfiability which over (<,Den), (≤,Non-S), and (≤,S,Den) was known to be NL-hard and in P, but no tight results were known. The second result, shows that over (<,Den), (≤,Non-S), and (≤,S,Den) the computational complexity of $\mathsf{HS}^{\Box,i,@}_{core}$ and its syntactical extension $\mathsf{HS}^{\Box,i,@}_{horn}$ is the same. The third result shows that over (<,Den) the computational complexity of $\mathsf{HS}^{\Box,i,@}_{core}$-satisfiability and $\mathsf{HS}^{\Box,i,@}_{horn}$-satisfiability is the same.

The paper is organized as follows. In Section 2 we present formally HS and its modifications, in particular we introduce $\mathsf{HS}^{\Box}_{core}$, $\mathsf{HS}^{\Box,i}_{core}$, and $\mathsf{HS}^{\Box,i,@}_{core}$. Then, in Section 3 we show new complexity results, namely in Section 3.1 we prove that $\mathsf{HS}^{\Box}_{core}$-satisfiability is P-complete over (<,Non-S,Den), in Section 3.2 that $\mathsf{HS}^{\Box,i,@}_{core}$-satisfiability is NP-complete over (<,Den), (≤,Non-S), and (≤,S,Den), whereas in Section 3.3 that $\mathsf{HS}^{\Box,i}_{core}$-satisfiability is NP-complete over (<,Den). Finally, in Section 4 we briefly summarize the paper and state the remaining open problems.

2 Core fragments of Halpern-Shoham logic

The language of Halpern-Shoham logic consists of the following pairwise disjoint sets of symbols:

- PROP – an infinite countable set of propositional variables;
- $\{\neg, \wedge\}$ – a set of standard propositional connectives, which consists of negation ($\neg$) and conjunction ($\wedge$);
- $\{\langle \mathsf{B}\rangle, \langle \overline{\mathsf{B}}\rangle, \langle \mathsf{D}\rangle, \langle \overline{\mathsf{D}}\rangle, \langle \mathsf{E}\rangle, \langle \overline{\mathsf{E}}\rangle, \langle \mathsf{O}\rangle, \langle \overline{\mathsf{O}}\rangle, \langle \mathsf{A}\rangle, \langle \overline{\mathsf{A}}\rangle, \langle \mathsf{L}\rangle\}$ – a set of twelve diamond modal operators.;
- $\{[\mathsf{B}], [\overline{\mathsf{B}}], [\mathsf{D}], [\overline{\mathsf{D}}], [\mathsf{E}], [\overline{\mathsf{E}}], [\mathsf{O}], [\overline{\mathsf{O}}], [\mathsf{A}], [\overline{\mathsf{A}}], [\mathsf{L}]\}$ – a set of twelve box modal operators.

We will use the standard abbreviations for disjunction ($\vee$), implication ($\rightarrow$), propositional constants "true" ($\top$), and "false" ($\bot$).

Well-formed HS-formulas are defined by the following abstract grammar:

$$\varphi \stackrel{\text{df}}{=} p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle \mathsf{R} \rangle \varphi \mid [\mathsf{R}]\varphi,$$

where $p \in \mathsf{PROP}$ and $\mathsf{R} \in \{\mathsf{B}, \overline{\mathsf{B}}, \mathsf{D}, \overline{\mathsf{D}}, \mathsf{E}, \overline{\mathsf{E}}, \mathsf{O}, \overline{\mathsf{O}}, \mathsf{A}, \overline{\mathsf{A}}, \mathsf{L}, \overline{\mathsf{L}}\}$.

An HS-frame is a tuple $\mathcal{F} = (\mathbb{D}, I(\mathbb{D}), \mathcal{R})$ such that:

- $\mathbb{D} = (D, \leq)$ is a non-strict linear ordering (a reflexive, antisymmetric, total, and transitive relation) which is unbounded (each element has a $<$-successor and a $<$-predecessor), where for any $x, y \in D$ we define:

  $$x < y \quad \text{iff} \quad (x \leq y) \wedge (x \neq y);$$

- $I(\mathbb{D})$ is a set of intervals over $\mathbb{D}$, which is either:

  $$I^{+}(\mathbb{D}) \stackrel{\text{df}}{=} \{\langle x, y \rangle \mid x, y \in D \text{ and } x \leq y\},$$

  or:

  $$I^{-}(\mathbb{D}) \stackrel{\text{df}}{=} \{\langle x, y \rangle \mid x, y \in D \text{ and } x < y\};$$

- $\mathcal{R}$ is a set of 12 binary relations between intervals, which either equals to:

  $$\mathcal{R}_{<} \stackrel{\text{df}}{=} \{\mathsf{B}, \overline{\mathsf{B}}, \mathsf{D}, \overline{\mathsf{D}}, \mathsf{E}, \overline{\mathsf{E}}, \mathsf{O}, \overline{\mathsf{O}}, \mathsf{A}, \overline{\mathsf{A}}, \mathsf{L}, \overline{\mathsf{L}}\},$$

  or to:

  $$\mathcal{R}_{\leq} \stackrel{\text{df}}{=} \{\mathsf{B}_{\leq}, \overline{\mathsf{B}}_{\leq}, \mathsf{D}_{\leq}, \overline{\mathsf{D}}_{\leq}, \mathsf{E}_{\leq}, \overline{\mathsf{E}}_{\leq}, \mathsf{O}_{\leq}, \overline{\mathsf{O}}_{\leq}, \mathsf{A}_{\leq}, \overline{\mathsf{A}}_{\leq}, \mathsf{L}_{\leq}, \overline{\mathsf{L}}_{\leq}\},$$

  where the relations from $\mathcal{R}_{<}$ and $\mathcal{R}_{\leq}$ are defined in Table 2,

If $\mathcal{R} = \mathcal{R}_{<}$ we say that the HS-frame is *irreflexive*. On the other hand, if $\mathcal{R} = \mathcal{R}_{\leq}$ we say that the frame is *reflexive*. We denote the class of irreflexive frames by $(<)$ and the class of reflexive frames by $(\leq)$. If $I(\mathbb{D}) = I^{+}(\mathbb{D})$ the frame is *non-strict*, i.e., punctual intervals are allowed, and if $I(\mathbb{D}) = I^{-}(\mathbb{D})$ the frame is *strict* and the punctual intervals are disallowed. We denote the classes of non-strict and strict frames by (Non-S) and (S), respectively. Finally, we will distinguish between *discrete* and *dense* frames. If $\mathbb{D}$ is discrete we call the frame *discrete*, and if $\mathbb{D}$ is dense, we call the frame *dense*. The corresponding classes of frames are denoted by (Dis) and (Den), respectively.

Combinations of the above 3 lines of division give rise to several classes of frames. Each combination will be denoted by a sequence of symbols abbreviating the chosen type of a frame, for example irreflexive, non-strict, and discrete frames will be denoted by $(<, \text{Non-S}, \text{Dis})$. Recall that if one of the elements in the tuple is missing, it means that it is not specified, for example $(<, \text{Dis})$ denotes irreflexive and discrete frames, which can be non-strict or strict.

An HS-model is a tuple $\mathcal{M} = (\mathbb{D}, I(\mathbb{D}), \mathcal{R}, V)$ such that $(\mathbb{D}, I(\mathbb{D}), \mathcal{R})$ is an HS-frame and:

$$V : \mathsf{PROP} \longrightarrow \mathcal{P}(I(\mathbb{D})).$$

The satisfaction relation for an HS-model $\mathcal{M} = (\mathbb{D}, I(\mathbb{D}), \mathcal{R}, V)$ and an interval $\langle x, y \rangle \in I(\mathbb{D})$ is defined inductively as follows:

$$\begin{array}{lll}
\mathcal{M}, \langle x, y \rangle \models p & \text{iff} & \langle x, y \rangle \in V(p), \text{ for any } p \in \mathsf{PROP}; \\
\mathcal{M}, \langle x, y \rangle \models \neg\varphi & \text{iff} & \mathcal{M}, \langle x, y \rangle \not\models \varphi; \\
\mathcal{M}, \langle x, y \rangle \models \varphi \wedge \psi & \text{iff} & \mathcal{M}, \langle x, y \rangle \models \varphi \text{ and } \mathcal{M}, \langle x, y \rangle \models \psi; \\
\mathcal{M}, \langle x, y \rangle \models \langle \mathsf{R} \rangle \varphi & \text{iff} & \text{there exists } \langle x', y' \rangle \in I(\mathbb{D}) \text{ such that } \langle x', y' \rangle \models \varphi \\
 & & \text{and if the semantics is irreflexive, then } \langle x, y \rangle \mathsf{R}_{<} \langle x', y' \rangle, \\
 & & \text{whereas if the semantics is reflexive, then } \langle x, y \rangle \mathsf{R}_{\leq} \langle x', y' \rangle.
\end{array}$$

for any HS-formulas φ, ψ and any $\mathsf{R} \in \mathcal{R}_{<}$.

Table 2 Definitions of interval relations in irreflexive and reflexive frames.

Irreflexive frames:		Reflexive frames:	
$\langle x,y\rangle \overline{\mathsf{L}} \langle x',y'\rangle$	iff $y' < x$	$\langle x,y\rangle \overline{\mathsf{L}}_{\leq} \langle x',y'\rangle$	iff $y' \leq x$
$\langle x,y\rangle \overline{\mathsf{A}} \langle x',y'\rangle$	iff $x' < y'$, $y' = x$	$\langle x,y\rangle \overline{\mathsf{A}}_{\leq} \langle x',y'\rangle$	iff $x' \leq y'$, $y' = x$
$\langle x,y\rangle \overline{\mathsf{O}} \langle x',y'\rangle$	iff $x' < x < y' < y$	$\langle x,y\rangle \overline{\mathsf{O}}_{\leq} \langle x',y'\rangle$	iff $x' \leq x \leq y' \leq y$
$\langle x,y\rangle \mathsf{B} \langle x',y'\rangle$	iff $x = x'$, $y' < y$	$\langle x,y\rangle \mathsf{B}_{\leq} \langle x',y'\rangle$	iff $x = x'$, $y' \leq y$
$\langle x,y\rangle \mathsf{D} \langle x',y'\rangle$	iff $x < x'$, $y' < y$	$\langle x,y\rangle \mathsf{D}_{\leq} \langle x',y'\rangle$	iff $x \leq x'$, $y' \leq y$
$\langle x,y\rangle \mathsf{E} \langle x',y'\rangle$	iff $x < x'$, $y = y'$	$\langle x,y\rangle \mathsf{E}_{\leq} \langle x',y'\rangle$	iff $x \leq x'$, $y = y'$
$\langle x,y\rangle \mathsf{O} \langle x',y'\rangle$	iff $x < x' < y < y'$	$\langle x,y\rangle \mathsf{O}_{\leq} \langle x',y'\rangle$	iff $x \leq x' \leq y \leq y'$
$\langle x,y\rangle \mathsf{A} \langle x',y'\rangle$	iff $y = x'$, $x' < y'$	$\langle x,y\rangle \mathsf{A}_{\leq} \langle x',y'\rangle$	iff $y = x'$, $x' \leq y'$
$\langle x,y\rangle \mathsf{L} \langle x',y'\rangle$	iff $y < x'$	$\langle x,y\rangle \mathsf{L}_{\leq} \langle x',y'\rangle$	iff $y \leq x'$
$\langle x,y\rangle \overline{\mathsf{E}} \langle x',y'\rangle$	iff $x' < x$, $y = y'$	$\langle x,y\rangle \overline{\mathsf{E}}_{\leq} \langle x',y'\rangle$	iff $x' \leq x$, $y = y'$
$\langle x,y\rangle \overline{\mathsf{D}} \langle x',y'\rangle$	iff $x' < x$, $y < y'$	$\langle x,y\rangle \overline{\mathsf{D}}_{\leq} \langle x',y'\rangle$	iff $x' \leq x$, $y \leq y'$
$\langle x,y\rangle \overline{\mathsf{B}} \langle x',y'\rangle$	iff $x = x'$, $y < y'$	$\langle x,y\rangle \overline{\mathsf{B}}_{\leq} \langle x',y'\rangle$	iff $x = x'$, $y \leq y'$

An HS-formula φ is satisfiable if and only if there exist an HS-model $\mathcal{M}$ and an interval $\langle x,y\rangle$ such that $\mathcal{M}, \langle x,y\rangle \models \varphi$.

An important representation of an HS-frame $(\mathbb{D}, I(\mathbb{D}), \mathcal{R})$, called a *compass representation*, is obtained by treating an interval $\langle x,y\rangle$ as a point in a two-dimensional Cartesian space $D \times D$ such that the abscissa of this point has a value x and its ordinate has a value y [19]. In compass representation non-punctual intervals correspond to points lying in the north-western half-plane of $D \times D$ (the points whose abscissa is strictly smaller than the ordinate) and the points lying on the diagonal correspond to punctual intervals. In such a setting the Allen's relations gain spatial interpretations. That is, intervals accessible from $\langle x,y\rangle$ with Allen's relations may be determined on the basis of a relative position of the corresponding points in the two-dimensional Cartesian space as presented in Figure 1. Compass representation turned out to be very convenient for proving formal properties of the logic, for instance its decidability and the computational complexity [19, 8].

Restricting the application of propositional connectives in a language is a common method for reducing the complexity of a logic [17, 18]. Recently, the language of HS was constrained in this manner by introducing the fragments $\mathsf{HS}^{\Box}_{horn}$ and $\mathsf{HS}^{\Box}_{core}$, among others [9].

Let $[\mathsf{U}]$ be the universal modality (which is easily expressible in HS), that is for any HS-formula φ, $[\mathsf{U}]\varphi$ is true whenever φ is satisfied in every $\langle x,y\rangle \in I(\mathbb{D})$.

▶ **Definition 1** ($\mathsf{HS}^{\Box}_{core}$-formula). $\mathsf{HS}^{\Box}_{core}$-formulas are generated by the following grammar:

$$\varphi \stackrel{\text{df}}{=} \lambda \mid [\mathsf{U}](\lambda \to \lambda) \mid [\mathsf{U}](\lambda \wedge \lambda \to \bot) \mid \varphi \wedge \varphi,$$

where for $p \in \mathsf{PROP}$ the grammar of *positive temporal intervals* is as follows:

$$\lambda \stackrel{\text{df}}{=} \top \mid \bot \mid p \mid [\mathsf{R}]\lambda.$$

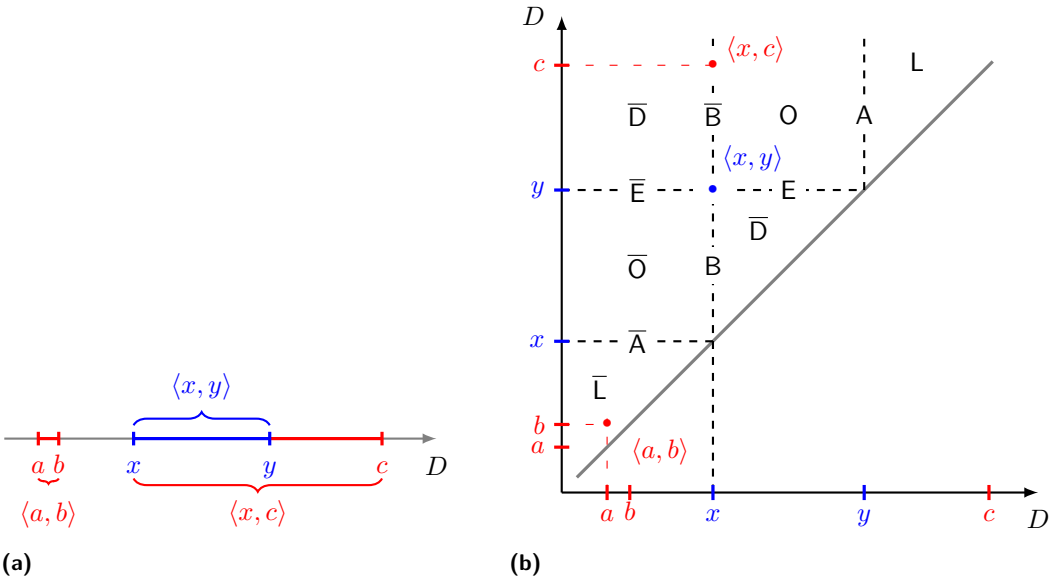


Figure 1 (a) one-dimensional and (b) compass representations of a frame of the Halpern-Shoham logic, in which $\langle x,y\rangle \mathsf{L} \langle a,b\rangle$, and $\langle x,y\rangle \overline{\mathsf{B}} \langle x,c\rangle$.

The idea of introducing the grammar from Definition 1 is based on Fisher's representation of linear temporal logic formulas in *separated normal form* (SNF in short) [12]. SNF enables us to view formulas as sets of initial conditions of the form λ together with universal rules, i.e., implications preceded by $[\mathsf{U}]$.

Although the diamond modal operators do not occur in the language of $\mathsf{HS}^{\Box}_{core}$ they may be express in the antecedent of a clause [8]. For any $\mathsf{R} \in \mathcal{R}_<$ and any positive temporal literals λ_1, λ_2 we define:

$$[\mathsf{U}](\langle \mathsf{R}\rangle \lambda_1 \to \lambda_2) \stackrel{\text{df}}{=} [\mathsf{U}](\lambda_1 \to [\overline{\mathsf{R}}]\lambda_2);$$
$$[\mathsf{U}](\langle \mathsf{R}\rangle \lambda_1 \wedge \lambda_2 \to \bot) \stackrel{\text{df}}{=} [\mathsf{U}](\lambda_1 \to [\overline{\mathsf{R}}]p) \wedge [\mathsf{U}](p \wedge \lambda_2 \to \bot),$$

where p is a new propositional variable, which does not occur in λ_1 and λ_2.

The equisatisfiability of the above formulas follow directly from the semantics of HS.

One of the crucial constructs in temporal knowledge representation is *referentiality*, that is the possibility to label time intervals and then to refer to a chosen interval with a concrete label [2, 3]. The most straightforward way to provide referentiality in a modal logic is to *hybridize a logic* by extending language with:

- NOM – a countable set of nominals different from PROP;
- $\{@_i \mid i \in \mathsf{NOM}\}$ – a set of satisfaction operators indexed with nominals.

In what follows, we consider HS languages with nominals or with nominals and satisfaction operators.

▶ **Definition 2** ($\mathsf{HS}^{\Box,i}_{core}$-formula). $\mathsf{HS}^{\Box,i}_{core}$-formulas are generated by the following grammar:

$$\varphi \stackrel{\text{df}}{=} \lambda \mid [\mathsf{U}](\lambda \to \lambda) \mid [\mathsf{U}](\lambda \wedge \lambda \to \bot) \mid \varphi \wedge \varphi,$$

where for $p \in \mathsf{PROP}$ and $i \in \mathsf{NOM}$:

$$\lambda \stackrel{\text{df}}{=} \top \mid \bot \mid p \mid [\mathsf{R}]\lambda \mid i.$$

▶ **Definition 3** ($\mathsf{HS}^{\Box,i,@}_{core}$-formula). $\mathsf{HS}^{\Box,i,@}_{core}$-formulas are generated by the following grammar:

$$\varphi \stackrel{\text{df}}{=} \lambda \mid [\mathsf{U}](\lambda \to \lambda) \mid [\mathsf{U}](\lambda \wedge \lambda \to \bot) \mid \varphi \wedge \varphi.$$

where for $p \in \mathsf{PROP}$ and $i \in \mathsf{NOM}$:

$$\lambda \stackrel{\text{df}}{=} \top \mid \bot \mid p \mid [\mathsf{R}]\lambda \mid i \mid @_i\lambda.$$

The Horn fragments $\mathsf{HS}^{\Box}_{horn}$, $\mathsf{HS}^{\Box,i}_{horn}$, and $\mathsf{HS}^{\Box,i,@}_{horn}$ are obtained by extending the grammars of $\mathsf{HS}^{\Box}_{core}$, $\mathsf{HS}^{\Box,i}_{core}$, and $\mathsf{HS}^{\Box,i,@}_{core}$ to the following form:

$$\varphi \stackrel{\text{df}}{=} \lambda \mid [\mathsf{U}](\lambda \wedge \lambda \wedge \ldots \wedge \lambda \to \lambda) \mid \varphi \wedge \varphi.$$

A hybrid HS-model $\mathcal{M}$ is a pair $(\mathcal{F}, V)$ such that $\mathcal{F}$ is an HS-frame, and the valuation $V : \mathsf{ATOM} \longrightarrow \mathcal{P}(I(\mathbb{D}))$, for $\mathsf{ATOM} \stackrel{\text{df}}{=} \mathsf{PROP} \cup \mathsf{NOM}$, assigns a set of intervals to each atom with an additional restriction that $V(i)$ is a singleton for any $i \in \mathsf{NOM}$. The satisfaction relation conditions for nominals and @ operators are defined for any hybrid HS-formula φ and any $i \in \mathsf{NOM}$ as follows:

$$\begin{array}{lll} \mathcal{M}, \langle x, y\rangle \models i & \text{iff} & V(i) = \{\langle x, y\rangle\}; \\ \mathcal{M}, \langle x, y\rangle \models @_i\varphi & \text{iff} & \mathcal{M}, \langle x', y'\rangle \models \varphi, \text{ where } \langle x', y'\rangle \text{ is such that} \\ & & V(i) = \{\langle x', y'\rangle\} \text{ and } i \in \mathsf{NOM}. \end{array}$$

We denote the set of propositional variables occurring in a formula φ by $\mathsf{PROP}(\varphi)$, the set of nominals occurring in φ by $\mathsf{NOM}(\varphi)$, and the set of atoms occurring in φ by $\mathsf{ATOM}(\varphi)$. Moreover, by $\mathsf{clauses}(\varphi)$ we denote the set of subformulas of φ which start with the universal modal operator $[\mathsf{U}]$.

Hybrid machinery usually extends expressive power of a modal language and enables to overcome the local nature of the standard modal logic [2, 4]. Interestingly, although the language of HS does not contain hybrid machinery, it is expressive enough to define nominals and satisfaction operators [2]. However, in $\mathsf{HS}^{\Box}_{core}$ we cannot express nominals nor the satisfaction operators over (<,Den), (≤,Non-S), and (≤,S,Den) [21].

3 Computational complexity

3.1 Non-hybrid fragment

In this section we will study the computational complexity of $\mathsf{HS}^{\Box}_{core}$-satisfiability over (<,Non-S,Den). So far, it was shown that this problem is in P and that it is NL-hard [8]. The former result follows from P-completeness of $\mathsf{HS}^{\Box}_{horn}$-satisfiability over (<,Non-S,Den) and the latter from NL-completeness of the satisfiability problem for the core formulas of classical propositional calculus PC. We will prove that $\mathsf{HS}^{\Box}_{core}$-satisfiability is P-hard over (<,Non-S,Den) which implies that this problem is P-complete. First, let us denote by SmallHornSAT the following problem:

input: a formula φ generated by the grammar:

$$\varphi \stackrel{\text{df}}{=} p \mid p \wedge q \to r \mid p \wedge q \to \bot \mid \varphi \wedge \varphi, \tag{1}$$

where $p, q, r \in \mathsf{PROP}$.

output: "yes" if φ is satisfiable in the classical propositional calculus, "no" otherwise.

The problem is in P because each input formula to SmallHornSAT is a Horn formula of PC, and checking whether such a formula is satisfiable is well-known to be P-complete [18]. On the other hand, it is easy to reduce the satisfiability problem of Horn PC-formulas to SmallHornSAT, so P-hardness of the latter problem follows.

▶ **Lemma 4.** SmallHornSAT *is* P*-complete.*

In what follows we will reduce SmallHornSAT to $\mathsf{HS}^{\Box}_{core}$-satisfiability over (<,Non-S,Den), which will imply that the latter problem is P-hard.

▶ **Lemma 5.** $\mathsf{HS}^{\Box}_{core}$*-satisfiability is* P*-hard over (<,Non-S,Den).*

Proof. Fix a formula φ generated by the grammar (1). For any PC-formulas ψ, χ and any $p, q, r \in \mathsf{PROP}$ we define the following translation τ:

$$\tau(p) \stackrel{\text{df}}{=} [\overline{\mathsf{E}}][\mathsf{E}]p; \tag{2}$$

$$\tau(p \wedge q \to r) \stackrel{\text{df}}{=} [\mathsf{U}](p \to [\mathsf{A}]c_{p\wedge q\to r}) \wedge \tag{3}$$

$$[\mathsf{U}](q \to [\mathsf{A}][\overline{\mathsf{E}}]c_{p\wedge q\to r}) \wedge \tag{4}$$

$$[\mathsf{U}]([\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]c_{p\wedge q\to r} \to r); \tag{5}$$

$$\tau(p \wedge q \to \bot) \stackrel{\text{df}}{=} [\mathsf{U}](p \wedge q \to \bot); \tag{6}$$

$$\tau(\varphi_1 \wedge \varphi_2) \stackrel{\text{df}}{=} \tau(\varphi_1) \wedge \tau(\varphi_2), \tag{7}$$

where $c_{p\wedge q\to r}$ is a new propositional variable not occurring in φ and which is distinct for any $p, q, r \in \mathsf{PROP}$. It follows that $\tau(\varphi)$ is an $\mathsf{HS}^{\Box}_{core}$-formula. We claim that the following conditions are equivalent:

1. φ is PC-satisfiable;
2. $\tau(\varphi)$ is HS-satisfiable over (<,Non-S,Den).

(1 ⇒ 2) Assume that φ is PC-satisfiable. Let $\mathsf{v} : \mathsf{PROP} \longrightarrow \{0, 1\}$ be a PC-model such that $\mathsf{v}(\varphi) = 1$. We construct an HS-model $\mathcal{M} = (\mathbb{D}, I^+(\mathbb{D}), \mathcal{R}_<, V)$ such that $\mathbb{D}$ is the set of rational numbers $\mathbb{Q}$ with their standard ordering and V is defined as follows. For any $p \in \mathsf{PROP}$ such that $\mathsf{v}(p) = 1$:

$$V(p) \stackrel{\text{df}}{=} \{\langle x, 0\rangle \mid x \leq 0\}, \tag{8}$$

for any $(p \wedge q \to r) \in \mathsf{clauses}(\varphi)$, such that $\mathsf{v}(p) = 1$ and $\mathsf{v}(q) = 0$:

$$V(c_{p\wedge q\to r}) \stackrel{\text{df}}{=} \{\langle 0, y\rangle \mid 0 \leq y\}, \tag{9}$$

for any $(p \wedge q \to r) \in \mathsf{clauses}(\varphi)$, such that $\mathsf{v}(p) = 0$ and $\mathsf{v}(q) = 1$:

$$V(c_{p\wedge q\to r}) \stackrel{\text{df}}{=} \{\langle x, y\rangle \mid x < 0 \text{ and } 0 < y\}, \tag{10}$$

and for any $(p \wedge q \to r) \in \mathsf{clauses}(\varphi)$, such that $\mathsf{v}(p) = 1$ and $\mathsf{v}(q) = 1$:

$$V(c_{p\wedge q\to r}) \stackrel{\text{df}}{=} \{\langle x, y\rangle \mid x \leq 0 \text{ and } 0 < y\}. \tag{11}$$

An example of a model constructed according to this procedure is depicted in Figure 2.

We claim that $\mathcal{M}, \langle 0, 0\rangle \models \tau(\varphi)$. Fix any $\psi \in \mathsf{clauses}(\tau(\varphi))$. By the construction of $\tau(\varphi)$ the formula ψ is of one of the following forms $[\overline{\mathsf{E}}][\mathsf{E}]p$, $[\mathsf{U}](p \to [\mathsf{A}]c_{p\wedge q\to r})$, $[\mathsf{U}](q \to [\mathsf{A}][\overline{\mathsf{E}}]c_{p\wedge q\to r})$, $[\mathsf{U}]([\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]c_{p\wedge q\to r} \to r)$, or $[\mathsf{U}](p \wedge q \to \bot)$, where $p, q, r \in \mathsf{PROP}(\varphi)$. Systematic inspection

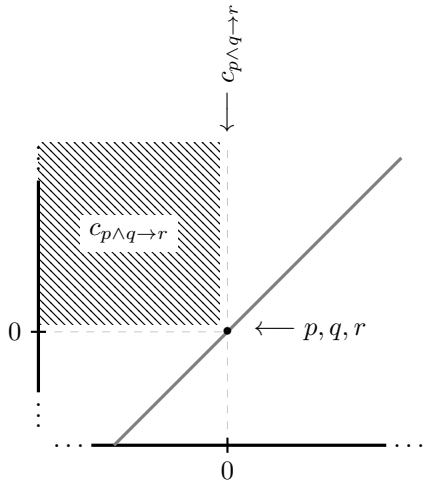


Figure 2 An HS-model constructed for $\varphi = p \wedge q \wedge (p \wedge q \to r)$, where $\longleftarrow p, q, r$ denotes a line in which p, q, and r are satisfied.

of all these cases allows us to show that $\mathcal{M}, \langle 0, 0\rangle \models \psi$ (because of space limits we leave the inspection to the reader). Then, we have $\mathcal{M}, \langle 0, 0\rangle \models \tau(\varphi)$, so $\tau(\varphi)$ is HS-satisfiable over (<,Non-S,Den).

$(1 \Leftarrow 2)$ Assume that $\tau(\varphi)$ is HS-satisfiable over (<,Non-S,Den). Let us fix an HS-model $\mathcal{M} = (\mathbb{D}, I^+(\mathbb{D}), \mathcal{R}_<, V)$ such that $\mathbb{D}$ is dense and $\langle x, y\rangle \in I(\mathbb{D})$ such that $\mathcal{M}, \langle x, y\rangle \models \tau(\varphi)$. We define a PC-model $\mathsf{v} : \mathsf{PROP} \longrightarrow \{0, 1\}$ as follows:

$$\mathsf{v}(p) \stackrel{\text{df}}{=} 1 \qquad \text{iff} \qquad \langle x, y\rangle \in V(p) \text{ and } p \in \mathsf{PROP}(\varphi).$$

We claim that $\mathsf{v}(\varphi) = 1$. Let us fix $\psi \in \mathsf{clauses}(\varphi)$, that is ψ is of the form p, $p \wedge q \to r$, or $p \wedge q \to \bot$ for $p, q, r \in \mathsf{PROP}$. It is easy to show that in all cases $\mathsf{v}(\psi) = 1$. Hence, $\mathsf{v}(\varphi) = 1$.

Conditions 1 and 2 are equivalent and $\tau(\varphi)$ may be constructed in logarythmic space L, so SmallHornSAT reduces in L to HS-satisfiability over (<,Non-S,Den), which ends the proof. ◀

As a result, we obtain the following tight complexity result.

▶ **Theorem 6.** $\mathsf{HS}^{\Box}_{core}$*-satisfiability is* P*-complete over (<,Non-S,Den).*

In the further sections we will study the computational complexity of the satisfiability problem in hybrid extensions of $\mathsf{HS}^{\Box}_{core}$.

3.2 Fragment with nominals and satisfaction operators

In what follows we will study the computational complexity of $\mathsf{HS}^{\Box,i,@}_{core}$-satisfiability and $\mathsf{HS}^{\Box,i}_{core}$-satisfiability. We will show that the former problem is NP-complete over (<,Den), (≤,Dis), and (≤,Den), whereas the latter problem is NP-complete over (<,Den). Since $\mathsf{HS}^{\Box,i}_{core}$-formulas and $\mathsf{HS}^{\Box,i,@}_{core}$-formulas are $\mathsf{HS}^{\Box,i,@}_{horn}$-formulas, the NP upper bound follows from NP-completeness of $\mathsf{HS}^{\Box,i,@}_{horn}$-satisfiability over (<,Den), (≤,Dis), and (≤,Den) [20]. In what follows we will determine the lower bounds.

First, we will establish the lower bound for the complexity of $\mathsf{HS}^{\Box,i,@}_{core}$-satisfiability. Let 3SAT be the following decision problem:

input: a 3CNF formula φ, i.e., a formula generated by the grammar:

$$\varphi \stackrel{\text{df}}{=} (l \vee l \vee l) \mid \varphi \wedge \varphi, \tag{12}$$

where l is a literal, i.e., a propositional variable or a negated propositional variable;
output: "yes" if φ is satisfiable in the classical propositional calculus, "no" otherwise.

To prove the next theorem, we will use the well-known fact that 3CNF is NP-complete [18].

▶ **Lemma 7.** $\mathsf{HS}^{\Box,i,@}_{core}$*-satisfiability is* NP*-hard.*

Proof. We will reduce 3SAT to the problem of $\mathsf{HS}^{\Box,i,@}_{core}$-satisfiability. Let us fix a 3CNF formula φ. We define the following translation τ:

$$\tau(\varphi) \stackrel{\text{df}}{=} i_0 \wedge \bigwedge_{p \in \mathsf{PROP}(\varphi)} [\mathsf{U}](i_0 \wedge i_p \to \bot) \wedge \tag{13}$$

$$\bigwedge_{p \in \mathsf{PROP}(\varphi)} \bigwedge_{\mathsf{R} \in \mathcal{R}_< \setminus \{\mathsf{L}, \overline{\mathsf{L}}\}} [\mathsf{U}](i_0 \wedge \langle \mathsf{R} \rangle i_p \to \bot) \wedge \tag{14}$$

$$\bigwedge_{p \in \mathsf{PROP}(\varphi)} \Big([\mathsf{U}](@_{i_0} \langle \mathsf{L} \rangle i_p \to @_{i_0} p) \wedge [\mathsf{U}](@_{i_0} \langle \overline{\mathsf{L}} \rangle i_p \to @_{i_0} \overline{p}) \Big) \wedge \tag{15}$$

$$p_0 \wedge [\overline{\mathsf{E}}] p_0 \wedge [\mathsf{U}]([\overline{\mathsf{E}}][\mathsf{E}] p_0 \to \bot) \wedge \tag{16}$$

$$\bigwedge_{s \in \mathsf{clauses}(\varphi)} \psi(s), \tag{17}$$

where i_0 and i_p for any $p \in \mathsf{PROP}(\varphi)$ are distinct nominals, p, p_0, and $\overline{p}$ for any $p \in \mathsf{PROP}(\varphi)$ are distinct propositional variables, and for any $s \in \mathsf{clauses}(\varphi)$ we define $\psi(s)$ as follows:

$$\psi(s) \stackrel{\text{df}}{=} p_s \wedge \tag{18}$$

$$[\mathsf{U}](\mathsf{neg}(l^1_s) \to [\overline{\mathsf{E}}] p_s) \wedge \tag{19}$$

$$[\mathsf{U}](\mathsf{neg}(l^2_s) \to [\mathsf{E}] p_s) \wedge \tag{20}$$

$$[\mathsf{U}]([\overline{\mathsf{E}}][\mathsf{E}] p_s \wedge \mathsf{neg}(l^3_s) \to \bot), \tag{21}$$

where for any $s \in \mathsf{clauses}(\varphi)$, p_s is a distinct propositional variable not occurring in φ, $s = (l^1_s \vee l^2_s \vee l^3_s)$ for l^1_s, l^2_s, and l^3_s propositional literals, and $\mathsf{neg}(l^m_s)$ is defined as follows for any $s \in \mathsf{clauses}(\varphi)$ and $m \in \{1, 2, 3\}$:

$$\mathsf{neg}(l^m_s) \stackrel{\text{df}}{=} \begin{cases} p & \text{if } l^m_s = \neg p \text{ for some } p \in \mathsf{PROP}(\varphi); \\ \overline{p} & \text{if } l^m_s = p \text{ for some } p \in \mathsf{PROP}(\varphi). \end{cases} \tag{22}$$

From the definition of the translation it follows that $\tau(\varphi)$ is an $\mathsf{HS}^{\Box,i,@}_{core}$-formula and the translation is in L. The intuition after the translation is as follows. By (13) the current interval is marked by i_0 and by (14) for each $p \in \mathsf{PROP}(\varphi)$ two cases may take place, namely (a) the nominal i_p is satisfied in an interval which is in relation interpreting $\langle \mathsf{L} \rangle$ with the current interval or (b) i_p is satisfied in an interval which is in relation interpreting $\langle \overline{\mathsf{L}} \rangle$ with the current interval. Moreover, by (13) i_p is not satisfied in the current interval. It follows that the cases (a) and (b) are distinct in any HS-frame since :

$$(\mathsf{L} \cap \overline{\mathsf{L}}) \setminus \{(\langle x, y \rangle, \langle x, y \rangle) \mid \langle x, y \rangle \in I(\mathbb{D})\} = \emptyset,$$

and

$$(\mathsf{L}_{\leq} \cap \overline{\mathsf{L}}_{\leq}) \setminus \{(\langle x, y\rangle, \langle x, y\rangle) \mid \langle x, y\rangle \in I(\mathbb{D})\} = \emptyset.$$

By (15) if (a) is the case, then p is satisfied in the current interval, whereas if (b) is the case, then $\overline{p}$ is satisfied in the current interval. The propositional variable $\overline{p}$ is used to simulate $\neg p$ (negation is disallowed in $\mathsf{HS}^{\Box,i}_{horn}$-formulas). The formula (16) forces the interval $\langle x, y\rangle$ in which i_0 is satisfied to be such that y is not the immediate $<$-successor of x. Then, (17) forces each clause of φ to be satisfied in the current interval.

We will show that the following statements are equivalent:

1. φ is PC-satisfiable;
2. $\tau(\varphi)$ is HS-satisfiable.

$(1 \Rightarrow 2)$ Assume that φ is PC-satisfiable and $\mathsf{v} : \mathsf{PROP} \longrightarrow \{0, 1\}$ is an PC-model such that $\mathsf{v}(\varphi) = 1$. We will construct an HS-model $\mathcal{M} = (\mathbb{D}, I(\mathbb{D}), \mathcal{R}, V)$ in which $\tau(\varphi)$ is satisfied. Let $a, b, c, d, e, f \in D$ be such that $a < b < c < d < e < f$ and d is not the immediate $<$-successor of c. Define V as follows:

$$V(i_0) \stackrel{\text{df}}{=} \{\langle c, d\rangle\}; \tag{23}$$

$$V(p_0) \stackrel{\text{df}}{=} \{\langle x, d\rangle \in I(\mathbb{D}) \mid x \leq c\}, \tag{24}$$

for any $p \in \mathsf{PROP}(\varphi)$ such that $\mathsf{v}(p) = 1$:

$$V(i_p) \stackrel{\text{df}}{=} \{\langle e, f\rangle\}; \tag{25}$$

$$V(p) \stackrel{\text{df}}{=} \{\langle c, d\rangle\}, \tag{26}$$

and for any $p \in \mathsf{PROP}(\varphi)$ such that $\mathsf{v}(p) = 0$:

$$V(i_p) \stackrel{\text{df}}{=} \{\langle a, b\rangle\}; \tag{27}$$

$$V(\overline{p}) \stackrel{\text{df}}{=} \{\langle c, d\rangle\}. \tag{28}$$

Moreover, for any clause $s = (l^1_s \vee l^2_s \vee l^3_s)$ of φ, if $\mathsf{v}(l^1_s) = 0$ and $\mathsf{v}(l^2_s) = 1$, then:

$$V(p_s) \stackrel{\text{df}}{=} \{\langle x, d\rangle \in I(\mathbb{D}) \mid x \leq c\}, \tag{29}$$

if $\mathsf{v}(l^1_s) = 1$ and $\mathsf{v}(l^2_s) = 0$, then:

$$V(p_s) \stackrel{\text{df}}{=} \{\langle x, d\rangle \in I(\mathbb{D}) \mid x \geq c\}, \tag{30}$$

and if $\mathsf{v}(l^1_s) = 0$ and $\mathsf{v}(l^2_s) = 0$, then:

$$V(p_s) \stackrel{\text{df}}{=} \{\langle x, d\rangle \in I(\mathbb{D}) \mid x \in D\}. \tag{31}$$

An example of an HS-model obtained by the above presented construction is depicted in Figure 3.

We claim that $\mathcal{M}, \langle c, d\rangle \models \tau(\varphi)$. The formulas (13) and (14) are satisfied in $\langle c, d\rangle$ by (23), (25), and (27). The formula (15) is satisfied in $\langle c, d\rangle$ by (26) and (28). The formula (16) is satisfied in $\langle c, d\rangle$ by the fact that $c < d$, c is not the immediate $<$-successor of d, and by (24). It remains to show that (17) is satisfied in $\langle c, d\rangle$. Towards a contradiction suppose that (17) is not satisfied in $\langle c, d\rangle$, that is for some $s \in \mathsf{clauses}(\varphi)$ it holds that $\mathcal{M}, \langle c, d\rangle \models [\overline{\mathsf{E}}][\mathsf{E}]p_s \wedge \mathsf{neg}(l^3_s)$. By the fact that d is not the immediate $<$-successor of c and

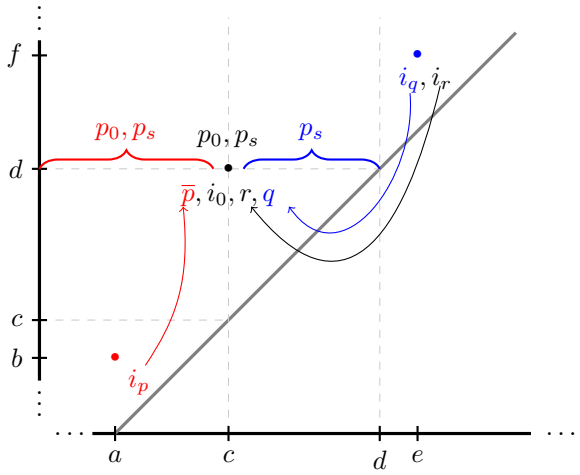


Figure 3 An HS-model constructed for $\varphi = (p \vee \neg q \vee r)$ and for a PC-model v such that $\mathsf{v}(q) = \mathsf{v}(r) = 1$, and $\mathsf{v}(p) = 0$, where a curly bracket denotes a set of points in the compass representation in which a given propositional variables is satisfied.

by (29) it follows that $\mathsf{v}(l_s^1) = \mathsf{v}(l_s^2) = 0$. By (26) and (28) we obtain $\mathsf{v}(l_s^3) = 0$. Hence, $\mathsf{v}(s) = 0$ and consequently $\mathsf{v}(\varphi) = 0$, which raises a contradiction. It follows that (17) is satisfied in $\langle c, d\rangle$.

$(1 \Leftarrow 2)$ Assume that $\tau(\varphi)$ is HS-satisfiable. Let $\mathcal{M} = (\mathbb{D}, I(\mathbb{D}), \mathcal{R}, V)$ be an HS-model under any semantics and $\langle x, y\rangle \in I(D)$ such that $\mathcal{M}, \langle x, y\rangle \models \tau(\varphi)$. We define a PC-model $\mathsf{v} : \mathsf{PROP} \longrightarrow \{0, 1\}$ for any $p \in \mathsf{PROP}(\varphi)$ as follows:

$$\mathsf{v}(p) = 1 \qquad \text{iff} \qquad \langle x, y\rangle \in V(p). \tag{32}$$

It remains to show that $\mathsf{v}(\varphi) = 1$. Towards a contradiction suppose that there is $(l_s^1 \vee l_s^2 \vee l_s^3) \in \mathsf{clauses}(\varphi)$ such that $\mathsf{v}(l_s^1) = \mathsf{v}(l_s^2) = \mathsf{v}(l_s^3) = 0$. By (22) and (32) we have $\mathcal{M}, \langle x, y\rangle \models \mathsf{neg}(l_s^1) \wedge \mathsf{neg}(l_s^2) \wedge \mathsf{neg}(l_s^3)$. By (19) and (20) we obtain $\mathcal{M}, \langle x, y\rangle \models [\overline{\mathsf{E}}][\mathsf{E}]p_s$. Hence, we have $\mathcal{M}, \langle x, y\rangle \models [\overline{\mathsf{E}}][\mathsf{E}]p_s \wedge \mathsf{neg}(l_s^3)$, which raises a contradiction by (21). As a result, for any $s \in \mathsf{clauses}(\varphi)$ we have $\mathsf{v}(s) = 1$, so $\mathsf{v}(\varphi) = 1$, which ends the proof. ◀

As shown in [20] $\mathsf{HS}_{horn}^{\Box,i}$-satisfiability is in NP over (<,Den), (≤,Dis), and (≤,Non-S,Den). Hence, by Lemma 7 we obtain the following tight complexity result.

▶ **Theorem 8.** $\mathsf{HS}_{core}^{\Box,i,@}$*-satisfiability is* NP*-complete over (<,Den), (≤,Dis), (≤,Non-S,Den).*

3.3 Fragment with nominals only

Next, we will study the computational complexity of $\mathsf{HS}_{core}^{\Box,i}$. We will modify the proof of Lemma 7 in order to show that $\mathsf{HS}_{core}^{\Box,i}$-satisfiability is NP-hard over (<).

▶ **Lemma 9.** $\mathsf{HS}_{core}^{\Box,i}$*-satisfiability is* NP*-hard over (<).*

Proof. We will modify the proof of Lemma 7 in order to reduce 3SAT to $\mathsf{HS}_{core}^{\Box,i}$-satisfiability

over $(<)$. Let us fix a 3CNF formula φ. We define the following translation τ:

$$\tau(\varphi) \stackrel{\text{df}}{=} i_0 \wedge \bigwedge_{p \in \mathsf{PROP}(\varphi)} [\mathsf{U}](i_0 \wedge i_p \to \bot) \wedge \tag{33}$$

$$\bigwedge_{p \in \mathsf{PROP}(\varphi)} \bigwedge_{\mathsf{R} \in \mathcal{R}_< \setminus \{\mathsf{B}, \overline{\mathsf{B}}\}} [\mathsf{U}](i_0 \wedge \langle \mathsf{R} \rangle i_p \to \bot) \wedge \tag{34}$$

$$\bigwedge_{p \in \mathsf{PROP}(\varphi)} \Big([\mathsf{U}](i_p \to [\overline{\mathsf{B}}]\overline{p}) \wedge [\mathsf{U}](i_p \to [\mathsf{B}]p) \Big) \wedge \tag{35}$$

$$p_0 \wedge [\overline{\mathsf{E}}]p_0 \wedge [\mathsf{U}]([\overline{\mathsf{E}}][\mathsf{E}]p_0 \to \bot) \wedge \tag{36}$$

$$\bigwedge_{s \in \mathsf{clauses}(\varphi)} \psi(s), \tag{37}$$

where i_0 and i_p for any $p \in \mathsf{PROP}(\varphi)$ are distinct nominals, p, p_0, and $\overline{p}$ for any $p \in \mathsf{PROP}(\varphi)$ are distinct propositional variables, and for any $s \in \mathsf{clauses}(\varphi)$ the formula $\psi(s)$ is defined in (18)–(21).

We claim that the following statements are equivalent:

1. φ is PC-satisfiable;
2. $\tau(\varphi)$ is HS-satisfiable.

$(1 \Rightarrow 2)$ Assume that φ is PC-satisfiable and $\mathsf{v} : \mathsf{PROP} \longrightarrow \{0, 1\}$ is an PC-model such that $\mathsf{v}(\varphi) = 1$. We will construct an HS-model $\mathcal{M} = (\mathbb{D}, I(\mathbb{D}), \mathcal{R}_<, V)$ in which $\tau(\varphi)$ is satisfied. Let $a, b, c, d \in D$ be such that $a < b < c < d$. Define V as follows:

$$V(i_0) \stackrel{\text{df}}{=} \{\langle a, c \rangle\}; \tag{38}$$

$$V(p_0) \stackrel{\text{df}}{=} \{\langle x, c \rangle \in I(\mathbb{D}) \mid x \leq a\}, \tag{39}$$

for any $p \in \mathsf{PROP}(\varphi)$ such that $\mathsf{v}(p) = 1$:

$$V(i_p) \stackrel{\text{df}}{=} \{\langle a, d \rangle\}; \tag{40}$$

$$V(p) \stackrel{\text{df}}{=} \{\langle a, y \rangle \in I(\mathbb{D}) \mid y < d\}, \tag{41}$$

and for any $p \in \mathsf{PROP}(\varphi)$ such that $\mathsf{v}(p) = 0$:

$$V(i_p) \stackrel{\text{df}}{=} \{\langle a, b \rangle\}; \tag{42}$$

$$V(\overline{p}) \stackrel{\text{df}}{=} \{\langle a, y \rangle \in I(\mathbb{D}) \mid b < y\}. \tag{43}$$

Moreover, for each clause $s = (l_s^1 \vee l_s^2 \vee l_s^3)$ of φ if $\mathsf{v}(l_s^1) = 0$ and $\mathsf{v}(l_s^2) = 1$, then:

$$V(p_s) \stackrel{\text{df}}{=} \{\langle a, c \rangle\} \cup \{\langle x, y \rangle \in I(\mathbb{D}) \mid x < a \text{ and } b \leq y\}, \tag{44}$$

if $\mathsf{v}(l_s^1) = 1$ and $\mathsf{v}(l_s^2) = 0$, then:

$$V(p_s) \stackrel{\text{df}}{=} \{\langle a, c \rangle\} \cup \{\langle x, y \rangle \in I(\mathbb{D}) \mid a < x \text{ and } y \leq d\}, \tag{45}$$

and if $\mathsf{v}(l_s^1) = 0$ and $\mathsf{v}(l_s^2) = 0$, then:

$$V(p_s) \stackrel{\text{df}}{=} \{\langle a, c \rangle\} \cup \{\langle x, y \rangle \in I(\mathbb{D}) \mid x < a \text{ and } b \leq y\} \cup \{\langle x, y \rangle \in I(\mathbb{D}) \mid a < x \text{ and } y \leq d\}. \tag{46}$$

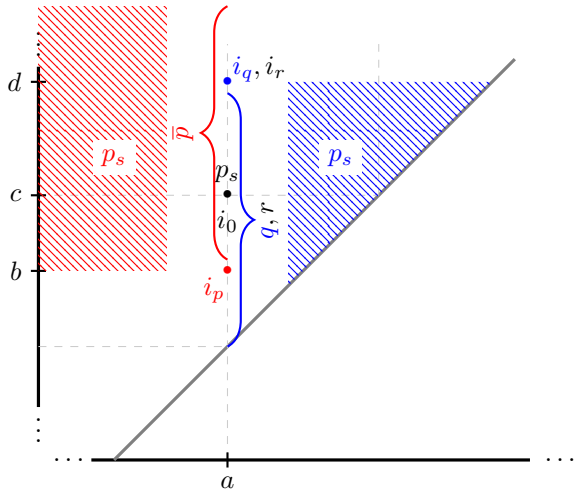


Figure 4 An HS-model constructed for a formula $\varphi = (p \vee \neg q \vee r)$ and for a PC-model v such that $\mathsf{v}(q) = \mathsf{v}(r) = 1$, and $\mathsf{v}(p) = 0$. For the sake of clarity we do not show on the picture that $V(p_0) = \{\langle x, c\rangle \mid x \leq a\}$.

An example of an HS-model obtained by this construction is depicted in Figure 4.
We will show that $\mathcal{M}, \langle a, c\rangle \models \tau(\varphi)$. Formulas (33) and (34) are satisfied in $\langle a, c\rangle$ by (38), (40), and (42). The formula (35) is satisfied in $\langle a, c\rangle$ by (41) and (43). The formula (36) is satisfied in $\langle a, c\rangle$ by the fact that $a < b < c$, that is c is not the immediate $<$-successor of a, and by (39). It remains to show that (37) is satisfied in $\langle a, c\rangle$. Towards a contradiction suppose that (37) is not satisfied in $\langle a, c\rangle$, that is for some $s \in \mathsf{clauses}(\varphi)$ it holds that $\mathcal{M}, \langle a, c\rangle \models [\overline{\mathsf{E}}][\mathsf{E}]p_s \wedge \mathsf{neg}(l_s^3)$. It follows that (46) was applied, so $\mathsf{v}(l_s^1) = \mathsf{v}(l_s^2) = 0$. Since $\mathcal{M}, \langle a, c\rangle \models \mathsf{neg}(l_s^3)$, by (22), (41), and (41) we have $\mathsf{v}(l_s^3) = 0$. Hence, $\mathsf{v}(s) = 0$ and consequently $\mathsf{v}(\varphi) = 0$ which raises a contradiction with the assumption that $\mathsf{v}(\varphi) = 1$. As a result, (37) is satisfied in $\langle a, c\rangle$ so $\mathcal{M}, \langle a, c\rangle \models \tau(\varphi)$.

$(1 \Leftarrow 2)$ Assume that $\tau(\varphi)$ is HS-satisfiable over $(<)$. Let $\mathcal{M} = (\mathbb{D}, I(\mathbb{D}), \mathcal{R}_<, V)$ be an HS-model and $\langle x, y\rangle$ such that $\mathcal{M}, \langle x, y\rangle \models \tau(\varphi)$. We define an PC-model $\mathsf{v} : \mathsf{PROP} \longrightarrow \{0, 1\}$ for any $p \in \mathsf{PROP}$ as follows:

$$\mathsf{v}(p) = 1 \qquad \text{iff} \qquad \langle x, y\rangle \in V(p). \tag{47}$$

We will show that $\mathsf{v}(\varphi) = 1$. Towards a contradiction let us suppose that there exists $(l_s^1 \vee l_s^2 \vee l_s^3) \in \mathsf{clauses}(\varphi)$ such that $\mathsf{v}(l_s^1) = \mathsf{v}(l_s^2) = \mathsf{v}(l_s^3) = 0$. By (47) and (22) we have $\mathcal{M}, \langle x, y\rangle \models \mathsf{neg}(l_s^1) \wedge \mathsf{neg}(l_s^2) \wedge \mathsf{neg}(l_s^3)$. By (19) and (20) we obtain $\mathcal{M}, \langle x, y\rangle \models [\overline{\mathsf{E}}][\mathsf{E}]p_s$. Hence, we have $\mathcal{M}, \langle x, y\rangle \models [\overline{\mathsf{E}}][\mathsf{E}]p_s \wedge \mathsf{neg}(l_s^3)$ which raises a contradiction due to (21). As a result, for any $s \in \mathsf{clauses}(\varphi)$ we have $\mathsf{v}(s) = 1$, so $\mathsf{v}(\varphi) = 1$, which ends the proof. ◀

As we have already stated $\mathsf{HS}_{horn}^{\Box,i,@}$-satisfiability is in NP over $(<,\mathsf{Den})$, $(\leq,\mathsf{Dis})$, and $(\leq,\mathsf{Den})$ [20]. Then, by Lemma 9 we obtain the following result.

▶ **Theorem 10.** *$\mathsf{HS}_{core}^{\Box,i}$-satisfiability is* NP*-complete over $(<,\mathsf{Den})$.*

4 Conclusions

In the paper we have studied the computational complexity of core fragments of Halpern-Shoham logic. We have showed that $\mathsf{HS}_{core}^{\Box}$-satisfiability is P-complete over $(<,\mathsf{Non\text{-}S},\mathsf{Den})$,

$\mathsf{HS}^{\Box,i,@}_{core}$-satisfiability is NP-complete over (<,Den), (≤,Non-S), and (≤,S,Den), whereas $\mathsf{HS}^{\Box,i}_{core}$-satisfiability is NP-complete over (<,Den).

Notice that the satisfiability problem for Horn PC-formulas is P-complete and for core PC-formulas it is NL-complete [18]. Similarly, the satisfiability problem for Horn formulas (without diamond modal operators) of modal logics K, T, K4, and S4 is P-complete and for core formulas it is NL-complete [10]. In the case of Halpern-Shoham logic it is known that $\mathsf{HS}^{\Box}_{horn}$-satisfiability is P-complete over (<,Den), (≤,Non-S), and (≤,S,Den) but our results implies that $\mathsf{HS}^{\Box}_{core}$-satisfiability is not NL-complete over these classes of frames. Moreover, our results for hybrid $\mathsf{HS}^{\Box}_{core}$ fragments show that (over particular classes of frames) the computational complexity of Horn and core hybrid HS fragments is also the same. Hence, to understand the interplay between the computational complexity of HS-fragments and the adopted structure of frames it is interesting to answer the following question:

- Is there a class of frames over which $\mathsf{HS}^{\Box}_{core}$-satisfiability is computationally easier than $\mathsf{HS}^{\Box}_{horn}$-satisfiability?

The same question may be asked according to the hybrid extensions of $\mathsf{HS}^{\Box}_{core}$, namely:

- Is there a class of frames over which $\mathsf{HS}^{\Box,i}_{core}$ ($\mathsf{HS}^{\Box,i,@}_{core}$) is computationally easier than $\mathsf{HS}^{\Box,i}_{horn}$ ($\mathsf{HS}^{\Box,i,@}_{horn}$)?

It is known that over (<,Dis) HS_{horn}-satisfiability is undecidable, but for HS_{core}-satisfiability only the PSPACE-hardness was shown [8], hence the question arises:

- Is HS_{core}-satisfiability decidable over (<,Dis)? If yes, then what is its complexity?

References

1. James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
2. Carlos Areces, Patrick Blackburn, and Maarten Marx. The computational complexity of hybrid temporal logics. *Logic Journal of IGPL*, 8(5):653–679, 2000.
3. Patrick Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3), 2000.
4. Patrick Blackburn, Maarten De Rijke, and Yde Venema. *Modal Logic*, volume 53. Cambridge University Press, 2002.
5. Davide Bresolin, Dario Della Monica, Angelo Montanari, Pietro Sala, and Guido Sciavicco. Interval temporal logics over strongly discrete linear orders: expressiveness and complexity. *Theoretical Computer Science*, 560:269–291, 2014.
6. Davide Bresolin, Dario Della Monica, Angelo Montanari, Pietro Sala, and Guido Sciavicco. The light side of interval temporal logic: The Bernays-Schönfinkel's fragment of CDT. *Annals of Mathematics and Artificial Intelligence*, 71:11–39, 2014.
7. Davide Bresolin, Dario Della Monica, Angelo Montanari, Pietro Sala, and Guido Sciavicco. On the complexity of fragments of the modal logic of Allen's relations over dense structures. In *LATA*, pages 511–523, 2015.
8. Davide Bresolin, Agi Kurucz, Emilio Muñoz-Velasco, Vladislav Ryzhikov, Guido Sciavicco, and Michael Zakharyaschev. Horn fragments of the Halpern-Shoham interval temporal logic. *ACM Transactions on Computational Logic (TOCL)*, 18(3):22:1–22:39, 2017.
9. Davide Bresolin, Emilio Muñoz-Velasco, and Guido Sciavicco. Sub-propositional fragments of the interval temporal logic of Allen's relations. In *European Workshop on Logics in Artificial Intelligence*, pages 122–136. Springer, 2014.
10. Davide Bresolin, Emilio Munoz-Velasco, and Guido Sciavicco. On the complexity of fragments of Horn modal logics. In *Temporal Representation and Reasoning (TIME), 2016 23rd International Symposium on*, pages 186–195. IEEE, 2016.

11 Dario Della Monica, Valentin Goranko, Angelo Montanari, Guido Sciavicco, et al. Interval temporal logics: a journey. *Bulletin of EATCS*, 3(105), 2013.

12 Michael Fisher. A resolution method for temporal logic. In *IJCAI*, volume 91, pages 99–104, 1991.

13 Valentin Goranko, Angelo Montanari, and Guido Sciavicco. A road map of interval temporal logics and duration calculi. *Journal of Applied Non-Classical Logics*, 14(1-2):9–54, 2004.

14 Joseph Y Halpern and Yoav Shoham. A propositional modal logic of time intervals. In *Proceedings of the First IEEE Symposium on Logic in Computer Science*, pages 279–292. Computer Society Press, 1986.

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

16 Roman Kontchakov, Laura Pandolfo, Luca Pulina, Vladislav Ryzhikov, and Michael Zakharyaschev. Temporal and spatial OBDA with many-dimensional Halpern-Shoham logic. In *IJCAI*, pages 1160–1166, 2016.

17 Linh A Nguyen. On the complexity of fragments of modal logics. *Advances in Modal logic*, 5:318–330, 2004.

18 Christos H Papadimitriou. *Computational Complexity*. John Wiley and Sons Ltd., 2003.

19 Yde Venema. Expressiveness and completeness of an interval tense logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990.

20 Przemysław A Wałęga. Computational complexity of a hybridized Horn fragment of Halpern-Shoham logic. In *Indian Conference on Logic and Its Applications*, pages 224–238. Springer, 2017.

21 Przemysław A Wałęga. On expressiveness of Halpern-Shoham logic and its Horn fragments. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 90. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

A Appendix

▶ **Lemma 4.** SmallHornSAT is P-complete.

Proof. The Horn PC-formulas are generated by the following abstract grammar, where $p, q, p_1, \ldots, p_n \in \mathsf{PROP}$:

$$\varphi \stackrel{\text{df}}{=} p \mid (p_1 \wedge p_2 \wedge \ldots \wedge p_n \to q) \mid (p_1 \wedge p_2 \wedge \ldots \wedge p_n \to \bot) \mid \varphi \wedge \varphi. \tag{48}$$

It is well known that satisfiability of Horn PC-formulas is P-complete [18]. Since each input formula to SmallHornSAT is a Horn formula of PC, SmallHornSAT is also in P.

To show that SmallHornSAT is P-hard we will reduce satisfiability problem of Horn PC-formulas to SmallHornSAT in L. Let φ be a Horn PC-formula. For any PC-formulas ψ, χ and any $p, q, p_1, p_2, \ldots, p_n \in \mathsf{PROP}$ we introduce the following translation τ:

$$\begin{aligned}
\tau(p) &= p;\\
\tau(p_1 \wedge p_2 \wedge \ldots \wedge p_n \to q) &\stackrel{\text{df}}{=} (p_1 \wedge p_2 \to p_{1,2}) \wedge (p_{1,2} \wedge p_3 \to p_{1,2,3}) \wedge \\
&\qquad \ldots \wedge (p_{1,\ldots,n-1} \wedge p_n \to q);\\
\tau(p_1 \wedge p_2 \wedge \ldots \wedge p_n \to \bot) &\stackrel{\text{df}}{=} (p_1 \wedge p_2 \to p_{1,2}) \wedge (p_{1,2} \wedge p_3 \to p_{1,2,3}) \wedge \\
&\qquad \ldots \wedge (p_{1,\ldots,n-1} \wedge p_n \to \bot);\\
\tau(\psi \wedge \chi) &\stackrel{\text{df}}{=} \tau(\psi) \wedge \tau(\chi),
\end{aligned}$$

where $p_{1,2}, p_{1,2,3}, \ldots, p_{1,2,3,\ldots,n} \in \mathsf{PROP}$ are new distinct propositional variables which do not occur in φ. It follows that $\tau(\varphi)$ belongs to the grammar (1). Moreover, $\tau(\varphi)$ is equisatisfiable with φ and it is constructed in L. Then, SmallHornSAT is P-hard, which ends the proof. ◀

▶ **Lemma 5.** $\mathsf{HS}^{\Box}_{core}$-satisfiability is P-hard over (<,Non-S,Den).

Proof. In $(1 \Rightarrow 2)$ implication we claimed that $\mathcal{M}, \langle 0,0\rangle \models \tau(\varphi)$. The proof is as follows. Fix any $\psi \in \mathsf{clauses}(\tau(\varphi))$. By the construction of $\tau(\varphi)$ the formula ψ is of one of the following forms $[\overline{\mathsf{E}}][\mathsf{E}]p$, $[\mathsf{U}](p \to [\mathsf{A}]c_{p\wedge q\to r})$, $[\mathsf{U}](q \to [\mathsf{A}][\overline{\mathsf{E}}]c_{p\wedge q\to r})$, $[\mathsf{U}]([\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]c_{p\wedge q\to r} \to r)$, or $[\mathsf{U}](p \wedge q \to \bot)$, where $p, q, r \in \mathsf{PROP}(\varphi)$.

(Case 1): $\psi = [\overline{\mathsf{E}}][\mathsf{E}]p$ for some $p \in \mathsf{PROP}(\varphi)$. Then, ψ was obtained by (2), so $p \in \mathsf{clauses}(\varphi)$. Since $\mathsf{v}(\varphi) = 1$, we obtain that $\mathsf{v}(p) = 1$. Then, by (8) $V(p) = \{\langle x, 0\rangle \in I(\mathbb{D})\}$. Hence, $\mathcal{M}, \langle 0,0\rangle \models [\overline{\mathsf{E}}][\mathsf{E}]p$.

(Case 2): $\psi = [\mathsf{U}](p \to [\mathsf{A}]c_{p\wedge q\to r})$ for some $p, q, r \in \mathsf{PROP}(\varphi)$. By (2)–(7) we have $(p \wedge q \to r) \in \mathsf{clauses}(\varphi)$. We want so show $\mathcal{M}, \langle 0,0\rangle \models \psi$. Let us fix any $\langle x, y\rangle \in I(\mathbb{D})$ and assume that $\mathcal{M}, \langle x, y\rangle \models p$. By the construction of $\mathcal{M}$ it follows that (8) was applied, hence $\mathsf{v}(p) = 1$, $y = 0$, and $x \leq 0$. We need to show that $\mathcal{M}, \langle x, 0\rangle \models [\mathsf{A}]c_{p\wedge q\to r}$, that is $\{\langle 0, y'\rangle \mid 0 < y'\} \subseteq V(c_{p\wedge q\to r})$.

- **(Case 2.1):** $\mathsf{v}(q) = 0$. Then, by (9) $V(c_{p\wedge q\to r}) = \{\langle 0, y'\rangle \mid 0 \leq y'\} \supseteq \{\langle 0, y'\rangle \mid 0 < y'\}$. As a result, $\mathcal{M}, \langle x, 0\rangle \models [\mathsf{A}]c_{p\wedge q\to r}$.
- **(Case 2.2):** $\mathsf{v}(q) = 1$. Then, by (11) we have $V(c_{p\wedge q\to r}) = \{\langle x', y'\rangle \mid x' \leq 0 \text{ and } 0 < y'\} \supseteq \{\langle 0, y'\rangle \mid 0 < y'\}$. It follows that $\mathcal{M}, \langle x, 0\rangle \models [\mathsf{A}]c_{p\wedge q\to r}$.

(Case 3): $\psi = [\mathsf{U}](q \to [\mathsf{A}][\overline{\mathsf{E}}]c_{p\wedge q\to r})$ for some $p, q, r \in \mathsf{PROP}(\varphi)$. By (2)–(7) we have $(p \wedge q \to r) \in \mathsf{clauses}(\varphi)$. We want so show $\mathcal{M}, \langle 0,0\rangle \models \psi$. Let us fix any $\langle x, y\rangle \in I(\mathbb{D})$ and assume that $\mathcal{M}, \langle x, y\rangle \models q$. By the construction of $\mathcal{M}$ it follows that (8) was applied, hence $\mathsf{v}(q) = 1$, $y = 0$, and $x \leq 0$. We need to show that $\mathcal{M}, \langle x, 0\rangle \models [\mathsf{A}][\overline{\mathsf{E}}]c_{p\wedge q\to r}$, that is $\{\langle x', y'\rangle \mid x' < 0 \text{ and } 0 < y'\} \subseteq V(c_{p\wedge q\to r})$.

- **(Case 3.1):** $\mathsf{v}(p) = 0$. Then by (10) we have $V(c_{p\wedge q\to r}) = \{\langle x', y'\rangle \mid x' < 0 \text{ and } 0 < y'\}$. Consequently, we have $\mathcal{M}, \langle x, 0\rangle \models [\mathsf{A}][\overline{\mathsf{E}}]c_{p\wedge q\to r}$.
- **(Case 3.2):** $\mathsf{v}(p) = 1$. Then, by V (11) we have $V(c_{p\wedge q\to r}) = \{\langle x', y'\rangle \mid x' \leq 0 \text{ and } 0 < y'\} \supseteq \{\langle x', y'\rangle \mid x' < 0 \text{ and } 0 < y'\}$. It follows that $\mathcal{M}, \langle x, 0\rangle \models [\mathsf{A}][\overline{\mathsf{E}}]c_{p\wedge q\to r}$.

(Case 4): $\psi = [\mathsf{U}]([\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]c_{p\wedge q\to r} \to r)$ for $p, q, r \in \mathsf{PROP}(\varphi)$. Fix any $\langle x, y\rangle \in I(\mathbb{D})$ and assume that $\mathcal{M}, \langle x, y\rangle \models [\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]c_{p\wedge q\to r}$. We will show that $\mathcal{M}, \langle x, y\rangle \models r$.

$\mathcal{M}, \langle x, y\rangle \models [\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]c_{p\wedge q\to r}$, so $\{\langle x', y'\rangle \mid x' \leq 0 \text{ and } y < y'\} \subseteq V(c_{p\wedge q\to r})$. By the definition of V the above condition may be satisfied only by application of (11), that is when $V(c_{p\wedge q\to r}) = \{\langle x', y'\rangle \mid x' \leq 0 \text{ and } 0 < y'\}$. But in this case $[\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]c_{p\wedge q\to r}$ is satisfied exactly in intervals $\langle u, w\rangle$ such that $u \leq 0$ and $w = 0$. It follows that $x' \leq 0$ and $y' = 0$.

Since (11) was applied, we have $\mathsf{v}(p) = 1$ and $\mathsf{v}(q) = 1$. By $\mathsf{v}(\varphi) = 1$, $(p \wedge q \to r) \in \mathsf{clauses}(\varphi)$, $\mathsf{v}(p) = 1$, and $\mathsf{v}(q) = 1$, we obtain $\mathsf{v}(r) = 1$. Then, by (8) we have $V(r) = \{\langle u, 0\rangle \mid u \leq 0\}$. Hence, $\mathcal{M}, \langle x', 0\rangle \models r$, so $\mathcal{M}, \langle x, y\rangle \models \psi$.

(Case 5): $\psi = [\mathsf{U}](p \wedge q \to \bot)$ for some $p, q \in \mathsf{PROP}(\varphi)$. Fix $\langle x, y\rangle \in I(\mathbb{D})$ and suppose towards a contradiction that $\langle x, y\rangle \in V(p)$ and $\langle x, y\rangle \in V(q)$. It follows by (8) that $\mathsf{v}(p) = \mathsf{v}(q) = 1$. We have $\psi = [\mathsf{U}](p \wedge q \to \bot) \in \mathsf{clauses}(\tau(\varphi))$, so by (6) we obtain $(p \wedge q \to \bot) \in \mathsf{clauses}(\varphi)$, which raises a contradiction with $\mathsf{v}(p) = \mathsf{v}(q) = 1$. It follows that $\mathcal{M}, \langle x, y\rangle \models \psi$.

Then, $\mathcal{M}, \langle 0,0\rangle \models \tau(\varphi)$, so $\tau(\varphi)$ is HS-satisfiable over ($<$,Non-S,Den).

In the $(1 \Leftarrow 2)$ implication we claimed that $\mathsf{v}(\varphi) = 1$. The proof is as follows. Let us fix $\psi \in \mathsf{clauses}(\varphi)$, that is ψ is of the form p, $p \wedge q \to r$, or $p \wedge q \to \bot$ for $p, q, r \in \mathsf{PROP}$.

(Case 1): $\psi = p$ for some $p \in \mathsf{PROP}$. It follows by (2) that $\tau(p) = [\overline{\mathsf{E}}][\mathsf{E}]p$ is a clause of $\tau(\varphi)$. Since $\mathcal{M}, \langle x,y\rangle \models \tau(\varphi)$, we have $\mathcal{M}, \langle x,y\rangle \models [\overline{\mathsf{E}}][\mathsf{E}]p$, that is $\{\langle x',y\rangle \mid x' \leq y'\} \subseteq V(p)$. It follows that $\langle x,y\rangle \in V(p)$, so by the definition of v we obtain $\mathsf{v}(\psi) = 1$.

(Case 2): $\psi = p \wedge q \to r$ for some $p, q, r \in \mathsf{PROP}$. Assume that $\mathsf{v}(p) = \mathsf{v}(q) = 1$. We will show that $\mathsf{v}(r) = 1$. By the definition of v we have $\langle x,y\rangle \in V(p)$ and $\langle x,y\rangle \in V(q)$. Then, by $p \wedge q \to r \in \mathsf{clauses}(\varphi)$ and (2)–(7) we obtain $\tau(p \wedge q \to r) \in \mathsf{clauses}(\tau(\varphi))$. Since $\mathcal{M}, \langle x,y\rangle \models \tau(\varphi)$, we have $\mathcal{M}, \langle x,y\rangle \models \tau(p \wedge q \to r)$.

By (3) we have $\mathcal{M}, \langle x,y\rangle \models [\mathsf{U}](p \to [\mathsf{A}]c_{p\wedge q\to r})$. Since $\mathcal{M}, \langle x,y\rangle \models p$, we obtain $\mathcal{M}, \langle x,y\rangle \models [\mathsf{A}]c_{p\wedge q\to r}$, that is for all z such that $y < z$ it holds that $\mathcal{M}, \langle y,z\rangle \models \tau(c_{p\wedge q\to r})$.

On the other hand, by (4) we have $\mathcal{M}, \langle x,y\rangle \models [\mathsf{U}](q \to [\mathsf{A}][\overline{\mathsf{E}}]c_{p\wedge q\to r})$. Since $\mathcal{M}, \langle x,y\rangle \models q$, we obtain $\mathcal{M}, \langle x,y\rangle \models [\mathsf{A}][\overline{\mathsf{E}}]c_{p\wedge q\to r}$, that is for all $x' < y$ and $y < y'$ we have $\mathcal{M}, \langle x',y'\rangle \models c_{p\wedge q\to r}$.

Hence, $\mathcal{M}, \langle x',y'\rangle \models c_{p\wedge q\to r}$ for all $x' \leq y$ and $y < y'$, so $\mathcal{M}, \langle x,y\rangle \models [\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]\tau(c_{p\wedge q\to r})$. We have $\mathcal{M}, \langle x,y\rangle \models \tau(\varphi)$, so by (5) we obtain $\mathcal{M}, \langle x,y\rangle \models [\mathsf{U}]([\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]c_{p\wedge q\to r} \to r)$. Since $\mathcal{M}, \langle x,y\rangle \models [\overline{\mathsf{E}}][\mathsf{E}][\overline{\mathsf{B}}]\tau(c_{p\wedge q\to r})$, we have $\mathcal{M}, \langle x,y\rangle \models r$. Then, by the definition of v we get $\mathsf{v}(r) = 1$.

(Case 3): $\psi = p \wedge q \to \bot$ for some $p, q \in \mathsf{PROP}$. Towards a contradiction suppose that $\mathsf{v}(\psi) = 0$, that is $\mathsf{v}(p) = \mathsf{v}(q) = 1$. By the definition of v we have $\mathcal{M}, \langle x,y\rangle \models p \wedge q$.

On the other hand, $(p \wedge q \to \bot) \in \mathsf{clauses}(\varphi)$, therefore by (6) we have $[\mathsf{U}](p \wedge q \to \bot) \in \mathsf{clauses}(\tau(\varphi))$. We have $\mathcal{M}, \langle x,y\rangle \models [\mathsf{U}](p \wedge q \to \bot)$ which raises a contradiction with $\mathcal{M}, \langle x,y\rangle \models p \wedge q$. It follows, that $\mathsf{v}(\psi) = 1$.

Hence, $\mathsf{v}(\varphi) = 1$. ◀