

31st International Symposium on Temporal Representation and Reasoning


TIME 2024, October 28–30, 2024, Montpellier, France

Edited by

Pietro Sala
Michael Sioutis
Fusheng Wang



Editors

Pietro Sala 

University of Verona, Italy
pietro.sala@univr.it

Michael Sioutis 

LIRMM UMR 5506, Université de Montpellier & CNRS, France
michael.sioutis@lirmm.fr

Fusheng Wang 

Stony Brook University, NY, USA
fusheng.wang@stonybrook.edu

ACM Classification 2012

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

ISBN 978-3-95977-349-2

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-349-2>.

Publication date

October, 2024

Bibliographic information published by the Deutsche Nationalbibliothek

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

License

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): <https://creativecommons.org/licenses/by/4.0/legalcode>.



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

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

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.TIME.2024.0

ISBN 978-3-95977-349-2

ISSN 1868-8969

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

LIPICs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Luca Aceto (Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université Paris Cité, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University, Brno, CZ)
- Meena Mahajan (*Chair*, Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Pierre Senellart (ENS, Université PSL, Paris, FR)

ISSN 1868-8969

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

■ Contents

Preface	
<i>Pietro Sala, Michael Sioutis, and Fusheng Wang</i>	0:vii
TIME Steering Committee	
.....	0:ix
Program Committee Members	
.....	0:xi–0:xiii
Authors	
.....	0:xiii–0:xiv

Invited Talks

A General Logical Approach to Learning from Time Series	
<i>Guido Sciavicco</i>	1:1–1:2
Strategic Reasoning Under Imperfect Information with Synchronous Semantics	
<i>Sophie Pinchinat</i>	2:1–2:2
Rule-Based Temporal Reasoning: Exploring DatalogMTL	
<i>Przemysław Andrzej Wałęga</i>	3:1–3:3

Regular Papers

Agile Controllability of Simple Temporal Networks with Uncertainty and Oracles	
<i>Johann Eder, Roberto Posenato, Carlo Combi, Marco Franceschetti, and Franziska S. Hollauf</i>	4:1–4:16
Open the Chests: An Environment for Activity Recognition and Sequential Decision Problems Using Temporal Logic	
<i>Ivelina Stoyanova, Nicolas Museux, Sao Mai Nguyen, and David Filliat</i>	5:1–5:19
Extending the Range of Temporal Specifications of the Run-Time Event Calculus	
<i>Periklis Mantenoglou and Alexander Artikis</i>	6:1–6:14
Fitting’s Style Many-Valued Interval Temporal Logic Tableau System: Theory and Implementation	
<i>Guillermo Badia, Carles Noguera, Alberto Paparella, Guido Sciavicco, and Ionel Eduard Stan</i>	7:1–7:16
A More Efficient and Informed Algorithm to Check Weak Controllability of Simple Temporal Networks with Uncertainty	
<i>Ajdin Sumic and Thierry Vidal</i>	8:1–8:15
A Faster Algorithm for Finding Negative Cycles in Simple Temporal Networks with Uncertainty	
<i>Luke Hunsberger and Roberto Posenato</i>	9:1–9:15

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang



Leibniz International Proceedings in Informatics

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

What Killed the Cat? Towards a Logical Formalization of Curiosity (And Suspense, and Surprise) in Narratives <i>Florence Dupin de Saint-Cyr, Anne-Gwenn Bosser, Benjamin Callac, and Eric Maisel</i>	10:1–10:16
Faster Algorithm for Converting an STNU into Minimal Dispatchable Form <i>Luke Hunsberger and Roberto Posenato</i>	11:1–11:14
Robust Execution of Probabilistic STNs <i>Luke Hunsberger and Roberto Posenato</i>	12:1–12:19
Introducing Interdependent Simple Temporal Networks with Uncertainty for Multi-Agent Temporal Planning <i>Ajdin Sumic, Thierry Vidal, Andrea Micheli, and Alessandro Cimatti</i>	13:1–13:14
Learning Temporal Properties from Event Logs via Sequential Analysis <i>Francesco Chiariello</i>	14:1–14:14
A Framework for Assessing Inconsistency in Disjunctive Temporal Problems <i>Jean-François Condotta and Yakoub Salhi</i>	15:1–15:18
Real-Time Higher-Order Recursion Schemes <i>Eric Alsmann and Florian Bruse</i>	16:1–16:20
Time Series Anomaly Detection Leveraging MSE Feedback with AutoEncoder and RNN <i>Ibrahim Delibasoglu and Fredrik Heintz</i>	17:1–17:12
Full Characterisation of Extended CTL* <i>Massimo Benerecetti, Laura Bozzelli, Fabio Mogavero, and Adriano Peron</i>	18:1–18:18
Model Checking Linear Temporal Properties on Polyhedral Systems <i>Massimo Benerecetti, Marco Faella, and Fabio Mogavero</i>	19:1–19:23
FastMinTC+: A Fast and Effective Heuristic for Minimum Timeline Cover on Temporal Networks <i>Giorgio Lazzarinetti, Sara Manzoni, Italo Zoppis, and Riccardo Dondi</i>	20:1–20:18

■ Preface

It is our pleasure to welcome you to the 31st International Symposium on Temporal Representation and Reasoning (TIME 2024), held in Montpellier, France, from October 28 to 30, 2024. For over three decades, TIME has been the premier forum for researchers from various disciplines to share their latest findings and insights in the field of temporal representation and reasoning.

This year's symposium continues the tradition of bringing together researchers from diverse areas such as artificial intelligence, database management, logic, and verification. The interdisciplinary nature of TIME fosters rich discussions and collaborations, pushing the boundaries of our understanding of temporal aspects in computer science and related fields.

We received a total of 26 submissions, representing a wide range of research topics. After a rigorous review process, 17 papers were selected for presentation at the symposium. The accepted papers cover various aspects of temporal representation and reasoning, including but not limited to temporal logics, planning, temporal databases, and spatio-temporal reasoning. We are honored to have three distinguished keynote speakers this year:

Sophie Pinchinat (Inria-IRISA, Rennes, France)

Guido Sciavicco (University of Ferrara, Italy)

Przemysław Wałęga (Queen Mary University of London & University of Oxford, UK)

Their talks promise to provide valuable insights into the current state and future directions of our field. We would like to express our gratitude to the Program Committee members and additional reviewers for their thorough and timely reviews. Their expertise and dedication have been crucial in maintaining the high scientific standards of the symposium. We also thank the authors for their high-quality submissions and their contributions to the field.

Our sincere appreciation goes to the local organizing committee for their hard work in ensuring a smooth and enjoyable symposium experience. We are also grateful to our sponsors - Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), Pôle Mathématiques, Informatique, Physique, Systèmes (MIPS) of the Université de Montpellier, the Department of Computer Science of the University of Verona, and EurAI - for their generous support.

Lastly, we would like to thank LIPICs for publishing the proceedings and for their continued support of open-access dissemination of scientific research.

We hope that TIME 2024 will be a stimulating and enriching experience for all participants, fostering new ideas, collaborations, and advances in the field of temporal representation and reasoning.

Pietro Sala, University of Verona, Italy

Michael Sioutis, LIRMM UMR 5506, Université de Montpellier & CNRS, France

Fusheng Wang, Stony Brook University, NY, USA

TIME 2024 Program Committee Co-Chairs

September 7th, 2024

■ TIME Steering Committee

Alexander Artikis

University of Piraeus & NCSR Demokritos
Greece
a.artikis@unipi.gr

Patricia Bouyer

CNRS & ENS Paris-Saclay
France
bouyer@lsv.fr

Carlo Combi (Chair)

University of Verona
Italy
carlo.combi@univr.it

Johann Eder

University of Klagenfurt
Austria
johann.eder@aau.at

Thomas Guyet

IRISA
France
thomas.guyet@irisa.fr

Luke Hunsberger

Vassar College
United States
hunsberger@vassar.edu

Martin Lange (Chair)

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

Angelo Montanari

University of Udine
Italy
angelo.montanari@uniud.it

Shankara Narayanan Krishna (Krishna S.)

IIT Bombay
India
krishnas@cse.iitb.ac.in

Mark Reynolds

University of Western Australia
Australia
mark.reynolds@uwa.edu.au



■ Program Committee Members

Alia Abdelmoty

Cardiff University
Wales
AbdelmotyAI@cardiff.ac.uk

Alexandros Artikis

NCSR Demokritos & University of Piraeus
Greece
a.artikis@iit.demokritos.gr

Luyi Bai

Northeastern University & University of Leicester
China & UK
baily@neuq.edu.cn

Nassim Belmecheri

Simula Research Laboratory
Norway
nassim@simula.no

Florian Bruse

University of Kassel
Germany
florian.bruse@uni-kassel.de

Davide Catta

University of Naples Federico II
Italy
davidecatta2@gmail.com

María Laura Cobo

National University of the South
Argentina
mlc@cs.uns.edu.ar

Jean-François Condotta

Artois University
France
jfrancois.condotta@univ-artois.fr

Esra Erdem

Sabancı University
Turkey
esraerdem@sabanciuniv.edu

LuoYi Fu

Shanghai Jiao Tong University
China
yiluofu@sjtu.edu.cn

Nicola Gigante

Free University of Bozen-Bolzano
Italy
nicola.gigante@unibz.it

S Akshay

Indian Institute of Technology Bombay
India
akshayss@cse.iitb.ac.in

Beatrice Amico

University of Verona
Italy
beatrice.amico@univr.it

Philippe Balbiani

CNRS-IRIT
France
Philippe.Balbiani@irit.fr

Béatrice Bérard

Sorbonne University
France
Beatrice.Berard@lip6.fr

Jaewook Byun

Sejong University
South Korea
jwbyun@sejong.ac.kr

Cindy Chen

University of Massachusetts Lowell
US
cchen@cs.uml.edu

Carlo Combi

University of Verona
Italy
carlo.combi@univr.it

Johann Eder

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

Zoe Falomir

Umeå University
Sweden
zoe.falomir@umu.se

Silvia García Méndez

University of Vigo
Spain
sgarcia@gti.uvigo.es

John Grant

University of Maryland
US
grant@cs.umd.edu

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang



Leibniz International Proceedings in Informatics

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

Thomas Guyet

Inria-AIstroSight
France
thomas.guyet@inria.fr

Zina Ibrahim

King's College London
UK
zina.ibrahim@kcl.ac.uk

Peter Jonsson

Linköping University
Sweden
peter.jonsson@liu.se

Orna Kupferman

Hebrew University
Israel
orna@cs.huji.ac.il

Martin Lange

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

Ruizhe Ma

University of Massachusetts Lowell
US
ruizhe_ma@uml.edu

Maria Chiara Meo

"G. d'Annunzio" University of Chieti-Pescara
Italy
mariachiara.meo@unich.it

Munyque Mittelmann

University of Naples Federico II
Italy
munyque.mittelmann@unina.it

Anastasia Paparrizou

CNRS-LIRMM
France
Anastasia.Paparrizou@lirmm.fr

Roberto Posenato

University of Verona
Italy
roberto.posenato@univr.it

Yakoub Salhi

Artois University
France
salhi@cril.fr

Kathleen Stewart

University of Maryland
US
stewartk@umd.edu

Shufang Zhu

University of Oxford
UK
shufang.zhu@cs.ox.ac.uk

Christopher Hahn

Google LLC
US
chrishahn@google.com

Tomi Janhunén

Tampere University
Finland
tomijanhunen@tuni.fi

Manolis Koubarakis

National and Kapodistrian University of Athens
Greece
koubarak@di.uoa.gr

Victor Lagerkvist

Linköping University
Sweden
victor.lagerkvist@liu.se

Jianwen Li

East China Normal University
China
jwli@sei.ecnu.edu.cn

Federica Mandreoli

University of Modena and Reggio Emilia
Italy
federica.mandreoli@unimore.it

Christian Molinaro

University of Calabria
Italy
cmolinaro@dimes.unical.it

Emilio Muñoz-Velasco

University of Málaga
Spain
ejmunoz@uma.es

Francesco Parisi

University of Calabria
Italy
fparisi@dimes.unical.it

Farimah Poursafaei

Mila, Quebec AI Institute & McGill University
Canada
farimah.poursafaei@mila.quebec

Enrico Scala


University of Brescia
Italy
enrico.scala@unibs.it


Matteo Zavatteri


University of Padua
Italy
matteo.zavatteri@unipd.it

■ List of Authors


Eric Alsmann  (16)
University of Kassel, Germany

Alexander Artikis  (6)
University of Piraeus, Greece; NCSR
"Demokritos", Athens, Greece

Guillermo Badia  (7)
School of Historical and Philosophical Inquiry,
University of Queensland, Brisbane, Australia

Massimo Benerecetti  (18, 19)
Università di Napoli Federico II, Italy

Anne-Gwenn Bosser  (10)
Lab-STICC CNRS UMR 6285, ENIB, Brest,
France

Laura Bozzelli  (18)
Università di Napoli Federico II, Italy

Florian Bruse  (16)
University of Kassel, Germany


Benjamin Callac (10)
Lab-STICC CNRS UMR 6285, Université de
Bretagne Occidentale, Brest, France


Francesco Chiariello  (14)
IRIT, ANITI, University of Toulouse, France


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

Carlo Combi  (4)
University of Verona, Italy


Jean-François Condotta  (15)
CRIL UMR 8188, Université d'Artois & CNRS,
Lens, France

Ibrahim Delibasoglu  (17)
Department of Computer and Information
Science (IDA), Linköping University, Sweden;
Software Engineering, Sakarya University,
Turkey


Riccardo Dondi  (20)
Università degli Studi di Bergamo, Bergamo,
Italy


Florence Dupin de Saint-Cyr  (10)
Lab-STICC CNRS UMR 6285, ENIB, Brest,
France; IRIT, Université de Toulouse, France

Johann Eder  (4)
University of Klagenfurt, Austria

Marco Faella  (19)
Università di Napoli Federico II, Italy


David Filliat (5)
U2IS, ENSTA Paris, Institut Polytechnique de
Paris, Palaiseau, France

Marco Franceschetti  (4)
University of St. Gallen, Switzerland


Fredrik Heintz  (17)
Department of Computer and Information
Science (IDA), Linköping University, Sweden

Franziska S. Hollauf  (4)
University of Klagenfurt, Austria

Luke Hunsberger  (9, 11, 12)
Vassar College, Poughkeepsie, NY, USA

Giorgio Lazzarinetti  (20)
Università degli Studi Milano-Bicocca, Milano,
Italy

Eric Maisel (10)
Lab-STICC CNRS UMR 6285, ENIB, Brest,
France


Periklis Mantenoglou  (6)
National and Kapodistrian University of Athens,
Greece; NCSR "Demokritos", Athens, Greece


Sara Manzoni  (20)
Università degli Studi Milano-Bicocca, Milano,
Italy


Andrea Micheli  (13)
Fundazione Bruno Kessler, Trento, Italy


Fabio Mogavero  (18, 19)
Università di Napoli Federico II, Italy

Nicolas Museux  (5)
THALES, Palaiseau, France

Sao Mai Nguyen  (5)
U2IS, ENSTA Paris, Institut Polytechnique de
Paris, Palaiseau, France

Carles Noguera  (7)
Department of Information Engineering and
Mathematics, University of Siena, Italy









Alberto Paparella  (7)
Department of Mathematics and Computer
Science, University of Ferrara, Italy

Adriano Peron  (18)
Università di Trieste, Italy

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).
Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang



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

- Sophie Pinchinat  (2)
IRISA Laboratory/University of Rennes, France
- Roberto Posenato  (4, 9, 11, 12)
University of Verona, Italy
- Yakoub Salhi  (15)
CRIL UMR 8188, Université d'Artois & CNRS,
Lens, France
- Guido Sciavicco  (1, 7)
Department of Mathematics and Computer
Science, University of Ferrara, Italy
- Ionel Eduard Stan  (7)
Faculty of Engineering, Free University of
Bozen-Bolzano, Italy
- Ivelina Stoyanova (5)
U2IS, ENSTA Paris, Institut Polytechnique de
Paris, Palaiseau, France; THALES, Palaiseau,
France
- Ajdin Sumic (8, 13)
Technological University of Tarbes, France
- Thierry Vidal  (8, 13)
Technological University of Tarbes, France
- Przemysław Andrzej Wałęga  (3)
University of Oxford, UK; Queen Mary
University of London, UK
- Italo Zoppis  (20)
Università degli Studi Milano-Bicocca, Milano,
Italy

A General Logical Approach to Learning from Time Series

Guido Sciavicco  

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

Abstract

Machine learning from multivariate time series is a common task, and countless different approaches to typical learning problems have been proposed in recent years. In this talk, we review some basic ideas towards logic-based learning methods, and we sketch a general framework.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases Machine learning, temporal logic, general approach

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.1

Category Invited Talk

Funding We acknowledge the support of the FIRD project *Methodological Developments in Modal Symbolic Geometric Learning*, funded by the University of Ferrara, and the INDAM-GNCS project *Symbolic and Numerical Analysis of Cyberphysical Systems* (code CUP_E53C23001670001), funded by INDAM; Guido Sciavicco is a GNCS-INDAM member. Moreover, this research has also been funded by the Italian Ministry of University and Research through PNRR - M4C2 - Investimento 1.3 (Decreto Direttoriale MUR n. 341 del 15/03/2022), Partenariato Esteso PE00000013 - “FAIR - Future Artificial Intelligence Research” - Spoke 8 “Pervasive AI”, funded by the European Union under the “NextGeneration EU programme”.

1 Extended Abstract

Time series are temporally ordered observations. Time series can be *univariate* or *multivariate*, depending on whether there is a single one or multiple measurements, and each measurement, known as *temporal variable* can be either numerical or categorical.

Time series are ubiquitous in computer science. In some areas, data have naturally the form of multivariate time series; this is the case, for example, of *predictive maintenance* [5] in industry, usually obtained via the recording of sensors’ values (e.g., vibration sensors, gas exhaust sensors), of *(hospitalized) patients’ monitoring* [2], during which the value of vital signs (e.g., oximetry, blood pressure, temperature) is recorded in order to quickly identify variations or deterioration of the condition. In other areas, time series arise from data whose temporal nature is often overlooked; examples range from *acoustic data* [3] (e.g., voice, cough samples, breath samples), in which case the data becomes temporal when audio is separated into its frequency component (for example via a Fourier transform) whose power changes over time – in the scale of the milliseconds, to *brain waves recording data* [4] (e.g from an electroencephalogram), in which the electrical power at different frequencies (again, after a Fourier transform) changes over time and across different electrodes, up to *textual data*: tokens (e.g., words) follow a linear order, and their different characterizations (e.g., syntactic type, semantic value) can be seen as temporal variables, implying that even text, in a sense, can be interpreted as a multivariate time series [1].

A collection of data instances, or *dataset*, is associated to several classic problems. Given a single multivariate time series, seen as its own dataset, the most natural machine learning problem is that of *forecasting*, defined as the problem of establishing the next value of a



© Guido Sciavicco;

licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 1; pp. 1:1–1:2

Leibniz International Proceedings in Informatics



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

specific temporal variable given the past values of all variables. Otherwise, given a dataset of several multivariate time series, one can formulate a classification problem: if the dataset is labelled, then the problem is *supervised classification/regression*, and if it is not, it becomes *unsupervised classification*. The typical approaches to forecasting are based on examining the values of the time series in the last temporal points preceding the one for which the forecasting is required; by partitioning the time series into several chunks, each labeled with the value of the variable to be forecasted, then a single time series becomes a dataset itself, and forecasting can be reduced to regression). Moreover, both supervised classification/regression and unsupervised classification are pattern-searching problems; in the former case the search of patterns is guided by some measure of information on the class (e.g., entropy), and in the latter case by some measure of information on the pattern itself (e.g., frequency). In this sense, it can be said that with time series, at some abstract level there is only one machine learning paradigm of interest, that is, *classification*, or pattern extraction.

In this talk we focus on *logical* pattern extraction of datasets of multivariate time series. Patterns can be written in different logical languages, from propositional, to modal (temporal), to first-order, and beyond. One common idea to all symbolic methods and techniques is that a multivariate time series can be seen as a model of a logical framework, and pattern extraction is essentially a model-checking exercise.

Although the logical languages may vary, it is possible to give a general definition of propositional letter/atomic relation, that serves as a starting point. To this end, we consider a set of time points (e.g., from moment t_1 to moment t_2), and the value of all variables at those points (e.g., the level of vibration picked up by sensors A and B); then we apply a function to the set all values (e.g., the average). Finally, we compare the result to a constant, ending up with an atomic sentence (e.g., the average vibration between sensors A and B is below $100Hz$). By varying the parameters that govern such generic atomic statements, one obtains a wide range of basic alphabets, which can be combined with different languages.

As we shall see, logical pattern extraction from datasets of time series has been and can be approached in ways that range from very simple (i.e., using propositional logic), to relatively complex (i.e., using point-based or interval-based temporal logics), to very complex (i.e., using higher order logics). Standard symbolic machine learning methods (e.g., decision trees and lists learning algorithms) are designed for propositional logic, but they are progressively adapting to more complex languages, and in this talk we give an overview of this landscape.

References

- 1 D. Del Fante, F. Manzella, G. Sciavico, and I. E. Stan. A post-modern approach to automatic metaphor identification. In *Proc. of the 9th Italian Conference on Computational Linguistics (CLIC)*, volume 3596 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: <https://ceur-ws.org/Vol-3596/short9.pdf>.
- 2 R. Hegde, S. Ranjana, and C. D. Divya. Survey on development of smart healthcare monitoring system in iot environment. In *Proc. of the 5th International Conference on Computing Methodologies and Communication (ICCMC)*, pages 395–399, 2021.
- 3 F. Manzella, G. Pagliarini, G. Sciavico, and I. E. Stan. The voice of COVID-19: breath and cough recording classification with temporal decision trees and random forests. *Artif. Intell. Medicine*, 137:102486, 2023. doi:10.1016/J.ARTMED.2022.102486.
- 4 S. Mazzacane, M. Coccagna, F. Manzella, G. Pagliarini, V. A. Sironi, A. Gatti, E. Caselli, and G. Sciavico. Towards an objective theory of subjective liking: A first step in understanding the sense of beauty. *PLOS ONE*, 18(6):1–20, June 2023.
- 5 Y. Ran, X. Zhou, P. Lin, Y. Wen, and R. Deng. A survey of predictive maintenance: Systems, purposes and approaches. *ArXiv*, abs/1912.07383, 2019. arXiv:1912.07383.

Strategic Reasoning Under Imperfect Information with Synchronous Semantics

Sophie Pinchinat   

IRISA Laboratory/University of Rennes, France

Abstract

Dynamic Epistemic Logic is a modal logic dedicated to specifying epistemic property changes along the dynamic behavior of a multi-agent system. The models that underlie this logic are (epistemic) states together with transitions caused by events, the occurrence of which may modify the current state. We first develop a setting where the entire dynamics of the system starting from an initial state is captured by a single infinite tree, in a way similar to what has been considered for Epistemic Temporal Logic, and second go through the current state-of-the-art regarding strategic reasoning, with a focus on planning problems in this infinite structure.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Logic and verification; Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking

Keywords and phrases Strategic reasoning, Imperfect information, chain-MSO, Automatic structures

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.2

Category Invited Talk

1 Extended Abstract

Strategic reasoning is a field of formal methods that focuses on the quest for mathematical settings that allow for specifying and verifying properties in a multiplayer game-like framework where the focus is put on quantifying over strategies of individual players to achieve some goals. The game arena arises from a compositional operational semantics of some multi-agent system, that gives rise to infinite-horizon computations one wants to reason about. Not surprisingly, the kind of goals involved in strategic reasoning resembles the one used in formal verification, and is considered to be specified in temporal logic, the simplest ones being reachability properties. Additionally, and contrary to many approaches for system verification, the need for handling *imperfect information* is central: this is because, in multi-agent systems, the limited information available to each agent/player of the system prevents her from knowing the global state of the system. With this limited player ability of observing the system, strategic reasoning becomes hard to deal with. Indeed, while in a full information setting a strategy can be seen as a subtree of the full computation tree of the system's behavior, in a partial information setting, one has to deal with an extra property of these subtrees that guarantees the consistency of players' decision with their observation, known as *uniformity* [5]: in two different executions of the system that look the same to a player, the player's decision should be the same.

The uniformity property requires to reasoning about infinite trees that involve binary relations between nodes not considered in classical logic for trees [9] and that threatens the decidability of the resulting logics. Typically, extending monadic second-order logic (MSO) on trees with the extra binary "equal level" predicate makes it undecidable [10].

The purpose of this talk is to describe a setting where the uniformity property of players' decision can be handled. This setting is borrowed from the automated planning field where the Dynamic Epistemic Logic (DEL) [11] was introduced to provide one-player games with



© Sophie Pinchinat;

licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 2; pp. 2:1–2:2

Leibniz International Proceedings in Informatics



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

imperfect information and compute (uniform) strategies (there named “plans”) to achieve reachability epistemic goals. One major feature of the DEL setting is to deal with synchronous semantics of the players’ observation.

The talk consists in presenting a restricted version of the DEL setting where one-player uniform strategies for arbitrary omega-regular linear-time temporal goals can be represented by finite-state automata and makes strategic reasoning computable. We resort to fairly recent results from the authors and colleagues [2, 3, 8] that exhibit decidable strategic reasoning decision problems and strategy synthesis. A central tool is the class of *automatic structures* [4, 1], a class of possible infinite-state models with a decidable first-order theory, as well as the subclass of *regular automatic trees* [3] where chain-MSO becomes decidable, as opposed to the former.

Due to time limitation, it is unlikely that we discuss the multi-player extensions as done afterward in [6, 7].

References

- 1 Achim Blumensath and Erich Grädel. Automatic structures. In *Logic in Computer Science, 2000. Proceedings. 15th Annual IEEE Symposium on*, pages 51–62. IEEE, 2000. doi:10.1109/LICS.2000.855755.
- 2 Thomas Bolander, Tristan Charrier, Sophie Pinchinat, and François Schwarzenrüber. Del-based epistemic planning: Decidability and complexity. *Artif. Intell.*, 287:103304, 2020. doi:10.1016/j.artint.2020.103304.
- 3 Gaëtan Douéneau-Tabot, Sophie Pinchinat, and François Schwarzenrüber. Chain-monadic second order logic over regular automatic trees and epistemic planning synthesis. In Guram Bezhanishvili, Giovanna D’Agostino, George Metcalfe, and Thomas Studer, editors, *Advances in Modal Logic 12, proceedings of the 12th conference on "Advances in Modal Logic," held in Bern, Switzerland, August 27-31, 2018*, pages 237–256. College Publications, 2018. URL: <http://www.aiiml.net/volumes/volume12/DoueneauTabot-Pinchinat-Schwarzenrüber.pdf>.
- 4 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *Logic and computational complexity*, pages 367–392. Springer, 1995.
- 5 Bastien Maubert. *Logical foundations of games with imperfect information: uniform strategies*. PhD thesis, Université Rennes 1, 2014.
- 6 Bastien Maubert, Aniello Murano, Sophie Pinchinat, François Schwarzenrüber, and Silvia Stranieri. Dynamic epistemic logic games with epistemic temporal goals. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 155–162. IOS Press, 2020. doi:10.3233/FAIA200088.
- 7 Bastien Maubert, Sophie Pinchinat, François Schwarzenrüber, and Silvia Stranieri. Concurrent games in dynamic epistemic logic. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1877–1883. ijcai.org, 2020. doi:10.24963/ijcai.2020/260.
- 8 Côme Neyrand and Sophie Pinchinat. On the role of postconditions in dynamic first-order epistemic logic. *CoRR*, abs/2205.00876, 2022. doi:10.48550/arXiv.2205.00876.
- 9 Michael O Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.
- 10 Wolfgang Thomas. Infinite trees and automaton-definable relations over ω -words. *Theoretical Computer Science*, 103(1):143–159, 1992.
- 11 Hans Van Ditmarsch, Wiebe van Der Hoek, and Barteld Kooi. *Dynamic epistemic logic*, volume 337. Springer Science & Business Media, 2007.

Rule-Based Temporal Reasoning: Exploring DatalogMTL

Przemysław Andrzej Wałęga  

University of Oxford, UK

Queen Mary University of London, UK

Abstract

I will introduce DatalogMTL – an extension of Datalog, augmenting it with operators known from metric temporal logic (MTL). DatalogMTL is an expressive language which allows us for complex temporal reasoning over a dense timeline and, at the same time, remains decidable. I will provide an overview of research on DatalogMTL by discussing its computational complexity, syntactic and semantic modifications, practical reasoning approaches, applications, and future research directions.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning

Keywords and phrases Temporal Datalog, Temporal Logic Programming, Temporal Reasoning

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.3

Category Invited Talk

Funding This research was partially supported by the EPSRC projects OASIS (EP/S032347/1), ConCuR (EP/V050869/1) and UK FIRES (EP/S019111/1), as well as SIRIUS Centre for Scalable Data Access and Samsung Research UK.

Acknowledgements The research surveyed in this talk is a result of a joint work with Bernardo Cuenca Grau, Mark Kaminski, Egor V. Kostylev, Michał Zawidzki, Dingmin Wang, David Tena Cucala, Pan Hu, Matthias Lanzinger, Markus Nissl, and Emanuel Sallinger.

Extended Abstract

DatalogMTL [2] is a temporal extension of Datalog which allows one to perform complex reasoning over a dense timeline, using metric temporal logic (MTL) operators. For example it makes it possible to write temporal rules involving recursion over time, as in the program:

$$\begin{aligned} \text{TransactionChain}(x, y) &\leftarrow \text{Transaction}(x, y) \wedge \text{RedList}(x), \\ \text{TransactionChain}(x, z) &\leftarrow \diamond_{[0,24]} \text{TransactionChain}(x, y) \wedge \text{Transaction}(y, z), \\ \boxplus_{[0,\infty)} \text{Suspect}(y) &\leftarrow \text{TransactionChain}(x, y) \wedge \text{HighRisk}(y). \end{aligned}$$

The first two rules recursively define relation *TransactionChain*. The first rule initialises the definition by stating that *TransactionChain* holds between x and y at some time point, if at that time point a financial *Transaction* was recorded between x and y , and moreover, x was on a bank's *RedList*. The second rule, in turn, states that *TransactionChain* holds between x and z at a time point t , if *TransactionChain*(x, y) did hold sometime in the interval $[t - 24, t]$ (expressed with the MTL operator $\diamond_{[0,24]}$) and *Transaction*(y, z) holds at t . Hence, *TransactionChain*(x, y) means that there is a chain of transactions from x to y , such that the period between consecutive transactions in this chain is at most 24 hours and x was on *RedList* at the time of the first transaction. The third rule states that if y is in a *TransactionChain* (started by some x) and they are a *HighRisk* customer, then y will be treated as a *Suspect* in indefinite future (expressed with $\boxplus_{[0,\infty)}$).



© Przemysław Andrzej Wałęga;

licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 3; pp. 3:1–3:3

Leibniz International Proceedings in Informatics



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

The above program shows some non-trivial temporal reasoning aspects allowed by DatalogMTL, which involve recursion over time as well as reasoning about time intervals and time distances. Due to succinct representation and high modeling capabilities, DatalogMTL has been applied to a number of tasks, including temporal ontology-based query answering, stream reasoning, modelling smart contracts, verification of banking agreements, fact-checking economic claims, and even for describing dance movements. On the other hand, the high expressive power of DatalogMTL leads to challenging computational behaviour, namely consistency checking and fact entailment are EXPSPACE-complete in combined complexity [2] and PSPACE-complete in data complexity [4]. As a result, research on DatalogMTL has been concentrated on establishing low complexity fragments and developing practical reasoning algorithms.

The main approaches to obtaining low complexity fragments consist in: restricting the form of rules, for example by assuming their linearity or by limiting the form of allowed temporal operators, assuming a discrete time line, and considering an alternative event-based semantics. There are also several extensions of DatalogMTL, allowing for additional features such as temporal aggregation, existential rules, and (stratified and non-stratified) negation-as-failure under the stable model semantics [6].

Regarding the practical reasoning approaches, non-recursive programs can be rewritten into SQL, which was implemented within a temporal extension of the Ontop platform [3]. For DatalogMTL programs which are finitely materialisable [8] – that is the forward-chaining procedure for them takes a finite number of steps – reasoning can be performed using a standard chase, which was implemented within Vadalog system [1]. The first system allowing for reasoning in full DatalogMTL, called MeTeoR [5], combines materialisation-based and automata-based techniques, and has also been applied for solving stream reasoning tasks [7].

During the talk, I will describe both the theoretical results and practical reasoning approaches established for DatalogMTL. I will also present the ongoing and future research directions.

References

- 1 Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The Vadalog system: Datalog-based reasoning for knowledge graphs. *Proc. VLDB Endow.*, 11(9):975–987, 2018. doi:10.14778/3213880.3213888.
- 2 Sebastian Brandt, Elem Güzel Kalaycı, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyashev. Querying log data with metric temporal logic. *J. Artif. Intell. Res.*, 62:829–877, 2018. doi:10.1613/JAIR.1.11229.
- 3 Elem Güzel Kalaycı, Sebastian Brandt, Diego Calvanese, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyashev. Ontology-based access to temporal data with Ontop: A framework proposal. *Int. J. Appl. Math. Comput. Sci.*, 29(1):17–30, 2019. doi:10.2478/AMCS-2019-0002.
- 4 Przemysław Andrzej Wałęga, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. DatalogMTL: Computational complexity and expressive power. In Sarit Kraus, editor, *Proc. IJCAI*, pages 1886–1892, 2019.
- 5 Dingmin Wang, Pan Hu, Przemysław Andrzej Wałęga, and Bernardo Cuenca Grau. MeTeoR: Practical reasoning in datalog with metric temporal operators. In *Proc. AAAI*, pages 5906–5913, 2022.
- 6 Przemysław Andrzej Wałęga, David J. Tena Cucala, Bernardo Cuenca Grau, and Egor V. Kostylev. The stable model semantics of Datalog with metric temporal operators. *Theory Pract. Log. Program.*, 24(1):22–56, 2024. doi:10.1017/S1471068423000315.

- 7 Przemysław Andrzej Wałęga, Mark Kaminski, Dingmin Wang, and Bernardo Cuenca Grau. Stream reasoning with DatalogMTL. *J. Web Semant.*, 76:100776, 2023. doi:10.1016/J.WEBSEM.2023.100776.
- 8 Przemysław Andrzej Wałęga, Michał Zawidzki, and Bernardo Cuenca Grau. Finite materialisability of Datalog programs with metric temporal operators. *J. Artif. Intell. Res.*, 76:471–521, 2023. doi:10.1613/JAIR.1.14040.

Agile Controllability of Simple Temporal Networks with Uncertainty and Oracles

Johann Eder  

University of Klagenfurt, Austria

Roberto Posenato  

University of Verona, Italy

Carlo Combi  

University of Verona, Italy

Marco Franceschetti  

University of St. Gallen, Switzerland

Franziska S. Hollauf  

University of Klagenfurt, Austria

Abstract

Simple temporal networks with uncertainty (STNUs) have achieved wide attention and are the basis of many applications requiring the representation of temporal constraints and checking whether they are conflicting. Dynamic controllability is currently the most relaxed notion to check whether a system can be controlled without violating temporal constraints despite uncertainties. However, dynamic controllability assumes that the actual duration of a contingent activity is only known when the end event of this activity takes place. The recently introduced notion of agile controllability considers when this duration is known earlier, leading to a more relaxed notion of temporal feasibility. We extend the definition of STNUs to STNUOs (Simple Temporal Networks with Uncertainty and Oracles) to represent the point in time at which information about a contingent duration is available. We formally define agile controllability as a generalization of dynamic controllability considering the timepoints of information availability. We propose a set of constraint propagation rules for STNUOs leading to an algorithm for checking agile controllability.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning; Theory of computation → Dynamic graph algorithms

Keywords and phrases Temporal constraint networks, contingent durations, agile controllability

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.4

Supplementary Material *Software*: <https://git-isys.aau.at/ics/Papers/stnuo.git>

Dataset: <https://profs.scienze.univr.it/~posenato/software/benchmarks/OSTNUBenchmarks2024.tgz>

1 Introduction

Simple temporal networks with uncertainty (STNUs) [17, 4] have gained wide recognition for modeling temporal constraints. They extend Simple Temporal Networks (STNs) [5] by allowing one to represent uncertainties, i.e., they include so-called contingent activities, finishing at contingent timepoints and having a duration set by the environment of the considered system. System controllers only know the admissible range of such contingent durations, but they can only observe but do not decide on their actual values. STNUs are comparatively easy to use and easy to understand but expressive enough to represent temporal constraints for a large number of applications, for example, constraint-based planning [17], business processes [14], requirements engineering [7], or legal smart contracts [15], to name but a few.



© Johann Eder, Roberto Posenato, Carlo Combi, Marco Franceschetti, and Franziska S. Hollauf; licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 4; pp. 4:1–4:16

Leibniz International Proceedings in Informatics



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

An important question in all these applications is whether the constraints modeled in an STNU are temporally correct: can the controller derive a schedule for a process observing all constraints? Can the controller steer the execution of a process without violating any constraint? Are the elicited temporal requirements in conflict? Although the notion of satisfiability is sufficient for STNs, uncertainty in STNUs requires a more sophisticated notion of correctness. Dynamic controllability [17] is currently the most studied notion for the temporal correctness of STNUs. It requires a viable execution strategy (assignment of values to timepoints) that does not violate any constraint and where later timepoints may depend on earlier timepoints but not vice versa. For checking dynamic controllability, effective and efficient methods with polynomial complexity have been proposed [16, 4, 12].

However, dynamic controllability assumes that the duration of contingent activities, hence the values of contingent timepoints, are only known when they happen. This is adequate in many applications where the actual duration of some activity is only known when the end event of this activity is observed. An example of such an activity is a money transfer in the EU, where there is a legal requirement that the transfer does not last longer than four days. However, the actual duration is only known when the amount is credited to the receiver's account. In this case, the controller cannot schedule an event that has to occur exactly one day before the contingent timepoint (the receipt of the transferred amount) without (potential) violation of this temporal constraint.

However, in other applications, the duration of a contingent activity can be known earlier. For example, a delivery time between 4 and 6 weeks is guaranteed in order processing. However, the delivery date is communicated within a week after placing the order. In this case, it is perfectly feasible for the controller to schedule an event precisely two days before the contingent timepoint (day of the delivery). For such a scheduling decision, the controller recognizes that the duration of the contingent activity is known before the contingent timepoint takes place.

Now the question arises of how the notion of dynamic controllability can be generalized such that in a viable execution strategy, a timepoint may depend not only on timepoints which *are* earlier, but also which *are known* earlier. We call this novel notion of controllability *agile* because information about future timepoints may be used as soon as it is available.

This notion of *agile controllability* has first been introduced in [22, 21] together with an algorithm to check agile controllability based on the propagation rules in [16]. Here, we further formalize the extension of STNUs by introducing the notion of oracles, which represent the timepoints when the duration of a contingent activity is revealed. We formally define agile controllability by extending the notion of a viable execution strategy based on the available information of durations rather than only the occurrence of events. Furthermore, we develop an algorithm for checking agile controllability based on an extension of the constraint propagation rules presented in [4].

Therefore, the original contributions of this paper are:

1. The formal definition of STNUO (Simple Temporal Network with Uncertainty and Oracles).
2. A formal definition of Agile Controllability.
3. ORUL, a set of rules for propagating constraints in STNUOs.
4. An algorithm for checking Agile Controllability of STNUOs.
5. A proof-of-concept implementation of the checking algorithm.

The rest of the paper is organized as follows. In Section 2, we review related work and introduce the basic terms and definitions. In Section 3, we define STNUOs as an extension of STNUs with so-called oracles and define Agile Controllability as an extension of the notion of viable execution strategy. Section 4.1 discusses using oracles in execution strategies and

presents ORUL, a set of propagation rules to derive implicit constraints, and a backtracking algorithm to check agile controllability based on ORUL. Section 5 discusses the experimental evaluation of a proof-of-concept implementation of the checking algorithm, and finally, in Section 6, we draw some conclusions.

2 Background and Related work

2.1 Simple Temporal Networks with Uncertainty

The Simple Temporal Network with Uncertainty (STNU) is a data structure that models temporal problems in which the execution of some events cannot be controlled. The STNU comprises a set of timepoints and a set of temporal constraints. The timepoint set is partitioned into controllable (executable) timepoints and uncontrollable (contingent) ones; the constraint set is partitioned into regular and contingent ones.

The following is a formal definition of the STNU adapted from [11]:

- ▶ **Definition 1 (STNU).** *An STNU is a triple $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, where:*
 - \mathcal{T} is a finite, non-empty set of real-valued variables called timepoints. \mathcal{T} is partitioned into \mathcal{T}_X , the set of executable timepoints, and \mathcal{T}_C , the set of contingent timepoints.
 - \mathcal{C} is a set of binary (ordinary) constraints, each of the form $Y - X \leq \delta$ for some $X, Y \in \mathcal{T}$ and $\delta \in \mathbf{R}$.
 - \mathcal{L} is a set of contingent links, each of the form (A, x, y, C) , where $A \in \mathcal{T}_X, C \in \mathcal{T}_C$ and $0 < x < y < \infty$. A is called the activation timepoint; C contingent timepoint. If (A_1, x_1, y_1, C_1) and (A_2, x_2, y_2, C_2) are distinct contingent links, then $C_1 \neq C_2$.

The tuple $(\mathcal{T}, \mathcal{C})$ forms a Simple Temporal Network (STN), a data structure proposed by Dechter et al. in [5] to study the Simple Temporal Problem, that is, the satisfiability of a set of (controllable) temporal constraints. An STN is *satisfiable* if it is possible to determine an assignment (schedule) to timepoints such that all the constraints are satisfied. We say that a controller *executes an STN* when it schedules its timepoints.

The STNU model extends the STN one by adding contingent timepoints and links. The contingent link bounds cannot be modified, and the schedule of contingent timepoints is decided by *nature/environment*, who determines the *duration* of each contingent link. Therefore, given a contingent link (A, x, y, C) , once the controller executes the activation timepoint A , the environment decides the duration $d \in [x, y]$ and reveals it at time $A + d$, that is $C = A + d$.

An important property of the STNU is the *dynamic controllability*. To define it, we must formally introduce some concepts we recall from [11].

- ▶ **Definition 2 (Situation).** *If $(A_1, x_1, y_1, C_1), \dots, (A_K, x_K, y_K, C_K)$ are the K contingent links in an STNU $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, then the corresponding space of situations for \mathcal{N} is $\Omega = [x_1, y_1] \times \dots \times [x_K, y_K]$. Each situation $\omega = (\omega_1, \dots, \omega_K) \in \Omega$ represents one possible complete set of values for the duration of the contingent links of \mathcal{N} (chosen by nature).*

- ▶ **Definition 3 (Schedule).** *A schedule for an STNU $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is a mapping $\xi: \mathcal{T} \cup \{\perp\} \rightarrow \mathbf{R}$, where we assume that $\xi(\perp) = +\infty$. Ξ denotes the set of all schedules for an STNU. For historical reasons, we represent $\xi(X)$ as $[X]_\xi$.*

After having formally introduced the durations decided by *nature*, i.e., a *situation*, and the schedules of an STNU, i.e., the assignments of all timepoints to real values, we have to merge such aspects, to consider a strategy that, given a situation decided by nature, finds a suitable schedule.

► **Definition 4** (Execution Strategy). An execution strategy S for an STNU $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is a mapping $S : \Omega \rightarrow \Xi$.

► **Definition 5** (Viable Execution Strategy). An execution strategy S for an STNU $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is viable if for each situation $\omega \in \Omega$ the schedule $S(\omega)$ is a solution for \mathcal{N} , i.e., an assignment that satisfies all the constraints in the network.

► **Definition 6** (Dynamic Execution Strategy). An execution strategy for an STNU $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is dynamic if, for any two situations ω', ω'' and any executable timepoint $X \in \mathcal{T}_X$, it holds that:

$$\text{if } [X]_{S(\omega')} = k \text{ and } S(\omega')^{\leq k} = S(\omega'')^{\leq k}, \text{ then } [X]_{S(\omega'')} = k,$$

where $S(\omega')^{\leq k}$ is the set of contingent link durations observed up to and including time k , called *history*¹ until k . Since history also considers contingent durations observed at instant k , we say that the dynamic execution strategy implements the instantaneous reaction semantics.

An STNU is *dynamically controllable* if there exists a viable dynamic execution strategy for it, that is, an execution strategy that assigns the executable timepoints with the guarantee that all constraints will be satisfied, irrespectively of the duration values (within the specified bounds) the contingent links will be revealed to take [11].

2.2 Checking Dynamic Controllability

Each STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ has a corresponding graph $\mathcal{G} = (\mathcal{T}, \mathcal{E}_o \cup \mathcal{E}_{lc} \cup \mathcal{E}_{uc})$, also called *distance graph*, where the timepoints in \mathcal{T} serve as the graph's nodes and the constraints in \mathcal{C} and \mathcal{L} correspond to labeled, directed edges. In particular:

- $\mathcal{E}_o = \{X \xrightarrow{\delta} Y \mid (Y - X \leq \delta) \in \mathcal{C}\}$
- $\mathcal{E}_{lc} = \{A \xrightarrow{c;x} C \mid (A, x, y, C) \in \mathcal{L}\}$, and
- $\mathcal{E}_{uc} = \{C \xrightarrow{C;-y} A \mid (A, x, y, C) \in \mathcal{L}\}$.

The so-called *lower-case* (LC) edge $A \xrightarrow{c;x} C$ represents the uncontrollable possibility that the duration $(C - A)$ might take on its minimum value x , while the so-called *upper-case* (UC) edge $C \xrightarrow{C;-y} A$ represents the uncontrollable possibility that $(C - A)$ might take on its maximum value y . Such edges may be respectively notated as $(A, c;x, C)$ and $(C, C;-y, A)$, while constraints in \mathcal{C} and edges in \mathcal{E}_o may be called *ordinary* constraints and edges to distinguish them from the LC and UC edges.

Constraint propagation algorithms based on applying constraint propagation rules on the corresponding graph have been proposed to check whether an STNU is dynamically controllable [17, 16, 3, 12, 13]. A constraint propagation algorithm applies constraint propagation rules to derive implicit constraints from existing ones in the STNU. The algorithm terminates when either reaching network quiescence, i.e., no new constraints can be derived (the network is dynamically controllable), or a *negative cycle* is found (the network is not dynamically controllable). From now on, given a set of propagation rules R , we will call *closure* of a set \mathcal{N} of temporal constraints the set of constraints derived by applying the propagation rules in R .

Morris and Muscettola were the first to propose in [17] an algorithm based on constraint propagation that exhibits time complexity $O(n^5)$. In contrast, Cairo et al. proposed in [3] a new set of rules that improve the time complexity of the dynamic controllability (DC) check

¹ Also called *pre-history* in previous work [11, 2].

■ **Table 1** Edge-generation rules used by RUL [4, Fig. 2].

Rule	Pre-existing and generated edges	Applicability Conditions
(R)	$ \begin{array}{c} W \xrightarrow{v} Y \xrightarrow{u} X \\ \underbrace{\hspace{10em}}_{u+v} \end{array} $	none
(U)	$ \begin{array}{c} X \xrightarrow{v} C \xrightarrow{C:-y} A \\ \underbrace{\hspace{10em}}_{\max\{v-y, -x\}} \\ \xleftarrow{c:x} \end{array} $	$(A, x, y, C) \in \mathcal{L}$
(L)	$ \begin{array}{c} X \xleftarrow{v} C \xleftarrow{c:x} A \\ \underbrace{\hspace{10em}}_{x+v} \end{array} $	$v \leq 0$ or $X \in \mathcal{T}_C$, $X \neq C$, $\exists(B, s, t, X) \in \mathcal{L}$, $v \leq t$

algorithm to $O(mn + k^2n + kn \log n)$, where n is the number of timepoints, m is the number of constraints, and k is the number of contingent links. Table 1 shows the propagation rules proposed by Cairo et al., which we will use in the following.

2.3 Related Work: Making Contingent Links Flexible

The motivation to study the flexibility for contingent links comes from different application domains, such as business process modeling [23, 21, 20], robotics [24], and so on. Here, we briefly introduce different approaches to managing such flexibility.

The first, proposed in [23, 19], introduces the concept of guarded link: it is a contingent link, and thus its duration is not controlled by the system but has bounds that can be shrunk until some specific durations, named guards. In [23], the authors extend STNUs' propagation rules to deal with such guarded links. In contrast, in [19], the propagation rules for checking DC are also extended to deal with conditional execution paths, where, according to some conditions set during the network execution, only specific time points are executed.

In a second research line, in [1], the authors discuss some degrees of strong and dynamic controllability for STNUs, evaluating how far a network is from being controllable. Such metrics approximate the probability that an STNU can be dispatched offline (strong controllability) or online (dynamic controllability). Here, the focus is on uncontrollable networks. Such metrics are further generalized to Probabilistic Simple Temporal Networks (PSTNs). Taking into account even more recent research results, in [24], the authors discuss the robustness measure of PSTNs, that is, the probability of success in execution. They introduce and discuss degrees of weak/strong/dynamic controllability, robustness under a predefined dispatching protocol, and the PSTN expected execution utility.

There are several approaches that allow the representation of a timepoint when a contingent duration is revealed. The approach proposed in [25] proposes weak controllability, where all contingent durations are already known at the beginning of the process. Weak controllability can be seen as a special case of agile controllability proposed here.

The temporal variables considered in [6] are means of receiving temporal information from the process environment, for example, as output of process activities.

In [9], the authors introduce a further character of flexibility in the context of temporal business processes, making the durations of non-contingent activities known earlier. Indeed, they introduce and discuss the concept of semi-contingent task duration: it is a duration under the system's control until the task is initiated. Then, such duration becomes only observable but not under the system's control. Simple Temporal Networks with Semi-Contingency and Uncertainty (STNSUs) are then introduced, and dynamic controllability is studied for this new kind of temporal constraint network.

In [8, 10], the authors introduce a further flexibility aspect as they extend STNUs and Conditional Simple Temporal Networks with Uncertainty (CSTNUs) also to include a new kind of timepoints named *parameters*, whose occurrence must be fixed as soon as the network execution starts. A dynamic controllability check algorithm is proposed for this new kind of network.

3 Extending STNUs with Oracles

As the STNU does not allow decoupling the value of a contingent duration and the time of occurrence of the associated timepoint, we introduce a new kind of timepoint called *oracle*. An oracle O_C is a timepoint associated with a contingent link (A, C) . When O_C is executed, it reveals the associated contingent link duration. In other words, O_C can reveal the duration of the contingent link before the contingent timepoint C occurs. We extend the formal definition of an STNU in [11] with oracles as follows:

► **Definition 7** (STNU with Oracles). *An STNU with Oracles (STNUO) is a tuple $(\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$, where:*

- \mathcal{T} is a finite, non-empty set of real-valued variables called timepoints. \mathcal{T} is partitioned into \mathcal{T}_X , the set of executable timepoints and \mathcal{T}_C , the set of contingent timepoints. $\mathcal{T}_O \subseteq \mathcal{T}_X$, is the set of oracle timepoints.
- \mathcal{C} is a set of binary (ordinary) constraints, each of the form $Y - X \leq \delta$ for some $X, Y \in \mathcal{T}$ and $\delta \in \mathbf{R}$.
- \mathcal{L} is a set of contingent links, each of the form (A, x, y, C) , where $A \in \mathcal{T}_X, C \in \mathcal{T}_C$ and $0 < x < y < \infty$. A is called the activation timepoint; C contingent timepoint. If (A_1, x_1, y_1, C_1) and (A_2, x_2, y_2, C_2) are distinct contingent links, then $C_1 \neq C_2$.
- $\mathcal{O}: \mathcal{T}_C \rightarrow \mathcal{T}_O \cup \{\perp\}$ is a function that associates a contingent timepoint with its corresponding oracle, if any. For the sake of simplicity and without loss of generality, we assume that each oracle is associated with a single contingent timepoint.

The environment decides the duration d of a contingent link (A_i, x_i, y_i, C_i) , revealed at time $A_i + d$ or when the associated oracle O_i is executed. The requirement that only non-contingent nodes can be oracles does not reduce expressiveness, as oracles can be closely linked to contingent nodes. If $\mathcal{O}(C) = \perp$, the contingent node C does not have an oracle.

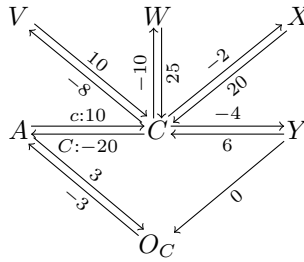
In the following, we extend the concept of dynamic execution strategy, replacing the concept of history with the concept of *Oracle-extended History* (OH) to consider also the presence of oracle timepoints. Therefore, we prefer to call this new dynamic execution strategy as *agile execution strategy*. Then, we introduce the concept of Agile Controllability.

The definitions of situation, schedule, execution strategy, and viable execution strategy are straightforward extensions of Definitions 2–5, respectively, to also include oracle time points.

In the definition of dynamic controllability, the *history until k* is the set of all contingent durations whose contingent timepoints occurred before or at time k . For STNUOs, the concept of history must also include all the durations revealed by oracles executed before or at k . Therefore, we call such history as *Oracle-extended History (OH) at time k* . Thus, OH contains information about the past and already-known information about the future.

► **Definition 8** (Oracle-extended History (OH)). *Given a schedule ξ for an STNUO $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$, and a time k , the Oracle-extended History (OH) until k is:*

$$\xi^{\leq k} = \{\omega_i \mid \omega_i = [C]_\xi - [A]_\xi \text{ and } \min\{[C]_\xi, [\mathcal{O}(C)]_\xi\} \leq k\}.$$



■ **Figure 1** An STNUO with contingent link $(A, 10, 20, C)$. Oracle O_C is associated to C .

► **Definition 9** (Agile Execution Strategy with Oracles). Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ be an STNUO. An execution strategy with oracles S_O for \mathcal{N} is agile if, for any two situations ω', ω'' and any executable timepoint $X \in \mathcal{T}$, it holds that:

$$\text{if } [X]_{S_O(\omega')} = k \text{ and } S_O(\omega')^{\leq k} = S_O(\omega'')^{\leq k}, \text{ then } [X]_{S_O(\omega'')} = k$$

where $S_O(\omega)$ is a schedule determined by the execution strategy with oracles S_O given the situation ω , and $S_O(\omega)^{\leq k}$ is OH until k . Since OH also considers contingent durations observed and revealed until time k , we say that the dynamic execution strategy implements the instantaneous reaction semantics.

► **Definition 10** (Agile Controllability (AC)). An STNUO $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ is agilely controllable if it admits a viable agile execution strategy with oracles. We refer to agile controllability (AC) as the property of being agilely controllable.

► **Example 11.** Let us consider an STNUO $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ as depicted in Figure 1, where O_C is the oracle for C that must be executed 3 time units after the activation timepoint A . Let $d \in [10, 20]$ be the duration revealed by the oracle O_C .

V can neither be scheduled with nor without an oracle because if the contingent link lasts 10, the oracle is executed too late to allow V to be executed, satisfying the constraint with C . W must be scheduled before the oracle to satisfy the constraint with C . Therefore, the oracle is not relevant for scheduling W . X can be scheduled without oracle (for example, $X = A + 2$) or with oracle (for example, $X = O_C - 3 + d - 4 = A + d - 4$, where 4 is one of the possible values to choose.) Y can be scheduled only with oracle: $Y = O_C - 3 + d - 5 = A + d - 5$. Thus, Y can be executed only after O_C .

The notion of agile controllability is strictly more general than the notion of dynamic controllability.

► **Lemma 12.** Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ be an STNUO. If the STNU $\mathcal{N}' = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is dynamically controllable, then \mathcal{N} is agilely controllable.

Proof. The lemma follows directly from the definition as if \mathcal{N} satisfies the requirements for dynamic controllability; it also satisfies those of agile controllability, as any viable dynamic execution strategy is also a viable agile execution strategy. ◀

4 Checking Agile Controllability of STNUO

This section presents a procedure for checking whether an STNUO is agilely controllable. The procedure is based on the rules introduced in Table 1. For the following considerations, let X be a non-contingent node, C a contingent node (with activation node A), and m/M the minimum/maximum duration of the considered contingent link (A, m, M, C) .

4.1 On The Usage of Oracles

The canonical problem we deal with here arises when a timepoint X depends on a future contingent timepoint C , i.e., there is a constraint $X \leq C + \delta$, where $\delta < 0$. The standard STNU semantics assumes that the value of a contingent timepoint, here C , is only revealed when C occurs. Therefore, the value for X must not violate the constraint for any possible C , i.e., $\exists X \forall C \mid X \leq C + \delta$. Such a set is only possible if $X \leq A + m + \delta$, which is the constraint derived by the L rule in Table 1, i.e., at the latest, X has to be executed at least δ time units before the earliest execution of C .

On the other hand, if there is a constraint $C \leq X + \delta'$, $0 \leq \delta'$, then STNU is DC only if $M - m \leq \delta + \delta'$, that is, if the difference between the smallest and the greatest distance between X and C is larger than the contingency of C (i.e., the difference between the maximum and minimum distances between A and C); otherwise, the constraints conflict.

► **Example 13.** The node X in Figure 1 can be scheduled since the contingency of C is 10 smaller than 18, the range of possible distances between C and X . For node Y , these values are 10 and 2. Applying the rules U and L to X, C , and A leads to a negative cycle.

However, if the duration of contingent activity is revealed earlier by an oracle at time point O_C , then it is sufficient that $\forall C \exists X \mid X \leq C + \delta$. However, the following constraint has to hold: $O_C \leq X$ and consequently $O_C \leq C + \delta$. So, the essence of using an oracle is that the sequence of quantifiers is changed from $\exists X \forall C$ to $\forall C \exists X$. However, the price is the introduction of an additional constraint, which could conflict with other constraints.

For the propagation of constraints, this has the following consequences:

- (1) If the oracle is not used, then the L and U rules must be applied.
- (2) If the oracle is used, then the L and U rules must not be applied on X and C , but the additional constraint $O_c \leq X$ must be added to the OSTNU.

Generally, the STNUs derived in (1) or (2) are not equivalent and admit different closures. Moreover, one closure could contain a negative cycle, and the other not.

► **Example 14.** Using the oracle avoids the negative cycle resulting from propagation in case (1), such as Y in Figure 1. On the other hand, using the oracle as in case (2) might lead to a conflict that was not there, such as W in Figure 1.

These possible configurations are all considered in the definition of a viable execution strategy: the value of X may be a function of the duration of a contingent activity d if C is in the history of X (i.e., is before X), or O_C is in the oracle-extended history of X (i.e., O_C is before X). The disjunction in the definition of oracle-extended history also leads to a choice in applying rules to propagate constraints.

► **Example 15.** In Figure 1, for timepoint X , there is the choice to either apply the rules L and U or consider the oracle O_C and introduce constraint $O_c \leq X$.

An interesting question is determining when an oracle is necessary to guarantee Agile Controllability. Basically, we can only use an oracle if there is one for a contingent node. Then, we only have to consider whether to use an oracle if there is a negative link from the considered contingent node to a non-contingent node. This link could be the result of constraint propagation.

We call $\mathcal{U} = \{(X, C) \mid X \in \mathcal{T}_X, C \in \mathcal{T}_C, \mathcal{O}(C) \neq \perp\}$ the set of all **potential oracle candidates**. If there is constraint $X \leq C + \delta$, or a constraint $C \leq X + \delta'$ with $C \in \mathcal{T}_C$, $\mathcal{O}(C) \neq \perp$, $X \in \mathcal{T}_X$, $\delta < 0$, and $0 \leq \delta'$, then we call (X, C) an **oracle candidate** since a viable execution strategy *could* require the usage of the oracle.

■ **Table 2** ORUL: propagation rules extending RUL ones for checking Agile Controllability of STNUO. \mathcal{U}^+ is the set of all oracle candidates for which the oracle should be used, and \mathcal{U}^- is the set of all oracle candidates for which the oracle should not be used.

Rule	Pre-existing and generated edges	Conditions
Relax (REL)	$ \begin{array}{c} W \xrightarrow{v} Y \xrightarrow{u} X \\ \underbrace{\hspace{10em}}_{u+v} \end{array} $	none
Upper (UPP)	$ \begin{array}{c} X \xrightarrow{v} C \xrightarrow{C:-y} A \\ \underbrace{\hspace{10em}}_{\max\{v-y, -x\}} \\ \xleftarrow{c:x} \end{array} $	$(X, C) \in \mathcal{U}^-$ or $\mathcal{O}(C) = \perp$ or $X \in \mathcal{T}_C$
Lower (LOW)	$ \begin{array}{c} X \xleftarrow{v} C \xrightarrow{c:x} A \\ \underbrace{\hspace{10em}}_{x+v} \end{array} $	$X \in \mathcal{T}_X$, $((X, C) \in \mathcal{U}^-$ or $\mathcal{O}(C) = \perp$), $v \leq 0$, or $X \in \mathcal{T}_C$, $X \neq C$, $\exists(B, w, y, X) \in \mathcal{L}$, $v \leq y$
Oracle (ORC)	$ \begin{array}{c} X \xleftarrow{v} C \xrightarrow{C:-y} A \\ \underbrace{\hspace{10em}}_{0} \\ \xrightarrow{\quad} O_C \end{array} $	$X \in \mathcal{T}_X$, $(X, C) \in \mathcal{U}^+$

► **Example 16.** In Figure 1, all pairs (V, C) , (W, C) , (X, C) , (Y, C) are oracle candidates.

Now consider the case where there are a pair of constraints, $X \leq C + \delta$ and $C \leq X + \delta'$, with $\delta < 0$, and $\delta + \delta' \leq M - m$, the contingency of C . In such a configuration, there is no viable execution strategy without using the oracle O_C . Therefore, we call (X, C) *oracle dependent*.

► **Example 17.** In Figure 1, (Y, C) is oracle dependent.

Constraints can only become stricter (and never removed) during constraint propagation. Therefore, if an oracle candidate (X, C) becomes oracle dependent due to the propagation of constraints, it will remain oracle dependent.

While for oracle-dependent pairs, any viable solution must use the oracle, constraint propagation could, but not necessarily, make oracle dependent some oracle candidates. Whether a pair becomes an oracle candidate or oracle dependent might also depend on which oracles are used for constraint propagation and which are not.

As there is no way of deciding upfront whether a (X, C) pair will become oracle dependent, it will be necessary for oracle candidates, which are not oracle dependent, to explore both options: to use oracle and not to use the oracle. For a particular constraint propagation, it is necessary to decide for which (X, C) pairs to use the oracle.

4.2 Propagation rules for STNUOs

In the previous section, we argued that exploring whether an oracle has to be used for an oracle candidate might be necessary. For guiding the propagation of constraints, we maintain two sets of oracle candidates:

- \mathcal{U}^+ is the set of all oracle candidates for which the oracle has to be used, and
- \mathcal{U}^- is the set of all oracle candidates for which the oracle has not to be used.

In Table 2, we propose ORUL, a set of constraint-propagation rules based on RUL set [4], modified for checking agile controllability. ORUL uses sets \mathcal{U}^+ and \mathcal{U}^- to guide the rules' application. Briefly, the REL rule is the same as the R rule in Table 1. The conditions for the UPP and LOW rules are extended with additional restriction $(X, C) \in \mathcal{U}^-$ such that the rules are only applied if oracles should *not* be used. The new ORC rule inserts the necessary constraints for using an oracle when $(X, C) \in \mathcal{U}^+$.

The following theorem states conditions on \mathcal{U}^+ and \mathcal{U}^- to guarantee AC.

► **Theorem 18.** *Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ be an STNUO and let $\mathcal{U} = \{(X, C) \mid X \in \mathcal{T}_X, C \in \mathcal{T}_C, \mathcal{O}(C) \neq \perp\}$ be the set of all possible pairs (ordinary node, contingent node) where the contingent node has its corresponding oracle.*

\mathcal{N} is Agilely Controllable if $\exists \mathcal{U}^+, \mathcal{U}^-$ such that $\mathcal{U} = \mathcal{U}^+ \cup \mathcal{U}^-$, $\mathcal{U}^+ \cap \mathcal{U}^- = \emptyset$, and the closure of \mathcal{N} considering $\mathcal{U}^+, \mathcal{U}^-$ for the propagation rules in Table 2 does not include a negative cycle.

Proof Sketch. We show that ORUL is sound, i.e., if ORUL does not lead to a negative cycle, then the STNUO is AC. It is straightforward to show that the propagated constraints are finite if no negative cycle is derived. Hence, let $\mathcal{N}_{\mathcal{O}}^*$ be the closure of \mathcal{N} with respect to ORUL, and $\mathcal{N}_{\mathcal{R}}^*$ be the set of all constraints derived by RUL. Cairo and Rizzi showed that the constraints of $\mathcal{N}_{\mathcal{R}}^*$ are necessary and sufficient to decide whether \mathcal{N} is DC [4]. We show that $\mathcal{N}_{\mathcal{O}}^*$ contains all the constraints of $\mathcal{N}_{\mathcal{R}}^*$ except those that are not necessary if oracles are used, while it includes all the constraints needed for the oracle candidates in \mathcal{U}^+ . It also excludes all constraints that are derived from unnecessary ones. Hence, $\mathcal{N}_{\mathcal{O}}^*$ is sufficient to decide whether \mathcal{N} is AC.

If $\mathcal{U}^+ = \emptyset$, and thus $\mathcal{U}^- = \mathcal{U}$, then the rules in Table 2 are the same as in Table 1, and $\mathcal{N}_{\mathcal{R}}^* = \mathcal{N}_{\mathcal{O}}^*$. As in [4], we can conclude that the STNUO is DC and hence AC if no negative cycle is derived.

If $\mathcal{U}^+ \neq \emptyset$, let $(X, C) \in \mathcal{U}^+$, $(A, m, M, C) \in \mathcal{L}$, $(X - C \leq \delta) \in \mathcal{C}$, $(C - X \leq \delta') \in \mathcal{C}$, $\delta < 0 < \delta'$, $0 \leq \delta' + \delta < M - m$, and $\mathcal{O}(C) = O$. Since oracle O is used for X , $(O - X \leq 0) \in \mathcal{N}_{\mathcal{O}}^*$ and the LOW and UPP rules are not applied for the triple $\langle X, A, C \rangle$. Hence, the LOW- and UPP-derived constraints between A and X are not in $\mathcal{N}_{\mathcal{O}}^*$. For a viable execution strategy, these constraints are not necessary, as for any duration d of the contingent activity (A, m, M, C) , $C = A + d$. There exists a value for X that satisfies constraints $(X - C \leq \delta)$ and $(C - X \leq \delta')$. In fact, both X and A are executable timepoints, and d is available before X , therefore $X = A + d + \delta$ is admissible and allows for the satisfaction of constraints $(X - C \leq \delta)$ and $(C - X \leq \delta')$. Therefore, if the propagation of such constraints does not determine negative cycles, then the network is AC.

We may also observe that any constraint that can be derived from \mathcal{N} by rules in Table 1 which is not in $\mathcal{N}_{\mathcal{O}}^*$, would be the result of a sequence of rule applications, starting by applying the UPP or LOW rules to some $\langle X, A, C \rangle$, where $(X, C) \in \mathcal{U}^+$, and hence is not necessary. ◀

4.3 A Checking Algorithm

We propose the backtracking Algorithm 1 to check whether an STNUO is agilely controllable.

The algorithm aims to check whether there exist sets \mathcal{U}^+ and its complementary \mathcal{U}^- such that the propagation with these sets does not lead to a negative cycle. As outlined at the end of Section 4.1, there is no way to know in advance sets \mathcal{U}^+ and \mathcal{U}^- . Therefore, the algorithm computes these sets incrementally and with backtracking. To reduce the effort of backtracking, the general strategy is to delay decisions about *potential* membership of a pair (X, C) in these sets as late as possible, but for those having an explicit condition for membership to \mathcal{U}^+ or \mathcal{U}^- . For this reason, \mathcal{U}^+ and \mathcal{U}^- are both empty when the algorithm starts, and only at the end, if the network is agilely controllable, they satisfy the conditions of Theorem 18².

² More precisely, $\mathcal{U}^+ \cap \mathcal{U}^- = \emptyset$ and $\mathcal{U}^+ \cup \mathcal{U}^- \subseteq \mathcal{U}$. Indeed, the algorithm does not consider pairs (X, C) for which there are no explicit constraints. Such possible pairs are not effective with respect to constraint propagations.

Algorithm 1 CheckAC.

```

Input:  $\mathcal{N}$  an STNUO,  $\mathcal{U}^-$ ,  $\mathcal{U}^+$ 
Output: Agile controllability status
1  $ok \leftarrow \text{applyRules}(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$ ;
2 if  $ok$  then
3    $\text{save}(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$ ;
4    $\mathcal{U}^0 \leftarrow \text{getOpenOracles}(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$ ;
5   if  $\mathcal{U}^0 \neq \emptyset$  then
6      $(X, C) \leftarrow \text{select}(\mathcal{U}^0)$ ;
7     if  $\text{interval}(X, C) > \text{contingencyInterval}(C)$  then
8        $\mathcal{U}^- \leftarrow \mathcal{U}^- \cup \{(X, C)\}$ ;
9        $ok \leftarrow \text{checkAC}(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$ ;
10      if not  $ok$  then
11         $\text{restore}(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$ ;
12      if  $(X, C) \notin \mathcal{U}^-$  then
13         $\mathcal{U}^+ \leftarrow \mathcal{U}^+ \cup \{(X, C)\}$ ;
14         $ok \leftarrow \text{checkAC}(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$ ;
15        if not  $ok$  then
16           $\text{restore}(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$ ;
17 return  $ok$ 

```

The algorithm is recursive. It starts with applying all the rules in Table 2. If the check is positive (no negative cycle was discovered), it determines which oracle-dependent pairs have not yet been considered and tries to assign each to \mathcal{U}^- or \mathcal{U}^+ recursively. When a negative cycle is discovered, the STNUO with the current sets \mathcal{U}^+ and \mathcal{U}^- is not agilely controllable and, therefore, backtracking is required (procedure `restore`). If there is no negative cycle, the algorithm checks whether there are still undecided oracle candidates. Heuristically, one undecided candidate pair (X, C) is chosen. Suppose a solution without applying the oracle for this pair is still possible (no oracle dependency). In that case, we decide not to use the oracle (i.e., inserting (X, C) in \mathcal{U}^-) and proceed recursively, invoking the checking procedure. If it fails, we restore the STNUO and continue with the decision to use the oracle (i.e., inserting (X, C) in \mathcal{U}^+) and recursively invoke the checking procedure.

When constraints lead to an oracle-dependent pair (X, C) (line 12 of Algorithm 1), such a pair is inserted into \mathcal{U}^+ . When a constraint $C \xrightarrow{v} X$ with a non-positive v is derived, (X, C) is inserted into \mathcal{U}^- (line 8 of Algorithm 1). The procedure terminates when either no additional constraints can be derived, and the procedure returns true, or a negative cycle is detected, and the procedure returns false.

The algorithm uses the following auxiliary procedures.

getOpenOracles($\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+$)

It returns the set of oracle candidates (X, C) , i.e., the set of all pairs (X, C) for which either a constraint $C \xrightarrow{v} X$ with a non-positive v or a constraint $X \xrightarrow{w} C$ with a non-negative w exist and (X, C) is neither in \mathcal{U}^- nor in \mathcal{U}^+ .

select(\mathcal{U}^0)

It returns one of the oracle candidate pairs (X, C) , not yet in \mathcal{U}^- neither in \mathcal{U}^+ . Heuristically, it returns the pair with the smallest difference between its interval and the contingency interval of C .

applyRules($\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+$)

It iteratively applies rules of Table 2 to generate additional constraints. It returns false if a negative cycle is discovered; true, otherwise.

save($\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+$)

It saves the current set of constraints such that **restore**($\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+$) can reconstruct the set of constraints as they were when the corresponding **save**($\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+$) was executed. These procedures support backtracking if a negative cycle is found based on a decision on the inclusion of pairs in \mathcal{U}^- resp. \mathcal{U}^+ .

interval(X, C)

It returns the value $\delta + \delta'$ derived by the constraints $(X - C \leq \delta)$ and $(C - X \leq \delta')$.

contingencyInterval(C)

It returns the value $u - l$ relative to the contingent link (A, l, u, C) associated with the input contingent timepoint C .

► **Example 19.** Figure 2 presents the process of applying the algorithm on an example STNUO where $(A, 20, 30, C)$ is a contingent link, O_C is its oracle, and X and Y are non-contingent nodes. Propagated self-loops (e.g., from X to X) are excluded from the figure for readability reasons except for negative cycles. In one step, newly propagated constraints by the rules in Table 2 are depicted as follows: rules propagated by REL are colored gray, by UPP orange, by LOW olive, and by ORC blue. Negative cycles are marked in red. Pre-existing or pre-propagated constraints are black.

The algorithm starts with empty \mathcal{U}^+ and \mathcal{U}^- . In the first step (line 1 of Algorithm 1), the rules are applied by propagating three new constraints via REL as presented in Figure 2 (step I). This intermediate STNUO is checked for a negative cycle. Since it does not have one (is *ok*), the algorithm proceeds with saving this state and getting the open oracles (lines 13 and 4 of Algorithm 1). \mathcal{U}^0 includes then the following pairs: (X, C) and (Y, C) .

Next, (X, C) is selected to check whether to use the oracle for it (line 6 of Algorithm 1). In this case, **interval**(X, C) = $15 - (-4) = 19$ and **contingencyInterval**(X, C) = $30 - 20 = 10$. Since the contingency interval is smaller, (X, C) is put in \mathcal{U}^- (lines 7-9 of Algorithm 1).

CheckAC calls itself. The rules are applied, and new constraints are propagated via REL as presented in Figure 2 (Step II). The STNUO now has a negative cycle. Indeed, the self-cycle in A equal to -1 is derived. More precisely, applying rule REL on timepoints A , X , and C , we obtain $(A, C) = 16 + 13 = 29$; then, applying rule UPP on A , C , A , we obtain the negative self-cycle $29 - 30 = -1$ in A .

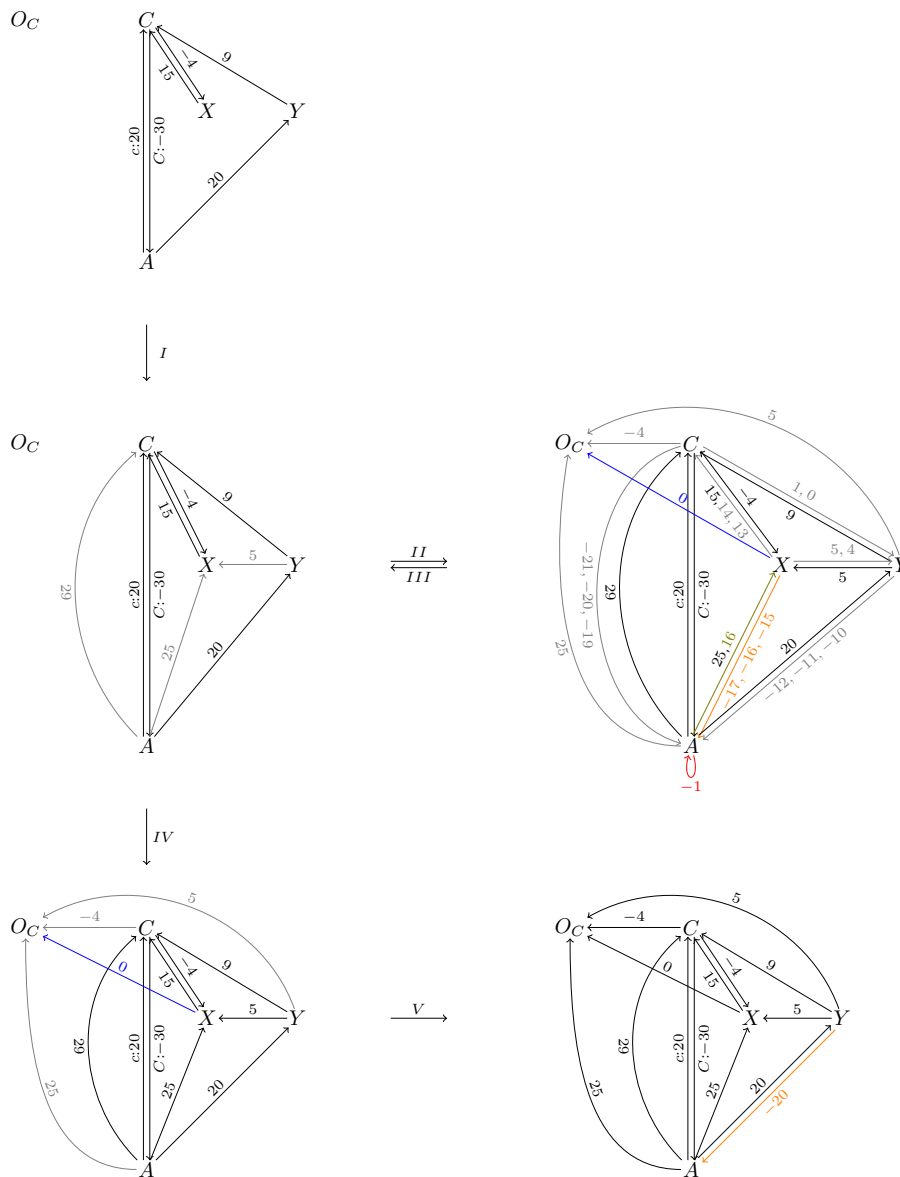
Thus, the status is not *ok*, and the algorithm backtracks (lines 10 and 11 of Algorithm 1): The STNUO from step (I) is restored (see step III in Figure 2) and (X, C) is removed from \mathcal{U}^- . Thus, the algorithm checks what happens if the oracle is used for (X, C) (lines 12-16 of Algorithm 1). (X, C) is put in \mathcal{U}^+ .

CheckAC calls itself. The rules are applied again, and new constraints are propagated (see step IV in Figure 2). This STNUO is without a negative cycle (status is *ok*). The algorithm stores this state and gets again the open oracles. \mathcal{U}^0 now includes only (Y, C) . For this pair, **interval**(Y, C) = ∞ (since there is only the constraint $C - Y \leq 9$) and **contingencyInterval**(Y, C) = $30 - 20 = 10$. Since the contingency interval is smaller, (Y, C) is put in \mathcal{U}^- .

CheckAC calls itself again, and the rules are applied (see Step V in Figure 2). No constraint leading to a negative cycle was propagated (status is *ok*). This state is saved, and the algorithm checks for open oracles. None are found; hence, \mathcal{U}^0 is empty. The algorithm ends with the result that the STNUO is *agilely controllable*.

4.3.1 Computational Complexity

Since CheckAC is a recursive backtracking algorithm, it is possible to give an upper bound to its space complexity assuming the worst case that each oracle must be paired with each other timepoint. In such a case, the depth of the recursion is $O(kn)$ (assuming that each of k contingent timepoints has an oracle). For each level of recursion, it is necessary to store



■ **Figure 2** Applying CheckAC on an STNUO example.

(save procedure) the configuration of the network and the two sets \mathcal{U}^+ and \mathcal{U}^- . Such an operation requires space $O(n^2)$, where n is the number of all timepoints. Therefore, the computational space required by the algorithm is $O(kn^3)$.

5 Experimental Evaluation

The algorithm for checking the agile controllability of an STNUO presented in Section 4.3 has been implemented as a proof-of-concept prototype to test the algorithm’s correctness and analyze its feasibility.

The algorithm has been implemented in Java. Experiments with this implementation were executed on an Ubuntu 22 machine having 16GB of RAM and an AMD EPYC-Rome (8) @ 2.6GHz CPU. The experiments use as input data from the OSTNU benchmark, which is available online³.

The benchmark includes 30 random STNUO instances with 30 nodes (5 contingent and 2 oracles). In all cases, the implementation produced the correct result. We ran the checking algorithm 100 times on each example and determined the average AC checking execution time. All execution average times are below 3 s. Therefore, our approach to determining the AC property is comparable to that presented by Posenato *et al.* [22]. This allows us to conclude that the algorithm is feasible for realistically sized STNUOs. Nevertheless, we will continue optimizing the algorithm and its implementation. For example, we will want to consider the DC checking algorithm in [13], which implements a DC checker based on the rules in Table 1 in a more efficient way, for the `applyRules($\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+$)` procedure, and to improve the heuristics for the `select(\mathcal{U}^0)` procedure.

The source code of the prototype implementation, the parser of the data sets used for the experiments, and the complete results are publicly available in an online repository⁴.

6 Discussion and Conclusions

We proposed agile controllability (AC) as a proper generalization of the well-established notion of dynamic controllability for STNUs, leading to a more relaxed notion of temporal correctness, which is still strong enough to guarantee that a controller can steer the execution of a process in a way that no temporal constraint is violated despite uncertainties. The main distinction to dynamic controllability is that available information about *future* durations of contingent links can be utilized for scheduling and dispatching timepoints. STNUOs, Simple Temporal Networks with Uncertainty and Oracles, can express *when* information about the timepoint of future contingent links is available. We presented a formal definition of viable execution strategies that utilize such advanced information. We also presented a set of rules for propagating constraints, leading to an algorithm that effectively checks whether an STNUO is agilely controllable.

AC is expected to support a wide range of applications as it provides a less restrictive notion of temporal correctness of plans, processes, requirements, contracts, etc. The presented algorithm seems feasible for typical problem sizes in many of these application areas. Nevertheless, improving implementations of this algorithm will further extend the approach’s applicability. Indeed, the algorithm `CheckAC` and its related proof-of-concept implementation,

³ <https://profs.scienze.univr.it/~posenato/software/benchmarks/OSTNUBenchmarks2024.tgz> [18]

⁴ <https://git-isys.aau.at/ics/Papers/stnuo.git>

as it is presented here, is intended to demonstrate the existence of an effective backtracking algorithm to check the agile controllability of an STNUO. It is not optimized, and exploring numerous possibilities to develop a significantly more efficient implementation of this basic algorithm is the subject of ongoing research.

References

- 1 Shyan Akmal, Savana Ammons, Hemeng Li, Michael Gao, Lindsay Popowski, and James C. Boerkoel Jr. Quantifying controllability in temporal networks with uncertainty. *Artif. Intell.*, 289:103384, 2020. doi:10.1016/J.ARTINT.2020.103384.
- 2 Arthur Bit-Monnot and Paul Morris. Dynamic Controllability of Temporal Plans in Uncertain and Partially Observable Environments. *Journal of Artificial Intelligence Research*, 77:1311–1369, August 2023. doi:10.1613/jair.1.13065.
- 3 Massimo Cairo, Luke Hunsberger, and Romeo Rizzi. Faster dynamic controllability checking for simple temporal networks with uncertainty. In *25th International Symposium on Temporal Representation and Reasoning (TIME 2018)*, volume 120 of *LIPICs*, pages 8:1–8:16, 2018. ISBN: 9783959770897. doi:10.4230/LIPICs.TIME.2018.8.
- 4 Massimo Cairo and Romeo Rizzi. Dynamic controllability of simple temporal networks with uncertainty: Simple rules and fast real-time execution. *Theoretical Computer Science*, 797:2–16, 2019. doi:10.1016/J.TCS.2018.11.005.
- 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 Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of business processes with temporal variables. In *ACM SAC 2019*, pages 40–47, 2019. doi:10.1145/3297280.3297286.
- 7 Johann Eder, Marco Franceschetti, and Josef Lubas. Time and processes: Towards engineering temporal requirements. In *Proceedings of the 16th International Conference on Software Technologies, ICSoft 2021*, pages 9–16. SCITEPRESS, 2021. doi:10.5220/0010625400090016.
- 8 Marco Franceschetti and Johann Eder. Checking temporal service level agreements for web service compositions with temporal parameters. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 443–445. IEEE, 2019. doi:10.1109/ICWS.2019.00080.
- 9 Marco Franceschetti and Johann Eder. Semi-contingent task durations: Characterization and controllability. In Marcello La Rosa, Shazia W. Sadiq, and Ernest Teniente, editors, *Advanced Information Systems Engineering - 33rd International Conference, CAiSE 2021, Melbourne, VIC, Australia, June 28 - July 2, 2021, Proceedings*, volume 12751 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2021. doi:10.1007/978-3-030-79382-1_15.
- 10 Marco Franceschetti, Roberto Posenato, Carlo Combi, and Johann Eder. Dynamic Controllability of Parameterized CSTNUs. In *ACM SAC 2023*, pages 965–973, 2023. doi:10.1145/3555776.3577618.
- 11 Luke Hunsberger. Efficient execution of dynamically controllable simple temporal networks with uncertainty. *Acta Informatica*, 53:89–147, 2016. doi:10.1007/S00236-015-0227-0.
- 12 Luke Hunsberger and Roberto Posenato. Speeding up the RUL⁻ Dynamic-Controllability-Checking Algorithm for Simple Temporal Networks with Uncertainty. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, volume 36, pages 9776–9785. AAAI Press, 2022. doi:10.1609/aaai.v36i9.21213.
- 13 Luke Hunsberger and Roberto Posenato. A Faster Algorithm for Converting Simple Temporal Networks with Uncertainty into Dispatchable Form. *Information and Computation*, 293, June 2023. doi:10.1016/j.ic.2023.105063.
- 14 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, 2015. doi:10.5220/0005200903700381.

- 15 Josef Lubas and Johann Eder. A time-aware model for legal smart contracts. In Han van der Aa, Dominik Bork, Henderik A. Proper, and Rainer Schmidt, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 121–135, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-34241-7_9.
- 16 Paul Morris. Dynamic controllability and dispatchability relationships. In *CPAIOR 2014*, volume 8451 of *LNCS*, 2014. doi:10.1007/978-3-319-07046-9_33.
- 17 Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *20th National Conf. on Artificial Intelligence (AAAI-2005)*, 2005. URL: <https://cdn.aaai.org/AAAI/2005/AAAI05-189.pdf>.
- 18 Roberto Posenato. CSTNU Tool: A Java library for checking temporal networks. *SoftwareX*, 17:100905, 2022. doi:10.1016/j.softx.2021.100905.
- 19 Roberto Posenato and Carlo Combi. Adding flexibility to uncertainty: Flexible simple temporal networks with uncertainty (FTNU). *Inf. Sci.*, 584:784–807, 2022. doi:10.1016/J.INS.2021.10.008.
- 20 Roberto Posenato and Carlo Combi. Flexible temporal constraint management in modularized processes. *Inf. Syst.*, 118:102257, 2023. doi:10.1016/J.IS.2023.102257.
- 21 Roberto Posenato, Marco Franceschetti, Carlo Combi, and Johann Eder. Some results and challenges Extending Dynamic Controllability to Agile Controllability in Simple Temporal Networks with Uncertainties. TechRep 1/2023, Dip. Informatica-Univ. di Verona, 2023. URL: <https://iris.univr.it/handle/11562/1116013>.
- 22 Roberto Posenato, Marco Franceschetti, Carlo Combi, and Johann Eder. Introducing agile controllability in temporal business processes. In *Enterprise, Business-Process and Information Systems Modeling - 25th International Conference, BPMDS 2024, and 29th International Conference, EMMSAD 2024*, volume 511 of *Lecture Notes in Business Information Processing*, pages 87–99. Springer, 2024. doi:10.1007/978-3-031-61007-3_8.
- 23 Roberto Posenato, Andreas Lanz, Carlo Combi, and Manfred Reichert. Managing time-awareness in modularized processes. *Softw. Syst. Model.*, 18(2):1135–1154, 2019. doi:10.1007/S10270-017-0643-4.
- 24 Michael Saint-Guillain, Tiago Vaquero, Steve A. Chien, Jagriti Agrawal, and Jordan R. Abrahams. Probabilistic temporal networks with ordinary distributions: Theory, robustness and expected utility. *J. Artif. Intell. Res.*, 71:1091–1136, 2021. doi:10.1613/JAIR.1.13019.
- 25 Thierry Vidal and H el ene Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.*, 11(1), 1999. doi:10.1080/095281399146607.

Open the Chests: An Environment for Activity Recognition and Sequential Decision Problems Using Temporal Logic

Ivelina Stoyanova ✉

U2IS, ENSTA Paris, Institut Polytechnique de Paris, Palaiseau, France
THALES, Palaiseau, France

Nicolas Museux ✉ 

THALES, Palaiseau, France

Sao Mai Nguyen ✉ 

U2IS, ENSTA Paris, Institut Polytechnique de Paris, Palaiseau, France

David Filliat ✉

U2IS, ENSTA Paris, Institut Polytechnique de Paris, Palaiseau, France

Abstract

This article presents **Open the Chests**, a novel benchmark environment designed for simulating and testing activity recognition and reactive decision-making algorithms. By leveraging temporal logic, **Open the Chests** offers a dynamic, event-driven simulation platform that illustrates the complexities of real-world systems. The environment contains multiple chests, each representing an activity pattern that an interacting agent must identify and respond to by pressing a corresponding button. The agent must analyze sequences of asynchronous events generated by the environment to recognize these patterns and make informed decisions. With the aim of theoretically grounding the environment, the Activity-Based Markov Decision Process (AB-MDP) is defined, allowing to model the context-dependent interaction with activities. Our goal is to propose a robust tool for the development, testing, and bench-marking of algorithms that is illustrative of realistic scenarios and allows for the isolation of specific complexities in event-driven environments.

2012 ACM Subject Classification Computing methodologies → Simulation environments

Keywords and phrases Event-Based Decision Making, Activity Recognition, Temporal Logic, Reinforcement Learning, Dynamic Systems, Complex Event Processing, Benchmark Environment, Real-Time Simulation

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.5

Supplementary Material *Software (Source Code)*: <https://github.com/ThalesGroup/open-the-chests>, archived at `swb:1:dir:4c883ad251fc88889142844a79d0d71cc667dd40`

1 Introduction

The emergence of smart technologies and automated information processing has generated an increasing interest in the fields of activity recognition [17, 8] and sequential event-based decision making [52, 12]. These fields are characterised by the identification and interpretation of behaviours as they occur and the optimisation of subsequent choices of reaction. They allow multiple applications ranging from monitoring and assisting in smart environments to enhance user convenience and safety [43, 13, 53], to automating cyber-security protocols for real time threat detection and response [15, 22, 18], and applying control in industrial settings to improve operational efficiency. However, the development of robust and reliable models for these domains has been challenged by the inherent complexity of their associated environments and the limited availability of suitable test-beds for evaluation [9, 33, 47].



© Ivelina Stoyanova, Nicolas Museux, Sao Mai Nguyen, and David Filliat;
licensed under Creative Commons License CC-BY 4.0

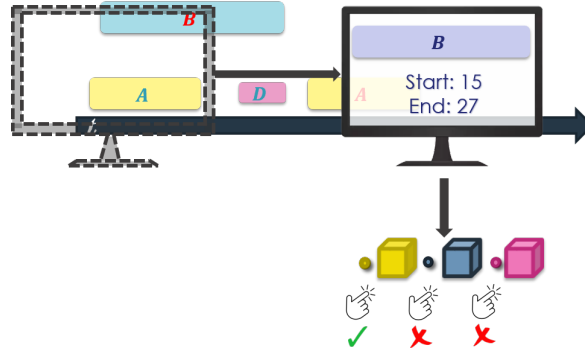
31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 5; pp. 5:1–5:19

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Overview of the **Open The Chests** environment. The figure shows *the event stream* being generated, the *observation of events* on the screen, and the corresponding *interactions with chests* based on significant sequence detection.

Environments in these fields are typically characterised by multiple interconnected activities evolving concurrently over time, observable indirectly through sensors and detection mechanisms. They often exhibit complex dynamics [44, 25], where activities are defined as significant patterns of observations characterized by complex temporal relationships and interdependencies. In the case of the event-driven paradigm [11, 9], observations are processed in order to extract significant changes in the environment under the form of data instances, also referred to as events. These events arrive asynchronously and provide temporal and attribute information on the occurred changes, constituting an event stream. The goal of the system interacting with the environment is to identify activities by analysing event signatures and selecting adapted reactions.

A key challenge for solutions and simulations lies in capturing the contextual and history-dependent nature of behaviors present in the event stream [5, 37, 42]. The significance and interpretation of an event are highly dependent on its surrounding context and the sequence of prior events. Additional challenges include heterogeneous data sources, complex temporal inter-dependencies [40], and uncertainty [2], which make it difficult to develop robust and adaptable systems. These challenges also complicate the process of obtaining large-scale datasets and conducting controlled simulations, thereby hindering the advancement of solutions [25, 20, 55]. Collecting and annotating datasets is expensive, time-consuming, and often impractical due to the vast number of scenario configurations and complexity levels required. Capturing the full range of variability, uncertainty, and asynchrony in event-driven environments often necessitates multiple datasets, further complicating the process. On the other hand, simulators must balance application specificity and simplicity. While a highly detailed, application-specific simulator might closely mimic real-world conditions, it can also make it difficult to isolate and evaluate specific complexities objectively. Conversely, overly simplified simulations may miss critical nuances, leading to gaps in testing.

To address this issue, we introduce **Open The Chests**¹, illustrated in Figure 1, a novel reinforcement learning environment that simulates a gamified scenario of activity recognition and sequential decision-making. Modeled after popular Reinforcement learning environments [51], the environment is designed to represent the complexities of real-world scenarios, where the significance of observations and appropriate reactions are highly dependent on temporal context and the history of events. Its configurable nature allows for multiple levels of complexity, enabling users to isolate specific challenges and systematically evaluate

¹ <https://github.com/ThalesGroup/open-the-chests>

various aspects of decision-making processes. This article details the core complexities of the environment, formalizes the main elements of the problem, and introduces `Open The Chests` as a versatile tool for such evaluations.

2 Background and Related Work

Multiple domains developing solutions to the challenges of reactive decision-making could benefit from the usage of standardized benchmarks, particularly in event-driven environments. Complex Event Processing (CEP) and Complex Event Recognition (CER) systems [36, 14, 24] are methods designed to identify and recognize intricate patterns within streams of events using predefined rules. Examining complexities such as uncertainty, noise, and varying context lengths could be crucial in assessing the robustness and reliability of these systems in diverse scenarios. Similarly, when extracting rules and patterns for these domains [19, 32], exploring how the accuracy and interpretability of results vary with scenario complexity could provide valuable insights into the adaptability of these approaches. For the temporal interval pattern mining community [38, 41, 6], comparisons across algorithms could be enriched by considering their computational efficiency and limits, especially in the context of standardized benchmarks. Other approaches to monitoring activities involve using probabilistic methods, often combined with logic-driven techniques [26, 2]. These methods typically estimate the likelihood of events, and exploring their adaptability in less predictable contexts can provide insights into their ability to maintain accuracy under varying conditions. Finally, Reinforcement Learning [48], which is showing growing interest in real-world scenarios [20], would gain from frameworks that allow thorough testing of long-term, contextual dependencies and delayed rewards.

Current benchmarks and datasets for event-driven environments typically fall into two distinct categories, focusing either on the classification of activities [54] or sequential decision-making [51]. Classification methods, which involve identifying and categorizing specific patterns of behavior or events, are often driven by deep learning techniques or knowledge-based methods [12, 8]. Several datasets are available for these purposes [8, 4], offering a range of scenarios for recognition. However, while effective in controlled, static settings, these classification methods often struggle to adapt to the dynamic nature of realistic environments, as they typically do not account for the impact of interaction. Conversely, sequential decision-making frameworks, such as those in Reinforcement Learning, focus on optimizing decisions based on the system's current state, making them well-suited for dynamic environments. Despite their strengths, these methods often rely on simplified environmental models, which can limit their real-world applicability [9]. Specifically, many existing simulators fail to account for the complexities of history dependence, context dependence, and inter-dependencies within the environment. Active research is working to address these challenges by incorporating more complex environmental features into these frameworks [7, 34]. These benchmarks challenge agents with higher-level tasks, requiring memory and goal abstraction. However, to our knowledge, none of them specifically address the task of activity recognition, motivating our development of the `Open The Chests` environment.

3 Formalising Activities using Temporal Logic and Attribute Filters

The development of robust algorithms and reliable simulations for event-driven environments requires the precise formalization of events, activities, and interactions. This formalization provides a foundation for understanding system dynamics and guiding the construction of

5:4 Open the Chests

the environment. In particular, *activities* are critical constructs in reactive decision-making systems, serving as the basis for identifying relevant scenarios and triggering appropriate responses. By integrating both temporal and attribute constraints on the event stream, activities enable the precise encoding of complex semantics as high-level abstractions, capturing meaningful patterns of behavior over time.

3.1 Definition of Events

Events are the fundamental building blocks of the environment, representing significant changes that occur over time. Formally, an event e can be represented as a tuple:

$$e = (sym, Attr, t_{start}, t_{end},)$$

where:








- sym is a symbolic identifier that categorizes the type of event, facilitating its recognition and interpretation within the system.
- $Attr = \{attr_1, \dots, attr_n\}$ represent a set of attributes that provide additional information about the event. These attributes can include, but are not limited to, spatial coordinates, intensity levels, source identifiers, and other domain-specific parameters.
- t_{start} and t_{end} denote the timestamps marking the initiation and conclusion of the event, respectively. These temporal markers are crucial for understanding the duration and sequence of events.

Events arrive asynchronously and are processed to form a history, or event stream. It is denoted as h_t , where t represents the current discrete time step, or the number of the last received event:

$$h_t = \langle e_1, e_2, \dots, e_t \rangle$$

The temporal aspect of events allows for identifying relationships between them, enabling the construction of higher-level abstractions. Attributes enrich event representation by encoding domain-specific information, facilitating more sophisticated analysis and reasoning about the state of the environment.

3.2 Temporal Relations between Events

Allen Statements		Example	Chronological sequence
Relations	Inverse Relations		
X before Y (<)	Y after X (>)		$X_{start} < X_{end} < Y_{start} < Y_{end}$
X equals Y (=)	Y equals X (=)		$X_{start} = Y_{start} < X_{end} = Y_{end}$
X meets Y (m)	Y met by X (mi)		$X_{start} < X_{end} = Y_{start} < Y_{end}$
X overlaps Y (o)	Y overlapped by X (oi)		$X_{start} < Y_{start} < X_{end} < Y_{end}$
X contains Y (c)	Y during X (d)		$X_{start} < Y_{start} < Y_{end} < X_{end}$
X starts Y (s)	Y started by X (si)		$X_{start} = Y_{start} < X_{end} < Y_{end}$
X finishes Y (f)	Y finished by X (fi)		$Y_{start} < X_{start} < X_{end} = Y_{end}$

■ **Figure 2** Allen's 13 temporal interval relations: *before*, *after*, *meets*, *met-by*, *overlaps*, *overlapped-by*, *starts*, *started-by*, *during*, *contains*, *finishes*, *finished-by*, *equal*.

Encoding temporal relationships between events is a key aspect of an activity’s representation, capturing their relative order, duration, and overlaps. Various formalisms have been proposed for representing temporality, each with its own trade-offs in expressiveness and computational complexity [16, 27]. For the **Open The Chests** environment, we leverage Allen’s Interval Algebra [3] due to its expressivity in capturing temporal relations and its established use in the activity recognition community [13, 39]. This algebra defines a set of thirteen mutually exhaustive temporal relationships between two time intervals. As depicted in Figure 2, each relation imposes a unique constraint on the start and end times of the two considered events. Leveraging these concepts, we establish the temporal relation \mathcal{T}_{allen} between two events e_i and e_j :

$$\mathcal{T}_{allen}(e_i, e_j) = \begin{cases} \text{True,} & \text{if } e_i.t_{start}, e_i.t_{end}, e_j.t_{start}, e_j.t_{end} \text{ respect conditions.} \\ \text{False,} & \text{otherwise} \end{cases} \quad (1)$$

Each relation captures a specific temporal interaction between intervals, enabling the precise modeling of complex temporal sequences. Specific Allen relations are denoted by using subscripts such as $\mathcal{T}_{before}(e_i, e_j)$.

3.3 Attribute Filtering

In addition to temporal relationships, activities are characterized by dependencies between event attributes. Specifically, recognizing an activity relies on recognizing the specific attribute values associated with events. For example, in a smart-home scenario, the location attribute may be crucial for identifying activities such as “cooking” or “watching TV”. Thus, the definition of an activity can be refined by imposing constraints on the attribute values of its constituent events, introducing the notion of filtering. To formalize this, we define an attribute filter function F_a that evaluates the relevance of an event’s attributes:

$$F_a(e) = \begin{cases} \text{True,} & \text{if the attributes of } e \text{ satisfy the filter conditions.} \\ \text{False,} & \text{otherwise} \end{cases} \quad (2)$$

Comparing the attribute values of two events can also determine their relevance to the same activity, which is crucial for accurately linking contextually connected events. For instance, in a surveillance system, two events occurring in the same area might need to share the same location attribute to accurately recognize a security breach or suspicious behavior. To formalize this, we define a relative attribute filter function Fr that evaluates the relevance of a pair of events’ attributes:

$$Fr(e_1, e_2) = \begin{cases} \text{True,} & \text{if the attributes of } e_1, e_2 \text{ jointly satisfy the filter conditions.} \\ \text{False,} & \text{otherwise} \end{cases} \quad (3)$$

3.4 Composition and Definition of Activities

Formally, an activity A is a temporally-structured sequence of m events which follow specific temporal and attribute relationships. The recognition of an activity is formalized by the function R_A , which combines these temporal and attribute relations to determine whether a given set of events constitutes a recognized activity. This can be expressed as:

$$R_A(e_1, \dots, e_m) = \bigwedge_{i=1}^m F_a(e_i) \wedge \bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^m Fr_{i,j}(e_i, e_j) \wedge \bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^m \mathcal{T}_{allen_{i,j}}(e_i, e_j) \quad (4)$$

We assume that an environment is constituted of n activities $\{A_1, \dots, A_n\}$ with corresponding recognition functions $\{R_{A_1}, \dots, R_{A_n}\}$.

4 Defining Interaction within the Environment

The interaction between the system and its environment evolves continuously. A stream of events is presented to the agent, generated by an underlying set of activities which prompt interaction. For realism, it is crucial to model not only the activities themselves but also the impact of interactions on the environment. Specifically, the agent's decisions have meaningful effects on the state of the environment, creating a closed-loop dynamic interaction. Traditionally, this decision-making process is modeled as a Markov Decision Process (MDP) [23] or a Partially Observable Markov Decision Process (POMDP) [46]. However, these approaches often fall short in capturing the complexities of event-driven environments, particularly those that exhibit rich contextual and historical dependencies [44].

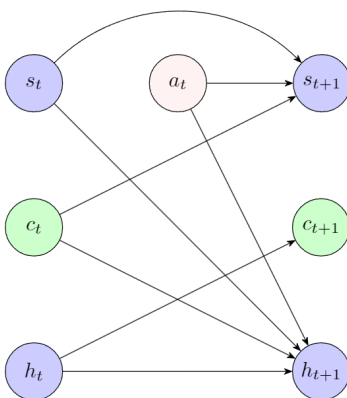
4.1 Challenges in Decision making and Modeling as an MDP

The challenges frameworks face can be categorized into three main areas:

- **State Space Complexity:** The asynchronous and concurrent nature of activities, along with their inter-dependencies, significantly expands the state space. Each activity, defined by its events, attributes, and temporal dependencies, has multiple states of advancement. With multiple activities occurring simultaneously in the environment, representing states as the compounded relations of these activities becomes complex. The coupling of events from different activities creates a multi-dimensional activity space, where the complexity increases due to their inter-dependencies and overlapping temporal dynamics. The number of possible event combinations and correlations grows exponentially with the number of actors, activities, and attributes, complicating the transition function and leading to modeling complexity [45, 35, 29].
- **Contextual and Historical Dependencies:** The state of activities is not directly observable by the agent, which must rely on the history of events to make inferences. Because individual events cannot be fully understood without their associated temporal and attribute relations, the agent is compelled to consider the entire event stream to accurately infer the underlying activities. This process underscores the critical role of context and history in determining the significance of events. The reliance on historical context and continuous event streams breaks the Markov assumption, which assumes that future states depend only on the current state and not on the sequence of past events [5, 50].
- **Temporally-Structured Nature of Activities:** Events and activities are inherently dependent on time and structured in complex temporal patterns, rather than being independent, instantaneous transitions. The temporal relations between events are crucial for recognizing activities and must be specifically captured. This temporal structuring is essential because activities often span multiple time steps and involve events that interact over time. Therefore, enriched state representations that incorporate these temporal dynamics are necessary to accurately reflect real-world scenarios and support effective decision-making processes [44, 49].

4.2 Defining an Activity-Based MDP

To address these limitations, we propose a novel formulation of the decision-making problem in event-driven environments by building on the fundamentals of activity recognition. We define the Activity-Based Markov Decision Process (AB-MDP), which integrates principles of Activity Recognition within the Markov Decision Process framework. Similar to Contextual Markov Decision Processes [28] and Dynamic Contextual Markov Decision Processes [50], the state space in the AB-MDP is expanded to include not only the observable state information but also additional contextual information. History dependence is captured by a contextual variable that indicates whether an activity has been completed, conditioning rewards and transitions. This contextual variable is latent to the agent, making it a special case of POMDP. In this version of the AB-MDP, illustrated in Figure 3, we assume that interventions are relevant only when activities have been completed. This simplification allows the model to focus on the state of recognized activities rather than the entire event sequence.



■ **Figure 3** Causal diagram depicting the dependencies in an AB-MDP. Green circles represent unobserved variables. Here s_t and s_{t+1} are the current and next states, c_t and c_{t+1} are the current and next context variables, h_t and h_{t+1} are the current and next histories and a_t is the applied action.

► **Definition 1** (Activity-Based MDP). *Supposing an environment is defined by the presence of n activities, an AB-MDP is defined by the tuple $\langle S, E, C, A, T, \mathcal{R} \rangle$*

- S is the observable space of the environment, constituted by any observable information outside of events.
- E is the space of events observations.
- C is a contextual vector of size n . Each element c_i indicates the completion status of the i -th activity, with $c_i \in \{\text{True}, \text{False}\}$. It is latent to the learning agent and must be inferred from the history of observations.
- A is the finite action space defined by the possible responses or reactions the system can take.
- T is a context dependent transition function defined as $T(s_t, a_t, c_t, s_{t+1}) = \Pr(s_{t+1} \mid s_t, a_t, c_t)$, with $s_t, s_{t+1} \in S$, $a_t \in A$ and $c_t \in C$. The transition is influenced by the actions taken and the currently active activities.
- \mathcal{R} is the reward function defined as $\mathcal{R}(s, a, c) = r$, which evaluates the success of actions in reacting to recognized activities.

► **Example 2.** To illustrate the components of the AB-MDP, consider the example of an autonomous drone monitoring a restricted area. The state at each step s_t represents the drone’s current position, battery level, and basic environmental data like wind speed. In contrast, events e_t capture instantaneous occurrences such as detecting movement near a perimeter fence, the activation of a door sensor, or an alarm signal triggered by unauthorized entry. The context variable c_t indicates whether specific activities, like unauthorized entry, have been completed, while the action a_t might involve the drone adjusting its position, zooming its camera, or sending an alert to security personnel.

At each time step t , the contextual variable c_i specifies whether there exists a subset of events $\{e_1, \dots, e_m\}$ in the event history $h_t \in \mathcal{H}$ that satisfy the predicate conditions for activity i .

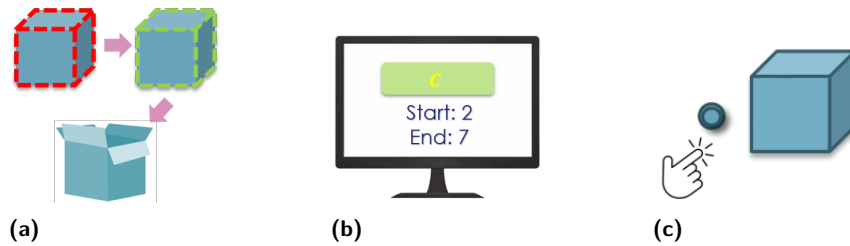
$$c_i = \begin{cases} \text{True,} & \{e_1, \dots, e_m\} \subset h_t | R_{A_i}(e_1, \dots, e_m) = \text{True} \\ \text{False,} & \text{otherwise} \end{cases} \quad (5)$$

This means that the next context occurs with a probability dependent on the history of events. Since this variable is latent to an observer of the environment, its values must be inferred from the observed events.

5 Description of the Open the Chests Environment

The **Open The Chests** environment implements an AB-MDP to simulate the complexities of real-world event-driven systems, providing a configurable platform for testing activity recognition and reactive decision-making algorithms. The environment models a scenario where an agent interacts with a series of chests that can be opened based on specific, unknown patterns of events. The agent’s primary goal is to recognize these patterns by processing the observable event stream and deciding which buttons to press to open the appropriate chest. The environment’s role is to generate events that respect the configured activities and allow interaction with their corresponding chests. This setup illustrates the dynamic and context-dependent nature of real-world systems, where recognizing event sequences allows the agent to modify the environment.

5.1 Game Mechanics



■ **Figure 4** The main elements of the **Open The Chests** environment: (a) boxes and their respective states (**active**, **ready**, **open**); (b) event observations with their symbol, attributes (**background color**, **foreground color**) and temporal information (**start**, **end**); (c) buttons for interacting with chests to modify their state.

The **Open the Chests** environment consists of several key elements, each playing a critical role in the simulation: chests, observations, buttons, and rewards. Figure 4 illustrates how these elements are presented in the environment.

- **Event Observations:** Event observations are the *event symbols, attributes and temporal information* presented to the agent. These observations form the event stream that the agent must analyze to recognize patterns and make decisions. At each step, the environment emits *last occurred event* (see Sec.5.2), generated according to the activities present in the environment (see Sec.5.3).
- **Chests:** Chests represent the *activities* present in the environment that the agent needs to recognize and address. Each chest corresponds to a specific *pattern of events* within the observable event stream. The state of chests is partially observable: the information of whether a chest is *open* or *active* is known, while the information of a chest being *ready* is hidden (see Sec.5.4).
- **Buttons:** Buttons are the interactive elements associated with each chest representing *actions*. Pressing the corresponding button allows opening the chest if the associated pattern of events is present in the event history, i.e. if the chest is **ready** (see Sec.5.4).
- **Rewards:** Rewards are the *feedback* which the environment provides to the agent based on its actions. If the button is pressed at the correct time, the corresponding chest opens and a positive reward is given; otherwise, a negative or no reward is received (see Sec.5.5).

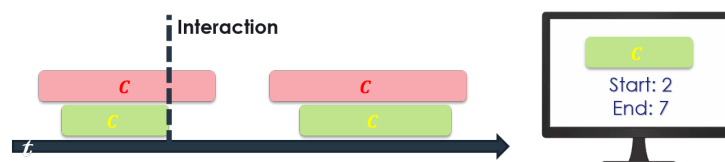
The environment continuously generates events based on predefined activities until all chests are recognized and opened. To solve this, the agent must monitor the event stream history, detect patterns, and infer the latent context. Upon recognizing a pattern corresponding to a chest activity, the agent must decide to press the appropriate button, linking the pattern, chest, and action.

5.2 Event Observation and Interaction Time

Events are displayed as symbols with varying values, colors, and background colors, representing different types of detections in the environment (Figure 5). Each event also carries continuous time information, indicating its start and end times. These events are presented to the agent one by one upon completion, allowing the agent to make decisions based on fully observed events. This approach ensures that the environment operates in discrete steps. While the timeline of events is not directly visible to the agent, it is implicitly understood through the sequence displayed on the observation screen. The environment allows for variation in event length during configuration, adding complexity and realism to the simulation. This variability challenges the agent to adapt to different event durations, enhancing the robustness of pattern recognition algorithms.



■ **Figure 5** Possible events symbols and their attributes represented by symbols and colors.

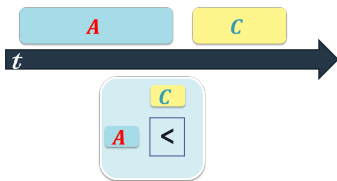


■ **Figure 6** An event is communicated to the agent at its completion.

► **Example 3.** Consider an activity defined by two overlapping events, both marked with the symbol C , as shown in the Figure 6. The activity recognition rule $R_A(e_1, e_2) = F_a(e_1, C, green, yellow) \wedge F_a(e_2, C, red, red) \wedge \mathcal{T}_{contains}(e_1, e_2)$ involves detecting event e_1 with the attributes C , $green$, and $yellow$, and event e_2 with the attributes C , red , and red , while maintaining the temporal relation $\mathcal{T}_{contains}(e_1, e_2)$. Since the *contains* relation requires that $t_{end_{e_1}} < t_{end_{e_2}}$, event e_1 will be generated and communicated first, along with its specified colors and timestamps.

5.3 Defining event patterns and generating events

When configuring event patterns for each chest, we use Temporal and Attribute relations as defined in Equation 4. To facilitate the management of these relations, we use Höppner’s matrix form [30] to transform them into a structured format, which makes it easier to check for continuity and detect any contradictions. To handle event generalisation and simplify the complexity of multiple concurrent activities, we represent patterns as memory-enriched automata [31]. Their goal is to track both current and pending events, while storing relevant past events that are needed for generation. During execution, the next event to generate is thus selected with respect to all activities and their current execution step. Each activity is configured to begin after a certain delay, which can be specified during its definition. Additionally, Allen Relations can be parameterized to introduce varying delays between events, adding complexity and realism to the simulation. Finally, empty filters can be defined, allowing for variations in attributes during event generation.



■ **Figure 7** An illustration of a defined pattern and its associated matrix representation.

```

INSTANTIATE
- name: e1
  type: A
  params:
    fg: red
    bg: blue
  duration:
    mu: 5
    sigma: 2
- name: e2
  type: C
  params:
    fg: blue
    bg: yellow
  duration:
    mu: 6
    sigma: 2
    
```

■ **Figure 8** Defining the events and attribute filters of a two-event pattern.

```

RELATIONSHIP:
- type: after
  events:
    - e1
    - e2
  other:
    gap_dist:
      mu: 4
      sigma: 1
    
```

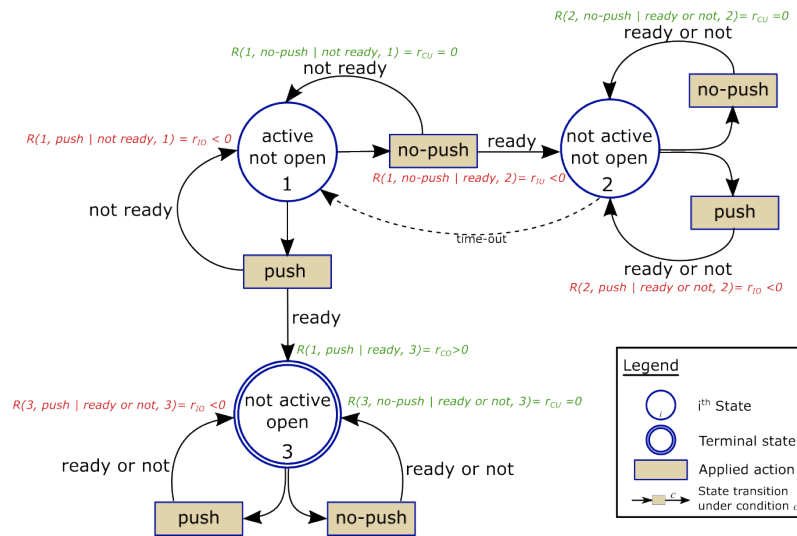
■ **Figure 9** Defining the relations of a two-event pattern.

► **Example 4.** To define a pattern consisting of two events where one occurs before the other, and where the events are represented by symbols A and C , we start by specifying their attributes and relations, as shown in Figures 8 and 9. We use the parameters μ and σ to define a normal distribution that will be used to sample the duration of events or the time between events. In this case, the matrix representation of this pattern (Figure 7) includes only one temporal relation between the two events.

Complexity is added to the environment by introducing noise and randomness into the event stream, simulating some of the inherent uncertainty of real-world systems. Specifically, patterns can be generated with varying degrees of noise, enabling the system to add events to the stream that are unrelated to the execution of current activities.

5.4 Chests and Buttons: Interaction and Box States

Chests represent activities in the environment, each with three binary state values: **active**, **ready**, and **open**. At the start of the game, all chests are closed (**not open**). A chest becomes **active** once it starts generating events that are observed in the event stream. When the sequence of events is fully generated and present in the stream, the corresponding chest becomes **ready** to open by pushing a button. If the button is pushed, the chest is marked as **open**, its pattern is removed from the environment, and it stops generating further events. If the button is not pushed, the pattern generation continues and eventually restarts. Until a chest is opened, its associated pattern continues to repeat, providing the agent with multiple opportunities to recognize and interact with the sequence. The **active** and **open** states represent the observable part of the environment and are communicated to the agent with each event observation. The **ready** state serves as the activity context, determining the outcome of button actions.



■ **Figure 10** The state transition graph of a single chest. A chest is initially **active** and **not open**. Its transitions are conditioned by the pushing buttons and on its associated **ready** value.

► **Example 5.** Suppose that only one chest is defined in the environment using the activity definitions in Example 3. Initially it is *active* and generates events one by one. Once both its associated events have been observed it will pass to the state *ready*, meaning its associated context variable $c_1 = True$ will indicate activity completion. If the button is correctly pressed, the chest will *open* and no further events will be generated. Otherwise, the chest is deactivated and goes back to the *active* state after a delay. This transition is illustrated in Figure 10.

5.5 Rewards

Reward is defined in terms of the number of correctly opened chests N_{CO} , incorrectly opened chests N_{IO} , correctly unopened chests N_{CU} , and incorrectly unopened chests N_{IU} . These notions are equivalent to true positives, true negatives, false positives, and false negatives, respectively. The action $a_t \in \{\text{True}, \text{False}\}^n$ defines the chests the agent attempted to open and the time at which the agent pressed the button corresponding to each chest. We use the vector $c_t \in \{\text{True}, \text{False}\}^n$ to represent which of the chests were ready to open at time t . We can thus define the following:

- $N_{CO} = a_t \wedge c_t$ meaning all correctly opened chests
- $N_{IO} = a_t \wedge (\neg c_t)$ meaning all incorrectly attempted chests
- $N_{CU} = (\neg a_t) \wedge (\neg c_t)$ meaning all correctly unopened chests
- $N_{IU} = (\neg a_t) \wedge c_t$ meaning all incorrectly unattempted chests

We define a reward for each type of chest: r_{CO} , r_{IO} , r_{CU} , r_{IU} , allowing us to calculate the final reward r .

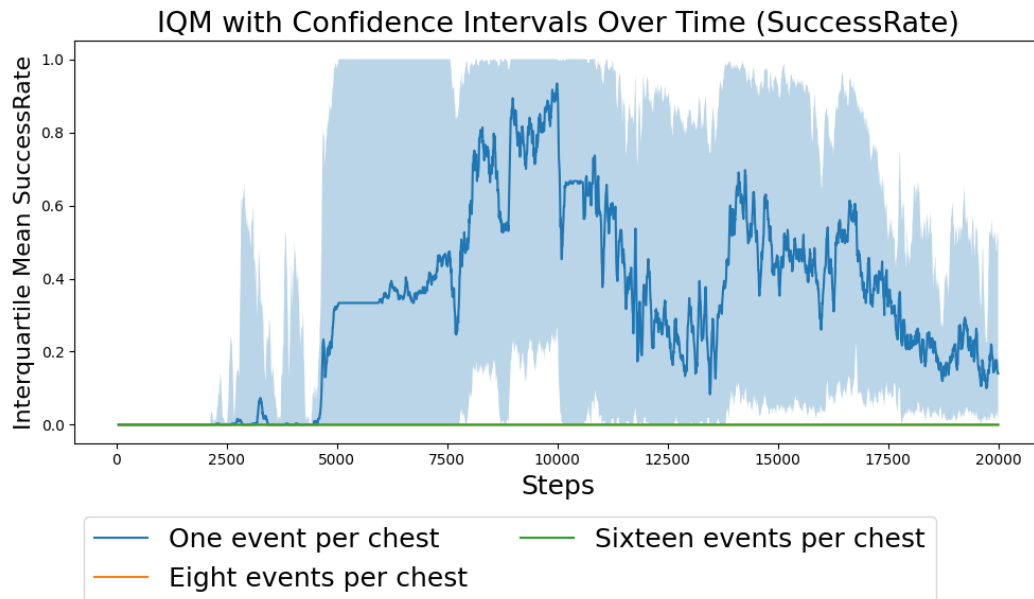
$$r = r_{CO} \cdot N_{CO} + r_{IO} \cdot N_{IO} + r_{CU} \cdot N_{CU} + r_{IU} \cdot N_{IU}$$

For the time being, we set the values $r_{CO} = 1$, $r_{IO} = -1$, $r_{CU} = 0$, and $r_{IU} = -1$.

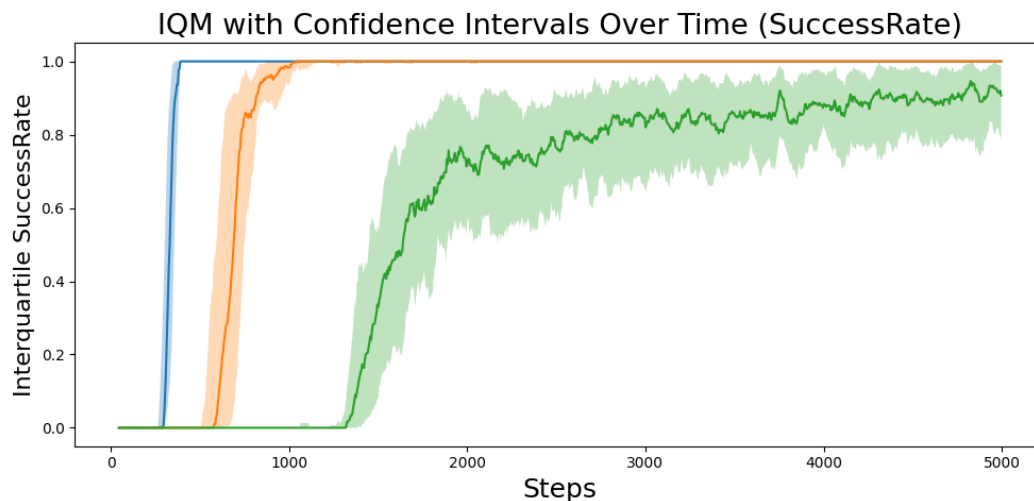
6 Validation of the Open The Chests Environment

The goal of our validation was to ensure the proper execution of the OpenTheChests environment and to establish performance baselines using two algorithms: Deep Q-Network (DQN) and Deep Transformer Q-Network (DTQN) [21]. Successfully generating meaningful and consistent results across both algorithms confirms that the environment functions as intended, providing a reliable platform for testing and comparing different algorithms. This validation also demonstrates the environment’s capability to effectively differentiate between the performance and behavior of distinct decision-making approaches, ensuring its integration with existing methods.

To achieve this, we configured three scenarios with varying levels of complexity. Each configuration contained 5 chests but differed in the number of events per activity: 1, 8, and 16 events. The simplest configuration associates one unique event per chest, meaning it doesn’t require additional temporal or context dependencies. Once the event is observed, the expected response is to identify the appropriate chest and press its button. Conversely, the environments with eight and sixteen events per chest require identifying sequences of appropriate length as well as their respective attribute and temporal relations. To configure these patterns, we randomly selected the corresponding number of event filters and Allen temporal interval relations. Finally, the environment was configured with varying levels of noise per activity, ranging from 0.1 to 0.3, meaning that a proportionate amount of non-relevant events was generated alongside the activity patterns. Our evaluations is shown in Figure 11, where each learning curve presents the success rate of the agents during training across the three different environments. To asses the performance in both algorithms, we measured the success rate in each scenario, reflecting the ability of agents to correctly identify and interact with the appropriate chests. We utilized the `rliable` library [1] to calculate the Interquartile Mean (IQM), which provides a robust measure of central tendency, along with stratified bootstrap confidence intervals to capture the variability of the results across five random seeds. These metrics were plotted over the course of training, allowing us to observe the learning progress and stability of each algorithm in handling the varying levels of complexity within the environments.



(a) DQN.



(b) DTQN.

■ **Figure 11** Success rates of DTQN and DQN across three environments during training with 1, 8, and 16 events per activity. Each pattern was configured with a noise value between 0.1 and 0.3. Values are measured across 5 random seeds.

Both DQN and DTQN algorithms were successfully integrated with the environment by leveraging the standardised gym framework. As expected, DQN performed poorly in scenarios requiring historical context and complex temporal dependencies. The algorithm struggled to effectively handle tasks that demanded memory of past events or intricate event interdependencies, which are critical in the `Open The Chests` environment. DQN's design, optimized for simpler, state-based decisions, lacks the mechanisms necessary to process and utilize long-term dependencies, leading to suboptimal performance in these context-sensitive scenarios. Interestingly, we also observed struggles in simpler scenarios without history

dependence, likely due to the challenges posed by multiple parallel activities and the multi-dimensional nature of the actions required. DTQN demonstrated superior performance, especially in more complex scenarios, due to its ability to incorporate past observations. The use of transformer architectures allows DTQN to maintain and utilize a memory of sequential events, enabling it to make more informed decisions based on the historical context. However, as the scenarios became increasingly complex with longer contexts, DTQN's performance began to decline, likely due to limitations in the model's parameter settings, such as the fixed context window size, which may not fully capture extended dependencies, as well as its capacity to effectively separate parallel activities. Further exploration of these methods, particularly through the use of custom configurations to isolate specific complexities, would allow for a better understanding of the algorithms' limitations. This underscores the value of the **Open The Chests** environment. Additionally, the lack of interpretability in both algorithms remains an important consideration for future development, as it impacts their practical applicability in real-world scenarios.

7 Conclusion and Perspectives

The **OpenTheChests** environment facilitates defining benchmarks of varying complexities, depending on the number of chests and the complexity of their corresponding patterns. Integrated with the `gym` [10] framework, it models dynamic, interactive scenarios where the agent must recognize patterns of events and make timely decisions, illustrating real-world system complexities. Through the integration of Activity-Based Markov Decision Processes (AB-MDP), **OpenTheChests** simulates context-dependent, sequential decision-making tasks, facilitating comprehensive testing and development of advanced algorithms.

Looking ahead, several enhancements are planned to further develop the capabilities of **OpenTheChests**. These include:

- **Dependence Between Activities and Model Expansion:** Introducing mechanisms where activities can influence each other, creating more complex inter-dependencies and richer scenarios for testing decision-making strategies. We aim to develop an advanced AB-MDP that considers intermediate activity states.
- **Event Sharing Between Activities:** Implementing various event consumption policies that govern how events are shared or partitioned among concurrent activities, enabling the exploration of different coordination strategies.
- **Advanced Attribute Relations:** Developing more sophisticated attribute relations that can span multiple events, enhancing the ability to model and recognize complex patterns and dependencies.

Future research will focus on expanding the complexity of the **OpenTheChests** environment and exploring its applications in various real-world scenarios. While initial results showed good performance, both DQN and DTQN lacked interpretability. Further goals include improving the interpretability of the algorithms, enabling clearer insights into decision-making processes, and enhancing the overall robustness of the environment.

References

- 1 Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

- 2 Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. Probabilistic complex event recognition: A survey. *ACM Computing Surveys (CSUR)*, 50(5):1–31, 2017. doi:10.1145/3117809.
- 3 James F Allen and George Ferguson. Actions and events in interval temporal logic. *Journal of logic and computation*, 4(5):531–579, 1994. doi:10.1093/LOGCOM/4.5.531.
- 4 Anindya Das Antar, Masud Ahmed, and Md Atiqur Rahman Ahad. Challenges in sensor-based human activity recognition and a comparative analysis of benchmark datasets: A review. In *2019 Joint 8th International Conference on Informatics, Electronics & Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, pages 134–139. IEEE, 2019.
- 5 Fahiem Bacchus, Craig Boutilier, and Adam Grove. Structured solution methods for non-markovian decision processes. In *AAAI/IAAI*, pages 112–117. Citeseer, 1997. URL: <http://www.aaai.org/Library/AAAI/1997/aaai97-018.php>.
- 6 Iyad Batal, Gregory F Cooper, Dmitriy Fradkin, James Harrison, Fabian Moerchen, and Milos Hauskrecht. An efficient pattern mining approach for event detection in multivariate temporal data. *Knowledge and information systems*, 46:115–150, 2016. doi:10.1007/S10115-015-0819-6.
- 7 Benjamin Beyret, José Hernández-Orallo, Lucy Cheke, Marta Halina, Murray Shanahan, and Matthew Crosby. The animal-ai environment: Training and testing animal-like artificial cognition. *arXiv preprint arXiv:1909.07483*, 2019. arXiv:1909.07483.
- 8 Damien Bouchabou, Sao Mai Nguyen, Christophe Lohr, Benoit LeDuc, and Ioannis Kanellos. A survey of human activity recognition in smart homes based on iot sensors algorithms: Taxonomies, challenges, and opportunities with deep learning. *Sensors*, 21(18):6037, 2021. doi:10.3390/S21186037.
- 9 Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999. doi:10.1613/JAIR.575.
- 10 Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. arXiv:arXiv:1606.01540.
- 11 Christos G Cassandras and Stéphane Lafortune. *Introduction to discrete event systems*. Springer, 2008.
- 12 Wuhui Chen, Xiaoyu Qiu, Ting Cai, Hong-Ning Dai, Zibin Zheng, and Yan Zhang. Deep reinforcement learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1659–1692, 2021. doi:10.1109/COMST.2021.3073036.
- 13 Jean-René Coffi, Christophe Marsala, and Nicolas Museux. Adaptive complex event processing for harmful situation detection. *Evolving Systems*, 3:167–177, 2012. doi:10.1007/S12530-012-9052-7.
- 14 Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):1–62, 2012. doi:10.1145/2187671.2187677.
- 15 Ala’ Darabseh and Nikolaos M Freris. A software-defined architecture for control of iot cyberphysical systems. *Cluster Computing*, 22(4):1107–1122, 2019. doi:10.1007/S10586-018-02889-8.
- 16 Dario Della Monica, Valentin Goranko, Angelo Montanari, and Guido Sciavicco. Interval temporal logics: a journey. *Bulletin of EATCS*, 3(105), 2013.
- 17 Florenc Demrozi, Graziano Pravadelli, Azra Bihorac, and Parisa Rashidi. Human activity recognition using inertial, physiological and environmental sensors: A comprehensive survey. *IEEE access*, 8:210816–210836, 2020. doi:10.1109/ACCESS.2020.3037715.
- 18 Derui Ding, Qing-Long Han, Yang Xiang, Xiaohua Ge, and Xian-Ming Zhang. A survey on security control and attack detection for industrial cyber-physical systems. *Neurocomputing*, 275:1674–1683, 2018. doi:10.1016/J.NEUCOM.2017.10.009.

- 19 Anton Dries and Luc De Raedt. Towards clausal discovery for stream mining. In *Inductive Logic Programming: 19th International Conference, ILP 2009, Leuven, Belgium, July 02-04, 2009. Revised Papers 19*, pages 9–16. Springer, 2010. doi:10.1007/978-3-642-13840-9_2.
- 20 Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021. doi:10.1007/S10994-021-05961-4.
- 21 Kevin Esslinger, Robert Platt, and Christopher Amato. Deep transformer q-networks for partially observable reinforcement learning. *arXiv preprint arXiv:2206.01078*, 2022. doi:10.48550/arXiv.2206.01078.
- 22 Ido Finder, Eitam Sheetrit, and Nir Nissim. Time-interval temporal patterns can beat and explain the malware. *Knowledge-Based Systems*, 241:108266, 2022. doi:10.1016/J.KNOSYS.2022.108266.
- 23 Frédéric Garcia and Emmanuel Rachelson. Markov decision processes. *Markov Decision Processes in Artificial Intelligence*, pages 1–38, 2013.
- 24 Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos Garofalakis. Complex event recognition in the big data era: a survey. *The VLDB Journal*, 29:313–352, 2020. doi:10.1007/S00778-019-00557-w.
- 25 Ben Goertzel, Nil Geisweiller, Lucio Coelho, Predrag Janičić, and Cassio Pennachin. *Real-World Reasoning: Toward Scalable, Uncertain Spatiotemporal, Contextual and Causal Inference*, volume 2. Springer Science & Business Media, 2011.
- 26 Shaogang Gong and Tao Xiang. Recognition of group activities using dynamic probabilistic networks. In *Proceedings ninth IEEE international conference on computer vision*, pages 742–749. IEEE, 2003. doi:10.1109/ICCV.2003.1238423.
- 27 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. doi:10.3166/JANCL.14.9-54.
- 28 Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015. arXiv:1502.02259.
- 29 Derek Hao Hu, Sinno Jialin Pan, Vincent Wenchen Zheng, Nathan Nan Liu, and Qiang Yang. Real world activity recognition with multiple goals. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 30–39, 2008. doi:10.1145/1409635.1409640.
- 30 Frank Höppner. Learning temporal rules from state sequences. In *IJCAI Workshop on Learning from Temporal and Spatial Data*, volume 25. Citeseer, 2001.
- 31 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- 32 Nikos Katzouris, Evangelos Michelioudakis, Alexander Artikis, and Georgios Paliouras. Online learning of weighted relational rules for complex event recognition. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II 18*, pages 396–413. Springer, 2019. doi:10.1007/978-3-030-10928-8_24.
- 33 Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022. doi:10.1613/JAIR.1.13673.
- 34 Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI conference on artificial intelligence*, pages 4501–4510, 2020.
- 35 Niels Landwehr. Modeling interleaved hidden processes. In *Proceedings of the 25th international conference on Machine learning*, pages 520–527, 2008. doi:10.1145/1390156.1390222.
- 36 D Luckham. The power of events: An introduction to complex event processing in distributed enterprise systems. additions-wesley. Reading, 2001.

- 37 Sultan Javed Majeed and Marcus Hutter. On q-learning convergence for non-markov decision processes. In *IJCAI*, volume 18, pages 2546–2552, 2018. doi:10.24963/IJCAI.2018/353.
- 38 Nijat Mehdiyev, Julian Krumeich, David Enke, Dirk Werth, and Peter Loos. Determination of rule patterns in complex event processing using machine learning techniques. *Procedia Computer Science*, 61:395–401, 2015. doi:10.1016/J.PROCS.2015.09.168.
- 39 Vlad I Morariu and Larry S Davis. Multi-agent event recognition in structured scenarios. In *CVPR 2011*, pages 3289–3296. IEEE, 2011. doi:10.1109/CVPR.2011.5995386.
- 40 Robert Moskovitch. Multivariate temporal data analysis-a review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(1):e1430, 2022. doi:10.1002/WIDM.1430.
- 41 Robert Moskovitch. Mining temporal data. *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*, pages 469–490, 2023.
- 42 Sindhu Padakandla. A survey of reinforcement learning algorithms for dynamically varying environments. *ACM Computing Surveys (CSUR)*, 54(6):1–25, 2021. doi:10.1145/3459991.
- 43 Carlos F Pfeiffer, Veralia Gabriela Sánchez, and Nils-Olav Skeie. A discrete event oriented framework for a smart house behavior monitor system. In *2016 12th International Conference on Intelligent Environments (IE)*, pages 119–123. IEEE, 2016. doi:10.1109/IE.2016.26.
- 44 Emmanuel Rachelson. *Temporal markov decision problems*. PhD thesis, Citeseer, 2009.
- 45 Emmanuel Rachelson, Gauthier Quesnel, Frédéric Garcia, and Patrick Fabiani. A simulation-based approach for solving generalized semi-markov decision processes. In *ECAI 2008*, pages 583–587. IOS Press, 2008. doi:10.3233/978-1-58603-891-5-583.
- 46 Matthijs TJ Spaan. Partially observable markov decision processes. In *Reinforcement learning: State-of-the-art*, pages 387–414. Springer, 2012. doi:10.1007/978-3-642-27645-3_12.
- 47 Richard S Sutton. The quest for a common model of the intelligent decision maker. *arXiv preprint arXiv:2202.13252*, 2022. arXiv:2202.13252.
- 48 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- 49 Kevin Tang, Li Fei-Fei, and Daphne Koller. Learning latent temporal structure for complex event detection. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1250–1257. IEEE, 2012. doi:10.1109/CVPR.2012.6247808.
- 50 Guy Tennenholtz, Nadav Merlis, Lior Shani, Martin Mladenov, and Craig Boutilier. Reinforcement learning with history dependent dynamic contexts. In *International Conference on Machine Learning*, pages 34011–34053. PMLR, 2023. URL: <https://proceedings.mlr.press/v202/tenneholtz23a.html>.
- 51 Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.
- 52 Aashma Uprety and Danda B Rawat. Reinforcement learning for iot security: A comprehensive survey. *IEEE Internet of Things Journal*, 8(11):8693–8706, 2020. doi:10.1109/JIOT.2020.3040957.
- 53 Antonio Vitale, Alpha Renner, Celine Nauer, Davide Scaramuzza, and Yulia Sandamirskaya. Event-driven vision and control for uavs on a neuromorphic chip. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 103–109. IEEE, 2021. doi:10.1109/ICRA48506.2021.9560881.
- 54 Michalis Vrigkas, Christophoros Nikou, and Ioannis A Kakadiaris. A review of human activity recognition methods. *Frontiers in Robotics and AI*, 2:28, 2015. doi:10.3389/FROBT.2015.00028.
- 55 Cuebong Wong, Erfu Yang, Xiu-Tian Yan, and Dongbing Gu. Autonomous robots for harsh environments: a holistic overview of current solutions and ongoing challenges. *Systems Science & Control Engineering*, 6(1):213–219, 2018.

A Example configuration

This appendix provides an example configuration for the **Open The Chests** environment, illustrating how to set up chests with patterns of multiple events. The first step to configuring the environment is to define a set of event types along with their foreground and background colors. This is done by specifying the sets of all possible event types along with all possible foreground and background colors, as follows:

$$\begin{aligned} \text{Types} &= \{A, B, C, D, E, F, G, H, I, J\} \\ \text{Background Colors} &= \{\text{red, blue, green, orange, pink}\} \\ \text{Foreground Colors} &= \{\text{red, blue, green, orange, pink}\} \end{aligned}$$

The next step would be to define activities using filters and Allen Relations. Below is an example for one chest, showing how to specify the events and their temporal relations.

$$\begin{aligned} R_{\text{activity}}(e_1, \dots, e_8) &= Fa_1(e_1, B, \text{pink}, \text{orange}) \\ &\quad \wedge Fa(e_2, D, \text{red}, \text{green}) \\ &\quad \wedge Fa(e_3, E, \text{orange}, \text{blue}) \\ &\quad \wedge Fa(e_4, G, \text{blue}, \text{pink}) \\ &\quad \wedge Fa(e_5, H, \text{green}, \text{red}) \\ &\quad \wedge Fa(e_6, I, \text{pink}, \text{orange}) \\ &\quad \wedge Fa(e_7, J, \text{green}, \text{blue}) \\ &\quad \wedge Fa(e_8, C, \text{orange}, \text{pink}) \\ &\quad \wedge Fr(e_2, e_3, \{4, 1\}) \\ &\quad \wedge Fr(e_4, e_5, \{5, 2\}) \\ &\quad \wedge Fr(e_7, e_8, \{3, 1\}) \\ &\quad \wedge \mathcal{T}_{\text{during}}(e_1, e_2) \\ &\quad \wedge \mathcal{T}_{\text{after}}(e_2, e_3) \\ &\quad \wedge \mathcal{T}_{\text{metBy}}(e_3, e_4) \\ &\quad \wedge \mathcal{T}_{\text{after}}(e_4, e_5) \\ &\quad \wedge \mathcal{T}_{\text{during}}(e_5, e_6) \\ &\quad \wedge \mathcal{T}_{\text{metBy}}(e_6, e_7) \\ &\quad \wedge \mathcal{T}_{\text{after}}(e_7, e_8) \end{aligned}$$


In this example, $Fa_i(e_i, \text{type}, \text{fg}, \text{bg})$ specifies the type, foreground, and background colors for each event e_i . This means that during generation, only events that satisfy the provided symbols and attributes will be selected. Relations like $\mathcal{T}_{\text{during}}(e_1, e_2)$ and $\mathcal{T}_{\text{metBy}}(e_6, e_7)$ guide the generation of the start and end times for the events e_1 through e_8 , ensuring that these events occur in a sequence that adheres to the specified temporal constraints. Finally relative filters, like $Fr(e_2, e_3, \{4, 1\})$, allow us to define the relative distance between events in relations like “before” and “after,” which imply a gap between the events, with this gap being determined by sampling from a normal distribution characterized by a mean of μ and standard deviation σ . The temporal relations between events in this configuration can also be summarized using a matrix of interval relations, as shown in Table 1.

■ **Table 1** Interval relation matrix representing the temporal relations between events e_1 to e_8 as specified in the configuration.


	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
e_1	=	d						
e_2		=	>					
e_3			=	mi				
e_4				=	>			
e_5					=	d		
e_6						=	mi	
e_7							=	>
e_8								=

The defined pattern is then transformed into a memory-enriched automaton, where each state corresponds to the generation of an event. Relevant temporal data, such as the initialization time of the activity and past generated events, is stored in memory, ensuring the consistent generation of complex event sequences.

Extending the Range of Temporal Specifications of the Run-Time Event Calculus

Periklis Mantenoglou ✉ 

National and Kapodistrian University of Athens, Greece
NCSR “Demokritos”, Athens, Greece

Alexander Artikis ✉ 

University of Piraeus, Greece
NCSR “Demokritos”, Athens, Greece

Abstract

Composite event recognition (CER) frameworks reason over streams of low-level, symbolic events in order to detect instances of spatio-temporal patterns defining high-level, composite activities. The Event Calculus is a temporal, logical formalism that has been used to define composite activities in CER, while RTEC_o is a formal CER framework that detects composite activities based on their Event Calculus definitions. RTEC_o , however, cannot handle every possible set of Event Calculus definitions for composite activities, limiting the range of CER applications supported by RTEC_o . We propose RTEC_β , an extension of RTEC_o that supports arbitrary composite activity specifications in the Event Calculus. We present the syntax, semantics, reasoning algorithms and time complexity of RTEC_β . Our analysis demonstrates that RTEC_β extends the scope of RTEC_o , supporting every possible set of Event Calculus definitions for composite activities, while maintaining the high reasoning efficiency of RTEC_o .

2012 ACM Subject Classification Computing methodologies → Temporal reasoning

Keywords and phrases Event Calculus, temporal pattern matching, composite event recognition

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.6

Supplementary Material *Software*: <https://github.com/aartikis/rtec>
archived at `swh:1:dir:71c0e451729b42a4ad25f6e6fad998fe5d35adba`

Funding This work was supported by the EU-funded CREXDATA project (No 101092749), and partly supported by the University of Piraeus Research Center.

1 Introduction

Composite event recognition (CER) involves the detection of composite activities by reasoning over streams of time-stamped, symbolic events [16, 20]. A CER framework employs an activity specification language, where it is possible to express the spatio-temporal combinations of input events that form each activity of interest in some application domain. In human activity recognition, e.g., we may specify the time periods during which two people are “gathering” using a pattern stating that at least one of the two people is walking towards the other one, while, at the same time, the distance between them is a few meters and they are facing each other. As another example, in the task of monitoring composite maritime activities, we may define “trawling”, i.e., a type of fishing activity that involves several consecutive turns, as a sequence of “change in heading” events.

The literature contains numerous CER frameworks [1, 20], several of which are automata-based [32, 39, 21]. CORE, e.g., is a formal automata-based CER system that has proven to be more efficient than other contemporary automata-based engines [10]. CORE is restricted to unary relations, while the composite activities derived by CORE cannot be used as building blocks in other patterns. In other words, CORE does not support relational and hierarchical



© Periklis Mantenoglou and Alexander Artikis;
licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 6; pp. 6:1–6:14

Leibniz International Proceedings in Informatics



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

composite activity specifications. There are also logic-based CER formalisms [17, 11, 8]. For instance, there are several frameworks supporting fragments of the LARS language [5] that are suitable for CER [6, 4, 18]. MeTeoR is a logic-based CER engine whose language extends DatalogMTL with windowing [37, 38]. The Chronicle Recognition System (CRS) represents composite activities as sets of events that are associated with time constraints [17]. The language of CRS includes several operators, such as sequencing, iteration and negation. These formalisms support relational composite activities, as well as compositional specifications, paving the way for hierarchical definitions. Moreover, logic-based formalisms typically exhibit a formal and declarative semantics, as opposed to automata-based approaches, which do not always come with a clear semantics, making them hard to evaluate and generalise [21].

The Event Calculus is a logic programming formalism for representing and reasoning about events and their effects over time [24]. The Event Calculus may be used as an activity specification language for CER, as it exhibits a formal, declarative semantics, while supporting relational and hierarchical activity specifications that may include background knowledge [27, 20]. Moreover, the Event Calculus includes a built-in representation of inertia, allowing for succinct composite activity patterns, and thus code maintenance. The Event Calculus has been employed in various settings, including mobility assistance [9], reactive and proactive health monitoring [13, 22] and simulations with cognitive agents [34]. The “Macro Event Calculus”, e.g., uses “macro-events” to support composite event operators, such as sequence, disjunction, parallelism and iteration [12]. The “Interval-based Event Calculus” incorporates durative events and supports sequencing, concurrency and negation [28]. jREC is a reactive implementation of the Cached Event Calculus [14] which is optimised for CER [7, 19]. The Run-Time Event Calculus (RTEC) extends the Event Calculus with optimisation techniques for CER, such as windowing, indexing and caching [3]. In order to perform CER with minimal latency, RTEC processes hierarchies of composite activity definitions bottom-up, while caching and reusing the derived instances of composite activities, thus avoiding re-computations. RTEC has proven highly efficient in demanding CER applications, including city transport management [3], maritime situational awareness [30] and commercial fleet management [36], outperforming the state-of-the-art [26, 25, 36].

RTEC does not support every possible composite activity definition that may be expressed in the Event Calculus. In human activity recognition, e.g., there is a need to model composite activities defined in terms of the concept “*movement*(P_1, P_2)”, expressing the relative movement between persons P_1 and P_2 . For instance, “*movement*(P_1, P_2) = *gathering*” expresses that P_1 and P_2 are moving towards one another in order to have a meeting, and “*movement*(P_1, P_2) = *abrupt_gestures*” denotes that, while P_1 and P_2 are talking to each other, one of them is moving his arms abruptly. Furthermore, it may be desirable to express that P_1 and P_2 may be making abrupt gestures to each other only after they have gathered close to one another, i.e., *movement*(P_1, P_2) = *abrupt_gestures* depends on *movement*(P_1, P_2) = *gathering*. RTEC does not support Event Calculus definitions where composite activities characterised by the same underlying concept, such as *movement*(P_1, P_2), depend on each other. To address this issue, we propose an extension of RTEC that supports an arbitrary set of Event Calculus definitions.

Our starting point is RTEC_o, an extension of RTEC that supports Event Calculus definitions with cyclic dependencies, which are often required for CER [26], and propose RTEC_{fl}, an extension of RTEC_o that supports every possible set of composite activity definitions in the Event Calculus. Our contributions are the following. First, we present the semantics of RTEC_{fl}. Second, we present a compiler for RTEC_{fl}, identifying the reasoning algorithm that needs to be used at run-time in order to resolve each condition of a composite

activity definition. Third, we outline the time complexity of $\text{RTEC}_{\mathcal{F}}$, demonstrating that its cost is the same as RTEC_{\circ} , while supporting a wider range of temporal specifications. $\text{RTEC}_{\mathcal{F}}$ and its compiler are publicly available¹.

2 Background

Our starting point is RTEC_{\circ} , i.e., a recent extension of the Run-Time Event Calculus (RTEC) that supports efficient reasoning over temporal specifications with cyclic dependencies [26] (the other extensions of RTEC are orthogonal to this work). We present the syntax, semantics and reasoning algorithms of RTEC_{\circ} . In Section 3, we outline the limitations of RTEC_{\circ} , and, in Section 4, we present an extension of RTEC_{\circ} that supports every set of Event Calculus definitions.

2.1 Syntax & Semantics

The language of RTEC_{\circ} follows the Event Calculus, which is many-sorted, including sorts for representing time, instantaneous events and “fluents”, i.e., properties that may have different values at different points in time. The time model comprises a linear time-line with non-negative integer time-points. $\text{happensAt}(E, T)$ signifies that event E occurs at time-point T . $\text{initiatedAt}(F = V, T)$ (resp. $\text{terminatedAt}(F = V, T)$) expresses that a time period during which a fluent F has the value V continuously is initiated (terminated) at time-point T . $\text{holdsAt}(F = V, T)$ states that F has value V at T , while $\text{holdsFor}(F = V, I)$ expresses that the “fluent-value pair” (FVP) $F = V$ holds continuously in the maximal intervals included in list I .

In CER, happensAt is used to express the input events of the stream, while FVPs express composite activities. A formalisation of the activity specification of a domain in the Event Calculus is called *event description*.

► **Definition 1** (Event Description). *An event description \mathcal{E} is a set of:*

- *ground $\text{happensAt}(E, T)$ facts, expressing a stream of event instances, and*
- *rules with head $\text{initiatedAt}(F = V, T)$ or $\text{terminatedAt}(F = V, T)$, expressing the effects of events on FVP $F = V$.*

► **Definition 2** (Syntax of the Rules in the Event Description). *$\text{initiatedAt}(F = V, T)$ rules have the following syntax:*

$$\begin{aligned} \text{initiatedAt}(F = V, T) \leftarrow & \\ & \text{happensAt}(E_1, T)[[\text{[not]} \text{ happensAt}(E_2, T), \dots, \text{[not]} \text{ happensAt}(E_n, T), \\ & \text{[not]} \text{ holdsAt}(F_1 = V_1, T), \dots, \text{[not]} \text{ holdsAt}(F_k = V_k, T)]]. \end{aligned} \quad (1)$$

The first body literal of an initiatedAt rule is a positive happensAt predicate; this is followed by a possibly empty set, denoted by “[]”, of positive/negative happensAt and holdsAt predicates. “not” expresses negation-by-failure [15], while “[not]” denotes that “not” is optional. All (head and body) predicates are evaluated on the same time-point T . The bodies of $\text{terminatedAt}(F = V, T)$ rules have the same form.

¹ <https://github.com/aartikis/RTEC>

► **Example 3** (Event Description for Human Activity Recognition). In human activity recognition, we apply rules on streams containing symbolic representations of video feeds [2]. In general, such rules are constructed in collaboration with domain experts or learned from data [23]. We use the fluent $interaction(P_1, P_2)$ to express that people P_1 and P_2 are interacting, while the value of $interaction(P_1, P_2)$ denotes the stage of the interaction. The “greeting” stage of $interaction(P_1, P_2)$ denotes that P_1 and P_2 are greeting each other at a distance. Below, we outline a set of rules included in the specification of FVP $interaction(P_1, P_2) = greeting$:

$$\begin{aligned} & \text{initiatedAt}(interaction(P_1, P_2) = greeting, T) \leftarrow \\ & \quad \text{happensAt}(active(P_1), T), \text{ happensAt}(active(P_2), T), \\ & \quad \text{holdsAt}(distance(P_1, P_2) = mid, T), \text{ holdsAt}(orientation(P_1, P_2) = facing, T). \end{aligned} \quad (2)$$

$$\begin{aligned} & \text{terminatedAt}(interaction(P_1, P_2) = greeting, T) \leftarrow \\ & \quad \text{happensAt}(walking(P_1), T), \\ & \quad \text{not holdsAt}(orientation(P_1, P_2) = facing, T). \end{aligned} \quad (3)$$

$$\begin{aligned} & \text{terminatedAt}(interaction(P_1, P_2) = greeting, T) \leftarrow \\ & \quad \text{happensAt}(walking(P_2), T), \\ & \quad \text{not holdsAt}(orientation(P_1, P_2) = facing, T). \end{aligned} \quad (4)$$

According to rule (2), P_1 and P_2 start greeting when both of them are “active”, i.e., moving their arms while in the same position, the distance between them is a few meters, denoted by the value “mid”, and they are facing towards one another. Rules (3)–(4) express that P_1 and P_2 stop greeting when one of them starts walking, while they are not facing each other. The FVPs $distance(P_1, P_2) = mid$ and $orientation(P_1, P_2) = facing$ are defined based on the coordinates and the orientation of the tracked people, which are provided in the input stream.

Moreover, we may use the fluent $movement(P_1, P_2)$ to express the relative movement between people P_1 and P_2 and the value “gathering” of $movement(P_1, P_2)$ to denote that P_1 and P_2 are approaching one another. The specification of FVP $movement(P_1, P_2) = gathering$ includes the following rules:

$$\begin{aligned} & \text{initiatedAt}(movement(P_1, P_2) = gathering, T) \leftarrow \\ & \quad \text{happensAt}(walking(P_1), T), \\ & \quad \text{holdsAt}(distance(P_1, P_2) = mid, T), \text{ holdsAt}(orientation(P_1, P_2) = facing, T). \end{aligned} \quad (5)$$

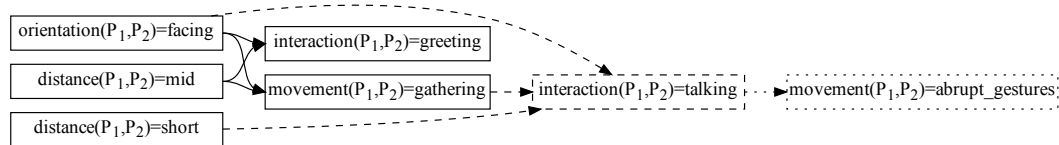
$$\begin{aligned} & \text{initiatedAt}(movement(P_1, P_2) = gathering, T) \leftarrow \\ & \quad \text{happensAt}(walking(P_2), T), \\ & \quad \text{holdsAt}(distance(P_1, P_2) = mid, T), \text{ holdsAt}(orientation(P_1, P_2) = facing, T). \end{aligned} \quad (6)$$

$$\begin{aligned} & \text{terminatedAt}(movement(P_1, P_2) = gathering, T) \leftarrow \\ & \quad \text{happensAt}(active(P_1), T), \text{ not happensAt}(walking(P_2), T). \end{aligned} \quad (7)$$

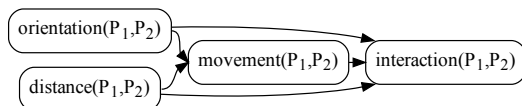
$$\begin{aligned} & \text{terminatedAt}(movement(P_1, P_2) = gathering, T) \leftarrow \\ & \quad \text{happensAt}(active(P_2), T), \text{ not happensAt}(walking(P_1), T). \end{aligned} \quad (8)$$

Rules (5)–(6) state that P_1 and P_2 start gathering when one of them is walking towards the other person, while their distance is a few meters and they are facing each other. Rules (7)–(8) express that P_1 and P_2 stop gathering when one of them is being active, while the other person is not walking.

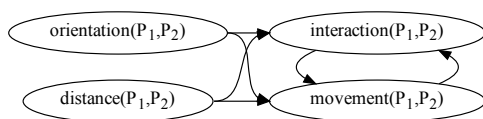
The dependencies among the FVPs in an event description can be expressed in the form of a *dependency graph*.



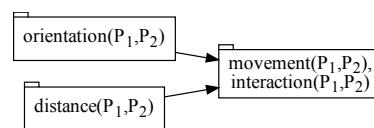
(a) The dependency graph $G_{\mathcal{E}_1}$ of event description \mathcal{E}_1 (continuous lines), the dependency graph $G_{\mathcal{E}_2}$ of event description \mathcal{E}_2 (continuous and dashed lines), and the dependency graph $G_{\mathcal{E}_3}$ of event description \mathcal{E}_3 (all lines). For simplicity, a vertex v_j is displayed as j .



(b) The fluent dependency graph $G_{\mathcal{E}_2}^{fl}$ of \mathcal{E}_2 . The contracted fluent dependency graph $G_{\mathcal{E}_2}^{cdf}$ of \mathcal{E}_2 is the same as $G_{\mathcal{E}_2}^{fl}$.



(c) The fluent dependency graph $G_{\mathcal{E}_3}^{fl}$ of \mathcal{E}_3 .



(d) The contracted fluent dependency graph $G_{\mathcal{E}_3}^{cdf}$ of \mathcal{E}_3 . We display a vertex v_{S_i} of a contracted fluent dependency graph, where S_i is a SCC of the corresponding fluent dependency graph, as the set of fluents whose vertices are in S_i .

■ **Figure 1** Dependency graphs, fluent dependency graphs and contracted fluent dependency graphs. We use distinct shapes for the vertices of each type of graph to aid the presentation.

► **Definition 4** (Dependency Graph). *The dependency graph of an event description is a directed graph $G=(\mathcal{V}, \mathcal{E})$, where:*

1. \mathcal{V} contains one vertex $v_{F=V}$ for each FVP $F=V$.
2. \mathcal{E} contains an edge $(v_{F_j=V_j}, v_{F_i=V_i})$ iff there is an *initiatedAt* or *terminatedAt* rule for $F_i=V_i$ having *holdsAt* $(F_j=V_j, T)$ as one of its conditions.

The vertices and edges of Figure 1a that are drawn with continuous lines, e.g., comprise the dependency graph $G_{\mathcal{E}_1}$ of event description \mathcal{E}_1 , which contains rules (2)–(8) of Example 3.

Based on the dependency graph of an event description, it is possible to define a function *level* that maps the FVPs of the event description to the positive integers. Towards defining an FVP level function, we define the *level* of a vertex in a directed acyclic graph as follows:

► **Definition 5** (Vertex Level). *Given a directed acyclic graph, the level of a vertex v is equal to:*

1. 1, if v has no incoming edges.
2. n , where $n > 1$, if v has at least one incoming edge from a vertex of level $n-1$, and zero or more incoming edges from vertices of levels lower than $n-1$.

A dependency graph may or may not be acyclic. Given an acyclic dependency graph, the level of an FVP $F=V$ is defined as the level of vertex $v_{F=V}$ in the dependency graph. In the acyclic dependency graph of Figure 1a, e.g., $v_{interaction(P_1, P_2)=greeting}$ has level 2, and thus FVP $interaction(P_1, P_2)=greeting$ has level 2. In order to handle cyclic dependency graphs,

we employ the *contracted dependency graph* of an event description, which is, by definition, acyclic. Then, we define the *level of an FVP* based on the level of the corresponding vertex in the contracted dependency graph.

A directed graph becomes acyclic by contracting its strongly connected components (SCC)s into single vertices.

► **Definition 6** (SCC Contracted Graph). *Given a directed graph $G=(\mathcal{V},\mathcal{E})$ and the SCCs S_1, S_2, \dots, S_n of G , the SCC contracted graph $G^{cd}=(\mathcal{V}^{cd}, \mathcal{E}^{cd})$ of G is defined as follows:*

1. $\mathcal{V}^{cd} = \bigcup_{1 \leq i \leq n} \{v_{S_i}\}$.
2. $(v_{S_i}, v_{S_j}) \in \mathcal{E}^{cd}$ iff $\exists v_i, v_j \in \mathcal{V}$, such that $v_i \in S_i, v_j \in S_j, S_i \neq S_j$ and $(v_i, v_j) \in \mathcal{E}$.

► **Definition 7** (Contracted Dependency Graph). *Consider an event description with dependency graph G . The contracted dependency graph of the event description is the SCC contracted graph of G .*

The dependency graph $G_{\mathcal{E}_l}$ in Figure 1a is acyclic, i.e., every SCCs of $G_{\mathcal{E}_l}$ contains one vertex. As a result, the contracted dependency graph $G_{\mathcal{E}_l}^{cd}$ of $G_{\mathcal{E}_l}$ is the same as $G_{\mathcal{E}_l}$.

► **Definition 8** (FVP Level in RTEC_o). *Consider an event description with dependency graph G and contracted dependency graph G^{cd} . The level of an FVP $F=V$, such that vertex $v_{F=V}$ is included in SCC S_i of G , is equal to the level of vertex v_{S_i} in G^{cd} .*

RTEC_o supports event descriptions where FVPs with the same fluent have the same FVP level. For such an event description, a local stratification may be constructed as follows. The first stratum contains all groundings of `happensAt`. The remaining strata are formed by following, in a bottom-up fashion, the levels of FVPs. For each FVP level l without cyclic dependencies, we have one stratum containing the ground predicates for FVPs with level l . For each FVP level l with cyclic dependencies, the ground predicates for FVPs with level l have to be stratified further in terms of their time-stamp. We introduce an additional stratum for each time-point of the window, i.e., the finite portion of the stream currently being processing by RTEC_o.

► **Proposition 9** (Semantics of RTEC_o). *Consider an event description \mathcal{E} where the FVPs with the same fluent have the same FVP level (see Definition 8). \mathcal{E} is a locally stratified logic program [33].*

2.2 Reasoning & Complexity

The key reasoning task of RTEC_o is the computation of `holdsFor($F=V, I$)`, i.e., the list of maximal intervals I during which each FVP $F=V$ of the event description holds continuously. Recall that, in CER, FVPs express the composite activities that we are interested in detecting. RTEC_o computes list I in `holdsFor($F=V, I$)` as follows. First, it computes the initiations of $F=V$ based on the rules of the event description with head `initiatedAt($F=V, T$)`. Second, if there is at least one initiation of $F=V$, then RTEC_o computes the terminations of $F=V$ based on the rules with head `terminatedAt($F=V, T$)`, as well as the rules with head `initiatedAt($F=V', T$)`, where $V' \neq V$. Third, RTEC_o computes the maximal intervals of $F=V$ by matching each initiation T_s of $F=V$ with the first termination T_e of $F=V$ after T_s , ignoring every intermediate initiation between T_s and T_e . `holdsAt($F=V, T$)` may then be evaluated by checking whether T belongs to one of the maximal intervals of FVP $F=V$.

RTEC_o processes FVPs in a bottom-up manner, computing and caching their intervals level-by-level. In order to derive the initiations and the terminations of an FVP $F=V$, we evaluate the `initiatedAt` and `terminatedAt` rules defining $F=V$. The body of such a rule may include

a $\text{holdsAt}(F' = V', T)$ condition (see rule schema (1)), leading to an edge $(v_{F'=V'}, v_{F=V})$ in the dependency graph (see Definition 4). We distinguish two cases for the evaluation of $\text{holdsAt}(F' = V', T)$:

1. Vertices $v_{F'=V'}$ and $v_{F=V}$ are not part of a cycle in the dependency graph. In this case, $v_{F'=V'}$ and $v_{F=V}$ are in different SCCs of the dependency graph and, based on edge $(v_{F'=V'}, v_{F=V})$, $F' = V'$ has a lower level than $F = V$ (see Definition 8). Since RTEC_o processes FVPs in ascending FVP level order, at the time of processing $F = V$, the intervals of $F' = V'$ that are required to compute $\text{holdsAt}(F' = V', T)$ have been derived and cached at a previous step. As a result, $\text{holdsAt}(F' = V', T)$ is resolved by fetching the intervals of $F' = V'$ from the cache and checking whether T belongs to one of those intervals, without the need for re-computation.
2. Vertices $v_{F'=V'}$ and $v_{F=V}$ are part of a cycle in the dependency graph. In this case, $v_{F'=V'}$ and $v_{F=V}$ are in the same SCC of the dependency graph, and thus $F' = V'$ and $F = V$ have the same level (see Definition 8). As a result, RTEC_o may process $F = V$ before $F' = V'$, in which case the intervals of $F' = V'$ are not be present in the cache at the time of processing $F = V$. To address this issue, RTEC_o computes $\text{holdsAt}(F' = V', T)$ using the incremental caching techniques presented in [26].

3 Problem Statement

Towards a more accurate domain specification for human activity recognition, we may extend event description \mathcal{E}_1 of Example 3 with a definition for an FVP expressing that two people are talking.

► **Example 10** (Representing $\text{interaction}(P_1, P_2) = \text{talking}$ (Example 3 cont'd)). After having approached one another, persons P_1 and P_2 may start talking, in which case the value of the $\text{interaction}(P_1, P_2)$ fluent should change from “greeting” to “talking”. The specification of FVP $\text{interaction}(P_1, P_2) = \text{talking}$ includes the following rules:

$$\begin{aligned} &\text{initiatedAt}(\text{interaction}(P_1, P_2) = \text{talking}, T) \leftarrow \\ &\quad \text{happensAt}(\text{active}(P_1), T), \\ &\quad \text{holdsAt}(\text{distance}(P_1, P_2) = \text{short}, T), \text{holdsAt}(\text{orientation}(P_1, P_2) = \text{facing}, T), \\ &\quad \text{not holdsAt}(\text{movement}(P_1, P_2) = \text{gathering}, T). \end{aligned} \quad (9)$$

$$\begin{aligned} &\text{initiatedAt}(\text{interaction}(P_1, P_2) = \text{talking}, T) \leftarrow \\ &\quad \text{happensAt}(\text{active}(P_2), T), \\ &\quad \text{holdsAt}(\text{distance}(P_1, P_2) = \text{short}, T), \text{holdsAt}(\text{orientation}(P_1, P_2) = \text{facing}, T), \\ &\quad \text{not holdsAt}(\text{movement}(P_1, P_2) = \text{gathering}, T). \end{aligned} \quad (10)$$

$$\begin{aligned} &\text{terminatedAt}(\text{interaction}(P_1, P_2) = \text{talking}, T) \leftarrow \\ &\quad \text{happensAt}(\text{inactive}(P_1), T), \text{happensAt}(\text{inactive}(P_2), T). \end{aligned} \quad (11)$$

According to rules (9)–(10), P_1 and P_2 start talking when one of them is being active, while their distance is about one meter, denoted by “short”, they are facing one another and their relative movement is not “gathering”, i.e., P_1 and P_2 are not moving towards one another. Rule (11) denotes that P_1 and P_2 stop talking when neither of them is being active.

A fluent cannot have more than one value at any time; an initiation of an FVP $F = V_1$ implies a termination of FVP $F = V_2$, where $V_1 \neq V_2$. As a result, there are implicit dependencies among FVPs with the same fluent. For instance, in the event description of Example 10, FVPs $\text{interaction}(P_1, P_2) = \text{greeting}$ and $\text{interaction}(P_1, P_2) = \text{talking}$ implicitly depend on each other.

The vertices and edges of Figure 1a that are drawn with continuous or dashed lines comprise the dependency graph $G_{\mathcal{E}_2}$ of event description \mathcal{E}_2 , i.e., the extension of event description \mathcal{E}_1 with rules (9)–(11) of Example 10. FVPs $\text{interaction}(P_1, P_2) = \text{greeting}$ and $\text{movement}(P_1, P_2) = \text{gathering}$ have level 2, while FVP $\text{interaction}(P_1, P_2) = \text{talking}$ has level 3 (see Definition 8).

Event description \mathcal{E}_2 contains FVPs with the same fluent and different levels, which is common in CER specifications. In city transport management, e.g., fluent “ $\text{punctuality}(Vh)$ ” may be used to monitor the punctuality level of a vehicle Vh over time [3]. $\text{punctuality}(Vh) = \text{low}$ may be initiated when Vh leaves a stop earlier than scheduled while $\text{punctuality}(Vh) = \text{mid}$ holds. As another example, in maritime activity monitoring, we may employ the fluent “ $\text{fishing_trip}(Vl)$ ” to survey a fishing trip of a vessel Vl [30]. FVP $\text{fishing_trip}(Vl) = \text{ended}$ may depend on FVP $\text{fishing_trip}(Vl) = \text{returning}$, which expresses the previous stage of the trip. In these cases, FVP $\text{punctuality}(Vh) = \text{low}$ has a higher level than FVP $\text{punctuality} = \text{mid}$, and FVP $\text{fishing_trip}(Vl) = \text{ended}$ has a higher level than FVP $\text{fishing_trip}(Vl) = \text{returning}$ (see Definition 8).

RTEC_o does not support event descriptions, such as \mathcal{E}_2 , where FVPs with the same fluent have different levels. Suppose that FVP $F = V_1$ has level n and FVP $F = V_2$ has level m , where $n < m$, and that RTEC_o is currently processing the FVPs with level n . When processing $F = V_1$, RTEC_o needs to evaluate the rules with head $\text{initiatedAt}(F = V_2, T)$, as the initiation of $F = V_2$ constitute terminations of $F = V_1$. Such a rule may include a body condition referring to an FVP $F' = V'$ with level n' , where $n \leq n' < m$. Since $F' = V'$ has a lower level than $F = V_2$, RTEC_o attempts to evaluate $\text{holdsAt}(F' = V', T)$ by retrieving the intervals of $F' = V'$ from the cache, in order to check whether T belongs to one of them. However, the cache of RTEC_o may not contain the intervals of $F' = V'$ at this time, because $F' = V'$ has level n' and RTEC_o is currently processing the FVPs with level n , where $n \leq n'$, compromising correctness.

In the case of event description \mathcal{E}_2 , when processing $\text{interaction}(P_1, P_2) = \text{greeting}$, RTEC_o evaluates the initiations of $\text{interaction}(P_1, P_2) = \text{talking}$, as they are terminations of $\text{interaction}(P_1, P_2) = \text{greeting}$. According to rules (9)–(10) of event description \mathcal{E}_2 , the initiations of $\text{interaction}(P_1, P_2) = \text{talking}$ depend on FVP $\text{movement}(P_1, P_2) = \text{gathering}$, whose intervals may not present in the cache at the time of processing $\text{interaction}(P_1, P_2) = \text{greeting}$. For this reason, RTEC_o does not support event description \mathcal{E}_2 .

One way to address this issue is to assign to FVP $\text{interaction}(P_1, P_2) = \text{greeting}$ a higher level than the level of FVP $\text{movement}(P_1, P_2) = \text{gathering}$. According to dependency graph $G_{\mathcal{E}_2}$ (see Figure 1a), since there is no FVP that depends on FVP $\text{interaction}(P_1, P_2) = \text{greeting}$, we may increase the level of $\text{interaction}(P_1, P_2) = \text{greeting}$ to 3 without producing an FVP level assignment that compromises the correctness of the bottom-up processing of RTEC_o. In this way, $\text{movement}(P_1, P_2) = \text{gathering}$ is processed before $\text{interaction}(P_1, P_2) = \text{greeting}$, and thus, at the time of processing $\text{interaction}(P_1, P_2) = \text{greeting}$, the maximal intervals of $\text{movement}(P_1, P_2) = \text{gathering}$ are present in the cache of RTEC_o, avoiding the aforementioned error.

However, it is not always possible to circumvent the issues introduced by FVPs with the same fluent and different levels by increasing the level of an FVP. Consider the following example, where we extend event description \mathcal{E}_2 with a definition for an FVP expressing that two people are making abrupt movements while talking.

► **Example 11** (Representing $\text{movement}(P_1, P_2) = \text{abrupt_gestures}$ (Example 10 cont'd)). While people P_1 and P_2 are talking, they may start moving their arms abruptly, possibly indicating that a fight between P_1 and P_2 is about to start. The specification of FVP $\text{movement}(P_1, P_2) = \text{abrupt_gestures}$ includes the following rules:

$$\begin{aligned} &\text{initiatedAt}(\text{movement}(P_1, P_2) = \text{abrupt_gestures}, T) \leftarrow \\ &\quad \text{happensAt}(\text{abrupt}(P_1), T), \\ &\quad \text{holdsAt}(\text{interaction}(P_1, P_2) = \text{talking}, T). \end{aligned} \quad (12)$$

$$\begin{aligned} &\text{initiatedAt}(\text{movement}(P_1, P_2) = \text{abrupt_gestures}, T) \leftarrow \\ &\quad \text{happensAt}(\text{abrupt}(P_2), T), \\ &\quad \text{holdsAt}(\text{interaction}(P_1, P_2) = \text{talking}, T). \end{aligned} \quad (13)$$

$$\begin{aligned} &\text{terminatedAt}(\text{movement}(P_1, P_2) = \text{abrupt_gestures}, T) \leftarrow \\ &\quad \text{happensAt}(\text{active}(P_1), T), \text{ not happensAt}(\text{abrupt}(P_2), T). \end{aligned} \quad (14)$$

$$\begin{aligned} &\text{terminatedAt}(\text{movement}(P_1, P_2) = \text{abrupt_gestures}, T) \leftarrow \\ &\quad \text{happensAt}(\text{active}(P_2), T), \text{ not happensAt}(\text{abrupt}(P_1), T). \end{aligned} \quad (15)$$

Rules (12)–(13) denote that $\text{movement}(P_1, P_2) = \text{abrupt_gestures}$ is initiated when one of the people P_1 and P_2 starts moving abruptly while the two of them are talking. Rules (14)–(15) express that we have a termination of $\text{movement}(P_1, P_2) = \text{abrupt_gestures}$ when one of the two people starts being active while the other one is not moving abruptly.

All the vertices and edges in Figure 1a compose dependency graph $G_{\mathcal{E}_3}$ of event description \mathcal{E}_3 , i.e., the extension of event description \mathcal{E}_2 with rules (12)–(15). According to dependency graph $G_{\mathcal{E}_3}$, FVP $\text{movement}(P_1, P_2) = \text{abrupt_gestures}$ has level 4.

Event description \mathcal{E}_3 contains FVPs with the same fluent and different levels. The FVPs $\text{interaction}(P_1, P_2) = \text{greeting}$ and $\text{interaction}(P_1, P_2) = \text{talking}$ have level 2 and 3, respectively, while FVPs $\text{movement}(P_1, P_2) = \text{gathering}$ and $\text{movement}(P_1, P_2) = \text{abrupt_gestures}$ have level 2 and 4. As a result, RTEC_o does not support event description \mathcal{E}_3 . When processing FVP $\text{movement}(P_1, P_2) = \text{gathering}$, RTEC_o may need to evaluate its terminations, which include the initiations of FVP $\text{movement}(P_1, P_2) = \text{abrupt_gestures}$. According to rules (12)–(13), the initiations of $\text{movement}(P_1, P_2) = \text{abrupt_gestures}$ depend on $\text{interaction}(P_1, P_2) = \text{talking}$, whose intervals are not present in the cache at this time.

In this case, it is not possible to set the level of $\text{movement}(P_1, P_2) = \text{gathering}$ to 4, with the goal of processing $\text{interaction}(P_1, P_2) = \text{talking}$ before $\text{movement}(P_1, P_2) = \text{gathering}$, because there is an edge $(v_{\text{movement}(P_1, P_2) = \text{gathering}}, v_{\text{interaction}(P_1, P_2) = \text{talking}})$ in $G_{\mathcal{E}_3}$, implying that we cannot process $\text{interaction}(P_1, P_2) = \text{talking}$ before $\text{movement}(P_1, P_2) = \text{gathering}$. These FVPs should have the same level. Moreover, $\text{interaction}(P_1, P_2) = \text{greeting}$ depends on $\text{interaction}(P_1, P_2) = \text{talking}$, and vice versa, which means that these FVPs should also have the same level. Therefore, $\text{movement}(P_1, P_2) = \text{gathering}$, $\text{interaction}(P_1, P_2) = \text{greeting}$, $\text{interaction}(P_1, P_2) = \text{talking}$ and $\text{movement}(P_1, P_2) = \text{abrupt_gestures}$ should have the same level, i.e., 2, implying that these FVPs must be processed with incremental caching (see the second case presented in Section 2.2).

4 Proposed Solution

We propose RTEC_f, an extension of RTEC_o that supports event descriptions where the vertices of FVPs with the same fluent may have different levels, such as event descriptions \mathcal{E}_2 and \mathcal{E}_3 . To achieve this, RTEC_f incorporates a new definition for FVP level that takes into account the implicit dependencies between FVPs with the same fluent. We demonstrate that, based on the definition of FVP level in RTEC_f, we may construct a local stratification for every possible event description. Afterwards, we propose a compiler for RTEC_f, identifying the $\text{holdsAt}(F = V, T)$ conditions that need to be resolved with the incremental caching technique proposed in [26], because the intervals of $F = V$ may not be present in the cache

at the time of evaluating $\text{holdsAt}(F = V, T)$. We outline the cost of RTEC_{fl} , showing that it is the same as the cost of RTEC_o . Therefore, RTEC_{fl} extends the range of temporal specifications supported by RTEC_o , while maintaining its high reasoning efficiency.

4.1 Syntax & Semantics

In RTEC_{fl} , all FVPs with the same fluent have the same level. This is achieved by determining FVP level based on the *fluent dependency graph* of the event description, which is defined as follows:

► **Definition 12** (Fluent Dependency Graph). *Consider an event description with dependency graph $G = (\mathcal{V}, \mathcal{E})$. The fluent dependency graph of the event description is a directed graph $G^{fl} = (\mathcal{V}^{fl}, \mathcal{E}^{fl})$, where:*

1. \mathcal{V}^{fl} contains one vertex v_F for each fluent F .
2. \mathcal{E}^{fl} contains an edge (v_{F_1}, v_{F_2}) , where $F_1 \neq F_2$, iff there is an edge $(v_{F_1} = V_1, v_{F_2} = V_2)$ in \mathcal{E} , where V_1 and V_2 are values of fluents F_1 and F_2 , respectively.

Figure 1b, e.g., depicts the fluent dependency graph $G_{\mathcal{E}_2}^{fl}$ of event description \mathcal{E}_2 of Example 10. Vertex $v_{interaction(P_1, P_2)}$ of $G_{\mathcal{E}_2}^{fl}$ corresponds to vertices $v_{interaction(P_1, P_2) = greeting}$ and $v_{interaction(P_1, P_2) = talking}$ of $G_{\mathcal{E}_2}$, inheriting their incoming edges.

The fluent dependency graph $G_{\mathcal{E}_2}^{fl}$ is acyclic. Therefore, we may assign to each FVP $F = V$ of event description \mathcal{E}_2 the level of vertex v_F in the fluent dependency graph $G_{\mathcal{E}_2}^{fl}$, which is derived by following Definition 5. It could be the case, however, that the fluent dependency graph of an event description contains cycles. Figure 1c, e.g., depicts the fluent dependency graph $G_{\mathcal{E}_3}^{fl}$ of event description \mathcal{E}_3 . $G_{\mathcal{E}_3}^{fl}$ includes a cycle, while, according to Definition 5, the level of a vertex is defined only on acyclic graphs. To address this issue, we contract the vertices of the fluent dependency graph that are in the same strongly connected component (SCC), leading to an acyclic graph. We define the *contracted fluent dependency graph* as follows:

► **Definition 13** (Contracted Fluent Dependency Graph). *Consider an event description with fluent dependency graph G^{fl} . The contracted fluent dependency graph G^{cdf} of the event description is the SCC contracted graph of G^{fl} .*

Consider, e.g., the fluent dependency graph $G_{\mathcal{E}_2}^{fl}$ of Figure 1b. $G_{\mathcal{E}_2}^{fl}$ is acyclic, and thus every SCC of $G_{\mathcal{E}_2}^{fl}$ contains one vertex. As a result, the contracted fluent dependency graph $G_{\mathcal{E}_2}^{cdf}$ of $G_{\mathcal{E}_2}^{fl}$ is the same as $G_{\mathcal{E}_2}^{fl}$. As another example, Figure 1d presents the contracted fluent dependency graph $G_{\mathcal{E}_3}^{cdf}$ corresponding to the fluent dependency graph $G_{\mathcal{E}_3}^{fl}$ in Figure 1c, which is produced by contracting vertices $v_{movement(P_1, P_2)}$ and $v_{interaction(P_1, P_2)}$ of $G_{\mathcal{E}_3}^{fl}$, as these vertices are in the same SCC of $G_{\mathcal{E}_3}^{fl}$. Due to this contraction of vertices, $G_{\mathcal{E}_3}^{cdf}$ is acyclic.

We may assign a level to each vertex in a contracted fluent dependency graph by following Definition 5. We define the *level of an FVP* in RTEC_{fl} as follows:

► **Definition 14** (FVP Level in RTEC_{fl}). *Consider an event description with fluent dependency graph G^{fl} and contracted fluent dependency graph G^{cdf} . The level of an FVP $F = V$, such that vertex v_F is included in SCC S_i of G^{fl} , is equal to the level of vertex v_{S_i} of G^{cdf} .*

Based on Definition 14, FVPs with the same fluent have the same level. In the case of event description \mathcal{E}_2 , e.g., where the contracted fluent dependency graph $G_{\mathcal{E}_2}^{cdf}$ of \mathcal{E}_2 matches with the fluent dependency graph in Figure 1b, FVPs $interaction(P_1, P_2) = greeting$ and

Algorithm 1 *compile*(\mathcal{E}).

```

1:  $G_{\mathcal{E}}^{cdf}$   $\leftarrow$  construct_contracted_fluent_dependency_graph( $\mathcal{E}$ )
2: level  $\leftarrow$  compute_fvp_level( $G_{\mathcal{E}}^{cdf}$ )
3: for each rule  $r$  in  $\mathcal{E}$  do
4:    $F = V \leftarrow$  get_fvp_in_head( $r$ )
5:   for each condition “[not] holdsAt( $F' = V', T$ )” in the body of  $r$  do  $\triangleright$  not is optional.
6:     if level[ $F' = V'$ ] = level[ $F = V$ ] then
7:       replace “[not] holdsAt( $F' = V', T$ )” with “[not] holdsAtCyclic( $F' = V', T$ )” in  $r$ 
8: return  $\mathcal{E}$ 

```

interaction(P_1, P_2) = *talking* have level 3 because the level of vertex $v_{interaction(P_1, P_2)}$ in $G_{\mathcal{E}_2}^{cdf}$ is 3. In the case of event description \mathcal{E}_3 , the vertex of the contracted fluent dependency graph corresponding to fluents *movement*(P_1, P_2) and *interaction*(P_1, P_2) has level 2 (see Figure 1d). Thus, FVPs *interaction*(P_1, P_2) = *greeting*, *movement*(P_1, P_2) = *gathering*, *interaction*(P_1, P_2) = *talking* and *movement*(P_1, P_2) = *abrupt_gestures* have level 2.

We can devise a local stratification of an event description by following bottom-up the levels of FVPs, as specified in Definition 14. For each level with cyclic dependencies, we introduce an additional stratum per time-point, following an ascending temporal order.

► **Proposition 15** (Semantics of $RTEC_{\mathcal{F}}$). *An event description is a locally stratified logic program.*

According to Proposition 15, $RTEC_{\mathcal{F}}$ supports every event description \mathcal{E} that follows Definition 1. If the dependency graph of \mathcal{E} contains FVPs with the same fluent whose vertices are in different levels of the graph, then these FVPs are assigned the same level, following the definition of FVP level in $RTEC_{\mathcal{F}}$ (see Definition 14), avoiding the issues described in Section 3.

4.2 Compiler

We developed a compiler that assigns a level to each FVP of an input event description \mathcal{E} and marks the holdsAt body conditions of the rules in \mathcal{E} that must be evaluated with incremental caching, in order to guarantee correct reasoning. The compilation is performed before the commencement of run-time reasoning, in a process transparent to the event description developer. Algorithm 1 outlines the compilation steps. First, we derive the levels of FVPs by following Definitions 13 and 14. We construct the contracted fluent dependency graph $G_{\mathcal{E}}^{cdf}$ of \mathcal{E} (line 1 of Algorithm 1). Then, we assign a level to each FVP in \mathcal{E} based on the level of the corresponding vertex of $G_{\mathcal{E}}^{cdf}$ (line 2). In order to identify the holdsAt conditions that need to be evaluated with incremental caching, the compiler works as follows. For each holdsAt($F' = V', T$) or “not holdsAt($F' = V', T$)” condition in the body of a rule in \mathcal{E} , the compiler checks whether the level of FVP $F' = V'$ is equal to the level of the FVP in the head of the rule (lines 3–6). If this is the case, then we translate condition holdsAt($F' = V', T$) (resp. “not holdsAt($F' = V', T$)”) into holdsAtCyclic($F' = V', T$) (resp. “not holdsAtCyclic($F' = V', T$)”) (line 7). At run-time, $RTEC_{\mathcal{F}}$ evaluates the conditions with holdsAtCyclic using incremental caching (recall the second case presented in Section 2.2) and the conditions with holdsAt using the interval retrieval operation (see the first case of Section 2.2). A further discussion on run-time reasoning is presented in the section that follows.

We tested the compiler of $\text{RTEC}_{\mathcal{F}}$ on event descriptions from various CER applications, including human activity recognition [2], city transport management [3] and maritime situational awareness [29, 30]. Moreover, we have used our compiler in applications that involve the monitoring of the normative positions of agents in multi-agent systems, such as e-commerce [35] and voting protocols [31]. In all cases, the compilation time amounted to a few milliseconds, and thus we do not show these times here. The compiler is available with the code of $\text{RTEC}_{\mathcal{F}}$ ¹.

4.3 Reasoning & Complexity

$\text{RTEC}_{\mathcal{F}}$ follows RTEC_{\circ} and processes FVPs in ascending FVP level order. When processing a rule that includes a $\text{holdsAtCyclic}(F' = V', T)$ condition, $\text{RTEC}_{\mathcal{F}}$ computes the changes in the value of F' between T_{leq} and T , where T_{leq} is the last time-point before T where the truth value of $\text{holdsAt}(F' = V', T_{\text{leq}})$ has been evaluated and cached. In the worst-case, the cost of this process is $\mathcal{O}(\omega k)$, where ω is the size of the window and k is the cost of computing whether an FVP is initiated or terminated at a given time-point (see [3] for an estimation of k). This is the same incremental caching technique as the one used in RTEC_{\circ} , thus yielding the same cost [26]. In the case of a $\text{holdsAt}(F' = V', T)$ condition, $\text{RTEC}_{\mathcal{F}}$ retrieves the maximal intervals of $F' = V'$ from its cache and checks whether T belongs to one of the retrieved intervals. Since the cached intervals are temporally sorted, this is achieved with a binary search, while the number of cached intervals of $F' = V'$ is bounded by ω . Therefore, the cost of an interval retrieval operation in $\text{RTEC}_{\mathcal{F}}$ is $\mathcal{O}(\log(\omega))$, which is the same as the cost of this operation in RTEC_{\circ} . As a result, $\text{RTEC}_{\mathcal{F}}$ yields the same worst-case time complexity as RTEC_{\circ} , while supporting a wider range of temporal specifications. By following Definition 14 for FVP level, RTEC_{\circ} reasons with incremental caching only when it is necessary, i.e., only when the required intervals may not be present in the cache.

5 Summary and Future Work

We proposed $\text{RTEC}_{\mathcal{F}}$, an extension of RTEC_{\circ} , which detects composite activities based on their Event Calculus definitions, in order to support every possible set of such definitions. We described the syntax and semantics of $\text{RTEC}_{\mathcal{F}}$, demonstrating that activity specifications in $\text{RTEC}_{\mathcal{F}}$ are locally stratified logic programs. Afterwards, we proposed a compiler for $\text{RTEC}_{\mathcal{F}}$, identifying the conditions of activity definitions that may be evaluated with an efficient cache operation, without sacrificing correctness, with the goal of improving reasoning efficiency at run-time. We outlined the worst-case time complexity of $\text{RTEC}_{\mathcal{F}}$, showing that it yields the same cost as RTEC_{\circ} . As a result, $\text{RTEC}_{\mathcal{F}}$ supports a wider range of temporal specifications than RTEC_{\circ} , while maintaining its high reasoning efficiency. The code of $\text{RTEC}_{\mathcal{F}}$ is publicly available¹.

In the future, we aim to compare $\text{RTEC}_{\mathcal{F}}$ with automata-based activity recognition frameworks, such as [10, 40].

References

- 1 Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. Probabilistic complex event recognition: A survey. *Commun. ACM*, 50(5):71:1–71:31, 2017. doi:10.1145/3117809.
- 2 Alexander Artikis, Marek J. Sergot, and Georgios Paliouras. A logic programming approach to activity recognition. In *EIMM Workshop in MM*, pages 3–8, 2010. doi:10.1145/1877937.1877941.

- 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. doi:10.1109/TKDE.2014.2356476.
- 4 Hamid R. Bazoobandi, Harald Beck, and Jacopo Urbani. Expressive stream reasoning with laser. In *ISWC*, volume 10587, pages 87–103, 2017. doi:10.1007/978-3-319-68288-4_6.
- 5 Harald Beck, Minh Dao-Tran, and Thomas Eiter. LARS: A logic-based framework for analytic reasoning over streams. *Artif. Intell.*, 261:16–70, 2018. doi:10.1016/J.ARTINT.2018.04.003.
- 6 Harald Beck, Thomas Eiter, and Christian Folie. Ticker: A system for incremental aspbased stream reasoning. *Theory Pract. Log. Program.*, 17(5-6):744–763, 2017. doi:10.1017/S1471068417000370.
- 7 Stefano Bragaglia, Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. Reactive event calculus for monitoring global computing applications. In *Logic Programs, Norms and Action - Essays in Honor of Marek J. Sergot on the Occasion of His 60th Birthday*, volume 7360, pages 123–146, 2012. doi:10.1007/978-3-642-29414-3_8.
- 8 Sebastian Brandt, Elem Güzel Kalayci, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyashev. Querying log data with metric temporal logic. *J. Artif. Intell. Res.*, 62:829–877, 2018. doi:10.1613/JAIR.1.11229.
- 9 Stefano Bromuri, Visara Urovi, and Kostas Stathis. icampus: A connected campus in the ambient event calculus. *Int. J. Ambient Comput. Intell.*, 2(1):59–65, 2010. doi:10.4018/JACI.2010010105.
- 10 Marco Bucchi, Alejandro Grez, Andrés Quintana, Cristian Riveros, and Stijn Vansummeren. CORE: a complex event recognition engine. *Proc. VLDB Endow.*, 15(9):1951–1964, 2022. doi:10.14778/3538598.3538615.
- 11 Francesco Calimeri, Marco Manna, Elena Mastria, Maria Concetta Morelli, Simona Perri, and Jessica Zangari. I-dlv-sr: A stream reasoning system based on I-DLV. *Theory Pract. Log. Program.*, 21(5):610–628, 2021. doi:10.1017/S147106842100034X.
- 12 Iliano Cervesato and Angelo Montanari. A calculus of macro-events: Progress report. In *TIME*, pages 47–58, 2000. doi:10.1109/TIME.2000.856584.
- 13 Hervé Chaudet. Extending the event calculus for tracking epidemic spread. *Artif. Intell. Medicine*, 38(2):137–156, 2006. doi:10.1016/J.ARTMED.2005.06.001.
- 14 L. Chittaro and A. Montanari. Efficient temporal reasoning in the cached event calculus. *Comput. Intell.*, 12(3):359–382, 1996. doi:10.1111/J.1467-8640.1996.TB00267.X.
- 15 Keith L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322. Plenum Press, 1977. doi:10.1007/978-1-4684-3384-5_11.
- 16 Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3), 2012. doi:10.1145/2187671.2187677.
- 17 C. Dousson and P. Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchisation. In *IJCAI*, pages 324–329, 2007.
- 18 Thomas Eiter, Paul Ogris, and Konstantin Schekotihin. A distributed approach to LARS stream reasoning (system paper). *Theory Pract. Log. Program.*, 19(5-6):974–989, 2019. doi:10.1017/S1471068419000309.
- 19 Nicola Falcionelli, Paolo Sernani, Albert Brugués de la Torre, Dagmawi Neway Mekuria, Davide Calvaresi, Michael Schumacher, Aldo Franco Dragoni, and Stefano Bromuri. Indexing the event calculus: Towards practical human-readable personal health systems. *Artif. Intell. Medicine*, 96:154–166, 2019. doi:10.1016/J.ARTMED.2018.10.003.
- 20 Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020. doi:10.1007/S00778-019-00557-w.
- 21 Alejandro Grez, Cristian Riveros, Martín Ugarte, and Stijn Vansummeren. A formal framework for complex event recognition. *ACM Trans. Database Syst.*, 46(4):16:1–16:49, 2021. doi:10.1145/3485463.

- 22 Özgür Kafali, Alfonso E. Romero, and Kostas Stathis. Agent-oriented activity recognition in the event calculus: An application for diabetic patients. *Comput. Intell.*, 33(4):899–925, 2017. doi:10.1111/COIN.12121.
- 23 Nikos Katzouris, Georgios Paliouras, and Alexander Artikis. Online learning probabilistic event calculus theories in answer set programming. *Theory Pract. Log. Program.*, 23(2):362–386, 2023. doi:10.1017/S1471068421000107.
- 24 Robert A. Kowalski and Marek J. Sergot. A logic-based calculus of events. *New Gener. Comput.*, 4(1):67–95, 1986. doi:10.1007/BF03037383.
- 25 Periklis Mantenoglou, Dimitrios Kelesis, and Alexander Artikis. Complex event recognition with allen relations. In *KR*, pages 502–511, 2023. doi:10.24963/KR.2023/49.
- 26 Periklis Mantenoglou, Manolis Pitsikalis, and Alexander Artikis. Stream reasoning with cycles. In *KR*, pages 544–553, 2022.
- 27 Adrian Paschke. Eca-ruleml: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics. *CoRR*, abs/cs/0610167, 2006.
- 28 Adrian Paschke and Martin Bichler. Knowledge representation concepts for automated SLA management. *Decis. Support Syst.*, 46(1):187–205, 2008. doi:10.1016/J.DSS.2008.06.008.
- 29 Kostas Patroumpas, Alexander Artikis, Nikos Katzouris, Marios Vodas, Yannis Theodoridis, and Nikos Pelekis. Event recognition for maritime surveillance. In *EDBT*, pages 629–640, 2015. doi:10.5441/002/EDBT.2015.63.
- 30 Manolis Pitsikalis, Alexander Artikis, Richard Dreo, Cyril Ray, Elena Camossi, and Anne-Laure Joussemme. Composite event recognition for maritime monitoring. In *DEBS*, pages 163–174, 2019. doi:10.1145/3328905.3329762.
- 31 J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Voting in multi-agent systems. *Comput. J.*, 49(2):156–170, 2006. doi:10.1093/COMJNL/BXH164.
- 32 Olga Poppe, Chuan Lei, Elke A. Rundensteiner, and David Maier. Event trend aggregation under rich event matching semantics. In *SIGMOD*, pages 555–572, 2019. doi:10.1145/3299869.3319862.
- 33 Teodor C. Przymusiński. On the declarative semantics of deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988. doi:10.1016/B978-0-934613-40-8.50009-9.
- 34 Nausheen Saba Shahid, Dan O’Keeffe, and Kostas Stathis. A knowledge representation framework for evolutionary simulations with cognitive agents. In *ICTAI*, pages 361–368. IEEE, 2023. doi:10.1109/ICTAI59109.2023.00059.
- 35 M. Sirbu. Credits and debits on the Internet. *IEEE Spectrum*, 34(2):23–29, 1997.
- 36 Efthimis Tsilonis, Alexander Artikis, and Georgios Paliouras. Incremental event calculus for run-time reasoning. *J. Artif. Intell. Res.*, 73:967–1023, 2022. doi:10.1613/JAIR.1.12695.
- 37 Przemyslaw Andrzej Walega, Mark Kaminski, and Bernardo Cuenca Grau. Reasoning over streaming data in metric temporal datalog. In *AAAI*, pages 3092–3099, 2019. doi:10.1609/AAAI.V33I01.33013092.
- 38 Przemyslaw Andrzej Walega, Mark Kaminski, Dingmin Wang, and Bernardo Cuenca Grau. Stream reasoning with datalogmtl. *J. Web Semant.*, 76:100776, 2023. doi:10.1016/J.WEBSEM.2023.100776.
- 39 Bo Zhao, Han van der Aa, Thanh Tam Nguyen, Quoc Viet Hung Nguyen, and Matthias Weidlich. EIRES: efficient integration of remote data in event stream processing. In *SIGMOD*, pages 2128–2141, 2021. doi:10.1145/3448016.3457304.
- 40 Bartosz Zielinski. Explanatory denotational semantics for complex event patterns. *Formal Aspects Comput.*, 35(4):23:1–23:37, 2023. doi:10.1145/3608486.

Fitting's Style Many-Valued Interval Temporal Logic Tableau System: Theory and Implementation

Guillermo Badia   

School of Historical and Philosophical Inquiry, University of Queensland, Brisbane, Australia

Carles Noguera   

Department of Information Engineering and Mathematics, University of Siena, Italy

Alberto Paparella   

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

Guido Sciavicco   

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

Ionel Eduard Stan   

Faculty of Engineering, Free University of Bozen-Bolzano, Italy

Abstract

Many-valued logics, often referred to as fuzzy logics, are a fundamental tool for reasoning about uncertainty, and are based on truth value algebras that generalize the Boolean one; the same logic can be interpreted on algebras from different varieties, for different purposes and pose different challenges. Although temporal many-valued logics, that is, the many-valued counterpart of popular temporal logics, have received little attention in the literature, the many-valued generalization of Halpern and Shoham's interval temporal logic has been recently introduced and studied, and a sound and complete tableau system for it has been presented for the case in which it is interpreted on some finite Heyting algebra. In this paper, we take a step further in this inquiry by exploring a tableau system for Halpern and Shoham's interval temporal logic interpreted on some finite FL_{ew} -algebra, therefore generalizing the Heyting case, and by providing its open-source implementation.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases Interval temporal logic, many-valued logic, tableau system

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.7

Funding This research has been partially funded by the Italian Ministry of University and Research through PNRR - M4C2 - Investimento 1.3 (Decreto Direttoriale MUR n. 341 del 15/03/2022), Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 8 "Pervasive AI", funded by the European Union under the "NextGeneration EU programme". Moreover, this research has also been founded by the FIRD project *Modal Geometric Symbolic Learning* (University of Ferrara), and the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (INDAM-GNCS) project *Symbolic and Numerical Analysis of Cyberphysical Systems*, CUP code E53C22001930001. Guido Sciavicco and Ionel Eduard Stan are GNCS-INdAM members; Carles Noguera is a GNSAGA-INdAM member.

1 Introduction

Many-valued logics, also called *fuzzy logics*, extend beyond the binary truth values of classical logic by allowing formula truth to be graded. These logics are defined over algebraic systems, or *algebras*. Unlike classical logic, which is grounded in the Boolean two-valued algebra, many-valued logics involve a more complex algebraic framework, thereby supporting a richer set of truth values. Such generalizations have led to the development of a sophisticated algebraic taxonomy, accommodating different types of underlying domains and the properties



© Guillermo Badia, Carles Noguera, Alberto Paparella, Guido Sciavicco, and Ionel Eduard Stan; licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 7; pp. 7:1–7:16

Leibniz International Proceedings in Informatics



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

of algebraic operators, which in turn influence the interpretation of logical connectives [10]. Noteworthy examples of varieties of algebras on which many-valued logics have been defined include *Gödel* algebras (G) [2], MV-algebras [8] on which *Lukasiewicz* logic is based [31], *product* algebras (II) [22], and *Heyting* algebras (H) on which *intuitionistic* logic is based [15].

In practical applications, the supporting algebra is often assumed to be linearly ordered, that is, a *chain*. Prominent examples falling under this assumption are algebras defined on the interval $[0, 1]$ in \mathbb{R} ; in this case, an algebra is termed *standard*. When an algebra only has a finite number of elements, instead, is known as a *finite* algebra; finite algebras may or may not be chains. Choosing among these types depends on the intended applications and the availability of reasoning tools.

Many-valued logics are crucial in several mathematics and computer science areas, particularly within artificial intelligence and symbolic machine learning. These logics are predominantly employed in rule-based classifier learning [24], enhancing the flexibility and expressive power of such systems. Less frequently, many-valued logics are used to refine decision-tree classifiers by supporting more granular decision-making processes [9]. In the context of subsymbolic machine learning, many-valued logics find their use in fuzzy neural networks, which aid in managing the inherent uncertainty in data, thereby improving network adaptability and performance [7].

In terms of temporal logics, only a few attempts have been made to study point-based temporal languages in the many-valued case. These include early contributions, such as [14, 26, 33], which are characterized by the fact that, in the proposed languages, only the propositional side of formulas is fuzzified, and more recent proposals, such as [18], in which the authors provide a generalization of LTL that allows one to express uncertainty in both atomic propositions and temporal relations. In all such cases, however, standard underlying algebras are assumed.

While many-valued logics based on standard algebras are relatively common in practical applications, the practical necessity for an infinite set of truth values is often debatable; for example, in the context of machine learning, it is immediate to observe that datasets contain only a finite number of distinct values, and therefore give rise to a finite number of distinct situations. At the same time, the conventional reliance on chain algebras can sometimes restrict modeling capabilities, for example disallowing the possibility of reasoning about different, and incomparable, experts' viewpoints. A more general approach to many-valued modal (and therefore also temporal) logics is that of Fitting [16]. Fitting's formalization is applicable to any modal (temporal, spatial, and so on) logic and any algebra. In particular, in the case of interval temporal logic, this has been done in [11, 12, 13], where a many-valued extension of Halpern and Shoham's interval temporal logic (HS [23]) over a Heyting algebra (the resulting logic was denoted FHS, that is, *fuzzy* HS) has been introduced and studied, along with a tableau system for it in the case of finite algebras.

In this paper, we explore a further generalization of HS based on FL_{ew} -algebras [10]. An FL_{ew} -algebra is defined over a bounded commutative integral residuated lattice and naturally generalizes several common frameworks, including G, MV, II, and H. In order to uniform the terminology, we shall use the term *Many-Valued Halpern and Shoham's interval temporal logic*, that is, MVHS. We extend Fitting's tableau system to deal with MVHS over some finite FL_{ew} -algebra, we provide a working implementation for it as a part of our open-source reasoning and learning framework, and we test it to assess its practical usefulness. It is important to notice that extending the tableau rules from H to FL_{ew} does not require much work; on the other hand, implementing the resulting system is not trivial.

■ **Table 1** Allen’s interval relations.

relation	definition	example
after	$[x, y]R_A[w, z] \Leftrightarrow y = w$	
later	$[x, y]R_L[w, z] \Leftrightarrow y < w$	
begins	$[x, y]R_B[w, z] \Leftrightarrow x = w \wedge z < y$	
ends	$[x, y]R_E[w, z] \Leftrightarrow y = z \wedge x < w$	
during	$[x, y]R_D[w, z] \Leftrightarrow x < w \wedge z < y$	
overlaps	$[x, y]R_O[w, z] \Leftrightarrow x < w < y < z$	

This paper is organized as follows. In Section 2 we recall some basic notions of both the crisp version of the interval temporal logic HS and the most common algebras that have some role in the literature of many-valued logic. In Section 3 we present MVHS, its syntax and semantics, several application examples, and our adaptation of Fitting’s tableau system for it. Finally, in Section 4 we present our implementation and the results of several systematic tests, before concluding.

2 Preliminaries

Halpern and Shoham’s Interval Temporal Logic. Several different interval temporal logics have been proposed in the recent literature [21], mostly in the point-based setting. In the interval-based setting, however, *Halpern and Shoham’s Modal Logic for Time Intervals* (HS) [23] can be considered the formalism that has received the most attention.

Let $\mathbb{D} = \langle D, < \rangle$ be a (strict) linear order with *domain* D ; in the following, we shall use D and \mathbb{D} interchangeably. A *strict interval* over \mathbb{D} is an ordered pair $[x, y]$, where $x, y \in \mathbb{D}$ and $x < y$. If we exclude the identity relation, there are 12 different binary ordering relations between two strict intervals on a linear order, often called *Allen’s interval relations* [1]: the six relations R_A (*adjacent to*), R_L (*later than*), R_B (*begins*), R_E (*ends*), R_D (*during*), and R_O (*overlaps*), depicted in Tab. 1, and their *inverses*, that is, $R_{\bar{X}} = (R_X)^{-1}$, for each $X \in \{A, L, B, E, D, O\}$. We interpret interval structures as Kripke structures, with Allen’s relations playing the role of accessibility relations. Thus, we associate an *existential modality* $\langle X \rangle$ with each one of Allen’s relations R_X . Moreover, for each $X \in \{A, L, B, E, D, O\}$, the *transpose* of modality $\langle X \rangle$ is the modality $\langle \bar{X} \rangle$ corresponding to the inverse relation $R_{\bar{X}}$ of R_X . Now, let $\mathcal{X} = \{A, \bar{A}, L, \bar{L}, B, \bar{B}, E, \bar{E}, D, \bar{D}, O, \bar{O}\}$; well-formed HS formulas are built from a set of *propositional letters* \mathcal{P} , the classical connectives \vee and \neg , and a modality for each Allen’s interval relation, as follows:

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

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

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

$$\begin{aligned} M, [x, y] \Vdash p & \quad \text{if and only if} \quad [x, y] \in V(p), \text{ for each } p \in \mathcal{AP}, \\ M, [x, y] \Vdash \neg\psi & \quad \text{if and only if} \quad M, [x, y] \not\Vdash \psi, \\ M, [x, y] \Vdash \psi_1 \vee \psi_2 & \quad \text{if and only if} \quad M, [x, y] \Vdash \psi_1 \text{ or } M, [x, y] \Vdash \psi_2, \\ M, [x, y] \Vdash \langle X \rangle \psi & \quad \text{if and only if} \quad \text{there is } [w, z] \text{ s.t. } [x, y] R_X [w, z] \text{ and } M, [w, z] \Vdash \psi, \end{aligned}$$

where $X \in \mathcal{X}$.

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

FL_{ew}-algebras. An algebraic structure

$$\langle A, \cap, \cup, \cdot, +, \leftrightarrow, 0, 1 \rangle,$$

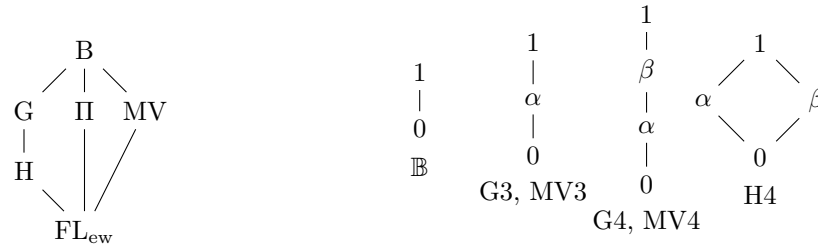
where we define a binary relation \preceq as $a \preceq b$ if and only if $a \cap b = a$, is called an FL_{ew}-*algebra* if

$$\begin{aligned} & \langle A, \cap, \cup, 0, 1 \rangle \text{ is a bounded lattice with upper bound } 1 \text{ and lower bound } 0 \text{ (and hence} \\ & \preceq \text{ is a partial order)} \\ & \langle A, \cdot, 1 \rangle \text{ and } \langle A, +, 0 \rangle \text{ are commutative monoids} \\ & \cdot \text{ and } + \text{ are monotone w.r.t. } \preceq, \text{ i.e., if } \gamma \preceq \alpha, \delta \preceq \beta, \text{ then } \gamma \cdot \delta \preceq \alpha \cdot \beta, \gamma + \delta \preceq \alpha + \beta \\ & \text{for each } a, b, c \in A, a \cdot b \preceq c \text{ if and only if } a \preceq b \leftrightarrow c \end{aligned}$$

We refer, as it is customary, to \cap as *meet*, \cup as *join*, \leftrightarrow as *implication*, \cdot as *t-norm*, and $+$ as *t-co-norm*. If the lattice order is complete (that is, each subset has infimum and supremum), we call the structure a *complete FL_{ew}-algebra*. In this case, given a subset $A' \subseteq A$, we denote the infimum and the supremum of A' respectively as $\bigwedge A$ and $\bigvee A$. An FL_{ew}-algebra is termed *linearly ordered* (or *chain*) if its lattice order is total, *standard* if its lattice reduct is the real unit interval $[0, 1]$, and *finite* if its lattice comprises only a finite number of elements. A many-valued logic, which generalizes Boolean logic, may derive its truth values from an FL_{ew}-algebra, interpreting logical conjunctions, disjunctions, and implications as the t-norm, t-co-norm, and implication operations of the algebra, respectively.

In the context of many-valued propositional logics, the most common and well-known examples include *Gödel algebra* (G) [2], *MV algebra* (MV) [31], and *product (II) algebra* [22].

In their typical formulation, they are based on the interval $[0, 1]$ in \mathbb{R} , and are, therefore, *chains*. However, they can all be considered special cases of FL_{ew}-chains. In particular, Gödel algebra, interprets conjunction as the minimum; so, for example, if *high fever* has truth value $1/3$ and *pain* has truth value $2/3$, then the sentence *high fever and pain* evaluates to $1/3$, illustrating Gödel's conservative nature: if we know something about two facts, we know the minimum of them about their conjunction. In the case of MV, the conjunction $\alpha \cdot \beta$ is computed as $\max\{0, \alpha + \beta - 1\}$, implying that, in the same scenario as before, the sentence would evaluate to 0, which highlights MV pessimistic approach: if we do not know enough about two facts, we do not know anything about their conjunction. Conversely, conjunction



■ **Figure 1** *On the left*, a taxonomy of well-known many-valued algebras. *On the right*, some examples of algebra lattices that will be used in our experiments. Note how G3 and MV3 (resp. G4 and MV4) differ because of the t-norm, but share the same lattice structure.

is the real product in product algebra, evaluating the same sentence as $2/9$, thus reflecting a probabilistic interpretation of independent events. Similarly, a Heyting algebra (H) [15, 17] is an FL_{ew} -algebra in which \cdot is defined as \cap and $+$ as \cup , and the underlying lattice is not necessarily a chain. For instance, if *high fever* is evaluated as α by some expert and *pain* as β some other expert, and α and β are not necessarily comparable to each other, then the sentence *high fever and pain* is evaluated as γ , where γ is the maximum value less than, and comparable with, both α and β . FL_{ew} -algebras thus generalize all four prominent cases.

Fig. 1 (left-hand side) provides a partial taxonomy of algebra varieties corresponding to various many-valued logics, with the variety corresponding to classical logic (the one generated by the two-valued Boolean algebra) denoted by B. In this diagram, a higher element is less general than a lower one. So, Boolean algebras are generalized by algebras in the Gödel variety, the product variety, and the MV variety, whereas the last three are incomparable to each other. Similarly, a Heyting algebra is a generalization of a Gödel algebra (precisely, the class of Gödel algebras is the variety generated by linearly ordered Heyting algebras), and, finally, the variety of FL_{ew} -algebras is more general than all of the above. In Fig. 1 (right-hand side), we give examples of specific algebra lattices from these varieties also used later in our experiments. For example, G3 is the Gödel algebra with 3 elements; it is linearly ordered, and meet/join are defined following the general specification for the Gödel variety. As we shall see, we can define a many-valued version of interval temporal logic by *plugging in* any specific algebra within FL_{ew} ; similarly, our tableau system works in any such case, for as long as the underlying lattice is finite.

3 Many-Valued Interval Temporal Logic

Many-valued linear orderings. In [11, 12, 13], a many-valued generalization of HS based on Heyting algebras has been proposed and studied. As we shall see, its further generalization to the case of FL_{ew} -algebras is syntactically and semantically quite similar; however, decoupling conjunction (resp., disjunction) and t-norm (resp., t-co-norm) increases the flexibility and adaptability of the language to practical situations.

We start off by defining the many-valued generalization of a linearly ordered set.

► **Definition 1.** *Let $\mathbf{A} = \langle A, \cap, \cup, \cdot, +, 0, 1 \rangle$ a complete FL_{ew} -algebra. A many-valued linear order is a structure of the type*

$$\tilde{\mathbb{D}} = \langle D, \tilde{<}, \tilde{=} \rangle,$$

where D is a domain (again, we identify D with $\tilde{\mathbb{D}}$) enriched with two functions $\tilde{<}, \tilde{=} : D \times D \rightarrow A$, for which the following conditions apply for every x, y , and z :

$$\begin{aligned}
& \cong(x, y) = 1 \text{ iff } x = y, \\
& \cong(x, y) = \cong(y, x), \\
& \tilde{<}(x, x) = 0, \\
& \tilde{<}(x, z) \succeq \tilde{<}(x, y) \cdot \tilde{<}(y, z), \\
& \text{if } \tilde{<}(x, y) \succ 0 \text{ and } \tilde{<}(y, z) \succ 0, \text{ then } \tilde{<}(x, z) \succ 0, \\
& \text{if } \tilde{<}(x, y) = 0 \text{ and } \tilde{<}(y, x) = 0, \text{ then } \cong(x, y) = 1, \\
& \text{if } \cong(x, y) \succ 0, \text{ then } \tilde{<}(x, y) \prec 1.
\end{aligned}$$

There are several possible definitions of non-crisp linear orders. For example, Zadeh [34], defines a similarity relation in a set, imposing that it is reflexive, symmetric, and transitive, as well as a notion of many-valued ordering, antisymmetry, and totality. Similarly, Bodenhofer [3] advocates for the use of similarity-based many-valued orderings, in which the linearity is in a strong form; the same notion is also used in [25]. Ovchinnikov [30] proposes a notion of many-valued ordering with a non-strict ordering relation. A common denominator to all such proposals is the definition of a very weak version of the transitivity property, which allows one to obtain very general definitions. A many-valued linear order defined as above is similar to Zadeh's, only slightly modified to take into account both the many-valued linear order and the many-valued equality in the same structure. This is motivated by the fact that many-valued Allen's relations involve both equality and linear order.

Given a many-valued linear order $\tilde{\mathbb{D}} = \langle D, \tilde{<}, \cong \rangle$, we define the *crispification* of $\tilde{\mathbb{D}}$ to be the crisp linear order $\mathbb{D} = \langle D, < \rangle$, where $x < y$ if and only if $\tilde{<}(x, y) \neq 0$. It is easy to verify that $\mathbb{D} = \langle D, < \rangle$, so defined, is, in fact, a linear order.

Later we shall refer to the set of all possible many-valued linear orders as \mathfrak{D} .

Many-valued Halpern and Shoham's Interval Temporal Logic. As in the crisp case, formulas of the many-valued version of Halpern and Shoham's interval temporal logic are based on a set of propositional letters.

► **Definition 2.** *Let \mathcal{P} be a set of propositional letters, and let \mathbf{A} be a complete FL_{ew} -algebra. Then, a well-formed many-valued Halpern and Shoham's interval temporal logic formula (or MVHS-formula, for short) is obtained by the following grammar:*

$$\varphi ::= \alpha \mid p \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \varphi \rightarrow \psi \mid \langle X \rangle \varphi \mid [X] \varphi,$$

where $\alpha \in A$, $p \in \mathcal{P}$, and, $X \in \mathcal{X}$.

In the following, we shall use $\neg\varphi$ to denote the formula $\varphi \rightarrow 0$, and $\tilde{\mathcal{L}}_{MVHS}$ to denote the smallest set that contains all formulas generated by the previous definition.

As for the semantics of MVHS-formulas, given a many-valued linearly ordered set we define the set of fuzzy strict intervals in $\tilde{\mathbb{D}}$ as

$$\mathbb{I}(\tilde{\mathbb{D}}) = \{[x, y] \mid \tilde{<}(x, y) \succ 0\},$$

and, generalizing classical Boolean evaluation, propositional letters are directly evaluated in the underlying algebra \mathbf{A} by defining a *many-valued valuation function*, as follows:

$$\tilde{V}: \mathcal{P} \times \mathbb{I}(\tilde{\mathbb{D}}) \rightarrow \mathbf{A}.$$

Moreover, given a fixed many-valued linear order we also need to define the many-valued generalization of Allen's relations between intervals (*many-valued Allen's relations*), which is obtained by substituting every $=$ with \cong and every $<$ with $\tilde{<}$ in the original, crisp definition, as shown in Tab. 2.

■ **Table 2** Many-valued version of Allen’s interval relations.

relation	definition	example
after	$\tilde{R}_A([x, y], [w, z]) = \tilde{\equiv}(y, w)$	
later	$\tilde{R}_L([x, y], [w, z]) = \tilde{<}(y, w)$	
begins	$\tilde{R}_B([x, y], [w, z]) = \tilde{\equiv}(x, w) \cdot \tilde{<}(z, y)$	
ends	$\tilde{R}_E([x, y], [w, z]) = \tilde{<}(x, w) \cdot \tilde{\equiv}(y, z)$	
during	$\tilde{R}_D([x, y], [w, z]) = \tilde{<}(x, w) \cdot \tilde{<}(z, y)$	
overlaps	$\tilde{R}_O([x, y], [w, z]) = \tilde{<}(x, w) \cdot \tilde{<}(w, y) \cdot \tilde{<}(y, z)$	

Many-valued Allen’s relations should be interpreted with caution. A many-valued linear order cannot be graphically represented as a crisp linear order, but it is abstractly defined by explicitly listing the value of each relation for each pair of points. Similarly, for example, one interval $[w, z]$ that is *after* an interval $[x, y]$ cannot be simply depicted “after” $[x, y]$ itself. However, to help the intuition, many-valued Allen’s relations can be used to interpret a crisp linear order; in this case, it would be natural to consider, for instance, as “equal” two points that are in fact “close” to each other, leading to the possibility of picturing Allen’s relations similarly to the crisp case.

► **Definition 3.** Let \mathcal{P} be a set of propositional letters, and let \mathbf{A} be a complete FL_{ew} -algebra. Then, a many-valued Halpern and Shoham’s interval temporal logic interpretation (MVHS-interpretation, for short) is a tuple of the type:

$$\tilde{M} = \langle \mathbb{I}(\tilde{\mathbb{D}}), \tilde{V} \rangle$$

where $\tilde{\mathbb{D}}$ is a many-valued linear order and \tilde{V} is a fuzzy valuation function. Given an MVHS-formula φ and an MVHS-interpretation, the valuation of φ on \tilde{V} and $[x, y] \in \mathbb{I}(\tilde{\mathbb{D}})$ is computed as follows:

$$\begin{aligned} \tilde{V}(\alpha, [x, y]) &= \alpha, \\ \tilde{V}(\varphi \wedge \psi, [x, y]) &= \tilde{V}(\varphi, [x, y]) \cdot \tilde{V}(\psi, [x, y]), \\ \tilde{V}(\varphi \vee \psi, [x, y]) &= \tilde{V}(\varphi, [x, y]) + \tilde{V}(\psi, [x, y]), \\ \tilde{V}(\varphi \rightarrow \psi, [x, y]) &= \tilde{V}(\varphi, [x, y]) \leftrightarrow \tilde{V}(\psi, [x, y]), \\ \tilde{V}(\langle X \rangle \varphi, [x, y]) &= \bigvee \{ \tilde{R}_X([x, y], [w, z]) \cdot \tilde{V}(\varphi, [w, z]) \}, \\ \tilde{V}([X] \varphi, [x, y]) &= \bigwedge \{ \tilde{R}_X([x, y], [w, z]) \leftrightarrow \tilde{V}(\varphi, [w, z]) \}, \end{aligned}$$

where $X \in \mathcal{X}$ and where $[w, z]$ varies in $\mathbb{I}(\tilde{\mathbb{D}})$. We say that a formula of FHS φ is α -satisfied at an interval $[x, y]$ in a fuzzy interval model \tilde{M} if and only if

$$\tilde{V}(\varphi, [x, y]) \succeq \alpha.$$

Moreover, a formula φ is α -satisfiable if and only if there exists a fuzzy interval model and an interval in that model in which it is α -satisfied, and it is satisfiable if it is α -satisfiable for some $\alpha \in \mathcal{H}$, $\alpha \neq 0$; similarly, a formula is α -valid if it is α -satisfied at every interval in every model, and valid if it is 1-valid.

Observe that since an FL_{ew} -algebra, in general, does not encompass classical negation, and since our definition of satisfiability is graded, instead of absolute, then the usual duality of satisfiability and validity does not hold anymore.

Applications. Interval temporal logic is designed to describe situations in which events have a duration, and are therefore not necessarily instantaneous. Even if a pure qualitative language such as the one of HS does not allow the specification of metric duration, the relative temporal position of different events can be expressed.

Temporal datasets are typically provided in the form of sets of multivariate time series. *Time series* are observations interpreted over a linear order; in some data science contexts, they are also called *single dimension* data. Observations can be *univariate* if there is only one measurement, or *multivariate* if there is more than one; observed data type can be numerical or categorical. Temporal datasets are common in several research areas, spanning from medicine to industry, among many others. In some cases, information can be extracted from a multivariate time series in the form of point-based temporal logic formulas, in which propositional letters symbolically represent instantaneous values (e.g., *the fever is over 37.5 degrees*, or *the vibration sensor marks below 3 mm/s*). More commonly, however, it is often the case that relevant events are better described as intervals so that a propositional letter represents some properties of some function of some variable (e.g., *the average fever is over 37.5 degrees*). Such functions are generally referred to as *feature extraction functions*. In this sense, interesting properties of a time series, or, better, of a temporal dataset of time series, could be expressed in propositional interval temporal logic.

One relevant example of how MVHS can be used to describe interesting situations is that of multivariate time series in psychology and psychiatry. A single patient can be described by several independent variables that help physicians identify his/her psychological status across some observation time. Each variable may be specific to some behavior trait, so that several symptoms can be identified. In the case of clinical depression, for example, such symptoms range from *depressed mood* to *diminished interest*, to *insomnia/hypersomnia*, among many others. Each one of them, clearly, can be associated with one or more periods of time during the observation. In major depression, in particular, the role of *overlapping* symptoms has been debated [4]. The presence of two such overlapping symptoms in a patient can be expressed, for example, as the HS formula

$$\langle G \rangle (\text{depressed mood} \wedge \langle O \rangle \text{insomnia}),$$

where $\langle G \rangle$ is the (definable) existential operator, that makes true a formula on some interval in the model. Statements such as the one above can, in fact, be automatically learned by a system such as [6, 32, 28], from suitable datasets of patients, and the problem of distinguishing, for instance, which pairs of overlapping symptoms more often lead to a major depression diagnosis can be solved as a temporal classification problem.

Operating with a crisp logic, however, one may incur in extracting less-than-optimal knowledge, for two reasons. First, single events labeled with symptoms may not necessarily be clearly identified, as is often the case in medicine. Second, minimal temporal variations may cause sharp changes in the labeled temporal relation (e.g., a patient with both symptoms, not overlapping but only by a relatively small amount of time, would not satisfy the above formula). Extracting many-valued logic statements, expressed in MVHS for example, would allow one to avoid both potential problems, as both the propositional letters and the temporal relations are given a degree of truth (in the above example, a patient with both symptoms, not overlapping but only by relatively small amounts of time, would still satisfy the formula, only to a slightly lower degree).

4 Reasoning in Many-Valued Halpern and Shoham's Interval Temporal Logic

A tableau system. In this section, we consider the problem of reasoning with MVHS formulas. Tableau systems have been introduced in [5, 19, 20, 29] for variants, fragments, and generalizations of crisp HS, and in [17] for many-valued modal logics. The case of MVHS interpreted on a finite Heyting algebra has been tackled in [13]. Here, we generalize the rules to cover the case of a finite FL_{ew} -algebra. From a theoretical point of view, the difference between the two systems is limited; therefore, we shall focus on the problems that having a functioning implementation raises, and, then, on the results of a set of systematic tests.

A tableau for an MVHS formula is a directed tree, in which every node is associated with a truth judgment, a pair formula/interval, and a finite many-valued linear order; these elements, altogether, form a decoration.

► **Definition 4 (decoration).** *Given an FL_{ew} -algebra \mathbf{A} , an MVHS formula φ , and a finite many-valued linear order $\tilde{\mathbb{D}}$, a decoration is an object of the type*

$$Q(\alpha \preceq \varphi, [x, y], \tilde{\mathbb{D}}), \text{ or } Q(\varphi \preceq \alpha, [x, y], \tilde{\mathbb{D}}),$$

where $\alpha \in \mathbf{A}$ and $Q \in \{T, F\}$ is a judgment. The expression $\alpha \preceq \varphi$ ($\varphi \preceq \alpha$) is an assertion on $[x, y] \in \mathbb{I}(\tilde{\mathbb{D}})$. The universe of all possible decorations is denoted by \mathcal{D} .

► **Definition 5 (tableau).** *Given a finite FL_{ew} -algebra \mathbf{A} and an MVHS formula φ , a tableau τ for φ and $\alpha \in \mathbf{A}$ is defined as a tuple*

$$\tau = \langle \mathcal{V}, \mathcal{E}, d, f, c \rangle,$$

where $\langle \mathcal{V}, \mathcal{E} \rangle$ is a tree with nodes in \mathcal{V} and edges in \mathcal{E} , and whose set of branches is denoted by \mathcal{B} . The vertices are partially ordered by a relation \triangleleft , which is induced by the edges. The function

$$d: \mathcal{V} \rightarrow \mathcal{D}$$

is a node labeling function that assigns each node ν a decoration of the form $Q(\beta \preceq \psi, [w, z], \tilde{\mathbb{D}})$ or $Q(\psi \preceq \beta, [w, z], \tilde{\mathbb{D}})$, where $\beta \in \mathbf{A}$ and ψ is a subformula of φ (denoted as $\psi \in \text{sub}(\varphi)$). The function

$$f: \mathcal{V} \rightarrow \{0, 1\}$$

is a node flag function indicating whether nodes are expanded (1) or not expanded (0). The function

$$c: \mathcal{B} \rightarrow \mathcal{D}$$

is a branch labeling function that associates to every branch the finite many-valued linear order that belongs to the decoration of its leaf. The initial tableau τ_0 is

$$\langle \{\nu_0\}, \emptyset, \{\nu_0 \mapsto (T(\alpha \preceq \varphi), [x, y], \{\tilde{\prec}(x, y) = \beta\})\}, \{\nu_0 \mapsto 0\}, \{\nu_0 \mapsto \{\tilde{\prec}(x, y) = \beta\}\} \rangle$$

or

$$\langle \{\nu_0\}, \emptyset, \{\nu_0 \mapsto (F(\alpha \preceq \varphi), [x, y], \{\tilde{\prec}(x, y) = \beta\})\}, \{\nu_0 \mapsto 0\}, \{\nu_0 \mapsto \{\tilde{\prec}(x, y) = \beta\}\} \rangle$$

for some $\beta \in \mathbf{A}$, and it evolves by iteratively applying the branch expansion rule (see Fig. 2) or, if not applicable, the reverse rule in (see Fig. 3). These rules are applied to the node ν closest to the root such that $f(\nu) = 0$, affecting every descendant leaf ν' for which $\nu \triangleleft \nu'$. An application of either rule to ν where $f(\nu) = 0$ will result in setting $f(\nu)$ to 1.

$$\begin{array}{l}
 (T\wedge) \frac{T(\beta \preceq (\psi \wedge \xi), [x, y], C)}{T(\beta_1 \preceq \psi, [x, y], C(B)) \mid \dots \mid T(\beta_n \preceq \psi, [x, y], C(B)) \\
 \quad T(\gamma_1 \preceq \xi, [x, y], C(B)) \mid \dots \mid T(\gamma_n \preceq \xi, [x, y], C(B))} \\
 \text{where } \beta \neq 0, (\beta_i, \gamma_i) \in \mathbf{A} \times \mathbf{A} \text{ so that } \beta \preceq \beta_i \cdot \gamma_i \text{ and there is no} \\
 \text{other } (\beta'_i, \gamma'_i) \in \mathbf{A} \times \mathbf{A} \text{ such that } \beta \preceq \beta'_i \cdot \gamma'_i, \beta'_i \preceq \beta_i \text{ and } \gamma'_i \preceq \gamma_i. \\
 \\
 (F\wedge) \frac{F(\beta \preceq (\psi \wedge \xi), [x, y], C)}{T(\psi \preceq \beta_1, [x, y], C(B)) \mid \dots \mid T(\psi \preceq \beta_n, [x, y], C(B)) \\
 \quad T(\xi \preceq \gamma_1, [x, y], C(B)) \mid \dots \mid T(\xi \preceq \gamma_n, [x, y], C(B))} \\
 \text{where } \beta \neq 0, (\beta_i, \gamma_i) \in \mathbf{A} \times \mathbf{A} \text{ so that } \beta \not\preceq \beta_i \cdot \gamma_i \text{ and there is no} \\
 \text{other } (\beta'_i, \gamma'_i) \in \mathbf{A} \times \mathbf{A} \text{ such that } \beta \not\preceq \beta'_i \cdot \gamma'_i, \beta_i \preceq \beta'_i \text{ and } \gamma_i \preceq \gamma'_i. \\
 \\
 (T\vee) \frac{T((\psi \vee \xi) \preceq \beta, [x, y], C)}{T(\psi \preceq \beta_1, [x, y], C(B)) \mid \dots \mid T(\psi \preceq \beta_n, [x, y], C(B)) \\
 \quad T(\xi \preceq \gamma_1, [x, y], C(B)) \mid \dots \mid T(\xi \preceq \gamma_n, [x, y], C(B))} \\
 \text{where } \beta \neq 1, (\beta_i, \gamma_i) \in \mathbf{A} \times \mathbf{A} \text{ so that } \beta_i + \gamma_i \preceq \beta \text{ and there is no} \\
 \text{other } (\beta'_i, \gamma'_i) \in \mathbf{A} \times \mathbf{A} \text{ such that } \beta'_i + \gamma'_i \preceq \beta, \beta_i \preceq \beta'_i \text{ and } \gamma_i \preceq \gamma'_i. \\
 \\
 (F\vee) \frac{F((\psi \vee \xi) \preceq \beta, [x, y], C)}{T(\beta_1 \preceq \psi, [x, y], C(B)) \mid \dots \mid T(\beta_n \preceq \psi, [x, y], C(B)) \\
 \quad T(\gamma_1 \preceq \xi, [x, y], C(B)) \mid \dots \mid T(\gamma_n \preceq \xi, [x, y], C(B))} \\
 \text{where } \beta \neq 1, (\beta_i, \gamma_i) \in \mathbf{A} \times \mathbf{A} \text{ so that } \beta_i + \gamma_i \not\preceq \beta \text{ and there is no} \\
 \text{other } (\beta'_i, \gamma'_i) \in \mathbf{A} \times \mathbf{A} \text{ such that } \beta'_i + \gamma'_i \not\preceq \beta, \beta'_i \preceq \beta_i \text{ and } \gamma'_i \preceq \gamma_i. \\
 \\
 (T \leftrightarrow) \frac{T(\beta \preceq (\psi \leftrightarrow \xi), [x, y], C)}{T(\psi \preceq \beta_1, [x, y], C(B)) \mid \dots \mid T(\psi \preceq \beta_n, [x, y], C(B)) \\
 \quad T(\gamma_1 \preceq \xi, [x, y], C(B)) \mid \dots \mid T(\gamma_n \preceq \xi, [x, y], C(B))} \\
 \text{where } \beta \neq 0, (\beta_i, \gamma_i) \in \mathbf{A} \times \mathbf{A} \text{ so that } \beta \preceq \beta_i \leftrightarrow \gamma_i \text{ and there is no} \\
 \text{other } (\beta'_i, \gamma'_i) \in \mathbf{A} \times \mathbf{A} \text{ such that } \beta \preceq \beta'_i \leftrightarrow \gamma'_i, \beta_i \preceq \beta'_i \text{ and } \gamma'_i \preceq \gamma_i. \\
 \\
 (F \leftrightarrow) \frac{F(\beta \preceq (\psi \leftrightarrow \xi), [x, y], C)}{T(\beta_1 \preceq \psi, [x, y], C(B)) \mid \dots \mid T(\beta_n \preceq \psi, [x, y], C(B)) \\
 \quad T(\xi \preceq \gamma_1, [x, y], C(B)) \mid \dots \mid T(\xi \preceq \gamma_n, [x, y], C(B))} \\
 \text{where } \beta \neq 0, (\beta_i, \gamma_i) \in \mathbf{A} \times \mathbf{A} \text{ so that } \beta \not\preceq \beta_i \leftrightarrow \gamma_i \text{ and there is no} \\
 \text{other } (\beta'_i, \gamma'_i) \in \mathbf{A} \times \mathbf{A} \text{ such that } \beta \not\preceq \beta'_i \leftrightarrow \gamma'_i, \beta'_i \preceq \beta_i \text{ and } \gamma_i \preceq \gamma'_i.
 \end{array}$$

■ Figure 2 Propositional rules.

$$\begin{array}{ll}
(T \succeq) \frac{T(\beta \preceq \psi, [x, y], C)}{F(\psi \preceq \gamma, [x, y], C(B))} & (F \succeq) \frac{F(\beta \preceq \psi, [x, y], C)}{T(\psi \preceq \gamma_1, [x, y], C(B)) \mid \dots \mid T(\psi \preceq \gamma_n, [x, y], C(B))} \\
\text{where } \varphi \neq \alpha, \beta \neq 0 \text{ and } \gamma \text{ is any maximal} & \text{where } \varphi \neq \alpha, \beta \neq 0 \text{ and } \gamma_1, \dots, \gamma_n \text{ are all maximal} \\
\text{element not above } \beta, \text{ i.e., } \gamma \not\preceq \beta & \text{elements not above } \beta, \text{ i.e., } \gamma_1, \dots, \gamma_n \not\preceq \beta
\end{array}$$

$$\begin{array}{ll}
(T \preceq) \frac{T(\psi \preceq \beta, [x, y], C)}{F(\gamma \preceq \psi, [x, y], C(B))} & (F \preceq) \frac{F(\psi \preceq \beta, [x, y], C)}{T(\gamma_1 \preceq \psi, [x, y], C(B)) \mid \dots \mid T(\gamma_n \preceq \psi, [x, y], C(B))} \\
\text{where } \varphi \neq \alpha, \beta \neq 1 \text{ and } \gamma \text{ is any minimal} & \text{where } \varphi \neq \alpha, \beta \neq 1 \text{ and } \gamma_1, \dots, \gamma_n \text{ are all minimal} \\
\text{element not below } \beta, \text{ i.e., } \gamma \not\preceq \beta & \text{elements not below } \beta, \text{ i.e., } \gamma_1, \dots, \gamma_n \not\preceq \beta
\end{array}$$

■ **Figure 3** Reverse rules.

$$\begin{array}{ll}
(T\Box) \frac{T(\beta \preceq [X]\psi, [x, y], C)}{T((\beta \cdot \gamma_1) \preceq \psi, [z_1, t_1], c(B)) \mid \dots \mid T((\beta \cdot \gamma_n) \preceq \psi, [z_n, t_n], c(B))} & (T\Diamond) \frac{T(\langle X \rangle \psi \preceq \beta, [x, y], C)}{T((\psi \preceq (\gamma_1 \leftrightarrow \beta), [z_1, t_1], c(B)) \mid \dots \mid T(\psi \preceq (\gamma_n \leftrightarrow \beta), [z_n, t_n], c(B))} \\
\text{where } \gamma_i = R_X([x, y], [z_i, t_i]), [z_i, t_i] \in o(c(B)), & \text{where } \gamma_i = R_X([x, y], [z_i, t_i]), [z_i, t_i] \in o(c(B)), \\
\gamma_i \succ 0, \text{ and } \beta \cdot \gamma_i \neq 0 & \gamma_i \succ 0, \text{ and } \gamma_i \leftrightarrow \beta \neq 1
\end{array}$$

$$\begin{array}{l}
(F\Box) \frac{F(\beta \preceq [X]\psi, [x, y], C)}{F((\beta \cdot \gamma_1) \preceq \psi, [z_1, t_1], c(B)) \mid \dots \mid F((\beta \cdot \gamma_n) \preceq \psi, [z_n, t_n], c(B))} \\
\text{where } \gamma_i = R_X([x, y], [z_i, t_i]), [z_i, t_i] \in o(c(B)) \cup n(c(B)), \\
\gamma_i \succ 0, \text{ and } \beta \cdot \gamma_i \neq 0
\end{array}$$

$$\begin{array}{l}
(F\Diamond) \frac{F(\langle X \rangle \psi \preceq \beta, [x, y], C)}{F(\psi \preceq (\gamma_1 \leftrightarrow \beta), [z_1, t_1], c(B)) \mid \dots \mid F(\psi \preceq (\gamma_n \leftrightarrow \beta), [z_n, t_n], c(B))} \\
\text{where } \gamma_i = R_X([x, y], [z_i, t_i]), [z_i, t_i] \in o(c(B)) \cup n(c(B)), \\
\gamma_i \succ 0, \text{ and } \gamma_i \leftrightarrow \beta \neq 1
\end{array}$$

■ **Figure 4** Temporal rules.

$$\begin{array}{lll}
(\mathbf{X1}) \frac{T(\beta \preceq \gamma, [x, y], C)}{\mathbf{X}} & (\mathbf{X2}) \frac{F(\beta \preceq \gamma, [x, y], C)}{\mathbf{X}} & (\mathbf{X3}) \frac{F(0 \preceq \psi, [x, y], C)}{\mathbf{X}} \\
\text{where } \beta \not\preceq \gamma & \text{where } \beta \neq 0, \gamma \neq 1, \text{ and } \beta \preceq \gamma &
\end{array}$$

$$\begin{array}{lll}
(\mathbf{X4}) \frac{F(\psi \preceq 1, [x, y], C)}{\mathbf{X}} & (\mathbf{X5}) \frac{T(\gamma \preceq \psi, [x, y], C)}{\mathbf{X}} & (\mathbf{X6}) \frac{Q(\cdot, \cdot, C)}{\mathbf{X}} \\
\text{where } \beta \preceq \gamma & \text{where } \beta \preceq \gamma & \text{where } C \text{ is inconsistent}
\end{array}$$

■ **Figure 5** Branch closing rules.

► **Definition 6** (open and closed tableau). *Given a finite FL_{ew} -algebra \mathbf{A} and an \mathbf{A} -formula φ , a tableau τ for φ and $\alpha \in \mathbf{A}$ is closed if the branch closing rules (see Fig. 5), can be applied to all of its branches. Conversely, τ is open if there exists at least one branch to which the branch closing rules cannot be applied.*

An open (resp., closed) tableau for φ and α , initiated with the decoration $T(\alpha \preceq \varphi)$ (resp., $F(\alpha \preceq \varphi)$), effectively demonstrates that φ is α -satisfiable (resp., α -valid). Conversely, the reversed decorations $T(\varphi \preceq \alpha)$ and $F(\varphi \preceq \alpha)$ are generally not employed as starting decorations to prove substantial statements. For example, $T(\varphi \preceq \alpha)$ might suggest the possibility of constructing a structure where φ has a value less than α . In classical logic, with a two-element Boolean algebra and $\alpha = 1$, this condition would imply that $\neg\varphi$ is valid, as φ would consistently take the value 0. However, in the many-valued case, such duality is absent, and there is no guarantee of a formula that consistently attains the value 1 when φ is valued strictly less than 1.

Despite their limited use as initial decorations, these reversed decorations, in conjunction with the reverse rule, facilitate reducing the number of necessary rules. For instance, a decoration including the judgment $T(p \preceq \beta)$ appearing in a branch of a tableau starting with the judgment $T(\alpha \preceq \varphi)$ informs us that in any model validating α -satisfiability of φ , the propositional variable p must take a value less than or equal to β . For any given formula φ and value α , a tableau starting with $T(\alpha \preceq \varphi)$ (resp., $F(\varphi \preceq \alpha)$) is termed a *SAT-tableau* (resp., *VAL-tableau*).

Two distinct notions of soundness and completeness are applicable: one for the SAT-tableau system and another for the VAL-tableau system. Although the foundational arguments for both systems bear strong similarities, our statement is focused on the former.

► **Theorem 7** (soundness and completeness for α -satisfiability). *The tableau system for the MVHS of finite FL_{ew} -algebras is sound and complete for proving α -satisfiability, that is, given a finite FL_{ew} -algebra \mathbf{A} , an \mathbf{A} -formula φ , and a value $\alpha \in \mathbf{A}$, φ is α -satisfiable if and only if some SAT-tableau for φ and α is open.*

Technically speaking, the proof of this theorem is essentially identical to that of soundness and completeness of the version of this tableau given in [13], and it is therefore omitted.

Implementation. The tableau system for MVHS with finite FL_{ew} -algebras has been implemented using the Julia programming language as part of a broader open-source project aimed at representing, reasoning, and learning from structured and unstructured data [27].

In our implementation, formulas are represented as syntax trees with leaves consisting of either propositional letters or algebra values. Finite FL_{ew} -algebras are configured by establishing a finite domain, defining operators through their tables, and ensuring, via a one-time check, that all axioms proper to an FL_{ew} -algebra are satisfied. Formulas can be generated randomly, while algebras are systematically created with progressively larger domains to explore the impact of algebra size on system performance.

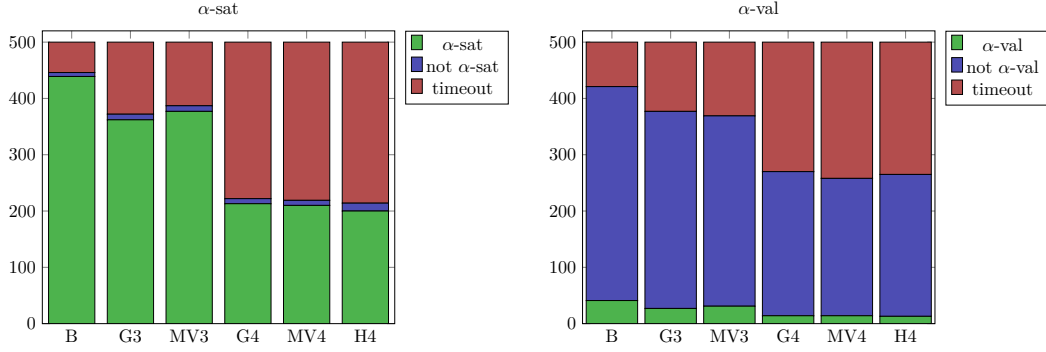
The tableau itself is structured as a rooted tree with variable arity. Each node in the tableau has a decoration and maintains two pointers, one to its parent node and the other to the head of a list of its children, which can be empty. Nodes also carry two flags to track whether they have been expanded (or not) and whether they are open (or closed). Specifically, a closed node indicates that all branches containing it are also closed. The tableau is linked to an array of priority queues where each queue node references a tableau node and carries a value determined by a priority function. This priority setting is crucial during the expansion phase to determine the next branch for expansion.

The construction of the tableau follows a standard search algorithm, beginning with an empty node to which several immediate children are attached; each child corresponds to a different algebra value for the relation $\tilde{<}$ between the two initial points. At each cycle, we evaluate all nodes at the top of any priority queue. We select one node from this group based on a predefined meta-policy (e.g., random or majority). The selected node, immediately removed from other queues where it was at the top, identifies a set of branches chosen for expansion. We first verify whether this node, denoted as ν , is already closed; if so, the cycle proceeds without action. Next, if ν has already been expanded, the process skips to the next cycle unless ν is a leaf, signaling an open branch in a fully expanded tableau. For a SAT-tableau, such an open branch identifies a model that α -satisfies the starting formula; for a VAL-tableau, it represents a counterexample to its α -validity.

If ν has not been expanded, we identify the nearest unexpanded ancestor ν' of ν and mark it as expanded. The subsequent steps are to apply the closing rule, thereby closing ν' and its descendants (including ν). If this is not applicable, the propositional rule extends the tree from all leaves containing ν' and inserts the fresh nodes into all priority queues checking that each fresh node was not already present in the branch; in such case, if no node is added to that branch, a virtual node containing $T(\top \preceq \top)$ is inserted to keep track of the presence of a branch. If no propositional rule is applicable, the temporal rule is applied with the only difference that when applying the $T\Box$ and the $T\Diamond$ rule, if at least one node has been added to the branch before $T(\beta \preceq [X]\psi, [x, y], C(B))$ (resp. $T(\beta \preceq \langle X \rangle \psi, [x, y], C(B))$) we still add the latter, as we may have new points in $c(B)$ w.r.t. C ; otherwise, we end the tableau procedure, as we have found an open branch with no contradictions within that is no longer expandable. Again, if this is not applicable, the reverse rule is applied with implications for all branch leaves containing ν' , inserting all fresh nodes in all priority queues with the same precautions taken for the expansion rules. Finally, if none of the above apply, ν is reinserted into the priority queues from which was extracted. The process concludes when all priority queues are empty, indicating no open branches remain, and the tableau is fully expanded and declared closed. For a SAT-tableau, this outcome proves the α -unsatisfiability of the starting formula; for a VAL-tableau, it demonstrates α -validity.

Optimizations. To improve both time and space efficiency of our implementation, we periodically perform a cleaning operation on all priority queues. For a parameter $K \in \mathbb{N}$, every K cycles all nodes within a queue that have already been expanded (unless they are leaves) or that are already closed are removed. This operation helps manage the computational overhead and optimizes the performance of the tableau construction. Moreover, since the creation of each new branch in these rules is independent from the others and has a non-negligible cost, as for each new node we have to check that no inconsistencies are introduced, this process has been parallelized, introducing also a locking system to manage the concurrent writing to both the tableau structure and the heaps.

All standard search strategies can be used within our system; different search strategies can be used at the same time, in a round-robin policy, or even with a further level of parallelization, in order to implement a virtual best solver policy. Given the high computation complexity of these problems, however, we designed (but not tested) a specific search strategy based on a weighting function that quantifies how ‘representative’, that is, how different a branch is from others already tested. Such a strategy can be used in a complete form (branches are explored in order of representativeness), or an incomplete form (only the most representative branches are explored). Future work includes a systematic assessment of the usefulness of such strategies.



■ **Figure 6** Results on Fig.1 algebras for formulas of height up to 5 with a timeout of 30 seconds.

5 Experiments

The objective of our experiments was to investigate the impact of different finite FL_{ew} -algebras as well as growth in formula height on computational performance. This analysis was conducted involving both SAT-tableaux and VAL-tableaux.

Initially, we generated six representative finite FL_{ew} -algebras: the Boolean algebra (\mathbb{B}), used as the baseline, the Gödel algebras and MV-algebras with $n \in \{3, 4\}$ values (Gn and MVn), and the diamond algebra with 4 values ($H4$). The lattice structures for these algebras are depicted in Fig. 1 (right-hand side). It is important to note that while $G3$ and $MV3$ (as well as $G4$ and $MV4$) are based on identical lattices, they are distinguished by their respective norms. We generated 100 random formulas for each algebra and for each height up to 5 (i.e., up to 32 symbols). Formula generation was governed by a weighted selection process: connectives $\{\wedge, \vee, \rightarrow\}$ were assigned a weight of 8, modalities $\{\langle A \rangle, \langle L \rangle, \langle B \rangle, \langle E \rangle, \langle D \rangle, \langle O \rangle, [A], [L], [B], [E], [D], [O]\}$ as well as their inverse were assigned a weight a 1 (so that one has the same probability to get a connective or a modality), algebra values received a weight of $1/|A|$ where A is the domain of the algebra, and each propositional letter in the set \mathcal{P} was weighted by $1/|\mathcal{P}|$. Selection probabilities for a symbol s were calculated as

$$P(s) = \frac{W[s]}{\sum_{\{t|t \in \{\wedge, \vee, \rightarrow\} \cup \{\langle X \rangle | X \in \mathcal{X}\} \cup A \cup \mathcal{P}\}} W[t]},$$

with formula expansion ceasing upon reaching the designated height or if further expansion is impossible (in which case it is discarded). The performance of each formula in terms of α -satisfiability and α -validity, with α chosen randomly, was analyzed; the results are depicted in Fig. 6. Throughout these experiments, the branch priority policy was kept random (and complete), and the choice of α was also randomized. All tests were conducted on a machine equipped with 2 Intel Xeon Gold 28-Core CPUs and 224GB of RAM.

6 Conclusions

In this paper, we expanded previous work on a tableau-based reasoning system tailored for many-valued interval temporal logic. In particular, we provided a reasoning system for MVHS in a very general case, and we focused on its implementation, which we made available as part of a comprehensive symbolic learning and reasoning framework [27]. We have also designed and carried out a series of tests to study the scalability of the system. As we have mentioned, future work includes exploring more elaborate search strategies and testing their effectiveness.

Moving forward, our aim is to build on the integration of symbolic learning models and reasoning systems and to provide end-to-end solutions for full data-driven learning, reasoning, and decision-making processes.

References

- 1 J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.
- 2 M. Baaz, N. Preining, and R. Zach. First-order Gödel logics. *Annals of Pure and Applied Logic*, 147:23–47, 2007. doi:10.1016/J.APAL.2007.03.001.
- 3 U. Bodenhofer. Representations and constructions of similarity-based fuzzy orderings. *Fuzzy Sets and Systems*, 137(1):113–136, 2003. doi:10.1016/S0165-0114(02)00436-0.
- 4 G. E. Brancati, E. Vieta, J. M. Azorin, J. Angst, C. L. Bowden, S. Mosolov, A. H. Young, and G. Perugi. The role of overlapping excitatory symptoms in major depression: are they relevant for the diagnosis of mixed state? *Journal of Psychiatric Research*, 115:151–157, 2019.
- 5 D. Bresolin, D. Della Monica, A. Montanari, and G. Sciavicco. A tableau system for right propositional neighborhood logic over finite linear orders: An implementation. In *Proc. of the 22th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 8123 of *LNCS*, pages 74–80. Springer, 2013. doi:10.1007/978-3-642-40537-2_8.
- 6 A. Brunello, G. Sciavicco, and I. E. Stan. Interval temporal logic decision tree learning. In *Proceedings of the 16th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 11468 of *LNCS*, pages 778–793. Springer, 2019. doi:10.1007/978-3-030-19570-0_50.
- 7 J. J. Buckley and Y. Hayashi. Fuzzy neural networks: A survey. *Fuzzy Sets and Systems*, 66:1–13, 1994.
- 8 C. C. Chang. Algebraic analysis of many valued logics. *Transactions of the American Mathematical society*, 88(2):467–490, 1958.
- 9 Y. Chen, T. Wang, B. Wang, and Z. Li. A survey of fuzzy decision tree classifier. *Fuzzy Information and Engineering*, 1(2):149–159, 2009.
- 10 P. Cintula, P. Hájek, and C. Noguera, editors. *Handbook of Mathematical Fuzzy Logic*, volume 37-38 of *Studies in Logic. Mathematical Logic and Foundation*. College publications, 2011.
- 11 W. Conradie, D. Della Monica, E. Muñoz-Velasco, and G. Sciavicco. An approach to fuzzy modal logic of time intervals. In *Proc. of the 24th European Conference on Artificial Intelligence (ECAI)*, volume 325 of *FAIA*, pages 696–703. IOS Press, 2020. doi:10.3233/FAIA200156.
- 12 W. Conradie, D. Della Monica, E. Muñoz-Velasco, G. Sciavicco, and I. E. Stan. Fuzzy Halpern and Shoham’s interval temporal logics. *Fuzzy Sets and Systems*, 456:107–124, 2023. doi:10.1016/J.FSS.2022.05.014.
- 13 W. Conradie, R. Monego, E. Muñoz-Velasco, G. Sciavicco, and I. E. Stan. A sound and complete tableau system for fuzzy Halpern and Shoham’s interval temporal logic. In *Proc. of the 30th International Symposium on Temporal Representation and Reasoning (TIME)*, volume 278 of *LIPICs*, pages 9:1–9:14. Schloss Dagstuhl, 2023. doi:10.4230/LIPICs.TIME.2023.9.
- 14 S. Dutta. An event based fuzzy temporal logic. In *Proc. of the 18th International Symposium on Multiple-Valued Logic*, pages 64–71, 1988.
- 15 L. Esakia, G. Bezhanishvili, W. H. Holliday, and A. Evseev. *Heyting Algebras: Duality Theory*. Springer, 2019.
- 16 M. Fitting. Many-valued modal logics. *Fundamenta Informaticae*, 15(3-4):235–254, 1991.
- 17 M. Fitting. Tableaus for many-valued modal logic. *Studia Logica*, 55(1):63–87, 1995. doi:10.1007/BF01053032.
- 18 A. Frigeri, L. Pasquale, and P. Spoletini. Fuzzy time in linear temporal logic. *ACM Transactions on Computational Logic*, 15:1–22, 2014. doi:10.1145/2629606.

- 19 V. Goranko, A. Montanari, P. Sala, and G. Sciavicco. A general tableau method for propositional interval temporal logics: Theory and implementation. *Journal of Applied Logics*, 4(3):305–330, 2006. doi:10.1016/J.JAL.2005.06.012.
- 20 V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighborhood temporal logics. *Journal of Universal Computer Science*, 9(9):1137–1167, 2003. doi:10.3217/JUCS-009-09-1137.
- 21 V. Goranko, A. Montanari, and G. Sciavicco. A road map of interval temporal logics and duration calculi. *Journal of Applied Non-Classical Logics*, 14(1–2):9–54, 2004. doi:10.3166/JANCL.14.9-54.
- 22 P. Hájek. *The Metamathematics of Fuzzy Logic*. Kluwer, 1998.
- 23 J. Y. Halpern and Y. Shoham. A Propositional Modal Logic of Time Intervals. *Journal of the ACM*, 38(4):935–962, 1991. doi:10.1145/115234.115351.
- 24 L. I. Kuncheva. *Fuzzy Classifier Design*, volume 49 of *Studies in Fuzziness and Soft Computing*. Springer, 2000. doi:10.1007/978-3-7908-1850-5.
- 25 S. Kundu. Similarity relations, fuzzy linear orders, and fuzzy partial orders. *Fuzzy Sets and Systems*, 109(3):419–428, 2000. doi:10.1016/S0165-0114(97)00370-9.
- 26 K. B. Lamine and F. Kabanza. Using fuzzy temporal logic for monitoring behavior-based mobile robots. In *Proc. of the IASTED International Conference, Robotics and Applications*, pages 116–121, 2000.
- 27 F. Manzella, G. Pagliarini, A. Paparella, G. Sciavicco, and I. E. Stan. Sole.jl – Symbolic Learning in Julia. <https://github.com/aclai-lab/Sole.jl>, 2024.
- 28 F. Manzella, G. Pagliarini, G. Sciavicco, and I. E. Stan. Interval Temporal Random Forests with an Application to COVID-19 Diagnosis. In *Proceedings of the 28th International Symposium on Temporal Representation and Reasoning (TIME)*, volume 206 of *LIPICs*, pages 7:1–7:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.TIME.2021.7.
- 29 E. Muñoz-Velasco, M. Pelegrín-García, P. Sala, G. Sciavicco, and I. E. Stan. On coarser interval temporal logics. *Artificial Intelligence*, 266:1–26, 2019. doi:10.1016/J.ARTINT.2018.09.001.
- 30 S. Ovchinnikov. Similarity relations, fuzzy partitions, and fuzzy orderings. *Fuzzy Sets and Systems*, 40(1):107–126, 1991.
- 31 A. Rose. Formalisations of further \aleph_0 -valued Łukasiewicz propositional calculi. *Journal of Symbolic Logic*, 43(2):207–210, 1978. doi:10.2307/2272818.
- 32 G. Sciavicco and I. E. Stan. Knowledge extraction with interval temporal logic decision trees. In *Proceedings of the 27th International Symposium on Temporal Representation and Reasoning (TIME)*, volume 178 of *LIPICs*, pages 9:1–9:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.TIME.2020.9.
- 33 H. Thiele and S. Kalenka. On fuzzy temporal logic. In *Proc. of the 2nd International Conference on Fuzzy Systems*, pages 1027–1032. IEEE, 1993.
- 34 L. A. Zadeh. Similarity relations and fuzzy orderings. *Information Sciences*, 3(2):177–200, 1971. doi:10.1016/S0020-0255(71)80005-1.

A More Efficient and Informed Algorithm to Check Weak Controllability of Simple Temporal Networks with Uncertainty

Ajdin Sumic ✉

Technological University of Tarbes, France

Thierry Vidal ✉

Technological University of Tarbes, France

Abstract

Simple Temporal Networks with Uncertainty (STNU) are a well-known constraint-based model expressing sets of activities (e.g., a schedule or a plan) related by temporal constraints, each having possible durations in the form of convex intervals. Uncertainty comes from some of these durations being *contingent*, i.e., the agent executing the plan cannot decide the actual duration at execution time. To check that execution will satisfy all the constraints, three levels of *controllability* exist: the Strong and Dynamic Controllability (SC/DC) has proven both useful in practice and provable in polynomial time, while Weak Controllability (WC) is co-NP-complete and has been left aside. Moreover, controllability checking algorithms are propagation strategies, which have the usual drawback, in case of failure, to prove unable to locate the contingents that explain the source of non-controllability. This paper has three contributions: (1) it substantiates the usefulness of WC in multi-agent systems (MAS) where another agent controls a contingent, and agents agree just before execution on the durations; (2) it provides a new WC-checking algorithm whose performance in practice depends on the network structure and is faster in loosely connected ones; (3) it provides the failing cycles in the network that explain non-WC.

2012 ACM Subject Classification Computing methodologies

Keywords and phrases Temporal constraints satisfaction, uncertainty, STNU, Controllability checking, Explainable inconsistency, Multi-agent planning

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.8

1 Introduction and Related Work

Temporal Constraint Satisfaction Problems (TCSP) are constraint-based problem formulations that allow to represent and reason on temporal constraints. They are used in a lot of domains, such as planning and scheduling (on which we will focus), supervision of dynamic systems, or workflow design. They are based on a graphical model, the reason why they are usually called Temporal Constraint Networks (TCN)[5]: variables/nodes are time-points for which one shall assign a timestamp. Constraints/edges express sets of possible durations relating them. A key issue is the ability to check the consistency of the whole network. The simplest class, called the Simple Temporal Network (STN), arises when they have only binary constraints with only convex intervals of values (no disjunctions). One of the main strengths of this restricted, but often sufficient in practice, model is that consistency checking is made through a polynomial propagation algorithm (the Floyd-Warhsall reduction) and provides a complete *minimal* network in which all inconsistent values are removed.

An STN with Uncertainty (*STNU*) is an extension in which one distinguishes a subset of constraints whose effective duration is not assigned but observed (uncontrollable durations). This is useful for addressing realistic dynamic and stochastic domains where such durations are usually set by the environment.



© Ajdin Sumic and Thierry Vidal;
licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 8; pp. 8:1–8:15

Leibniz International Proceedings in Informatics



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

In STNUs, the notion of temporal consistency has been redefined in the form of *controllability*: an STNU is controllable if there exists a strategy for executing the schedule, whatever the values are taken by the contingent durations. In [14], the authors introduce three levels of controllability that express how and when the uncertainties are resolved: the Weak Controllability (WC) proves that a solution exists for any possible combination of contingent values. Which requires that some “oracle” provides those values before the timing of controllable time-points is decided; Dynamic Controllability (DC) assumes that at execution time, a strategy can be built based on past observations only thus, whatever the contingent durations still to be observed; Strong Controllability (SC) is more demanding as it enforces that there is one unique assignment of controllable timepoints values, which defines a static control strategy that works whatever the contingent durations will be at execution time. WC has often appeared unrealistic in dynamic applications that assume full progressive observability at execution time, where DC looks more relevant and have received much attention in previous works. In contrast, SC fits perfectly application domains with partial or non observability, or when some strict commitment must be made on the execution schedule timing for some client.

Previous works prove that SC and DC can be resolved with specifically designed propagation-based algorithms that run in polynomial time [11, 3, 14]. While WC is a co-NP-complete problem [12], and only exponential algorithms exist to check WC [4, 14]. This is another reason why WC has been disregarded [2, 14].

This paper tackles Weak Controllability by first exhibiting its relevance in several contexts (e.g., multi-agent task management) and providing a more efficient algorithm for realistic networks, i.e., loosely connected networks. Contrary to the complete propagation algorithms proposed for SC and DC, our algorithm maintains and reasons only on the input constraints, which form network paths. As in any graph, such paths join and form cycles. We prove that it is possible to check the global Weak controllability by locally checking the elementary cycles of an STNU. This way, the algorithm can also diagnose the source of uncontrollability of a non-WC STNU by detecting the set of constraints (here, cycles) that make the STNU not Weakly controllable. This explainability issue was recently addressed and is important to repair non-controllable STNUs [9, 2, 1, 13].

The paper is organized as follows: Section 2 first recalls the necessary background on STNU. Section 3 then discusses the usefulness in practical applications of WC. Then, we prove in Section 4 how local controllability on cycles is equivalent to global WC. Next, Section 5 will present how to locally check WC, and Section 6 will present the new algorithm for globally checking WC. Some experimental evaluation will be displayed in Section 7 before concluding our contribution with some prospects.

2 Background

A Simple Temporal Network (*STN*) is a pair, (V, E) , where V is a set of time-points v_i representing event occurrence times, and E a set of temporal constraints between these time-points, in the form of convex intervals of possible durations [5], in the form $v_j - v_i \in [l_{ij}, u_{ij}]$, with lower bounds $l_{ij} \in \mathbb{R} \cup \{-\infty\}$ and upper bounds $u_{ij} \in \mathbb{R} \cup \{+\infty\}$. Interestingly enough, this model encompasses the qualitative precedence constraint, since v_i *precedes* v_j , noted $v_i \preceq v_j$, iff $l_{ij} \geq 0$. A reference time-point v_0 is usually added to V , which is the “origin of time”, depending on the application (might be, e.g., the current day at 0:00). The goal is to assign values to time-points such that all constraints are satisfied, i.e., to assign a value to each constraint in its interval domain.

An STN with Uncertainty (*STNU*) is an extension in which one distinguishes a subset of constraints whose values are parameters that cannot be assigned but will be observed [14].

- **Definition 1** (STNU). *An STNU is a tuple (V, E, C) with:*
- *V a set of time-points $\{v_0, v_1, \dots, v_n\}$, partitioned into controllable (V_c) and uncontrollable (V_u) and where v_0 is the reference time-point: $\forall i, v_0 \preceq v_i$;*
 - *E a set of requirement constraints $\{e_1, \dots, e_{|E|}\}$, where each e_k relates two time-points $e_k = v_j - v_i \in [l_{ij}, u_{ij}]$ with, $v_i, v_j \in V$.*
 - *C a set of contingent constraints $\{c_1, \dots, c_{|C|}\}$, where each c_k relates two time-points $c_k = v_j - v_i \in [l_{ij}, u_{ij}]$ with, $v_i \in V_c, v_j \in V_u$, and necessarily $v_i \preceq v_j : 0 \leq l_{ij} \leq u_{ij}$.*

Intuitively, controllable time points (V_c) are moments in time to be decided by the scheduling agent, which is trying to satisfy all the requirement constraints (E) under any possible instantiation of the contingent constraints (C). Moreover, having a contingent duration between two unordered time-points is semantically impossible. Figure 1a is the graphical representation of an STNU.

In addition, an STN (and hence an STNU too) has an equivalent *distance graph* representation [5, 7]. Each constraint of the form $[l, u]$ between v_i and v_j would be represented as $v_i \xrightarrow{[l, u]} v_j$ in the STN, or equivalently through two corresponding edges in its distance graph: $v_i \xrightarrow{u} v_j$ and $v_j \xrightarrow{-l} v_i$.

In STNUs, consistency has been redefined through three levels of *controllability*, which we will recall hereafter before focusing on one of them, namely the Weak controllability.

- **Definition 2** (Schedule). *A schedule δ of an STNU \mathcal{X} is the assignment of one value for each controllable time-point $\delta = \{\delta(v) \mid v \in V_c\}$.*

- **Definition 3** (Situation and Projection). *Given an STNU \mathcal{X} , the **situations** of \mathcal{X} is a set of tuples Ω defined as the cartesian product of contingent domains:*

$$\Omega = \prod_{c \in C} [l_c, u_c]$$

*A **situation** is an element ω of Ω and we write $\omega(c)$ with $c \in C$ to indicate the element in ω associated with c in the cross product. A **projection** $\mathcal{X}_\omega = (V, E \cup C_\omega)$ of \mathcal{X} is an STN where $C_\omega = \{[\omega(c), \omega(c)] \mid c \in C\}$. Last, a schedule δ_ω which satisfies all the constraints in \mathcal{X}_ω is called a **solution** of \mathcal{X}_ω .*

Intuitively, the set of situations defines the space of uncertainty, i.e., the possible values of contingent constraints; a projection substitutes all contingent links with a singleton, forcing its duration to the value appearing in ω . Now, a network shall be deemed controllable if it is possible to schedule the controllable time points to satisfy all requirement constraints in any possible projection. But that depends on how and when the contingent durations are observed/known by the execution supervisor.

- **Definition 4** (Weak Controllability (WC)). *An STNU \mathcal{X} is **Weakly controllable** iff $\forall \omega \in \Omega, \exists \delta_\omega$ such that δ_ω is a solution of \mathcal{X}_ω .*

This definition implies that an “oracle” communicates contingents’ durations to the scheduler before execution time, which requires all projections to be independently consistent.

We provide the two other controllability levels only for the sake of completeness, though they will not be addressed in this paper. Dynamic controllability (DC) demands that the assignment of a controllable time-point only depends on past observations, and Strong controllability (SC) demands a unique schedule that is totally independent from any observation [14].

► **Definition 5** (Dynamic Controllability (DC)). *An STNU \mathcal{X} is **Dynamically controllable** iff it is **Weakly controllable** and $\forall v_i \in V_c, \forall \omega, \omega' \in \Omega, \omega \preceq^{v_i} = \omega' \preceq^{v_i} \implies \delta_\omega(v_i) = \delta_{\omega'}(v_i)$ where $\omega \preceq^v = \{\omega_k \in \omega \text{ s.t. } \text{end}(c_k) \preceq v\}$ is the part of the situation ω which contingent constraints ending time-points precede v .*

► **Definition 6** (Strong Controllability (SC)). *An STNU \mathcal{X} is **Strongly controllable** iff $\exists \delta$ such that $\forall \omega \in \Omega, \delta$ is a solution of \mathcal{X}_ω .*

As said before, polynomial-time propagation-based checking algorithms exist for SC and DC [14][11][3]. But not for WC checking, which is co-NP-complete [12]. The original algorithm to check WC checks the consistency of all $2^{|C|}$ STNs obtained by replacing the contingents with one of their bounds (upper or lower), which is an exponential algorithm. This is enough to check WC as it has been proven in [14] that considering only the bounds of contingents is enough to verify any level of controllability in STNUs.

3 Relevance of Weak Controllability

In this section, we will argue that WC may be more relevant than DC and SC for some applications and, thus, deserves to be investigated.

In classical planning and scheduling applications, uncertainties come from external causes; they are somehow “controlled by Nature” and can only be observed at their time of occurrence. For instance, the duration of a truck ride to deliver some goods depends on exogenous traffic conditions that no one has control over. There, the real duration will be observed only at execution time, which calls for DC enforcement. However, in many domains (logistics, transport, services), one may have a first strategic phase that builds a plan without assigning all real resources; a more precise tactical version will do that later. For instance, in a health service or construction site, one needs a weekly plan for visiting patient rooms or for construction tasks. Still, the assigned teams (number of people, skills) are unknown, resulting in flexible and large enough intervals of possible durations. The precise assignment is only known each day for the next day, which allows for a more precise plan just before execution, which is exactly the definition of WC.

Moreover, uncontrollable durations also appear in multi-agent systems, when some activity duration might be controlled by another agent instead of Nature. Thus, some tasks might be controllable (requirement) for one agent but uncontrollable for another (contingent). For instance, in collaborating hospital services that share common resources: one service might need to wait before another one sends a patient. For the other agent controlling the duration, that represents a degree of freedom, i.e., the *flexibility*, that some agent wishes to keep as long as possible to be more robust. Then, collaboration may rely on the timely communication of effective durations at execution time. But it is also possible that they plan in advance their weekly operations with maximum flexibility but must set their own schedules each day for the next one. They will communicate their decisions to the agents that depend on them, for better coordination. Therefore checking WC instead of DC/SC enables the agents to be more robust through least-commitment strategies.

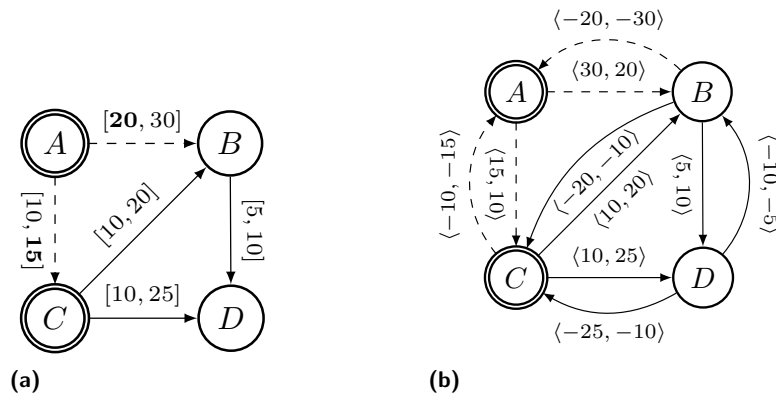
4 From local controllability to global controllability

4.1 Updated STNU graphical model

A starting point for resolving the issue of WC is to add some features to STNU's graphical representation and adapt the model accordingly. Nodes in an STNU will not only be divided between controllable and uncontrollable time-points but also by *divergent* time-points and *convergent* ones. From Definition 7, a divergent node has at least two outgoing edges in the input graph modeling the STNU, and a convergent one has at least two incoming edges.

► **Definition 7** (Convergent and Divergent time-points). *In a STNU $\mathcal{X} = (V, E, C)$:*

- $v_i \in V$ is called a **divergent** time-point iff $\exists j, k, i \neq j \neq k$ with $v_i \rightarrow v_j \in E \cup C$ and $v_i \rightarrow v_k \in E \cup C$. We denote V_{dv} as the set of divergent time-points with $V_{dv} \subseteq V$;
- $v_i \in V$ is called a **convergent** time-point iff $\exists j, k, i \neq j \neq k$ with $v_j \rightarrow v_i \in E \cup C$ and $v_k \rightarrow v_i \in E \cup C$. We denote V_{cv} , the set of convergent time-points with $V_{cv} \subset V$;



■ **Figure 1** An STNU is presented in (a) where time-point A can be seen as the reference point v_0 , $V_{dv} = \{A, C\}$ (doubly circled nodes) and $V_{cv} = \{D, B\}$. Dotted arrows express contingent constraints. Hence, C and B are uncontrollable time points, while A and D are controllable ones. The STNU is not Weakly controllable due to the projection highlighted in bold on the contingent constraints $A \xrightarrow{[10, 15]} C$ and $A \xrightarrow{[20, 30]} B$ that violate the synchronization on B. We show in 1b the controllable bounds graph of the STNU.

Please note that if a contingent link is necessarily a directed edge (implicit precedence), a requirement link may be a non-directed edge: e.g., $v_i \xrightarrow{[-5, 10]} v_j$, imposing some constraint on the temporal distance between the time-points but allowing any order between them at execution time. Hence, in this example, v_i or v_j may be considered a divergent time-point, depending on the order between them in the input link defined at the design level (here, the link will be an outgoing edge from v_i). As shown in the next subsection, the beginning and end points of the two paths that form a cycle will only change, but the cycle will still remain.

In addition, $V_{dv} \cap V_{cv}$ may not be void, i.e. any $v \in V$ may be convergent, divergent, convergent *and* divergent, or neither convergent nor divergent : these definitions are orthogonal to the distinction between controllable and contingent time-points, i.e., a controllable time-point might be convergent or divergent, etc., and an uncontrollable one alike.

Of course, by definition, v_0 cannot be a convergent time-point, but usually, a divergent one, even though the model does not enforce it, as v_0 is used to define the absolute time of any time-point v_i as a constraint between v_0 and v_i .

One can see that such a characterization is very similar to what is done in flow networks [8]. Still, there the problem is to check that the sum of labels (capacities) that converge on a point equals the sum of the labels that exit that node. Here, we will instead use this distinction to look for cycles, i.e., identify that two *paths* which diverge from one node and reunite in a convergent node have compatible overall durations whatever values the contingents in those paths will take, which is a local WC condition.

In Figure 1(a), we present an STNU as defined in definition 1 augmented by definition 7. Figure 1(b) exhibits an alternative way to represent the STNU that will be explained later.

4.2 Weak controllability on cycles

First, we assume there is at least one convergent point (and hence at least one divergent point). Otherwise the STNU is necessarily WC since there is no cycle among the input constraints and hence no negative one. That means there are **paths** that diverge at some point and merge at another point.

► **Definition 8 (Path).** *A path ρ in \mathcal{X} is a sequence of time-points v_1, \dots, v_p such that $\forall i = 1 \dots p - 1, v_i \rightarrow v_{i+1} \in E \cup C$ or $v_{i+1} \rightarrow v_i \in E \cup C, v_1 \in V_{dv}$ and $v_p \in V_{cv}$.*

In that definition, we allow a path to follow edges in the graph in any direction, thus ensuring that all possible cycles in the STNU will not be forgotten. For example, in Figure 1(a), considering divergent node C and convergent node D, there is obviously a path C-B-D, but C-A-B-D should also be considered, which is equivalent to stating that there is a path in the corresponding distance graph. Somehow, Figure 1(b), if one disregards, for now, the labels, can be viewed as such a distance graph, where the path C-A-B-D appears.

Then, any cycle of input constraints in the STNU can be defined as a pair of distinct paths with the same starting $v_1 \in V_{dv}$ and ending $v_p \in V_{cv}$ time-points. It is a peculiar way of defining those cycles that will be useful for our algorithm.

► **Definition 9 (WC Divergent Cycle).** *A **divergent cycle** \mathcal{M} is a pair (ρ_1, ρ_2) such that ρ_1 and ρ_2 are two **paths** starting at the same divergent time point $v_d \in V_{dv}$ and ending at the same converging time point $v_c \in V_{cv}$, where v_d, v_c are the only common time points in ρ_1, ρ_2 , i.e. $\rho_1 \cap \rho_2 = \{v_d, v_c\}$.*

*A cycle \mathcal{M} is said to be **Weakly controllable** if the sub-STNU restricted to the set of time-points and constraints involved in both paths is WC.*

For example, in Figure 1a one has a cycle (ρ_1, ρ_2) with $\rho_1 = \text{A-B}$ and $\rho_2 = \text{A-C-B}$.

Then, an STNU is WC only if all divergent cycles are WC. We will present this result in two steps, first defining a local property that might be checked for a divergent node and then generalizing to all divergent nodes, which will be useful for better explaining our algorithm.

► **Definition 10 (Local divergent-WC).** *Let $\mu(v_d) = \{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ the set of all cycles starting from $v_d \in V_{dv}$, converging on a set of convergent nodes of V_{cv} that are necessarily ordered (topological ordering) after v_d in the STNU \mathcal{X} . We say that \mathcal{X} is **locally divergent-WC** on v_d iff $\forall \mathcal{M}_i \in \mu(v_d), \mathcal{M}_i$ is Weakly controllable.*

For example, Figure 3d shows the two cycles starting from the divergent time-point A.

Local divergent-WC does not imply WC, as the corresponding sub-STNU might contain other divergent nodes.

► **Theorem 11 (Global controllability).** *\mathcal{X} is Weakly controllable (WC) iff $\forall v_d \in V_{dv}, \mathcal{X}$ is locally divergent-WC on v_d .*

Theorem 11 implies that checking the local divergent-WC property of all the divergent nodes of an STNU is enough to check the global WC.

Proof. The forward implication is straightforward to prove: if there is a divergent node for which at least one divergent cycle (sub-STNU) is not WC, that means there is at least one projection for which there is no consistent local schedule. Hence, the STNU will not be WC.

For the reverse implication, suppose the global STNU is not WC. Then there is at least one projection for which the corresponding STN is inconsistent; that is equivalent to having a negative cycle somewhere in that STN[14]; and that negative cycle necessarily relates time-points that form a divergent cycle in the STNU, which in turn is not WC following Definition 9. ◀

5 Local Weak Controllability

In this section, we show how to check the local WC of a cycle by exploiting the convexity of the problem, only considering the contingents bounds [14].

► **Definition 12** (Controllable Bounds). *Given an STNU $\mathcal{X} = (V, E, C)$, and $v_j - v_i \in E \cup C$. The **controllable bounds** of $v_j - v_i$, denoted Π_{ij}^{ctl} , is the pair of discrete values*

$$\Pi_{ij}^{ctl} = \langle \min_{ij}^{ctl}, \max_{ij}^{ctl} \rangle$$

where, \min_{ij}^{ctl} and \max_{ij}^{ctl} respectively represent the minimal and maximal duration that can be guaranteed for $v_j - v_i$.

Any requirement constraint $e_k = [l_{ij}, u_{ij}]$ has a minimal and maximal duration that can be guaranteed with $\min_{ij}^{ctl} = l_{ij}$ and $\max_{ij}^{ctl} = u_{ij}$. For a contingent constraint $c_k \in C$, we cannot guarantee that at execution time its duration will be lower (resp. greater) than its maximum bound u_{ij} (resp. its minimal bound l_{ij}). Hence, we have $\min_{ij}^{ctl} = u_{ij}$ and $\max_{ij}^{ctl} = l_{ij}$. Intuitively, e.g., \min_{ij}^{ctl} is the worst-case scenario for a contingent duration when trying to control the maximum possible total duration of a path it belongs to. We generalize Π_{ij}^{ctl} as follows:

$$\Pi_{ij}^{ctl} = \begin{cases} \langle u_{ij}, l_{ij} \rangle & \text{iff } v_j - v_i \in C \\ \langle l_{ij}, u_{ij} \rangle & \text{iff } v_j - v_i \in E \end{cases} \quad (1)$$

Then, from Equation 1, it is actually possible to represent an STNU \mathcal{X} in terms of its **controllable bounds graph** denoted $\Pi_{\mathcal{X}}^{ctl}$, similar to a distance graph but more suited to our algorithm, which is shown in Figure 1 (b). This graph considers each original constraint and its inverse. A requirement constraint $e_k = [l_{ij}, u_{ij}]$, equivalently $l_{ij} \leq (v_j - v_i) \leq u_{ij}$, has an inverse constraint $e'_k: -u_{ij} \leq (v_i - v_j) \leq -l_{ij}$ equivalently represented as $e'_i = [-u_{ij}, -l_{ij}]$. The same transformation is applied to contingent constraints.

From this transformation, it is possible to compute the controllable bounds of a path ρ composed of constraints in $E \cup C$ by propagating such bounds from v_1 to v_p .

► **Definition 13** (Controllable Path Bounds). *Let ρ be a path in $\Pi_{\mathcal{X}}^{ctl}$, with v_1, \dots, v_p the sequence of time-points of ρ . The **controllable path bounds** denoted Π_{ρ}^{ctl} is defined as follows:*

$$\Pi_{\rho}^{ctl} = \langle \sum \min_{ij}^{ctl}, \sum \max_{ij}^{ctl} \rangle$$

From this point, it's possible to check the WC controllability of a cycle $\mathcal{M} = (\rho_1, \rho_2)$ through the controllable paths bounds $\Pi_{\rho_1}^{ctl}$ and $\Pi_{\rho_2}^{ctl}$. Indeed, we need to guarantee that the minimum controllable duration of ρ_1 is less than or equal to the maximum controllable duration of ρ_2 and vice-versa. Intuitively, if the condition is not satisfied, then there exists a projection of \mathcal{M} such that ρ_1 and ρ_2 cannot synchronize on v_p as Π_{ρ}^{ctl} represent the worst-case scenarios of ρ : the worst cases for synchronizing two paths are when, for one path, its contingents take their minimal bounds l_{ij} and for the second one, their maximal bounds u_{ij} .

► **Theorem 14 (Cycle WC property).** *Given a cycle $\mathcal{M} = (\rho_1, \rho_2)$ and the controllable paths bounds $\Pi_{\rho_1}^{ctl} = \langle \min_{\rho_1}^{ctl}, \max_{\rho_1}^{ctl} \rangle$ and $\Pi_{\rho_2}^{ctl} = \langle \min_{\rho_2}^{ctl}, \max_{\rho_2}^{ctl} \rangle$, \mathcal{M} is weakly controllable iff:*

$$(\min_{\rho_1}^{ctl} \leq \max_{\rho_2}^{ctl}) \wedge (\min_{\rho_2}^{ctl} \leq \max_{\rho_1}^{ctl}) \quad (2)$$

Proof. If \mathcal{M} is WC, then whatever the bounds of the contingents in \mathcal{M} , there always exists a schedule that satisfies the constraints of \mathcal{M} . Let's suppose Equation 2 is false. It means there exists a projection of ρ_1 and ρ_2 such that the synchronization on v_p is impossible and forms a negative cycle. Thus, such a projection is inconsistent, and \mathcal{M} is not WC.

For the reverse implication, let us suppose \mathcal{M} is not WC, but Equation 2 is satisfied. Then, it means that the projections of the two worst-case scenarios of \mathcal{M} are consistent as there exists at least one schedule that guarantees the synchronization on v_p . Thus, any projection satisfies the synchronization on v_p . This is not possible as \mathcal{M} is not WC, which implies the sub-STNU has a negative cycle [14]. ◀

Obviously, one can see that only one of the literal can be false, i.e., either $(\min_{\rho_1}^{ctl} \leq \max_{\rho_2}^{ctl})$ or $(\min_{\rho_2}^{ctl} \leq \max_{\rho_1}^{ctl})$ is false. For the sake of simplicity, we denote M^{ctl} a worst-case scenario of \mathcal{M} . The left network of Figure 3d forms a non-WC cycle. The controllable bounds are $\{30, 20\}$ on (A-B) that forms a path ρ_1 , $\{10, 20\}$ on (C-B) and $\{15, 10\}$ on (A-C) that together form a path ρ_2 . We have $\Pi_{\rho_1}^{ctl} = \{30, 20\}$ and $\Pi_{\rho_2}^{ctl} = \{25, 30\}$, which does not satisfy $\min_{\rho_2}^{ctl} \leq \max_{\rho_1}^{ctl}$.

6 The WC-Checking algorithm

6.1 Description of the algorithm

In this section, we present the new WC-checking algorithm for an STNU \mathcal{X} , which comprises two parts: the first finds the cycles from a divergent time-point, and the second checks those cycles. It is based on the following basic structures:

- A path ρ is divided into two **projection paths** ρ_{min} and ρ_{max} where only the minimal (ρ_{min}) or maximal (ρ_{max}) controllable bounds are computed: $\Pi_{\rho_{max}}^{ctl} = \max_{\rho}^{ctl}$ and $\Pi_{\rho_{min}}^{ctl} = \min_{\rho}^{ctl}$. Given $\eta = \{min, max\}$, a projection path is of the form $\rho_{\eta} = \langle \eta_{\rho}^{ctl}, C_{\rho_{\eta}}, \mathcal{V}_{\rho_{\eta}} \rangle$ such that
 - η_{ρ}^{ctl} is the controllable bound of ρ_{η} (\max_{ρ}^{ctl} or \min_{ρ}^{ctl});
 - $C_{\rho_{\eta}}$ is the set of contingent constraints of ρ_{η} ($C_{\rho_{\eta}} \subseteq C$);
 - $\mathcal{V}_{\rho_{\eta}}$ the set of time-points of ρ_{η} ($\mathcal{V}_{\rho_{\eta}} \subseteq \mathcal{V}$).

One can notice that ρ_{min} and ρ_{max} represent the two worst-case scenarios of ρ .

- $\mathcal{P}(v_d)$ is the set of **projection paths** $\mathcal{P}(v_d) = \{\rho_{\eta_1}, \dots, \rho_{\eta_m}\}$ from the divergent time-point v_d .

- the **minimal divergent cycles** $D_{min}(v_d)$ is a mapping of convergent time points $v_c \in \mathcal{V}_{cv}$ to a set of projection paths ($\mathcal{P}_{v_c}^{min}$) that converge on v_c from v_d such that each of them $\eta = \min(\rho_{min})$: $\forall \rho_\eta \in \mathcal{P}_{v_c}^{min}, \eta_\rho^{ctl} = \min_\rho^{ctl}$.
- the **maximal divergent cycles** $D_{max}(v_d)$ is similar as $D_{min}(v_d)$ but each projection path in $\mathcal{P}_{v_c}^{max}$, $\eta = \max(\rho_{max})$: $\forall \rho_\eta \in \mathcal{P}_{v_c}^{max}, \eta_\rho^{ctl} = \max_\rho^{ctl}$.

We introduce in Algorithm 1 the *findDivergentCycles* algorithm in charge of finding the cycles from a divergent time-point v_d . To avoid going through all possible paths in the controllable bounds graph $\Pi_{\mathcal{X}}^{ctl}$, we prune the number of paths in two ways:

- We first add the notion of *rank*, which is common in qualitative temporal networks [6]: it is possible to define a partial order of all time-points with regard to the precedence relation; $rank(v_z) = 0$, then for all v_i such that $v_z \preceq v_i \in E \cup C$ and there is no v_j such that $v_z \preceq v_j \in E \cup C$ and $v_j \preceq v_i \in E \cup C$, $rank(v_i) = 1$, and so on and so forth.
- Using that rank, a forward search is then applied by ordering the time-points through a topological ordering algorithm from v_z (rank 0). This enables us to avoid any time-point v_i with a lower rank than the current divergent time-point v_d . Figures 3a to 3c highlight only the edges considered by the forward search.
- To distinguish between the minimal and maximal controllable bounds of a path, we apply two forward searches: one that computes the paths with only the maximal controllable bound and one with the minimal controllable bound. This allows us to prune the paths that converge to any convergent time-point to keep only stricter ones. For example, it is easy to see that for two projection paths ρ_η and ρ'_η such that $C_{\rho_\eta} = C_{\rho'_\eta} = \{\emptyset\}$ (only requirement constraints) ρ_η is stricter than ρ'_η if $\eta = \min$ and $\min_\rho^{ctl} > \min_{\rho'}^{ctl}$ (respectively, $\eta = \max$ and $\max_\rho^{ctl} < \max_{\rho'}^{ctl}$). Hence, it's useless to consider further ρ'_η as ρ_η is a stricter projection path, and only ρ_η is kept in $D_{min}(v_d)$ or $D_{max}(v_d)$ depending on the computed controllable bound (η). This also holds for a path ρ'_η such that $C_{\rho'_\eta} \neq \{\emptyset\}$ (with contingent constraints). However, when C_{ρ_η} and $C_{\rho'_\eta}$ are not empty, applying these rules is impossible as it might result in removing an inconsistent cycle in the graph. Suppose we have the minimal controllable bounds of ρ and ρ' (\min_ρ^{ctl} and $\min_{\rho'}^{ctl}$) and the maximal controllable bounds of a path ρ'' ($\max_{\rho''}^{ctl}$) such that the pair $\langle \rho', \rho'' \rangle$ forms a cycle M^{ctl} . Then, if ρ_{min} is stricter than ρ'_{min} and ρ'_{min} is not kept, M^{ctl} will never be checked likewise for the WC of \mathcal{X} . Therefore, both ρ_{min} and ρ'_{min} must be kept in $D_{min}(v_d)$.¹ We illustrate such case in Figure 2.

Lines 1 to 3 initialize the maps $D_{max}(v_d)$ and $D_{min}(v_d)$, and the set of paths $\mathcal{P}(v_d)$. Then, lines 5-16 propagate the paths in $\mathcal{P}(v_d)$ to find and keep all stricter paths of v_d in $D_{max}(v_d)$ until $\mathcal{P}(v_d) = \{\emptyset\}$. In fact, in line 14, we also update $\mathcal{P}(v_d)$ and \mathcal{P}_{v_j} by removing the paths that are not stricter anymore. A second forward search is done for $D_{min}(v_d)$ where $\mathcal{P}(v_d)$ is reset. Once the forward searches are over, the maps $D_{max}(v_d)$ and $D_{min}(v_d)$ contain all the restrictive paths from v_d to a convergent time-point v_c . Then, we execute the *checkCycles* algorithm (see Algorithm 2) in charge of checking the WC of the cycles of v_d . This algorithm is trivial as it simply searches and checks for each v_c in $D_{max}(v_d)$ and $D_{min}(v_d)$ all the pairs of paths (ρ_{min}, ρ_{max}) that converge on v_c and form a cycle M^{ctl} where $\mathcal{V}_{\rho_{min}} \cap \mathcal{V}_{\rho_{max}} = \{v_d, v_c\}$.

Finally, Algorithm 3 presents the *WC-checking* algorithm that, for a given STNU \mathcal{X} , computes its controllable bounds graph $\Pi_{\mathcal{X}}^{ctl}$ (line 1), determines the topological ordering of the time-points (line 2), and find and check the cycles of each divergent time-point in \mathcal{V}_{dv} .

¹ This is actually the reason why full reduction of intervals through the intersection of different edges is not possible, and hence, a polynomial time algorithm cannot be found, unlike DC and SC.

■ **Algorithm 1** findDivergentCycles algorithm.

Input: v_d :(time-point), $\Pi_{\mathcal{X}}^{ctl}$: (graph), rank: map
Output: Boolean

- 1 $D_{min}(v_d) = D_{max}(v_d) = \{\}$
- 2 $\mathcal{P}(v_d) = [\langle 0, [], [v_d] \rangle]$
- 3 *A first forward search for D_{max}*
- 4 **while** $\mathcal{P}(v_d)$ not empty **do**
- 5 $\rho_{max} = P(v_d)[0]$ ρ_{max} is removed in $\mathcal{P}(v_d)$
- 6 **for each child** v_j of $v_m \in \mathcal{V}_{\rho_{max}}$ with $rank(v_j) \geq rank(v_d)$ and $v_j \notin \mathcal{V}_{\rho_{max}}$ **do**
- 7 $\rho_{max} = \text{propagateMaxPath}(\Pi_{\mathcal{X}}^{ctl}, \rho_{max}, max_{m_j}^{ctl})$
- 8 **if** v_j is a convergent time point ($v_j \in \mathcal{V}_{cv}$) **then**
- 9 **if** v_j not in $D_{max}(v_d)$ **then**
- 10 add $v_j \rightarrow [\rho_{max}]$ in $D_{max}(v_d)$
- 11 **else**
- 12 **if** ρ_{max} is a restrictive path in $\mathcal{P}_{v_j}^{max}$ **then**
- 13 add ρ_{max} to $\mathcal{P}_{v_j}^{max}$ and to $\mathcal{P}(v_d)$
- 14 **else**
- 15 add ρ_{max} to $\mathcal{P}(v_d)$
- 16 *A second forward search for $D_{min}(v_d)$ with ρ_{min}*
- 17 **return** checkCycles($D_{max}(v_d)$, $D_{min}(v_d)$)

■ **Algorithm 2** checkCycles algorithm.

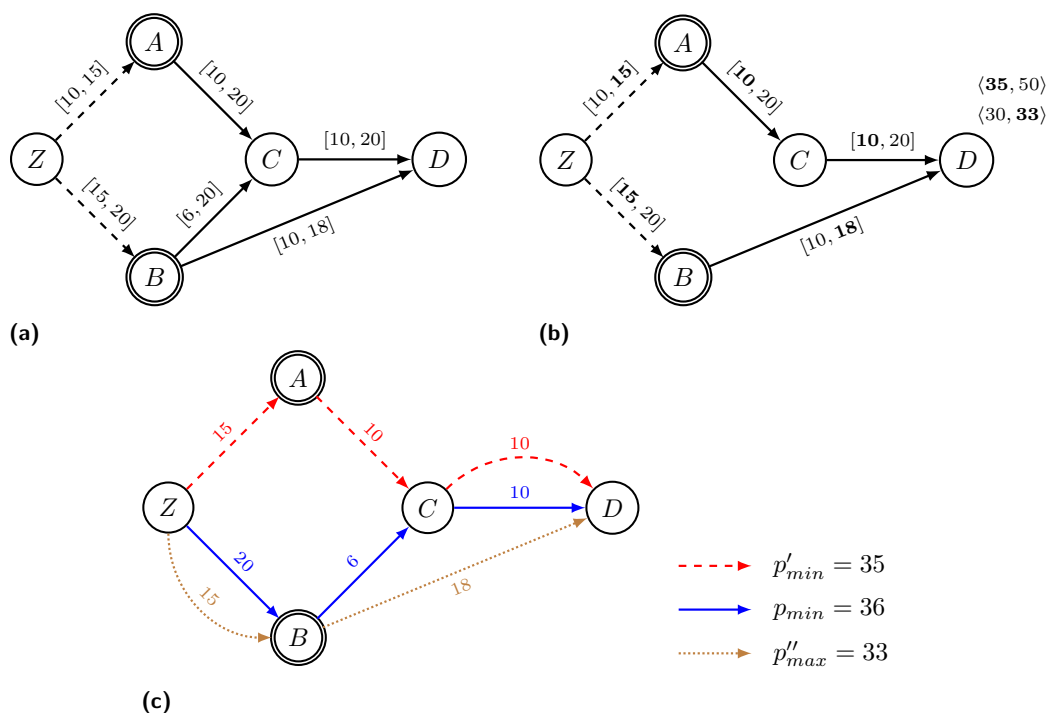
Input: $D_{max}(v_d)$, $D_{min}(v_d)$
Output: Boolean

- 1 **for each** $v_c \rightarrow \mathcal{P}_{v_c}^{min}$ in $D_{min}(v_d)$ **do**
- 2 **for each** ρ_{min} in $\mathcal{P}_{v_c}^{min}$ **do**
- 3 **for each** ρ_{max} in $\mathcal{P}_{v_c}^{max}$ in $D_{max}(v_d)$ **do**
- 4 **if** (ρ_{min}, ρ_{max}) is of the form M^{ctl} **then**
- 5 **if** $min_{\rho}^{ctl} > max_{\rho}^{ctl}$ **then**
- 6 **return** False *Or the cycle*
- 7 **return** True

■ **Algorithm 3** WC-Checking algorithm.

Input: \mathcal{X} : STNU($\mathcal{V}, \mathcal{E}, \mathcal{C}$)
Output: Boolean

- 1 $\Pi_{\mathcal{X}}^{ctl} = \text{getDistanceGraph}(\mathcal{X})$
- 2 rank = orderFromRank(\mathcal{X})
- 3 **for each** v_d in \mathcal{V}_{dv} **do**
- 4 **if** findDivergentCycles(v_d , $\Pi_{\mathcal{X}}^{ctl}$, rank) == False **then**
- 5 **return** False *Or non-WC cycles of v_d*
- 6 **return** True *Or all non-WC cycles*



■ **Figure 2** This figure illustrates the special case of the pruning rules when C_{ρ_η} and $C_{\rho'_\eta}$ are not empty. Figure a) shows a non-Weakly controllable STNU, whereas Figure b) shows its only non-WC cycle. Figure c) highlights the computed projection paths p_{min} , p'_{min} , and p''_{max} of the given example. One can see that if p'_{min} is not kept in $D_{min}(v_d)$, the inconsistent cycle will never be checked as the pair $\langle p_{min}, p''_{max} \rangle$ do not form a cycle M^{ctl} . Hence, such pruning rules cannot be applied when C_{ρ_η} and $C_{\rho'_\eta}$ are not empty.

We show the execution of our algorithm for Divergent time-point A in Figure 3 using $A \rightarrow C \rightarrow B \rightarrow D$ as the order for the forward searches. We highlight the search and the paths (min and max) forming the non-Weakly controllable cycle. Please note that we simplified the example by not showing how D_{min} and D_{max} are incrementally changed.

6.2 Features and Complexity

The algorithm presented in the previous section returns the set of negative cycles of a non-Weakly controllable STNU (see Algorithms 2, 3), which is important for explainability, i.e., necessary for the repair problem. Moreover, divergent time-points are independent, which makes parallelization possible. In addition, the usual pseudo-controllability step from Morris [10] is not required for constraint bounds with finite values ($l_{ij} \neq -\infty$ and $u_{ij} \neq +\infty$). Thus, an incremental execution is possible as divergent time-points are independent. Indeed, when adding new constraints, it's not necessary to recompute the minimal network; hence, checking only the cycles of divergent time points of the same rank or lesser (topological ordering) is enough to guarantee WC. Still, it is not optimal as unnecessary cycles might be checked. The drawback of the algorithm is that the minimal network is not computed.

The temporal complexity of the algorithm depends on the number of cycles to check, which is related to multiple parameters such as the number of contingents, the number of divergent time-points, and the number of successors per divergent time-point. For a complete graph, the algorithm is exponential and not better than the original algorithm

8:12 A More Efficient and Informed Algorithm to Check Weak Controllability

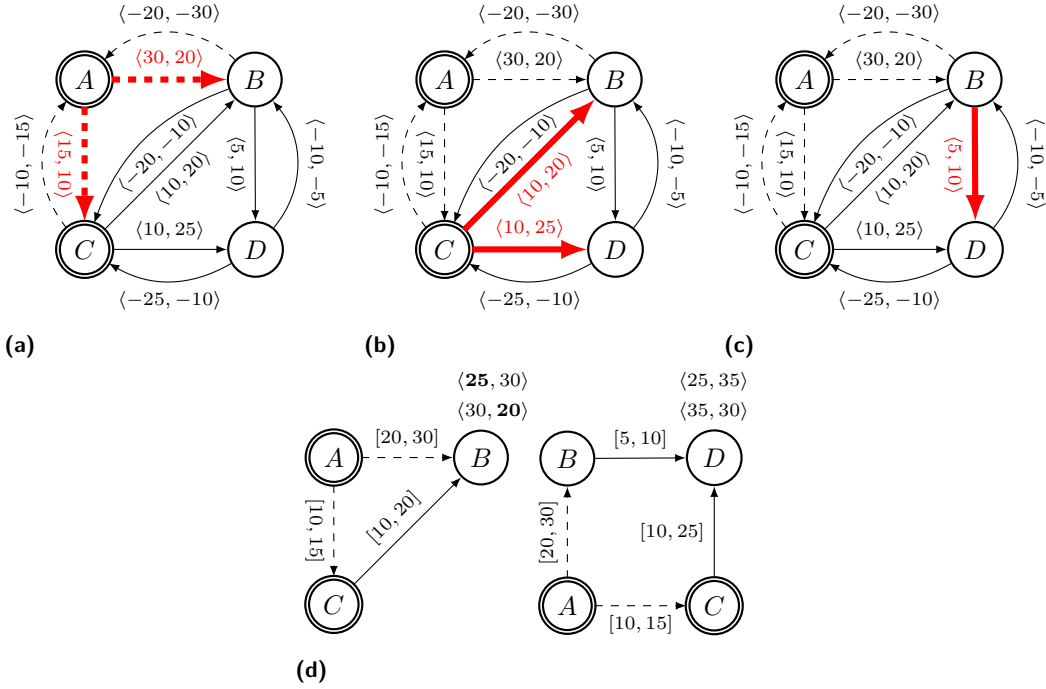


Figure 3 This Figure shows, in a simplified manner, a running example of Algorithm 1 with divergent time-point A. We highlight the edges taken at each step according to line 7. Figures 3a to 3c show the search, while Figure 3d shows the cycles to check for A, with the left one being not Weakly controllable. After step 3, D_{min} and D_{max} contain all the restrictive paths (only those that need to be kept) with, in a simplified manner, $D_{min} = \{C : \langle 15, AC \rangle, B : [\langle 20, AB \rangle, \langle 25, ACB \rangle], D : [\langle 15, ACD \rangle, \langle 35, ABD \rangle, \langle 30, ACBD \rangle]\}$ and $D_{max} = \{C : \langle 10, AC \rangle, B : [\langle 20, AB \rangle, \langle 30, ACB \rangle], D : [\langle 25, ACD \rangle, \langle 30, ABD \rangle, \langle 40, ACBD \rangle]\}$. We highlight the paths of the non-Weakly controllable cycle.

$(2^{|C|})$. However, our interest lies in realistic graphs where the sparsity of the graph is low by restricting these parameters. Thus, the next section compares our algorithm (new_WC) with the original one (old_WC) using the Floyd-Warshall algorithm (APSP) as a time metric only to see how close they are to a polynomial behavior when parameters are restricted enough.

7 Experiments

To empirically test the effectiveness of the proposed algorithm, we consider the execution time as the execution of all computations and not after finding an inconsistency as existing checking algorithms do. The benchmark comes from a random generator we implemented that can generate sparse STNUs. It creates an STNU in the form of a complete directed acyclic graph (DAG), then randomly removes several edges depending on parameters: the number of time points n , the percentage of divergent time points r_d , the maximum number of their successors n_c , and the percentage of contingent constraints r_c .

All the experiments have been performed on a machine equipped with an Intel Core processor: 11th Gen Intel(R) Core(TM) i7-11850H @ 2.50GHz 2.50 GHz. We used a time/memory limit of 10 minutes/4GB and sequential, single-core computation.

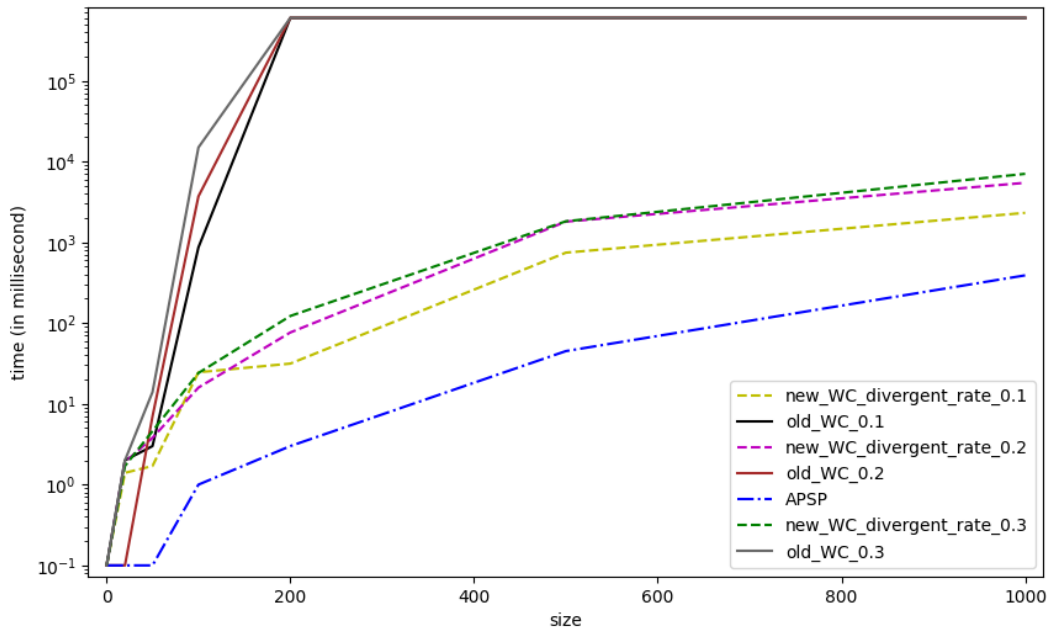
We experiment under different settings: $n = \{20, 50, 100, 200, 500, 1000\}$, $r_d = \{0.1, 0.2, 0.3\}$ meaning 10 to 30% of divergent time-points, $r_c = \{0.2, 0.3\}$, and $n_c = 3$. For each combination of parameters, we generate 20 STNUs and compute the average exe-

cution time. We show in Figure 4b that, in general, our algorithm clearly outperforms the *old-WC* algorithm and has a behavior slightly worse than the APSP algorithm up to 20% of contingent constraints. This shows that the parameters were bounded enough to have a polynomial-like behavior. However, beyond this threshold, our algorithm starts to show its limit. This shows the sensitivity of our algorithm to the parameters (see Figure 4c). In addition, we observe from the experimentation that the position of contingents can impact the number of cycles to check. The closer to v_0 contingents are, the higher the number of cycles to check. Such a case is shown in Figure 4a where the dotted line for the case of 10% of divergent time-points (*new-WC*) overlaps the other two (20 and 30 %).

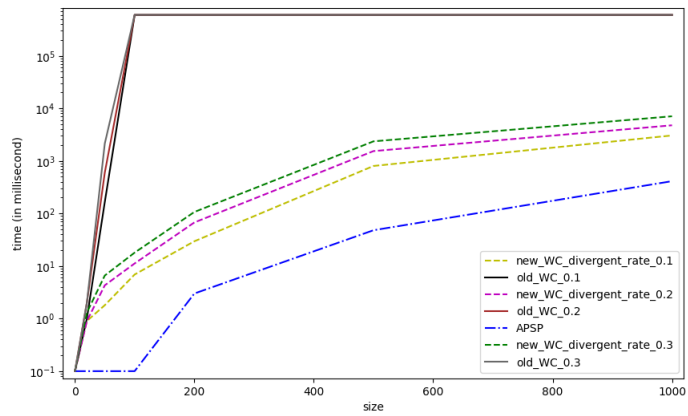
8 Conclusion

This paper introduced a novel approach for checking the WC of an STNU by checking the consistency of its elementary cycles. Interesting features of our algorithm to consider further are as follows: it can identify the constraints causing the uncontrollability, and it can be executed in an incremental way (not optimal) and in a parallelized way. However, it is not capable of computing the minimal network of an STNU. Moreover, we exhibited that the algorithm's complexity depends on the sparsity of the STNU, which makes it exponential in the worst cases. However, experiments show that in loosely connected STNU, the algorithm tends to behave in a polynomial-like way. Finally, the paper argues the relevance of the problem of WC in a multi-agent setting, where uncontrollable events are not controlled by Nature but by other agents in the system. Further work will tackle the problem of repairing negative cycles by negotiating the duration of the uncontrollable events, whose duration depends on the other agents' decisions.

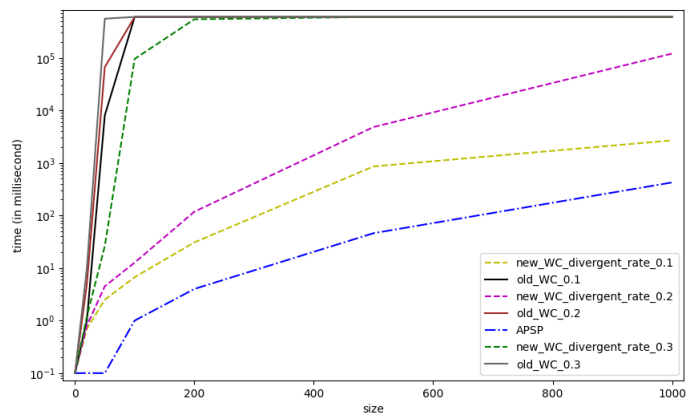
8:14 A More Efficient and Informed Algorithm to Check Weak Controllability



(a)



(b)



(c)

■ **Figure 4** Experimentation with 10% (a), 20% (b), and 30% (c) of contingent constraints.

References

- 1 Shyan Akmal, Savana Ammons, Hemeng Li, and James C Boerkoel Jr. Quantifying degrees of controllability in temporal networks with uncertainty. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2019.
- 2 Shyan Akmal, Savana Ammons, Hemeng Li, Michael Gao, Lindsay Popowski, and James C. Boerkoel. Quantifying controllability in temporal networks with uncertainty. *Artificial Intelligence*, 2020.
- 3 Arthur Bit-Monnot and Paul Morris. Dynamic controllability of temporal plans in uncertain and partially observable environments. *J. Artif. Intell. Res.*, 2023. doi:10.1613/JAIR.1.13065.
- 4 Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Solving temporal problems using smt: weak controllability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2012.
- 5 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 1991.
- 6 Malik Ghallab and A. Mounir Alaoui. Managing efficiently temporal relations through indexed spanning trees. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, pages 1297–1303. Morgan Kaufmann, 1989. URL: <http://ijcai.org/Proceedings/89-2/Papers/072.pdf>.
- 7 Luke Hunsberger and Roberto Posenato. Speeding up the rul dynamic-controllability-checking algorithm for simple temporal networks with uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.
- 8 Jsen-Shung Lin, Chin-Chia Jane, and John Yuan. On reliability evaluation of a capacitated-flow network in terms of minimal pathsets. *Networks*, 1995. doi:10.1002/NET.3230250306.
- 9 Josef Lubas, Marco Franceschetti, and Johann Eder. Resolving conflicts in process models with temporal constraints. In *Proceedings of the ER Forum and PhD Symposium*, 2022.
- 10 Paul Morris. A structural characterization of temporal dynamic controllability. In *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of *Lecture Notes in Computer Science*, pages 375–389. Springer, 2006. doi:10.1007/11889205_28.
- 11 Paul Morris. Dynamic controllability and dispatchability relationships. In *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings*. Springer, 2014. doi:10.1007/978-3-319-07046-9_33.
- 12 Paul H. Morris and Nicola Muscettola. Managing temporal uncertainty through waypoint controllability. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 1253–1258. Morgan Kaufmann, 1999. URL: <http://ijcai.org/Proceedings/99-2/Papers/083.pdf>.
- 13 Ajdin Sumic, Alessandro Cimatti, Andrea Micheli, and Thierry Vidal. SMT-based repair of disjunctive temporal networks with uncertainty: Strong and weak controllability. In *Proceedings of the The 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2024)*, 2024.
- 14 Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.*, 11(1):23–45, 1999. doi:10.1080/095281399146607.

A Faster Algorithm for Finding Negative Cycles in Simple Temporal Networks with Uncertainty

Luke Hunsberger   

Vassar College, Poughkeepsie, NY, USA

Roberto Posenato   

University of Verona, Italy

Abstract

Temporal constraint networks are data structures for representing and reasoning about time (e.g., temporal constraints among actions in a plan). Finding and computing negative cycles in temporal networks is important for planning and scheduling applications since it is the first step toward resolving inconsistent networks. For Simple Temporal Networks (STNs), the problem reduces to finding *simple* negative cycles (i.e., no repeat nodes), resulting in numerous efficient algorithms. For Simple Temporal Networks with Uncertainty (STNUs), which accommodate actions with uncertain durations, the situation is more complex because the characteristic of a *non-dynamically controllable* (non-DC) network is a so-called *semi-reducible negative* (SRN) cycle, which can have repeat edges and, in the worst case, an exponential number of occurrences of such edges. Algorithms for computing SRN cycles in non-DC STNUs that have been presented so far are based on older, less efficient DC-checking algorithms. In addition, the issue of repeated edges has either been ignored or given scant attention. This paper presents a new, faster algorithm for identifying SRN cycles in non-DC STNUs. Its worst-case time complexity is $O(mn + k^2n + kn \log n)$, where n is the number of timepoints, m is the number of constraints, and k is the number of actions with uncertain durations. This complexity is the same as that of the fastest DC-checking algorithm for STNUs. It avoids an exponential blow-up by efficiently dealing with repeated structures and outputting a compact representation of the SRN cycle it finds. The space required to compactly store accumulated path information while avoiding redundant storage of repeated edges is $O(mk + k^2n)$. An empirical evaluation demonstrates the effectiveness of the new algorithm on an existing benchmark.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning; Theory of computation → Dynamic graph algorithms

Keywords and phrases Temporal constraint networks, overconstrained networks, negative cycles

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.9

Supplementary Material *Software (Source code)*: <https://profs.scienze.univr.it/~posenato/software/cstnu/> [17]

1 Introduction

A Simple Temporal Network with Uncertainty (STNU) is a data structure for representing and reasoning about time [14]. STNUs are attractive for planning and scheduling applications because they accommodate not only a wide variety of temporal constraints (e.g., duration constraints, deadlines, and inter-action constraints), but also actions with uncertain durations (e.g., taxi rides or battery-charging actions) [5, 15, 6, 11, 18]. In STNUs, actions with uncertain durations are represented by *contingent links*. Each STNU has a graphical form where nodes represent timepoints; labeled, directed edges represent temporal constraints; and additional edges (called LC and UC edges) represent bounds on uncontrollable action durations.

The most important property of an STNU is called *dynamic controllability* (DC). An STNU is DC if there exists a dynamic strategy for executing its controllable timepoints that guarantees that all relevant constraints will be satisfied no matter how the uncertain durations



© Luke Hunsberger and Roberto Posenato;

licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 9; pp. 9:1–9:15

Leibniz International Proceedings in Informatics



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

turn out – within their specified bounds. There are many polynomial-time algorithms, called DC-checking algorithms, for determining whether any given STNU is DC. The fastest is the $O(mn + k^2n + kn \log n)$ -time RUL^- algorithm due to Cairo *et al.* [3], where n is the number of timepoints; m , the number of constraints; and k , the number of contingent links. Hunsberger and Posenato subsequently presented a modification of RUL^- , called RUL2021 , that has the same worst-case complexity, but is an order of magnitude faster in practice [10].

The characteristic feature of a non-DC STNU is that it must contain a *semi-reducible negative* (SRN) cycle [12]. In general, any path from X to Y in an STNU graph is semi-reducible if it entails a path of the same length from X to Y that contains no LC edges. Such entailments can be discovered by generating new edges using constraint-propagation (equivalently, edge-generation) rules. Although finding negative cycles in Simple Temporal Networks (STNs) reduces to finding *simple* negative cycles (i.e., no repeat nodes), finding SRN cycles in STNUs is more complex, given that even *indivisible* SRN cycles in a non-DC STNU can have repeat edges and, in the worst case, an *exponential* number of such edges [9]. (An SRN cycle is indivisible if each proper sub-cycle is non-negative or non-semi-reducible.)

When given a *non-DC* STNU, DC-checking algorithms simply report that the network is not DC; they do not produce an SRN cycle [12, 13, 3, 10]. For applications, it is important to identify SRN cycles so that they can be resolved (e.g., by accepting the cost of weakening constraints or tightening uncertain durations). Existing algorithms for finding SRN cycles in non-DC STNUs [22, 23, 21, 1, 2] are based on older, less efficient DC-checking algorithms; and the issue of repeated edges has been ignored or given scant attention. This paper presents a new, faster algorithm for computing SRN cycles in non-DC STNUs while also rigorously addressing the compact representation of SRN cycles having a large number of repeated edges. The new algorithm modifies the RUL2021 algorithm to accumulate path information without impacting its time complexity. The additional space required to compactly store path information, while avoiding redundant storage of repeated edges, is $O(mk + k^2n)$.

2 Background

This section summarizes the basic definitions and results for STNUs and then describes the RUL2021 DC-checking algorithm that is the starting point for our new algorithm.

2.1 Simple Temporal Networks with Uncertainty

A Simple Temporal Network with Uncertainty (STNU) is a triple $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ where \mathcal{T} is a set of n real-valued variables; \mathcal{C} is a set of m binary difference constraints, each of the form $Y - X \leq \delta$, where $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$; and \mathcal{L} is a set of k *contingent links*, each of the form (A, x, y, C) , where $A, C \in \mathcal{T}$ and $0 < x < y < \infty$ [14]. The timepoints typically represent starting or ending times of actions; the constraints can represent deadlines, release times, and duration or inter-action constraints. The contingent links represent actions with *uncertain* durations. For each contingent link (A, x, y, C) , A is called the *activation* timepoint and C the *contingent* timepoint. We let $\Delta_C = y - x$. The executor of the network typically controls A , but not C . The executor only *observes* the execution of C in real-time, knowing only that C will be executed such that $C - A \in [x, y]$. For example, your taxi ride might be represented by the contingent link $(A, 15, 25, C)$, where A is when you enter the taxi, C is when you arrive at your destination, and $C - A \in [15, 25]$ is the uncertain duration, learned only when you arrive.

Each STNU has a graph $(\mathcal{T}, \mathcal{E})$ where the timepoints serve as nodes and the constraints in \mathcal{C} and the contingent links in \mathcal{L} correspond to different kinds of labeled, directed edges. For convenience, edges such as $X \xrightarrow{\alpha} Y$ will be notated as (X, α, Y) , where $X, Y \in \mathcal{T}$ and $\alpha \in \mathbb{R}$, possibly annotated with an alphabetic letter. In particular, $\mathcal{E} = \mathcal{E}_o \cup \mathcal{E}_\ell \cup \mathcal{E}_u$, where: $\mathcal{E}_o = \{(X, \delta, Y) \mid (Y - X) \leq \delta \in \mathcal{C}\}$ is the set of *ordinary* edges; $\mathcal{E}_\ell = \{(A, c:x, C) \mid (A, x, y, C) \in \mathcal{L}\}$, the set of lower-case (LC) edges; and $\mathcal{E}_u = \{(C, C:-y, A) \mid (A, x, y, C) \in \mathcal{L}\}$, the set of upper-case (UC) edges. Note that each contingent link has a corresponding pair of edges: an LC edge representing that the contingent duration might take on its minimum value x , and a UC edge representing that it might take on its maximum value y .

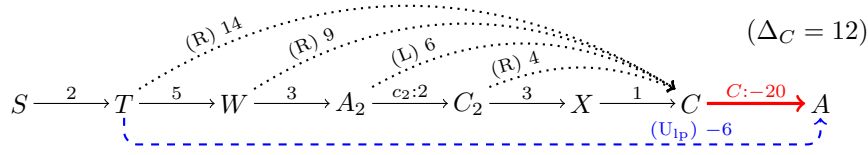
\mathcal{T}_c denotes the set of contingent timepoints; and $\mathcal{T}_x = \mathcal{T} \setminus \mathcal{T}_c$ the set of executable (or controllable) timepoints. An STNU is *dynamically controllable* (DC) if there exists a dynamic strategy for executing its controllable timepoints such that all constraints in \mathcal{C} will necessarily be satisfied no matter how the contingent durations turn out within their specified bounds [14, 7]. An execution strategy is *dynamic* if it can react, in real-time, to observations of contingent executions. The RUL^- algorithm [3] is the DC-checking algorithm with the best worst-case time complexity: $O(mn + k^2n + kn \log n)$. However, RUL2021 , which is a modification of RUL^- , has been shown to be an order-of-magnitude faster on a variety of STNU benchmarks, although having the same theoretical complexity [10].

2.2 The RUL2021 DC-Checking algorithm

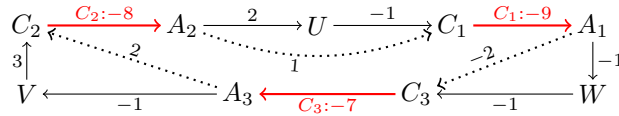
This section summarizes important features of the RUL2021 algorithm. Like all DC-checking algorithms, it operates on the STNU graph, using edge-generation rules to generate new edges representing constraints that must be satisfied by any dynamic execution strategy. Table 1 shows the edge-generation rules used by RUL2021. The R and L rules (for *Relax* and *Lower Case*, respectively) are used to back-propagate distance information in the LO-graph (i.e., the subgraph comprising the LC and ordinary edges). The wavy arrows represent paths in the LO-graph that have already been explored. In the R rule, back-propagation continues along the ordinary edge (P, v, Q) to generate the distance information represented by the dotted edge $(P, v + w, C_i)$. In the L rule, back-propagation continues along the LC edge $(A_j, c_j:x_j, C_j)$ to generate the distance information represented by the dotted edge $(A_j, x_j + w, C_i)$. RUL2021 uses the L and R rules only to accumulate distance information; the dotted edges are *not* inserted into the STNU graph. But RUL2021 *does* insert the edges generated by the U_{lp} rule: ordinary edges that effectively *bypass* UC edges. For example, in the table, the wavy path (P, v, C_i) represents distance information previously generated by the R and L rules. This “edge” combines with the UC edge $(C_i, C_i:-y_i, A_i)$ to generate the (blue and dashed) bypass edge $(P, v - y_i, A_i)$.

■ **Table 1** The edge-generation rules for the RUL2021 algorithm.

Rule	Graphical representation	Applicability Conditions
R	$P \xrightarrow{v} Q \xrightarrow{w} C_i$ $\xrightarrow{v+w}$	$Q \in \mathcal{T}_x, w < \Delta_{C_i}, C_i \in \mathcal{T}_c$
L	$A_j \xrightarrow{c_j:x_j} C_j \xrightarrow{w} C_i$ $\xrightarrow{x_j+w}$	$C_j \neq C_i, w < \Delta_{C_i}, C_i \in \mathcal{T}_c$
U_{lp}	$P \xrightarrow{v} C_i \xrightarrow{C_i:-y_i} A_i$ $\xrightarrow{v-y_i}$	$(A_i, x_i, y_i, C_i) \in \mathcal{L}, v \geq \Delta_{C_i}$



■ **Figure 1** RUL2021 generating a (blue and dashed) bypass edge for a (red) UC edge.



■ **Figure 2** A cycle of interruptions detected by RUL2021.

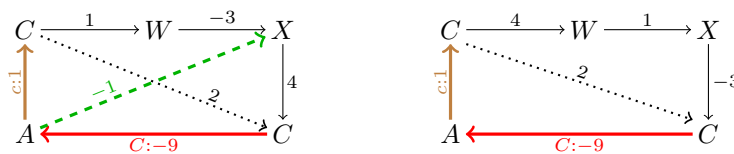
Figure 1 shows how RUL2021 processes a (red) UC edge, assuming that $\Delta_C = 12$. First, it uses the R and L rules to back-propagate from C along LO-edges, collecting distance information indicated by the dotted arrows. (The rules used to generate these edges are in parentheses.) Back-propagation continues as long as the distance stays less than Δ_C . Since the path from T to C has length $14 \geq \Delta_C$, back-propagation stops. Then the U_{ip} rule is applied to $(T, 14, C)$ and $(C, C:-20, A)$ to generate the (dashed) bypass edge $(T, -6, A)$.

There can be many paths emanating backward from C in the LO-graph. To ensure that only *shortest* distances are accumulated, back-propagation is guided by a priority queue and a potential function to re-weight the edges to non-negative values, as in Johnson’s algorithm [4], except that here the potential function is a solution to the LO-graph, viewed as an STN. The potential function is initialized by a one-time call to the Bellman-Ford algorithm [4].

Once all of the bypass edges are computed for a given UC edge, they are inserted into the LO-graph, which typically requires updating the potential function. Since all of the new edges terminate at A , this updating can be carried out by a separate Dijkstra-like back-propagation from A using a priority queue and the pre-existing potential function. If all of the UC edges can be successfully processed in this way, then RUL2021 declares the STNU to be DC.

However, three kinds of events can signal that the STNU is not DC:

1. **Failure to update the potential function.** Inserting new bypass edges might cause the LO-graph to become inconsistent (as an STN), which would be detected by encountering a negative cycle in the LO-graph while trying to update the potential function.
2. **Cycle of interruptions.** When processing a UC edge E_1 , back propagation from its contingent timepoint C_1 might bump into a different UC edge E_2 . If so, RUL2021 *interrupts* its processing of E_1 to process E_2 . After finishing with E_2 , back propagation from E_1 continues. However, should a *cycle* of such interruptions occur, for example, as illustrated in Figure 2, then the network cannot be DC. In the figure, the UC edges are colored red; distances along LO-paths computed by back-propagation using the L and R rules are indicated by dotted arrows; and the relevant contingent links are $(A_1, 1, 9, C_1)$, $(A_2, 2, 8, C_2)$ and $(A_3, 3, 7, C_3)$. Now back propagation from C_1 should continue as long as the dotted distances are less than $\Delta_{C_1} = 9 - 1 = 8$, but is interrupted by the UC edge $(C_2, C_2:-8, A_2)$. Similarly, back-propagation from C_2 should continue as long as the dotted distances are less than $\Delta_{C_2} = 8 - 2 = 6$, but is interrupted by the UC edge $(C_3, C_3:-7, A_3)$. Finally, back propagation from C_3 should continue as long as the dotted distances are less than $\Delta_{C_3} = 7 - 3 = 4$, but is interrupted by the first UC edge,



■ **Figure 3** Two different CC loops associated with a contingent link $(A, 1, 9, C)$.

■ **Algorithm 1** The FindSRNC algorithm.

Input: $\mathcal{G} = (\mathcal{T}, \mathcal{E} = \mathcal{E}_o \cup \mathcal{E}_\ell \cup \mathcal{E}_u)$, an STNU graph
Output: $(negCycle, edgeAnn)$, where $negCycle$ is an SRN cycle and $edgeAnn$ is a hash table of path annotations for edges in the cycle; or (\emptyset, \emptyset) if the STNU is DC

```

1 glo := new global data structure           // Fields: pf, status, intBy, edgeAnnotation
2 glo.pf := BellmanFord( $\mathcal{G}_{\ell o}$ )         // Initialize potential function for LO-graph
3 if glo.pf ==  $\perp$  then return BFCT( $\mathcal{G}_{\ell o}$ )
4 glo.edgeAnnotation := new empty hash table
5 glo.status := [nYet, ..., nYet]           // Initial processing status of the  $k$  UC edges
6 glo.intBy := [ $\perp$ , ...,  $\perp$ ]              //  $k$ -vector: records interruptions
7 foreach  $(C, C:-y, A) \in \mathcal{E}_u$  do
8    $negCycle := \text{RulBackProp}(\mathcal{G}, (C, C:-y, A), glo)$ 
9   if  $negCycle \neq \emptyset$  then return  $(negCycle, glo.edgeAnnotation)$ 
10 return  $(\emptyset, \emptyset)$ 

```

thereby completing the cycle. At this point, RUL2021 signals that the STNU is not DC. This is justified since each dotted distance being less than the corresponding Δ_{C_i} value ensures that the length of the cycle is negative; and a negative cycle in the OU-graph (i.e., the subgraph comprising ordinary and UC edges) represents an impossible-to-satisfy constraint for a dynamic execution strategy.

3. **CC loops.** Back propagation from a UC edge $(C, C:-y, A)$ can also be blocked if an LO-path from C back to C of length less than Δ_C is encountered. Such a path is called a *CC loop* [10]. A CC loop does not necessarily imply that the STNU is not DC; but it sometimes does. Figure 3 illustrates two scenarios in which back-propagation from C reveals a CC loop of length $2 < 8 = \Delta_C$. However, the lefthand STNU is not DC, while the righthand one is. The key difference, according to Morris' analysis of semi-reducible paths [12], is that the lefthand graph contains a *negative* LO-path emanating from C (to X) which can be used to generate the (dashed, green) bypass edge $(A, -1, X)$, thereby creating a negative cycle in the OU-graph from A to X to C to A , whereas the righthand graph has no such path.

3 The FindSRNC (Find Semi-Reducible Negative Cycle) Algorithm

This section introduces our new FindSRNC algorithm, which modifies RUL2021 to efficiently accumulate path information. To contrast FindSRNC and RUL2021, we have preserved the general structure of RUL2021, although to improve readability we have expanded the rather cryptic names of the original helper algorithms. Modifications are highlighted in green.

The pseudocode for FindSRNC is in Algorithm 1. When given a non-DC STNU as input, it outputs a compact representation of an SRN cycle in the form $(negCycle, edgeAnn)$, where $negCycle$ is a negative cycle of edges in the LO- or OU-graph, depending on how the cycle arose; and $edgeAnn$ is a hash table of $(key, value)$ pairs, where each *key* identifies an (ordinary)

bypass edge generated by the algorithm, and *value* is the path used to generate that edge. It is efficient to present the SRN cycle in this way since, in the worst case, unpacking all of the edges in the cycle could result in an exponential number of repeated edges.

Like RUL2021, `FindSRNC` starts by calling the Bellman-Ford algorithm to create an initial potential function for the LO-graph which, if successful, is stored in the `pf` field of a `glo` data structure. (The `glo` data structure contains `global` information accessible across multiple recursive calls to process UC edges.) If Bellman-Ford fails, then `FindSRNC` calls the $O(mn)$ -time BFCT algorithm [19] to return a negative cycle for the LO-graph (an STN). A negative cycle in the LO-graph is a trivial case of an SRN cycle for an STNU.

If Bellman-Ford succeeds, `FindSRNC` initializes `glo.edgeAnnotation` to a new hash table that will record the paths from which any bypass edges are derived. The `glo.status` field tracks the processing status of each UC edge, as in RUL2021. The `glo.intBy` field, initially a vector of \perp entries, stores information about when the processing of one UC edge is interrupted by another. Finally, `FindSRNC` iterates through the UC edges, processing each with a call to `RulBackProp` (Algorithm 2). Because `RulBackProp` recursively processes any interrupting UC edges, by the time `FindSRNC` calls `RulBackProp` on some UC edge, it may have already been processed. The `status` field is used to avoid redundant processing.

RulBackProp

The pseudocode for the `RulBackProp` algorithm is in Algorithm 2. It processes a single UC edge $\mathbf{E} = (C, C:-y, A)$ while integrating the recursive processing of any interrupting UC edges. At Line 2, if \mathbf{E} has already been processed, it immediately returns \top . At Line 3, it checks whether the processing of \mathbf{E} has already been started, but not yet completed, which implies a negative cycle of interruptions. In this case, `RulBackProp` calls `AccNegCycle` (Algorithm 3) to collect the relevant path information accumulated in the `glo.intBy` vector, which is then returned as a compact representation of an SRN cycle. (More will be said about how the information in `glo.intBy` is generated.)

In the pseudocode, we use $+$ as a concatenation operator that can be applied to edges or paths. For example, if e_1 is an edge, and π_1 and π_2 are paths, then $\pi_1 + e_1 + \pi_2$ represents their concatenation into a single path. In addition, we use $\langle \rangle$ to denote the empty path.

At Lines 4–7, `RulBackProp` prepares to process a UC edge $\mathbf{E} = (C, C:-y, A)$. As in RUL2021, `ccLoop` is a flag used to signal the discovery of a *CC loop*; and `dist` records, for each encountered timepoint X , the distance from X to C in the LO-graph. A new field, `path`, records the paths from each X to A (via C). Back-propagation from C is governed by a priority queue \mathcal{Q} , initialized at Lines 8–10 to include each X connected to C by an edge.

In each iteration of the `while` loop (Lines 12–23), `RulBackProp` either starts *or resumes* the processing of \mathbf{E} , first (at Line 13) by calling `TryBackProp` (Algorithm 4). `TryBackProp` (described later) back-propagates along LO-edges, but does *not* generate or insert any bypass edges. Instead, it simply collects the relevant distance *and path* information, while also keeping track of whether it encountered any unstarted (i.e., interrupting) UC edges or *CC loops*. At Line 14, `RulBackProp` checks whether `TryBackProp` found an SRN cycle, in which case `RulBackProp` returns that cycle. Otherwise, at Line 15, `RulBackProp` checks whether `TryBackProp` encountered any interrupting UC edges. If so, for each interrupting UC edge \mathbf{E}_X (Lines 16–19), it uses `glo.intBy` [\mathbf{E}] to record the interruption and then attempts to recursively process \mathbf{E}_X . If all interrupting UC edges are successfully processed, it clears the `glo.intBy` [\mathbf{E}] entry (Line 20) and prepares for the next iteration of the `while` loop by re-initializing the priority queue (Lines 21–22) so that processing \mathbf{E} can be resumed, starting from the activation timepoints of the no-longer-interrupting UC edges.

■ **Algorithm 2** The RulBackProp algorithm.

Input: $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, STNU graph; $\mathbf{E} = (C, C:-y, A) \in \mathcal{E}_u$, a UC edge; \mathbf{glo} , a *global* structure

Output: *negCycle*, an SRN cycle; or \emptyset if \mathbf{E} successfully processed

```

1  $h := \mathbf{glo.pf}$  // Potential function for LO-graph
2 if  $\mathbf{glo.status}[\mathbf{E}] == \text{done}$  then return  $\top$  //  $\mathbf{E}$  already done
3 if  $\mathbf{glo.status}[\mathbf{E}] == \text{started}$  then return  $\text{AccNegCycle}(\mathbf{glo.intBy}, \mathbf{E})$  // Cycle of interrupts
4  $\mathbf{glo.status}[\mathbf{E}] := \text{started}$  // Prepare to start processing the UC edge  $\mathbf{E}$ 
5  $\text{loc} := \text{new local struct}; \text{loc.ccLoop} := \perp$  // No CC loop found yet
6  $\text{loc.dist} := [\infty, \dots, \infty]$  // distance from each TP to  $C$ 
7  $\text{loc.path} := [\langle \rangle, \dots, \langle \rangle]$  // path from each TP to  $A$  (via  $\mathbf{E}$ )
8  $\mathcal{Q} := \text{a new priority queue}$  // Priority of each  $X$  is  $h(X) + \delta_{xc}$ , adjusted dist. from  $X$  to  $C$ 
9 foreach  $(X, \delta_{xc}, C) \in \mathcal{E}_o$  do
10    $\mathcal{Q.ins}(X, h(X) + \delta_{xc}); \text{loc.path}[X] := (X, \delta_{xc}, C) + (C, C:-y, A)$ 
11  $\text{continue?} := \top$ 
12 while  $\text{continue?}$  do // Start or resume processing of UC edge  $\mathbf{E}$ 
13    $\text{negCycle} := \text{TryBackProp}(\mathcal{G}, \mathbf{E}, \mathcal{Q}, \mathbf{glo}, \text{loc})$ 
14   if  $\text{negCycle} \neq \emptyset$  then return  $\text{negCycle}$ 
15   if  $\text{loc.UnstartedUCs} \neq \emptyset$  then // Process unstarted UC-edges
16     foreach  $(\mathbf{E}_X, X) \in \text{loc.UnstartedUCs}$  do
17        $\mathbf{glo.intBy}[\mathbf{E}] := (\mathbf{E}_X, \text{loc.path}[X])$ 
18        $\text{negCycle} := \text{RulBackProp}(\mathcal{G}, \mathbf{E}_X, \mathbf{glo})$ 
19       if  $\text{negCycle} \neq \emptyset$  then return  $\text{negCycle}$ 
20      $\mathbf{glo.intBy}[\mathbf{E}] := \perp$  // All interruptions of  $\mathbf{E}$  completed
21      $\mathcal{Q.clear}()$  // Prepare  $\mathcal{Q}$  for next iteration of WHILE
22     foreach  $(\mathbf{E}_X, X) \in \text{loc.UnstartedUCs}$  do  $\mathcal{Q.ins}(X, \text{loc.dist}[X] + \mathbf{glo.pf}[X])$ 
23   else  $\text{continue?} := \perp$  // Back-prop. from  $\mathbf{E}$  completed
24 if  $\text{loc.ccLoop}$  then // CC-loop found; must initiate forward propagation
25    $(X, \mathcal{P}_X) := \text{FwdPropNDC}(\mathcal{G}, C, \Delta_C, \text{loc}, \mathbf{glo.pf})$  //  $\Delta_C = y - x$  for cont. link  $(A, x, y, C)$ 
26   // If  $(A, c:x, C)$  can be reduced away, then return SRN cycle
27   if  $(X, \mathcal{P}_X) \neq \emptyset$  then return  $(A, c:x, C) + \mathcal{P}_X + \text{loc.path}[X]$ 
28 foreach  $X \in \mathcal{T} \setminus \{C\}$  do // Generate bypass edges using  $U_{lp}$  rule
29    $\delta_{xc} := \text{loc.dist}[X]$  //  $\delta_{xc} = \infty$  means node not reachable
30   if  $\Delta_C \leq \delta_{xc} < \infty$  then
31      $\mathcal{G.insertOrdEdge}(X, \delta_{xc} - y, A)$ 
32      $\mathbf{glo.edgeAnnotation.put}((X, A), \text{loc.path}[X])$ 
33      $\text{edges?} := \top$ 
34 if  $\text{edges?}$  then  $(\mathbf{glo.pf}, \text{negCycle}) := \text{UpdatePotFn}(\mathcal{G}, A, \mathbf{glo.pf})$ 
35 if  $\mathbf{glo.pf} == \perp$  then return  $\text{negCycle}$ 
36  $\mathbf{glo.status}[\mathbf{E}] := \text{done}$ 
37 return  $\emptyset$  // Processing of  $\mathbf{E}$  successfully completed

```

Once all back-propagation from \mathbf{E} is done, RulBackProp checks, at Line 24, whether any *CC loops* were encountered. If so, it calls FwdPropNDC to carry out a separate forward propagation from C along LO-edges, checking whether any LO-path, \mathcal{P}_{CX} , from C to some X , can be used to bypass the LC edge $e = (A, c:x, C)$. If so, there must be an SRN cycle, $e + \mathcal{P}_{CX} + \text{loc.path}[X]$, where $\text{loc.path}[X]$ is the LO-path from X to A obtained by the earlier back-propagation from C [10]. Hence, FwdPropNDC returns (X, \mathcal{P}_{CX}) . For the STNU on the left of Figure 3, \mathcal{P}_{CX} is $(C, 1, W) + (W, -3, X)$ and $\text{loc.path}[X]$ is $(X, 4, C) + (C, C:-9, A)$.

■ **Algorithm 3** The `AccNegCycle` algorithm (new).

```

Input: glo.intBy, vector recording a cycle of interruptions;  $\mathbf{E} \in \mathcal{E}_u$ , a UC edge in the cycle
Output: negCycle, an SRN cycle containing  $\mathbf{E}$ 
1 negCycle :=  $\langle \rangle$ 
2  $(\mathbf{E}', P') := \text{glo.intBy}[\mathbf{E}]$  //  $P'$  is path used to generate  $\mathbf{E}'$ 
3 while  $\mathbf{E}' \neq \mathbf{E}$  do
4   negCycle :=  $P' + \text{negCycle}$  // Accumulate  $P'$  into cycle
5    $(\mathbf{E}', P') := \text{glo.intBy}[\mathbf{E}']$  // Fetch next interrupter
6 return  $P' + \text{negCycle}$ 

```

If forward propagation fails to find an SRN cycle, then `RulBackProp` finally uses the information in `loc.dist` to generate edges that bypass the UC edge \mathbf{E} (Lines 28–33). These are the only edges that `FindSRNC` actually inserts into the STNU. For each bypass edge $(X, \delta_{xc} - y, A)$, the corresponding path that has been accumulated in `loc.path[X]` is recorded in the `glo.edgeAnnotation` hash table (Line 32). (As discussed below, it is `TryBackProp` that accumulates the path information in `loc.path[X]`.) If any bypass edges are inserted, then `RulBackProp` (at Line 34) calls `UpdatePotFn` (Algorithm 7, discussed later) to update the potential function for the LO-graph, whence the processing of \mathbf{E} is completed (Line 36).

TryBackProp

Pseudocode for `TryBackProp` (called `phaseOne` in RUL2021) is given as Algorithm 4. For a UC edge $\mathbf{E} = (C, C:-y, A)$, it propagates *backward* from C along LO-edges as long as the accumulated distance remains less than $\Delta_C = y - x$. (Recall the condition $w < \Delta_{C_i}$ for the R and L rules in Table 1.) Its `while` loop (Lines 3–19) uses the priority queue initialized by `RulBackProp` and the potential function updated by `RulBackProp` to explore shortest paths in the LO-graph. At Lines 4–6, it pops a node X off the queue, converts its key into the distance from X to C , and assigns it to `loc.dist[X]`. At Line 7, it checks whether X is an activation timepoint and, if so, sets \mathbf{E}_X to the corresponding UC edge. Next, if `loc.dist[X] < Δ_C` (Line 8), `TryBackProp` considers four cases (Lines 9–19).

In Case 1, back propagation has circled back to C , prompting `TryBackProp` to set the `ccLoop` flag. In Case 2, back propagation has encountered another UC edge \mathbf{E}_X whose processing has not yet been started; so `TryBackProp` pushes the interrupting edge onto a list of as-yet-unstarted UC edges (to be processed later by `RulBackProp`). In Case 3, back propagation has hit a UC edge \mathbf{E}_X whose processing has already been started, but not yet finished. This implies a cycle of interruptions and, hence, an SRN cycle. In preparation for terminating, the algorithm records the interruption of \mathbf{E} by \mathbf{E}_X , along with the *path* from X to A accumulated in `loc.path[X]`. Then it calls `AccNegCycle` (Algorithm 3) to recursively collect the information accumulated in the cycle of interruptions to return an SRN cycle. In Case 4, back propagation continues past X , and path information is accumulated. At Line 15, `TryBackProp` calls `ApplyRL` (Algorithm 5), which applies the R rule to all ordinary edges coming into X and, if X happens to be a contingent timepoint, applies the L rule to the corresponding LC edge coming into X . `ApplyRL` returns a list of pairs, each of the form (e, δ_{WC}) , where e is an LO-edge from W to X , and δ_{WC} is the length of the LO-path from W to C via X . For each such pair, `TryBackProp` (at Lines 15–19) first checks whether the path from W to C represents a new shorter LO-path and, if so, updates the key for W in the priority queue and incrementally accumulates the relevant path information in `loc.path[W]`.

■ **Algorithm 4** The TryBackProp algorithm.

Input: $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, an STNU graph; $\mathbf{E} = (C, C: -y, A) \in \mathcal{E}_u$; \mathcal{Q} , a priority queue; glo , *global* struct; loc , *local* struct

Output: *negCycle*, an SRN cycle; or \emptyset if no SRN cycle found.

```

1  $h := \mathit{glo}.pf$  // Potential function, a solution to the LO-graph
2  $\mathit{loc}.UnstartedUCs := \{\}$  // Will collect unstarted UC edges
3 while  $\mathcal{Q} \neq \emptyset$  do
  //  $key_X$  = distance from  $X$  to  $C$ , adjusted by  $h$ 
  4  $(X, key_X) := \mathcal{Q}.extractMinNode()$ 
  5  $\delta_{xc} := key_X - h(X)$  //  $\delta_{xc}$  = distance from  $X$  to  $C$  in  $\mathcal{G}_{lo}$ 
  6  $\mathit{loc}.dist[X] := \delta_{xc}$  // Record shorter length
  // If  $X$  is an ATP, then  $\mathbf{E}_X$  is corresponding UC-edge; else  $\perp$ 
  7  $\mathbf{E}_X := \mathcal{G}.UCEdgeFromATP(X)$ 
  8 if  $\delta_{xc} < \Delta_C$  then // Continue back-propagation
    // Case 1: Found CC loop of length  $\delta_{xc} < \Delta_C$ ; signal need for fwd prop
    9 if  $X \equiv C$  then  $\mathit{loc}.ccLoop := \top$ 
    // Case 2:  $\mathbf{E}_X$  is an unstarted UC-edge; accumulate it
    10 else if  $\mathit{glo}.status[\mathbf{E}_X] == \mathit{nYet}$  then  $\mathit{loc}.UnstartedUCs.add((\mathbf{E}_X, X))$ 
    // Case 3: Cycle of interruptions: not DC
    11 else if  $\mathit{glo}.status[\mathbf{E}_X] == \mathit{started}$  then
      12  $\mathit{glo}.intBy[\mathbf{E}] := (\mathbf{E}_X, \mathit{loc}.path[X])$ 
      13 return  $\mathit{AccNegCycle}(\mathit{glo}.intBy, \mathbf{E}_X)$ 
    14 else // Case 4: Continue back-propagation along LO-edges
      15 foreach  $(e, \delta_{wc}) \in \mathit{ApplyRL}(\mathcal{G}, X, \Delta_C, \delta_{xc})$  do
        16  $newKey := \delta_{wc} + h(W)$ 
        17 if  $\delta_{wc} < \mathit{loc}.dist[W]$  and  $(W \notin \mathcal{Q} \text{ or } newKey < \mathcal{Q}.key(W))$  then
          // Accumulate new path from  $W$  to  $C$ 
          18  $\mathcal{Q}.insOrDecrKey(W, newKey)$ 
          19  $\mathit{loc}.path[W] := e + \mathit{loc}.path[X]$ 
  20 return  $\emptyset$ 

```

■ **Algorithm 5** The ApplyRL algorithm.

Input: \mathcal{G} , an STNU graph; $X \in \mathcal{T}$; Δ_C ; and $\delta_{xc} < \Delta_C$

Output: A list of pairs, (e, δ_{wc}) , where e is an LO-edge from W to X , and $\delta_{wc} = |e| + \delta_{xc}$.

```

1  $edgeDistPairs := \{\}$ 
  // If  $X$  is a contingent timepoint  $C_i$ , then apply the L rule to  $(A_i, c_i: x_i, C_i)$  and  $(C_i, \delta_{xc}, C)$ 
  2 if  $X \equiv C_i \in \mathcal{T}_C$  then  $edgeDistPairs.add(((A_i, c_i: x_i, C_i), x_i + \delta_{xc}))$ 
  3 else // Otherwise, apply the R rule to  $(W, \delta_{wx}, X)$  and  $(V, \delta_{vc}, C)$ 
  4  $\mathit{foreach} (W, \delta_{wx}, X) \in \mathcal{E}_o$  do  $edgeDistPairs.add(((W, \delta_{wx}, X), \delta_{wx} + \delta_{xc}))$ 
  5 return  $edgeDistPairs$ 

```

FwdPropNDC

The FwdPropNDC algorithm (Algorithm 6) propagates *forward* from C along LO-edges checking whether there is a negative-length path from C to some X that can be used to bypass the LC edge $(A, c:x, C)$. It is the same as in RUL2021, except that it accumulates path information in a vector called `fwdPath`. At Lines 2–3, a priority queue is initialized to contain just C , with $\mathit{fwdPath}[C] = \langle \rangle$. The priority queue uses the same potential function as TryBackProp to effectively re-weight the LO-edges. As each timepoint X is popped from the queue (Line 5),

■ **Algorithm 6** The FwdPropNDC algorithm.

```

Input:  $\mathcal{G}$ , an STNU graph;  $C \in \mathcal{T}_C$ ;  $\Delta_C = y - x$ ; loc, local struct;  $h$ , potential function.
Output:  $(X, \mathcal{P}_{CX})$ , if path  $\mathcal{P}_{CX}$  can be used to reduce away the LC edge  $(A, c:x, C)$ ; else  $\emptyset$ 
1 fwdPath :=  $\{\langle \rangle, \dots, \langle \rangle\}$  // For each  $X$ ,  $\text{fwdPath}[X]$  is an LO-path from  $C$  to  $X$ 
2  $\mathcal{Q}$  := new priority queue // Key  $\text{key}_X = d(C, X) - h(C)$ 
3  $\mathcal{Q}.\text{insert}(C, -h(C))$  // Queue initially contains only  $C$ 
4 while  $\mathcal{Q} \neq \emptyset$  do
5    $(X, \text{key}_X) := \mathcal{Q}.\text{extractMinNode}()$ 
6    $d(C, X) := \text{key}_X + h(X)$  // Distance from  $C$  to  $X$  in  $\mathcal{G}_{\ell o}$ 
7   if  $\text{loc}.\text{dist}[X] < \Delta_C$  then // If distance from  $X$  to  $C < \Delta_C$ 
8     // Check if the path  $CX$  can reduce-away the LC-edge
9     if  $d(C, X) < 0$  then return  $(X, \text{fwdPath}[X])$ 
10    foreach  $(X, \delta_{xy}, Y) \in \mathcal{E}_\ell \cup \mathcal{E}_o$  do // Iterate over LO-edges emanating from  $X$ 
11       $\text{newKey} := d(C, X) + \delta_{xy} - h(Y)$ 
12      if  $Y \notin \mathcal{Q}$  or  $\text{newKey} < \mathcal{Q}.\text{key}(Y)$  then
13         $\mathcal{Q}.\text{insOrDecrKey}(Y, \text{newKey})$ 
14         $\text{fwdPath}[Y] := \text{fwdPath}[X] + (X, \delta_{xy}, Y)$ 
14 return  $\emptyset$  // Was unable to reduce-away the LC-edge

```

the distance from X to C that was determined during back-propagation and stored in $\text{loc}.\text{dist}[X]$ is compared to Δ_C . (Generating an edge to bypass the LC edge using the path from C to X will only create an SRN cycle if $\text{dist}[X] < \Delta_C$ [10].) If $\text{dist}[X] < \Delta_C$ and $d(C, X) < 0$ (i.e., an appropriate negative-length path has been found), then FwdPropNDC terminates, returning $(X, \text{fwdPath}[X])$ (Line 8). Otherwise, forward propagation continues from X , accumulating relevant path information (Lines 9–13). If the queue is exhausted without finding a way to bypass the LC edge, FwdPropNDC returns \emptyset (Line 14).

UpdatePotFn

When `Ru1BackProp` inserts edges that bypass a UC edge \mathbf{E} , it changes the LO-graph. Hence, the potential function for the LO-graph typically needs to be updated. The pseudocode for the `UpdatePotFn` function is given as Algorithm 7.

Since all bypass edges for \mathbf{E} necessarily point at its activation timepoint A , `UpdatePotFn` propagates backward from A along LO-edges as long as changes to the potential function, h , are required. This function and its helper `UpdateVal` (Algorithm 8) are the same as in RUL2021 except that path information is accumulated (Algorithm 8, Line 6) so that if back-propagation ever cycles all the way back to A , the implied SRN cycle can be returned (Algorithm 8, Line 4).

Computational Complexity

`FindSRNC` performs more operations than RUL2021, mostly by accumulating path information during propagation. For lack of space, we simply note that the most time-consuming operation is prepending an edge onto the front of an existing path, which happens at most once per edge visited. Since the prepending operation (+) can be realized in constant time, the worst-case time complexity of `FindSRNC` is the same as that of RUL2021: $O(mn + k^2n + kn \log n)$.

Regarding the *extra* space requirements of `FindSRNC`, the most costly is the space needed by `TryBackProp` for accumulating path information in the `loc.path` structures. `TryBackProp` is called at most $2k$ times [3, 10]. Each call explores at most $(m + nk)$ edges. (`FindSRNC`

■ **Algorithm 7** The UpdatePotFn algorithm.

Input: \mathcal{G} , an STNU graph; A , an activation timepoint; h , a potential function for $\mathcal{G}_{\ell o}$, excluding edges ending at A

Output: $(h', negCycle)$, where h' is either a potential function for $\mathcal{G}_{\ell o}$ (including edges terminating at A); or \perp , the latter indicating that $negCycle$ is a negative cycle

```

1  $h' :=$  copy of  $h$ ;  $path := [\langle \rangle, \dots, \langle \rangle]$ 
2  $\mathcal{Q} :=$  new priority queue;  $\mathcal{Q}.insert(A, 0)$  // Initialize queue for back-prop from  $A$ 
3 while  $\mathcal{Q} \neq \emptyset$  do
4    $(V, key_V) := \mathcal{Q}.extractMinNode()$ 
5   foreach  $((U, \delta, V) \in \mathcal{E}_o)$  do // Back-propagate along ordinary edges ending at  $V$ 
6      $negCycle := UpdateVal((U, \delta, V), h, h', \mathcal{Q}, path)$ 
7     if  $negCycle \neq \emptyset$  then return  $(\perp, negCycle)$ 
8   if  $V \in \mathcal{T}_C$  then //  $V$  is contingent; back-propagate along LC edge  $(A_V, v:x_V, V)$ 
9      $negCycle := UpdateVal((A_V, x_V, V), h, h', \mathcal{Q}, path)$ 
10    if  $negCycle \neq \emptyset$  then return  $(\perp, negCycle)$ 
11 return  $(h', \emptyset)$ 

```

■ **Algorithm 8** The UpdateVal algorithm.

Input: (U, δ, V) , an edge; h, h' , potential fns.; \mathcal{Q} , priority queue; and $path$, a vector of path info

Output: $negCycle$, an SRN cycle; or \emptyset if h' was successfully updated to satisfy (U, δ, V)

Side Effect: Modifies \mathcal{Q} , h' and $path$

```

1 if  $h'(U) < h'(V) - \delta$  then
2    $h'(U) := h'(V) - \delta$ 
3   // If back propagation has cycled back to  $A$ , return the cycle
4   if  $\mathcal{Q}.state(U) == alreadyPopped$  then return  $(U, \delta, V) + path[V]$ 
5    $\mathcal{Q}.insOrDecrKey(U, h(U) - h'(U))$ 
6    $path[U] := (U, \delta, V) + path[V]$ 
7 return  $\emptyset$ 

```

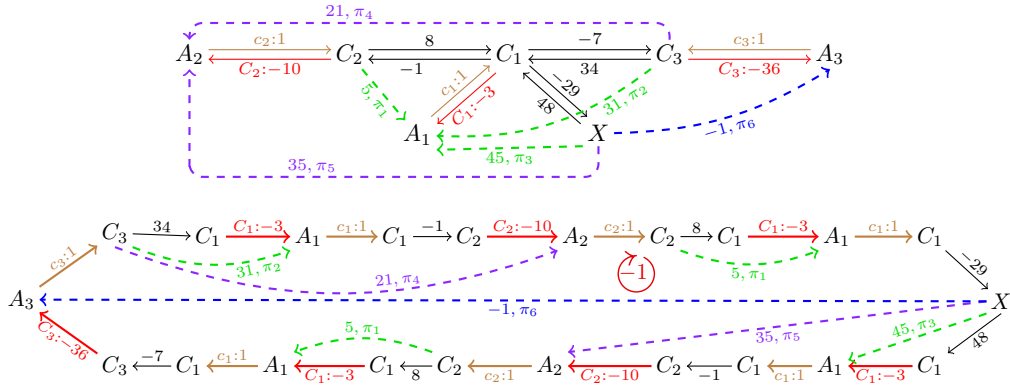
inserts at most nk edges overall.) Each edge exploration involves prepending an existing path with an edge, which uses only constant space. So the overall space complexity across *all* calls to TryBackProp is $O(mk + k^2n)$. Similar remarks apply to FwdPropNDC and UpdatePotFn.

The edgeAnnotation hash table has at most nk entries: one for each bypass edge. Each entry is a pointer to a loc.path entry. So the total space required is $O(nk)$. The compact SRN cycle generated by AccNegCycle is the concatenation of at most k paths, each with at most n edges, for a total of at most nk edges, which is dominated by the $O(mk + k^2n)$ space discussed above. This compact cycle, together with the information in the edgeAnnotation hash table, avoids redundantly storing repeated structures. In this way, it uses polynomial space to *implicitly* represent a cycle that, if fully expanded, might have exponentially many edges. Similar remarks apply to the cycles returned by FwdPropNDC and UpdatePotFn.

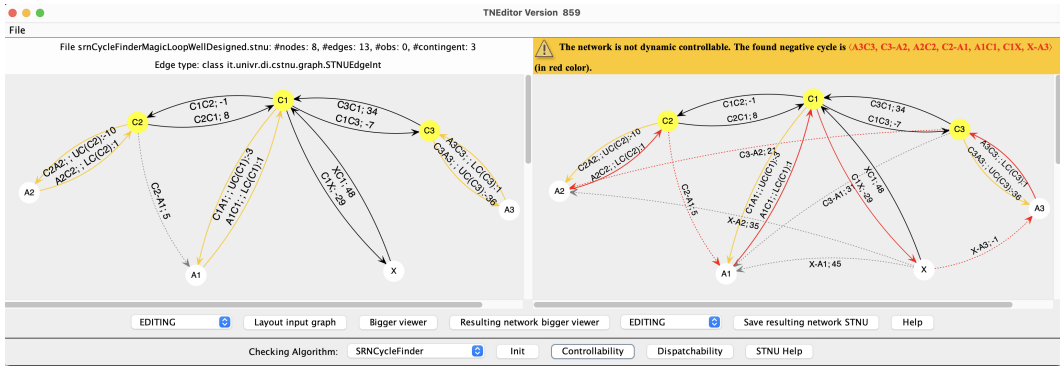
Magic Loop Example

Hunsberger [8] identified a family of STNUs in which the only SRN cycle, called a *magic loop*, has an *exponential* number of edges. Since each STNU has at most $n^2 + 2k$ edges, magic loops necessarily contain a large number of repeated edges. In particular, a magic loop of order k has k contingent links, but $3(2^k) - 2$ edges. The top of Figure 4 shows an STNU whose (brown) LC edges are $\mathbf{e}_1 = (A_1, c_1:1, C_1)$, $\mathbf{e}_2 = (A_2, c_2:1, C_2)$, and $\mathbf{e}_3 = (A_3, c_3:1, C_3)$; and whose (red)

9:12 Finding Negative Cycles in STNUs



■ **Figure 4** A sample STNU (top) and the magic loop of order 3 (bottom) hiding within it.



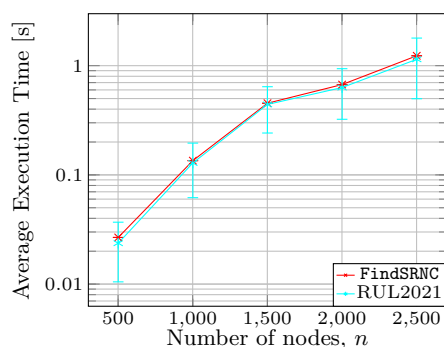
■ **Figure 5** A screenshot of FindSRNC in action.

UC edges are $\mathbf{E}_1 = (C_1, C_1: -3, A_1)$, $\mathbf{E}_2 = (C_2, C_2: -10, A_2)$, and $\mathbf{E}_3 = (C_3, C_3: -36, A_3)$. The bypass edges generated by FindSRNC are dashed: those bypassing \mathbf{E}_1 in green, \mathbf{E}_2 in purple, and \mathbf{E}_3 in blue. Each bypass edge is also annotated by a path, where: $\pi_1 = (C_2, 8, C_1) + \mathbf{E}_1$; $\pi_2 = (C_3, 34, C_1) + \mathbf{E}_1$; $\pi_3 = (X, 48, C_1) + \mathbf{E}_1$; $\pi_4 = \pi_2 + \mathbf{e}_1 + (C_1, -1, C_2) + \mathbf{E}_2$; $\pi_5 = \pi_3 + \mathbf{e}_1 + (C_1, -1, C_2) + \mathbf{E}_2$; and $\pi_6 = \pi_5 + \mathbf{e}_2 + \pi_1 + \mathbf{e}_1 + (C_1, -7, C_3) + \mathbf{E}_3$. The magic loop for this STNU is at the bottom of the figure. It has 22 edges. \mathbf{E}_1 and \mathbf{e}_1 appear four times each; several other edges, twice each. After all UC edges have been processed, UpdatePotFn finds a negative cycle in the LO-graph: $\pi_6 + \mathbf{e}_3 + \pi_4 + \mathbf{e}_2 + \pi_1 + \mathbf{e}_1 + (C_1, -29, X)$. This information is compactly stored in the cycle returned by FindSRNC. For higher-order magic loops, the number of edges grows exponentially, but the space used by FindSRNC is bounded by $mk + nk^2$.

4 Empirical Evaluation

In this section, we present a possible implementation of the FindSRNC algorithm and one its evaluation in a public benchmark.

The proposed algorithm was implemented as a proof-of-concept prototype in the (freely available) CSTNU Tool, version 1.42 [17]. The tool enables users to create different kinds of temporal constraint networks and to verify automatically some properties like dynamic controllability or consistency (for some kinds of networks). In particular, as concerns STNUs, it allows one to verify the dynamic controllability and, in case the network is not DC, to obtain the semireducible negative cycle that determines the non-controllability.



n	Avg. SRN cycle length	Pct. Simple SRN cycles	Avg. SRN expanded cycle length
500	8.0	63%	14.1
1000	7.9	64%	14.4
1500	7.6	74%	15.3
2000	8.5	72%	14.0
2500	8.6	63%	14.4

■ **Figure 6** Experimental results.

The screenshot Figure 5 shows the CSTNU Tool after the execution of `FindSRNC` algorithm on the STNU depicted in Figure 4. On the left side, there is the initial network that can be edited. On the right side, there is the checked network with the semireducible negative cycle emphasized in red. The status bar above the network on the right gives a summary of the `FindSRNC` result. The extended result (like the expanded semireducible negative cycle) is saved in a logging file associated with the execution.

We empirically evaluated `FindSRNC` on a published benchmark [16] to confirm that the execution times of `FindSRNC` and `RUL2021` are equivalent, and to highlight the characteristics of the SRN cycles in non-DC instances. Our implementations are publicly available [17]. We ran them on a JVM 21 with 8 GB of heap memory on a Apple PowerBook/M1 Pro.

For each $n \in \{500, 1000, 1500, 2000, 2500\}$, the benchmark contains 200 randomly generated non-DC STNUs, each having n nodes, $n/10$ contingent links, and $m \approx 3n$ edges. For each sub-benchmark (i.e., for each n), we used the first 100 instances. For each instance, `RUL2021` checked only the non-DC status; `FindSRNC` also returned an SRN cycle.

The left-hand plot of Figure 6 shows the average execution times of the two algorithms for each sub-benchmark. These results highlight that computing the SRN cycle does not require significant computational overhead. More interesting is that by analyzing the cycles computed by `FindSRNC`, we can evaluate the characteristics of the non-DC instances in the benchmark. The table in Figure 6 shows that, for each n , the average number of edges in the SRN cycle (i.e., the SRN cycle length) is quite small (less than 9); and most instances present a *simple* SRN cycle (i.e., an SRN cycle having no (annotated) bypass edges and, hence, comprising only edges that were already present in the input STNU).

`FindSRNC` outputs a *non-simple* SRN cycle very compactly. However, we also computed the *fully expanded* version of each cycle, recursively replacing each bypass edge by the annotated path from which it was derived. The average length of the expanded cycles increased to a maximum of 16 in each sub-benchmark, revealing that an SRN cycle can involve more edges from the original STNU than one might suspect from the compact version.

Finally, we checked that *no* instance leads to an expanded SRN cycle with *any* repeated edges. Since the benchmark was built to simulate temporal business processes organized on five lanes, the absence of complex SRN cycles in 500 random instances suggests that such instances may only rarely appear in practice; but if they do, `FindSRNC` will find them.

5 Conclusion

This paper presented the FindSRNC algorithm that modifies the fastest DC-checking algorithm for STNUs to accumulate path information while also rigorously addressing the compact representation of the SRN cycles it outputs. When given an overconstrained STNU, FindSRNC can be used to identify constraints to relax or contingent durations to tighten. It can also be used as a supporting process in an iterative algorithm for finding a DC STNU that well approximates a Probabilistic Simple Temporal Network [20, 23, 21, 1].

References

- 1 Shyan Akmal, Savanna Ammons, Hemeng Li, and James C. Boerkoel, Jr. Quantifying degrees of controllability in temporal networks with uncertainty. In *29th International Conference on Automated Planning and Scheduling (ICAPS 2019)*, pages 22–30, 2019. URL: <https://ojs.aaai.org/index.php/ICAPS/article/view/3456>, doi:10.1609/icaps.v29i1.3456.
- 2 Nikhil Bhargava, Tiago Vaquero, and Brian C. Williams. Faster conflict generation for dynamic controllability. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 4280–4286, 2017. doi:10.24963/ijcai.2017/598.
- 3 Massimo Cairo, Luke Hunsberger, and Romeo Rizzi. Faster Dynamic Controllability Checking for Simple Temporal Networks with Uncertainty. In *25th International Symposium on Temporal Representation and Reasoning (TIME-2018)*, volume 120 of *LIPICs*, pages 8:1–8:16, 2018. doi:10.4230/LIPICs.TIME.2018.8.
- 4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 4th Edition*. MIT Press, 2022. URL: <https://mitpress.mit.edu/9780262046305/introduction-to-algorithms>.
- 5 Johann Eder, Marco Franceschetti, and Josef Lubas. Dynamic Controllability of Processes without Surprises. *Applied Sciences*, 12(3):1461, January 2022. doi:10.3390/app12031461.
- 6 Cheng Fang, Andrew J. Wang, and Brian C. Williams. Chance-constrained Static Schedules for Temporally Probabilistic Plans. *Journal of Artificial Intelligence Research*, 75:1323–1372, 2022. doi:10.1613/jair.1.13636.
- 7 Luke Hunsberger. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *16th International Symposium on Temporal Representation and Reasoning (TIME-2009)*, pages 155–162, 2009. doi:10.1109/TIME.2009.25.
- 8 Luke Hunsberger. Magic Loops in Simple Temporal Networks with Uncertainty—Exploiting Structure to Speed Up Dynamic Controllability Checking. In *5th International Conference on Agents and Artificial Intelligence (ICAART-2013)*, volume 2, pages 157–170, 2013. doi:10.5220/0004260501570170.
- 9 Luke Hunsberger. Magic Loops and the Dynamic Controllability of Simple Temporal Networks with Uncertainty. In Joaquim Filipe and Ana Fred, editors, *Agents and Artificial Intelligence*, volume 449 of *Communications in Computer and Information Science (CCIS)*, pages 332–350, 2014. doi:10.1007/978-3-662-44440-5_20.
- 10 Luke Hunsberger and Roberto Posenato. Speeding up the RUL⁻ Dynamic-Controllability-Checking Algorithm for Simple Temporal Networks with Uncertainty. In *36th AAAI Conference on Artificial Intelligence (AAAI-22)*, volume 36-9, pages 9776–9785. AAAI Pres, 2022. doi:10.1609/aaai.v36i9.21213.
- 11 Erez Karpas, Steven J. Levine, Peng Yu, and Brian C. Williams. Robust Execution of Plans for Human-Robot Teams. In *25th Int. Conf. on Automated Planning and Scheduling (ICAPS-15)*, volume 25, pages 342–346, 2015. doi:10.1609/icaps.v25i1.13698.
- 12 Paul Morris. A Structural Characterization of Temporal Dynamic Controllability. In *Principles and Practice of Constraint Programming (CP-2006)*, volume 4204, pages 375–389, 2006. doi:10.1007/11889205_28.


- 13 Paul Morris. Dynamic controllability and dispatchability relationships. In *Int. Conf. on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR-2014)*, volume 8451 of *LNCS*, pages 464–479. Springer, 2014. doi:10.1007/978-3-319-07046-9_33.
- 14 Paul Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *17th Int. Joint Conf. on Artificial Intelligence (IJCAI-2001)*, volume 1, pages 494–499, 2001. URL: <https://www.ijcai.org/Proceedings/01/IJCAI-2001-e.pdf>.
- 15 Jun Peng, Jingwei Zhu, and Liang Zhang. Generalizing STNU to Model Non-functional Constraints for Business Processes. In *2022 International Conference on Service Science (ICSS)*, pages 104–111. IEEE, May 2022. doi:10.1109/ICSS55994.2022.00024.
- 16 Roberto Posenato. STNU Benchmark version 2020, 2020. Last access 2022-12-01. URL: <https://profs.scienze.univr.it/~posenato/software/cstnu/benchmarkWrapper.html>.
- 17 Roberto Posenato. CSTNU Tool: A Java library for checking temporal networks. *SoftwareX*, 17:100905, 2022. doi:10.1016/j.softx.2021.100905.
- 18 Christoph Strassmair and Nicholas Kenelm Taylor. Human Robot Collaboration in Production Environments. In *23rd IEEE International Symposium on Robot and Human Interactive Communication 2014*, 2014. URL: <https://researchportal.hw.ac.uk/en/publications/human-robot-collaboration-in-production-environments>.
- 19 Robert Endre Tarjan. Shortest Paths. Technical report, AT&T Bell Laboratories, 1981.
- 20 Ioannis Tsamardinos. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence (SETN 2002)*, volume 2308 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 97–108, 2002. doi:10.1007/3-540-46014-4_10.
- 21 Andrew J. Wang. *Risk-bounded Dynamic Scheduling of Temporal Plans*. PhD thesis, Massachusetts Institute of Technology, 2022. URL: <https://hdl.handle.net/1721.1/147542>.
- 22 Peng Yu, Cheng Fang, and Brian Charles Williams. Resolving uncontrollable conditional temporal problems using continuous relaxations. In *24th International Conference on Automated Planning and Scheduling, ICAPS 2014*. AAAI, 2014. URL: <http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7895>, doi:10.1609/icaps.v24i1.13623.
- 23 Peng Yu, Brian C. Williams, Cheng Fang, Jing Cui, and Patrick Haslum. Resolving over-constrained temporal problems with uncertainty through conflict-directed relaxation. *Journal of Artificial Intelligence Research*, 60:425–490, 2017. doi:10.1613/jair.5431.

What Killed the Cat?

Towards a Logical Formalization of Curiosity (And Suspense, and Surprise) in Narratives

Florence Dupin de Saint-Cyr ✉ 🏠 

Lab-STICC CNRS UMR 6285, ENIB, Brest, France
IRIT, Université de Toulouse, France

Anne-Gwenn Bosser ✉ 🏠 

Lab-STICC CNRS UMR 6285, ENIB, Brest, France

Benjamin Callac ✉

Lab-STICC CNRS UMR 6285, Université de Bretagne Occidentale, Brest, France

Eric Maisel ✉

Lab-STICC CNRS UMR 6285, ENIB, Brest, France

Abstract

We provide a unified framework in which the three emotions at the heart of narrative tension (curiosity, suspense and surprise) are formalized. This framework is built on non-monotonic reasoning which allows us to compactly represent the default behavior of the world and to simulate the affective evolution of an agent receiving a story. After formalizing the notions of awareness, curiosity, surprise and suspense, we explore the properties induced by our definitions and study the computational complexity of detecting them. We finally propose means to evaluate these emotions' intensity for a given agent listening to a story.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases Knowledge Representation, Narration, Cognition

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.10

Acknowledgements The authors want to thank the anonymous reviewers for their relevant suggestions that helped them to improve the paper.

1 Introduction

Humans tell stories to make sense of the world and communicate their understanding of what happens. Storytelling supposes to be able to sort out which events are worth telling, deciding on a level of detail for describing events, selecting among possible causes the ones which are deemed worth telling. It also supposes to make use of an affective machinery for capturing an audience's attention (emotional contagion, suspense elicitation...). In the act of storytelling, structural and affective phenomena are thus combined with communicative goals in mind. This combination has indeed shown its effectiveness in this respect: the phenomenon of narrative transportation (the experience of being immersed in a story) has been linked to persuasion [27]. The narrative paradigm therefore provides an appropriate framework, in which causal reasoning about the situations narrated [53] is combined with narrative devices to encourage the audience's emotional involvement [51], to study and model how opinion is formed and evolves. Building a framework for reasoning about and unveiling storytelling mechanics could pave the way for intellectual self-defense supporting tools, enabling citizens to arm themselves against hostile disinformation or influence campaigns.



© Florence Dupin de Saint-Cyr, Anne-Gwenn Bosser, Benjamin Callac, and Eric Maisel; licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 10; pp. 10:1–10:16

Leibniz International Proceedings in Informatics



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

Previous works in structural narratology have studied the way stories are conveyed to their audience and seminal work from (for instance) Genette [25] or Propp [45] have previously served as the backbone inspiration for computational narrative models and storytelling systems [43]. Whilst the operationalization of narrative theories is still subject to debate and caution, such works have shed light on how the story material to tell and the manner in which it is told interacts with a model of the listener¹ (which, depending on the media used for conveying the story can also be a reader, a spectator, or even a gamer): the act of storytelling can thus be understood as knowledge transfer and manipulation of her beliefs.

According to Sternberg [51] or Baroni [5], emotions more specific to narratives which are *suspense*, *curiosity* and *surprise* are critical to retain the interest of the listener. Drivers of the *narrative tension*, they are paramount in maximizing her engagement. In this paper we focus on these narrative tension's building blocks.

In the field of computational narratives, numerous studies and frameworks exist to tell interactive stories, a number of them as an application of planning technologies [13] allowing to adapt the narrative to each user's actions. However, adapting a narrative to a model of the user's emotions remains largely a challenge that needs to be addressed to favor engagement: narrative engagement depends partly on the appropriate maintenance of narrative tension, itself based on the uncertainty occurring in a narrative [9], and listener's models based on a formalization of related emotions have comparatively been less addressed so far in the literature. While suspense and surprise have been the object of previous studies [15] [23], there is – to our knowledge – still no curiosity model applicable to narratives.

In the following, we present a preliminary study for characterizing these emotions from an epistemic standpoint, with a focus on modeling the listener's curiosity depending on her beliefs and knowledge using a propositional language. Our overarching aim is to provide a unifying framework allowing to represent emotions relevant to the characterization of narrative tension and its evolution, which would enable to discuss their relationships and ultimately help establish dramatic metrics about a narrative.

In Section 2, we describe the main emotions supporting narrative tension. We also describe the problems and solutions for formalizing reasoning about action and change, as well as the ways in which the notions of surprise and awareness have been treated in the literature. Section 3 details our proposal, which relies on a non-monotonic framework resulting from extending propositional logic with default rules. The properties of the framework are presented in Section 4, along with some preliminary ideas for developing metrics.

2 Background on reasoning about change and narrative tension

In order to reason about a story, it is useful to dispose of a way to handle the concepts involved for understanding a sequence of facts and events. We first briefly outline the background to this vast subject, before discussing the formalization of emotions in the literature.

2.1 Reasoning about action and change

The formalization of action and change is an old field of research in the domain of knowledge representation and reasoning in AI. There are many different reasoning tasks in this field (see e.g. [18]) like prediction of the new state of the world after an action (which is related to *belief*

¹ For sake of homogeneity, we use the term listener in all the paper, while this kind of agent is called interpret by Baroni.

update [55, 30]), or integration of an observation (which is related to *belief revision* [2]), or *event abduction* which consists in guessing which event took place, or *scenario extrapolation* [19, 17] which consists in taking a partial description of facts and events that occurred and complete it (by prediction or event abduction) or *scenario recognition* [16].

These reasoning tasks were studied in various frameworks, the representation of actions in a compact way has given rise to some problems known as the frame, the ramification and the qualification problems [38, 24, 37]. In propositional logic, these problems were solved by a majority of approaches by introducing a special symbol for expressing a causal rule relating preconditions of an action to its effect (indeed classical implication cannot separate a cause from a consequence due to contraposition). Actions are first described by such rules, then given the set of causal rules, a set of formulas (called frame axioms) are generated stating that any fluent f is true at time $t + 1$ if and only if it was (a) true at t and no causal rule concluding $\neg f$ can be fired at t or (b) false at t and a causal rule concluding f can be fired.

As a proof of concept, we choose to use *propositional logic* in this article where we face a problem that can be viewed as an extension of belief extrapolation with narrative tension analysis. However in order to perform non-monotonic reasoning (which allows agents to change their minds and thus accept surprises), we use default rules of the form $a \rightsquigarrow b$ to encode causal relations. Note that another prominent formalization of default rules was given by Reiter in [46], but we choose to rely on a simpler formalism at first.

Logical approaches to computational narratives have been proposed in the past. In [10], (Intuitionistic) Linear logic has been argued to be a suitable representational model for narratives for its capacity to finely represent narrative actions through the production and consumption of resources. This language provides the symbol \multimap which can be used in $A \multimap B$ to express the validity of transforming resource A into resource B , the flow of resources consumption through the associated sequent calculus allowing to establish causal relations. Dynamic logic [29] and its epistemic extensions [8] are formalisms with higher expressiveness. In this work, we propose to characterize narrative tension phenomenon in *propositional logic* (extended with default rules) to demonstrate the representational uniformity of these concepts and their relationships with each other. We will explore how to encode them in aforementioned logics in the future, keeping in mind the challenges raised by their operationalization in their most expressive fragments [3].

2.2 Emotions supporting narrative tension

Psychological models of emotions are often used in the field of affective computing such as models from Ekman [22] or Plutchik [44] (which include surprise). Other works [41] consider that every emotion should have a valence and, as a consequence, surprise, which is inherently neither good nor bad, is considered as a different affective phenomenon. As we position ourselves in the context of studying the emotional states of a listener, we will rely on the characterizations given by Baroni in [5]. Curiosity occurs when there is a partial omission of crucial knowledge: at a given stage when experiencing the storytelling experience, the listener knows they are missing important information. Suspense arises when an event could potentially lead to an impacting result – be it good or bad – to the storyline, and is correlated with anticipation. Surprise results from a rupture from previous expectations, which retroactively invalidates some of the predictions made by the listener: the listener has expectations about how the story will develop, based on story genre or common sense. Going against these expectations while maintaining coherence is what causes surprise. Baroni distinguishes curiosity and suspense from surprise, as the former two are tied to anticipation and an urge to know, whilst the latter arises sporadically as the narrative progresses, which will be reflected in our model.

Related to suspense, the concept of *narrative closure* also reflects the epistemic nature of storytelling (as theorized by Carroll [14]): this encompasses the phenomenological feeling of finality that is generated when all the questions saliently posed by the narrative to the listener are answered. Previous work in the psychology of narrative understanding [53] has also tied the perception of the importance of story events to causal relationships' perception. In this paper we borrowed from this work, especially tracing a graph representing the narrative with nodes being actions, preconditions and effects and edges being causal relations. We will consider the degree of a node as reflecting its importance in the narrative, reading it as, the more an action is a consequence and has consequences the more important it is.

2.3 Logical models of emotions, surprises and awareness

The logical representation of emotions has already received some attention, see e.g. Lorini [34] or Adam [1] who formalized emotions based on the OCC theory [41]. In these works (which relies on a modal logic for BDI agents), an agent has beliefs, including beliefs about what is *good for herself*, and expresses different emotions such as joy or sadness.

The particular case of *surprise* was studied by several authors in computer science, but the first study is due to an economist named Shackle [48] who defined the degree of surprise associated with an event as the degree of impossibility of this event given the uncertain knowledge about the situation considered. In Lorini and Castelfranchi [33], the role of surprise is investigated in the context of belief update. They associate a surprise with a difficulty to integrate the new piece of information, this occurs when there is a form of inconsistency between expectation and perception. Surprise was recently formalized in the context of the analysis of jokes by [20], indeed surprise has been considered as an important ingredient for laughter by many authors, the model of surprise of [20] is based on a revision operator and non-monotonic reasoning: to be surprised the listener of a joke should be able to jump to conclusions that can be questioned and even revised.

The characterization of *curiosity* provided by Baroni emphasizes that the listener is aware of its incomplete knowledge and that surprise is linked to a notion of disturbance which makes the agent to question his assumptions/beliefs and leads him to reconsider his understanding of the story. This reconsideration reminds the operation of *awareness raising* introduced by [54] to allow agents "to make their implicit knowledge explicit". Logical models taking into account agent's awareness have previously been defined in the literature. As Halpern [28] states, traditionally when reasoning about agents' beliefs, it is assumed they are aware of every proposition. Modica and Ristichini [39] first came up with a definition of awareness based on knowledge, stating that an agent is aware of p if he knew p or if he knew he did not know p . Halpern extends on this by introducing *implicit* knowledge, where agents are aware of all propositions and can reason with them ; and *explicit* knowledge, which captures the conclusions of which the agent is explicitly aware of. In this system, explicit knowledge is also implicit, while the reverse is not necessarily true.

Previous works have proposed models for agents in computational narratives such as [40] or [12] based on Belief, Desire and Intention (BDI). In [47], a BDI agent aiming to simulate player behavior in interactive stories takes into account the player personality. Other work has assigned personality stereotypes to users [52, 4] according to their interactions with the system. Whilst such models allow personalizing an interactive narrative, they would not enable a storytelling engine to finely drive the narrative tension. By contrast, the *Suspenser* system by [15] offers an operationalization for suspense elicitation, one of the three drivers of narrative tension. In *Suspenser*, suspense is maximized by ordering multiple story bits at the discourse level. We lay in this paper the groundwork for ultimately representing in

a unified logical framework suspense, curiosity and surprise, the three drivers of narrative tension. This will build strong foundations for future generative and interactive systems able to operate both at the story and discourse levels.

We approach the modelization of curiosity, suspense and surprise as constructs at given moments of a narrative experience from the listener's beliefs and by non-monotonic reasoning about these beliefs. Doing so, we believe our model is compatible with previous formalization while providing new insights.

3 Formalizing curiosity, surprise and suspense

We first present an example to illustrate the concepts introduced throughout this article.

► **Example 1 (The box).** *To illustrate the framework, we present a short story involving three agents, Albert, Erwin as well as a protagonist Cecilia² (respectively agents A, E and C). A short narrative : “Cecilia enters her office. She sees a box lying on her desk that was not there when she last left the room.” Our hypothesis is that this event sparks curiosity in Cecilia's mind. We look at it from the point of view of Cecilia who reasons in a closed world where nothing, except three particular events (Albert putting a box on Cecilia's desk, Erwin doing it, Cecilia opening the box) can interact with the state of the world.*

We consider a set of variable symbols \mathcal{V} denoted by Latin lower case letters, from this set of symbols we build the vocabulary \mathcal{V}_T containing all variables of \mathcal{V} indexed by all the integers taken in the set $T = \{0, 1, \dots, N\}$ representing time points. \mathcal{L} is the propositional language based on \mathcal{V}_T with the usual connectors and constants $\vee, \wedge, \neg, \rightarrow, \equiv, \perp$ and \top denoting respectively the logical connectors “or”, “and”, “not”, material implication and logical equivalence, contradiction, and tautology. The symbol \models represent satisfiability. Let Ω denote the set of interpretations induced by \mathcal{V}_T , we will often use ω for naming a particular interpretation in Ω , each interpretation will be described by the list of literals satisfied by it, e.g., considering the vocabulary $\mathcal{V} = \{a, b\}$, and a set of two time points $T = \{0, 1\}$ $\omega = (a_0, \neg b_0, \neg a_1, \neg b_1)$ is an interpretation in Ω that associates the truth value True to a and False to b at time step 0 and False to a and b at time step 1. The set $Mod(A) \subseteq \Omega$ is the set of interpretations satisfying the set of propositional formulas $A \subseteq \mathcal{L}$ ($Mod(A) = \{\omega \in \Omega \mid \omega \models \bigwedge_{\varphi \in A} \varphi\}$), the same notation is used to represent the set of models of a formula $Mod(\varphi) = \{\omega \in \Omega \mid \omega \models \varphi\}$.

► **Example 1 (continued).** *To study this flow of events taking place in 4 time steps denoted $T = \{0, 1, 2, 3\}$ we need a vocabulary $\mathcal{V} = \{box, A, E, C, empty, visible\}$ meaning respectively there is a box on Cecilia's desk, agent A puts a closed box on the desk, agent E puts a closed box on the desk, agent C opens the box, there is nothing in the box and something inside the box is uncovered (and thus the box has been opened). In the language \mathcal{L} built on \mathcal{V} and T , the following expression is an example of a well-formed formula: $(A_0 \vee E_0) \wedge box_1 \wedge C_2 \wedge \neg empty_2$.*

Default rules are rules that tolerate exceptions and allow us to reason in presence of incomplete information, by assuming that the situation is not exceptional when there is no evidence for the contrary. The notation $\alpha \rightsquigarrow \beta$ (with $\alpha, \beta \in \mathcal{L}$) is used to represent a default rule interpreted as when α is true, it is more plausible that β is true than false.

² We consider a story involving Albert Einstein, Erwin Schrödinger and Cecilia Payne-Gaposchkin, hence the cat in the title.

► **Example 1** (continued). In order to be able to encode this example we propose to use default rules to express that by default some fluents keep their value: the following rule is expressing that when there is no box at time point 0 then by default there is no box at time point 1: $\neg box_0 \rightsquigarrow \neg box_1$. This rule admits exceptions: namely, if A puts a box on the desk at time point 0 then generally there is a box at time point 1: $A_0 \wedge \neg box_0 \rightsquigarrow box_1$.

Given a set of default rules Δ it is possible to define a ranking of these rules according to their specificity, thanks to “System Z” algorithm [42], the default base is then called stratified, its strata are the formulas with the same rank³. Note that there are sets of default rules that do not admit a Z ordering, such default sets are called “inconsistent” in [26]. In this paper, we restrict ourselves to consistent default sets. From a stratified default base lexicographic-entailment [6, 32] is a non-monotonic inference relation which imposes that the more specific the rules, the more mandatory it is to comply with them:

► **Definition 1** (Lex-inference [6]). Let $\Delta = \Delta_1 \cup \dots \cup \Delta_n$ be a stratified default base with n strata ordered from the most specific strata Δ_1 to the least specific one Δ_n , and let A and B be two subsets of Δ , and α, β be two formulas of \mathcal{L} ,

■ Notations: *str* (for “strict”) is a function that translates a set of default rules into a set of formulas of \mathcal{L} , i.e., $str(A) = \bigcup_{\alpha \rightsquigarrow \beta \in A} \{\neg \alpha \vee \beta\}$. For all $i \in [1, n]$, and any $E \subseteq \Delta$, E_i denotes the i th strata of E : $E_i = E \cap \Delta_i$.

■ A is Lex-preferred to B given Δ , denoted $A \succ_{\Delta} B$,

$$\text{iff there exists } k \in [1, n] \text{ s.t. } \begin{cases} |A_k| > |B_k| \text{ and} \\ \forall i < k, |A_i| = |B_i| \end{cases}$$

■ A is a Lex-preferred α -consistent subbase of Δ if $A \subseteq \Delta$ and $str(A) \cup \{\alpha\} \not\models \perp$ and for any $B \subseteq \Delta$ s.t. $str(B) \cup \{\alpha\} \not\models \perp$, $B \not\succeq_{\Delta} A$ holds

■ $\alpha \sim_{\Delta} \beta$ iff for any Lex-preferred α -consistent subbase B of Δ , $str(B) \cup \{\alpha\} \models \beta$

► **Example 1** (continued). Let us consider that the common knowledge about the world consists only in the default persistence of the fluents *box*, *empty* and *visible* and on the default effects of the occurrences of events A , E and C when their preconditions hold.

$$\Delta = \left\{ \begin{array}{ll} \neg box_0 \rightsquigarrow \neg box_1 & (A_0 \vee E_0) \wedge \neg box_0 \rightsquigarrow box_1 \\ box_0 \rightsquigarrow box_1 & C_0 \wedge \neg visible_0 \rightsquigarrow visible_1 \\ \neg empty_0 \rightsquigarrow \neg empty_1 & C_0 \wedge \neg visible_0 \wedge empty_0 \rightsquigarrow \neg visible_1 \\ empty_0 \rightsquigarrow empty_1 & \neg box_1 \rightsquigarrow \neg box_2 \\ \neg visible_0 \rightsquigarrow \neg visible_1 & \dots \\ visible_0 \rightsquigarrow visible_1 & C_2 \wedge \neg visible_2 \wedge empty_2 \rightsquigarrow \neg visible_3 \end{array} \right\}$$

System Z will give a stratification in three strata where all persistence rules (of the form $v_t \rightsquigarrow v_{t+1}$ or $\neg v_t \rightsquigarrow \neg v_{t+1}$) are in the least specific stratum Δ_3 (since they are tolerated by all the other rules). As seen before, $(A_0 \vee E_0) \wedge \neg box_0 \rightsquigarrow box_1$ describes an exception to the persistence of $\neg box$, just as $C_0 \wedge \neg visible_0 \rightsquigarrow visible_1$ describes an exception to the persistence of $\neg visible$ which leads us to place them in Δ_2 , the latter itself admits an exception described by rule $C_0 \wedge \neg visible_0 \wedge empty_0 \rightsquigarrow \neg visible_1$ making it the most specific rule thus placed in Δ_1 by System Z algorithm. At the end, we get:

³ System Z ordering method is based on the tolerance notion between rules. More precisely, a rule $r = \alpha \rightsquigarrow \beta$ is tolerated by a set of n rules $R \subseteq \Delta$ iff $\alpha \wedge \beta \wedge \bigwedge_{\alpha_i \rightsquigarrow \beta_i \in R} (\neg \alpha_i \vee \beta_i)$ is consistent. The process continues until Δ contains only rules tolerated by all the other ones, they constitute the most specific stratum called Δ_1 (Δ_n being the least specific stratum, with n being the number of iterations).

$$\Delta_1 = \{C_t \wedge \neg \text{visible}_t \wedge \text{empty}_t \rightsquigarrow \neg \text{visible}_{t+1}\}_{t \in \{0,1,2\}}$$

$$\Delta_2 = \left\{ \begin{array}{l} C_t \wedge \neg \text{visible}_t \rightsquigarrow \text{visible}_{t+1} \\ (A_t \vee E_t) \wedge \neg \text{box}_t \rightsquigarrow \text{box}_{t+1} \end{array} \right\}_{t \in \{0,1,2\}} \quad \Delta_3 = \left\{ \begin{array}{l} v_t \rightsquigarrow v_{t+1} \\ \neg v_t \rightsquigarrow \neg v_{t+1} \end{array} \right\}_{\substack{t \in \{0,1,2\} \\ v \in \{\text{box}, \text{empty}, \text{visible}\}}}$$

Using lexicographic inference we get: $\neg \text{box}_0 \sim_{\Delta} \neg \text{box}_1$ and $\neg \text{box}_0 \wedge (A_0 \vee E_0) \sim_{\Delta} \text{box}_1$, meaning that a priori if there was no box at time 0, there is no box at time 1, but knowing that either A or E has placed a box makes it more plausible that there is a box at time 1.

Note that in this example, for the sake of simplicity, we want to make a closed world assumption (CWA) in order to express that the only possible way to change the variable box (respectively visible) from false to true is the occurrence of A or E (respectively the performance of action C):

$$\text{CWA} = \{(\neg \text{box}_t \wedge \text{box}_{t+1}) \rightarrow (A_t \vee E_t), (\neg \text{visible}_t \wedge \text{visible}_{t+1}) \rightarrow C_t\}_{t \in \{0,1,2\}}$$

From the set of default rules and the close world assumption, we can then obtain: $\{\text{box}_1\} \cup \text{CWA} \sim_{\Delta} \text{box}_0$ meaning that the most plausible interpretation is that when there is a box at time point 1 it means that there was already a box at time 0. However, if we know that there were no box at time 0 then $\{\text{box}_1, \neg \text{box}_0\} \cup \text{CWA} \sim_{\Delta} (A_0 \vee E_0)$

We choose to use the lexicographic entailment in this paper, because [6] have shown that it is a powerful non-monotonic inference relation that satisfies the set of rational properties called System P. The System P, introduced by Kraus, Lehmann and Magidor [31], gathers properties that should follow rationally when one wants to deduce new inferences from a set of existing inference rules. The following definition describes an agent epistemic states via the pieces of information that she believes.

► **Definition 2** (Agent epistemic state and inference). *A user is represented by a tuple $B = (F, B_{\mathcal{L}}, B_{\Delta})$ composed of a set $F \subseteq \mathcal{L}$ of formulas representing facts, and two sets $B_{\mathcal{L}} \subseteq \mathcal{L}$ and B_{Δ} respectively representing the strict and default rules known by the agent, the default rules of B_{Δ} are expressions of the form $\alpha \rightsquigarrow \beta$ with $\alpha, \beta \in \mathcal{L}$.*

When $F \cup B_{\mathcal{L}}$ and B_{Δ} are both consistent⁴, the user is equipped with an inference relation between formulas of \mathcal{L} denoted \sim_B defined by:

$$\alpha \sim_B \beta \quad \text{iff} \quad \left\{ \begin{array}{l} \{\alpha\} \cup F \cup B_{\mathcal{L}} \text{ is consistent and} \\ \text{for any Lex-preferred } (\alpha \wedge \bigwedge_{\varphi \in F \cup B_{\mathcal{L}}} \varphi)\text{-consistent subbase } A \in B_{\Delta}, \\ A \cup \{\alpha\} \cup F \cup B_{\mathcal{L}} \models \beta \end{array} \right.$$

In the following, $\sim_B \varphi$ is a shortcut for $\top \sim_B \varphi$.

In order to formally introduce curiosity, we need to define awareness. This will be done by simply stating that an agent is aware of a variable if this variable appears in the facts contained in its epistemic state, and we assume that when an agent is aware of a variable then it becomes also aware of every variable of the strict or default rules of its epistemic state containing this variable (mimicking a kind of introspection). We use the notation $v \in \varphi$ to express that the variable v appears in the formula φ , this notation can be applied to variables of \mathcal{V}_T as well as of \mathcal{V} .

⁴ Here consistent is not used with the same meaning: for the propositional formulas it means classical logic consistency, while for the default rules base it means that the base can be stratified.

- **Definition 3** (awareness). *An agent represented by $B = (F, B_{\mathcal{L}}, B_{\Delta})$ is*
- aware of a variable $v \in \mathcal{V}$ if
 - there is a formula $\varphi \in F$ s.t. $v \in \varphi$ or
 - there is a formula $\varphi \in B_{\mathcal{L}} \cup \text{str}(B_{\Delta})$ s.t. $v \in \varphi$ and there is a variable $v' \in \varphi$ of which the agent is aware; and
 - aware of a formula $\varphi \in \mathcal{L}$ iff for any variable $v_t \in \varphi$, the agent is aware of v .

► **Example 1** (continued). *Let us consider that the epistemic state of agent C is $(\emptyset, \text{CWA}, \Delta = \Delta_1 \cup \Delta_2 \cup \Delta_3)$, in this case it does not know any fact, which means that it is not aware of anything. Assume now that at time point 1, our agent Cecilia comes to her office and sees a box on her desk, then the epistemic state of agent C is $(\{\text{box}_1\}, \text{CWA}, \Delta)$. In this state, it is aware that a box is on the desk, moreover by introspection it is aware of the possibility to open it due to rule concerning C , the possibility that Albert or Erwin are able to put it on the desk, the possibility that this box is empty or that something inside of it could be visible.*

The following definition enables us to keep only formulas that do not concern a time point later than a given time point t , i.e., keep the formulas such that all their variables are indexed by time points no later than t .

► **Definition 4** (epistemic state until t). *Given an epistemic state $(F, B_{\mathcal{L}}, B_{\Delta})$ and a time point $t \in [0, N]$, the epistemic state until t , denoted $B_{\rightarrow t} = (F_{\rightarrow t}, B_{\mathcal{L} \rightarrow t}, B_{\Delta \rightarrow t})$, is defined by: $F_{\rightarrow t} = \{\varphi \in F \mid \text{for all } v_{t'} \in \varphi, t' \leq t\}$, $B_{\mathcal{L} \rightarrow t} = \{\varphi \in B_{\mathcal{L}} \mid \text{for all } v_{t'} \in \varphi, t' \leq t\}$, $B_{\Delta \rightarrow t} = \{\delta \in B_{\Delta} \mid \text{for all } v_{t'} \in \text{str}(\{\delta\}), t' \leq t\}$.*

In the following, we use $[\varphi]_{<t}$ (and respectively $[\varphi]_{\leq t}$, $[\varphi]_{>t}$, $[\varphi]_{\geq t}$ and $[\varphi]_t$) to denote that φ is a formula containing only variables indexed by time points earlier than t (resp. earlier than or equal to t , strictly later than t , later than or equal to t , equal to t).

► **Remark 1.** For any formula $\varphi \in F_{\rightarrow t} \cup B_{\mathcal{L} \rightarrow t} \cup \text{str}(B_{\Delta \rightarrow t})$, $[\varphi]_{\leq t}$ holds.

An agent is curious about a formula at time point t if according to its epistemic state until t it is aware of this formula but it is not able to deduce its truth value at time t .

► **Definition 5** (curiosity). *An agent with state B is curious about $\varphi \in \mathcal{L}$ at $t \in T$ if, according to $B_{\rightarrow t}$, it is aware of φ and $\not\vdash_{B_{\rightarrow t}} \varphi$ and $\not\vdash_{B_{\rightarrow t}} \neg\varphi$.*

► **Example 2.** *Coming back to Example 1, the epistemic state of C being $B = (\{\text{box}_1\}, \text{CWA}, \Delta)$, its state at 0 is $B_{\rightarrow 0} = (\emptyset, \emptyset, \emptyset)$, meaning that at 0 it is aware of nothing, thus according to Definition 5 it is not curious about anything at 0. In the epistemic state B , she first thinks that $\{\text{box}_1\} \cup \text{CWA} \vdash_{\Delta} \text{box}_0$. However, she remembers that there was no box on her desk at time 0 before she left her office. Meaning that her epistemic state is now $B' = (\{\neg\text{box}_0, \text{box}_1\}, \text{CWA}, \Delta)$ which enables her to draw the inference $\{\neg\text{box}_0, \text{box}_1\} \vdash_{B'} (A_0 \vee E_0)$, however there is no way of knowing which of Albert or Bernard (or both) dropped off the box. More formally, we can say that Cecilia is curious about the possibility that Albert dropped off the box at time 0 because she is aware of this possibility and $\{\neg\text{box}_0, \text{box}_1\} \not\vdash_{B'} A_0$ and $\{\neg\text{box}_0, \text{box}_1\} \not\vdash_{B'} \neg A_0$.*

Now if we consider that Albert told Cecilia that he placed a box on her desk at 0 before she entered her office. In that case, the epistemic state of Cecilia is $B'' = (\{A_0, \neg\text{box}_0, \text{box}_1\}, \text{CWA}, \Delta)$, there is no more ambiguity as she knows who put it there, hence she is not curious about A_0 .

To define suspense, we propose for this formalization to use Baroni's description of *primary suspense* [5] which relies solely on temporal and belief factors. Baroni also describes other types of suspense involving different emotional components (empathy and identification with a protagonist for instance). These components affect suspense by strengthening the intensity of curiosity, and we will leave them for further study at the time being. The following definition expresses that an agent feels suspense about a formula φ when this agent is curious about it at time t , and thinks that it is not impossible for facts or events (below denoted ψ) to come to light and reveal the truth of φ (satisfying curiosity about it at last).

► **Definition 6** (suspense). *An agent represented by an epistemic state $B = (F, B_{\mathcal{L}}, B_{\Delta})$ feels suspense about $\varphi \in \mathcal{L}$ at time point t if*

1. *according to B , the agent is curious about φ at time t and*
2. *there is a formula $\psi \in \mathcal{L}$ such that $[\psi]_{>t}$ and $F_{\rightarrow t} \cup B_{\mathcal{L}} \cup \{\psi\}$ consistent and*
3. *there is $t' > t$ s.t. either $\vdash_{B'} \varphi_{t'}$ or $\vdash_{B'} \neg \varphi_{t'}$ holds, with $B' = (F \cup \{\psi\}, B_{\mathcal{L}}, B_{\Delta})$.*

► **Example 3.** *In the context of Example 1, assume now that agent C has the following epistemic state $B = (\{\neg \text{box}_0, \text{box}_1, \neg \text{visible}_1\}, \text{CWA}, \Delta)$. Here, at time 1 agent C is aware of the box, she is also aware that it is either empty or not, but has no way at this time to know which is true. Formally, $\not\vdash_B \text{empty}$ and $\not\vdash_B \neg \text{empty}$. Hence she is curious about the variable empty at time point 1. Still according to Definition 3, the agent is also aware of the formulas $(C_2 \wedge \neg \text{visible}_2 \rightsquigarrow \text{visible}_3)$ and $(C_2 \wedge \neg \text{visible}_2 \wedge \text{empty}_2 \rightsquigarrow \neg \text{visible}_3)$. Meaning she is aware she will know the content of the box once she opens it.*

More precisely, the formula $\psi = C_2 \wedge \text{visible}_3$ can be added to the facts of the epistemic state because $\{\neg \text{box}_0, \text{box}_1, \neg \text{visible}_1\} \cup \text{CWA} \cup \{C_2 \wedge \text{visible}_3\}$ is consistent. Now, $B' = (\{\neg \text{box}_0, \text{box}_1, \neg \text{visible}_1, C_2, \text{visible}_3\}, \text{CWA}, \Delta)$ yields $\vdash_{B'} \neg \text{empty}_2$. Hence Cecilia feels suspense at time 1 about the truth value of empty.

In order to formalize surprise, following [20], we propose to exploit our non-monotonic setting that enables agents to imagine several more or less plausible situations, i.e., enables them to incorporate new contradicting information by revising previous conclusions. This is required in order to avoid locking the agent in a state of total incomprehension. Surprise can then be defined by the occurrence of a formula that was unexpected but which is completely plausible.

► **Definition 7** (surprise). *An agent represented by $B = (F, B_{\mathcal{L}}, B_{\Delta})$ is surprised at time t about a formula $\varphi \in \mathcal{L}$ if $\varphi \in F_{\rightarrow t}$ and $B_{\rightarrow t}$ is consistent (φ occurred and it was not impossible) and $B' = (F_{\rightarrow t-1}, B_{\mathcal{L} \rightarrow t}, B_{\Delta \rightarrow t})$ is such that: $\vdash_{B'} \neg \varphi$ (φ was unexpected)*

► **Example 2** (continued). *Cecilia is surprised to find the box at time 1. Indeed, given the epistemic state $B = (\{\neg \text{box}_0, \text{box}_1\}, \text{CWA}, \Delta)$, before seeing the box at time 1, the persistence of $\neg \text{box}_0$ into $\neg \text{box}_1$ was the most plausible evolution. More formally, we can check that $\text{box}_1 \in F$ and $B' = (\{\neg \text{box}_0\}, \text{CWA}_{\rightarrow 1}, \Delta_{\rightarrow 1})$ is such that $\vdash_{B'} \neg \text{box}_1$, and B is consistent (hence $B_{\rightarrow 1}$ as well).*

4 Properties and graduality

In this section we show several simple properties relating the three emotions, moreover we establish the computational complexity of their detection.

4.1 Properties derived from the definitions

An agent who knows nothing is aware of nothing.

► **Proposition 1.** *If the epistemic state of an agent has no fact, i.e., $B = (\emptyset, B_{\mathcal{L}}, B_{\Delta})$ then the agent is not aware of any variable.*

Proof. Even if $B_{\mathcal{L}}$ or B_{Δ} are non-empty, there is no awareness since no variable appears in F . ◀

This kind of agent is not curious nor able to feel suspense since curiosity requires awareness, and suspense requires curiosity.

► **Corollary 2** (of Proposition 1). *If the epistemic state of an agent has no fact, i.e., $B = (\emptyset, B_{\mathcal{L}}, B_{\Delta})$, then the agent is not curious and does not feel suspense about any formula at any time point.*

The following proposition states that an omniscient agent (i.e., an agent with complete information about the world) is never curious nor able to feel suspense.

► **Proposition 2.** *If the epistemic state $B = (F, B_{\mathcal{L}}, B_{\Delta})$ of an agent admits only one most plausible interpretation in Ω , then for any finite formula, there is no time point where the agent is curious or feels suspense about it.*

Proof. In order to be curious, there should exist at least one variable whose truth value is unknown. Hence there should be at least two interpretations that are equally most plausible. ◀

Because surprise occurs when the agent expects something and then the opposite happens, it means that it is not curious about it (because the surprise makes it know it).

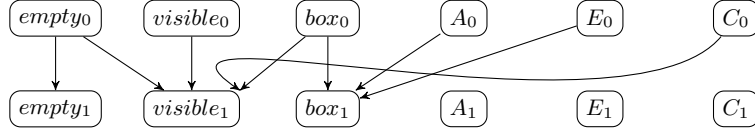
► **Proposition 3.** *Given an epistemic state B of an agent, if the agent is surprised about φ at time t then the agent is not curious about φ neither at time $t - 1$ nor at time t .*

Proof. Let us assume that $B = (F, B_{\mathcal{L}}, B_{\Delta})$ is surprised at time t about a formula $\varphi \in \mathcal{L}$, it means that $\varphi \in F_{\rightarrow t}$ and $B_{\rightarrow t}$ is consistent and $B' = (F_{\rightarrow t-1}, B_{\mathcal{L} \rightarrow t}, B_{\Delta \rightarrow t})$ is such that: $\vdash_{B'} \neg\varphi$. It means that the agent could infer the truth value of φ at time $t - 1$, hence she was not curious at $t - 1$. Now since $\varphi \in F$ and $B_{\rightarrow t}$ consistent then $\vdash_{B_{\rightarrow t}} \varphi$ hence she is not curious about it at t . ◀

This proposition shows that surprise and curiosity are antagonists in a given epistemic state, however we can imagine stories where the same event sequence may produce curiosity (e.g. by keeping some information hidden, namely the name of the murderer) when told in a given way and surprise when told differently (e.g. revealing this same information at start). The following proposition shows the complexity class of the decision problems associated to awareness, curiosity, suspense and surprise.

► **Proposition 4.** *Given an epistemic state B , a formula $\varphi \in \mathcal{L}$ and a time point $t \in T$,*
 ■ *Deciding whether B is aware of φ at time point t is linear*
 ■ *Deciding whether B is curious or feels suspense or surprise about φ at t is P^{NP} -complete.*

Proof. In order to check awareness about a variable, it is enough to check membership of this variable to a set of formulas, which is linear in the size of the epistemic state, this process should be repeated for all the variables of a formula to check formula awareness. Concerning



■ **Figure 1** Causal graph induced by the epistemic state $B_{\rightarrow 1}$.

curiosity, in addition to a test of awareness, it uses two lexicographic inference tests which have been shown to be in P^{NP} by [21]. Suspense requires a curiosity check and a consistency check of the strict part of the base B , which is a SAT problem hence NP-complete. It then requires several lexicographic inferences in order to find the time point where $\vdash_{B'} \varphi_{t'}$ or $\vdash_{B'} \neg \varphi_{t'}$ holds. Surprise requires a consistency check of the default base of B (which is a P^{NP} -complete problem according to [21]) and a lexicographic inference, hence the result. ◀

The complexity P^{NP} of these decision problems is due to the use of the lexicographic inference in their definition. Note that the upper bound (N) on time steps could relieve the computational complexity as obtained in traditional STRIPS planning [11] where the complexity of certain decision problems drops from PSPACE-complete to NP-complete. Note also that formulation of AI planning in answer set programming gives rise to similar complexity [50].

4.2 Towards defining measures

For further characterizing narrative tension, we need to quantify the intensity of the emotions generated in an agent when listening to a story. This section is a first attempt towards this goal. We propose three definitions of the emotional intensity of curiosity, suspense and surprise. In the following definition we propose to rely on findings from Trabasso and Sperry [53] as a heuristic in order to evaluate the intensity of the curiosity. We first define the causal graph associated with an epistemic state as the one relating variables of \mathcal{V}_T with the links induced by the default rules and strict rules of the epistemic state.

► **Definition 8** (causal graph). *The causal graph \mathcal{G}_B induced by an epistemic state $B = (F, B_{\mathcal{L}}, B_{\Delta})$ is a pair (V_B, E_B) with*

- $V_B = \{v_t \in \mathcal{V}_T \mid v_t \in \varphi, \varphi \in F \cup B_{\mathcal{L}} \cup \text{str}(B_{\Delta})\}$ is the set of vertices of \mathcal{G}_B
- $E_B = \{(v_t, v_{t'}) \in \mathcal{V}_T \times \mathcal{V}_T \mid v_t \in \alpha, v_{t'} \in \beta, \alpha \rightsquigarrow \beta \in B_{\Delta}\} \cup \{(v_t, v_{t'}) \in \mathcal{V}_B \times \mathcal{V}_B \mid \{t\} \cup F \cup B_{\mathcal{L}} \models l_{t'} \text{ with } l_t \in \{v_t, \neg v_t\}, l_{t'} \in \{v_{t'}, \neg v_{t'}\}\}$

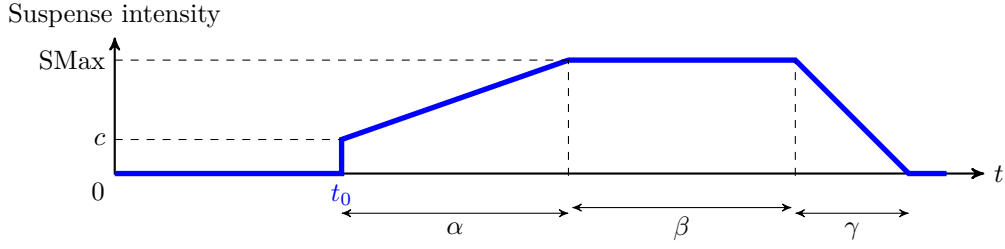
We illustrate this definition on the epistemic state of Cecilia until time point 1.

► **Example 3** (continued). *Considering $B = (\{-box_0, box_1, \neg visible_1\}, CWA, \Delta)$, the causal graph induced by $B_{\rightarrow 1}$ is shown in Figure 1.*

► **Definition 9** (curiosity intensity). *Given an agent with state $B = (F, B_{\mathcal{L}}, B_{\Delta})$ and curious about $\varphi \in \mathcal{L}$ at $t \in T$, her curiosity intensity level is $c_B(\varphi, t) = \sum_{v_{t'} \in \varphi} \text{deg}(v_{t'})$ where $\text{deg}(x)$ is the degree of the node x in the causal graph induced by B .*

► **Example 3** (continued). *Given the epistemic state of Cecilia $B = (\{-box_0, box_1, \neg visible_1\}, CWA, \Delta)$, she is the most curious about $\neg visible_1$ with intensity 5, denoted $c_B(\neg visible_1, 1) = 5$. Note that the degree of $visible_1$ is only four on Figure 1 but there is a supplementary outgoing arc from $visible_1$ to $visible_2$ when considering B instead of $B_{\rightarrow 1}$.*

10:12 What Killed the Cat? Towards a Logical Formalization of Curiosity in Narratives.



■ **Figure 2** Suspense intensity along time (c being the level of curiosity felt at time t_0).

Lets us now consider an example of suspense evolution. As previously explained, we base our definition only on beliefs and time. According to Baroni [5], once curiosity is aroused then the suspense begins and lasts until it reaches a plateau, at which point it diminishes and gradually fades away, unless the suspense is resolved in the meantime. We propose to consider that the suspense profile of an agent is available under the form of a quadruplet $(\alpha, \beta, \gamma, SMax)$ where α is the duration before reaching the maximum of intensity $SMax$, β the length of the plateau and γ the descent duration (see Figure 2). Thus, suspense intensity is a function of the curiosity intensity at the time it is first felt and of the duration between its triggering and its resolution.

► **Definition 10** (suspense intensity). *Given an epistemic state $B = (F, B_{\mathcal{L}}, B_{\Delta})$ and a suspense profile $p = (\alpha, \beta, \gamma, SMax)$ then the intensity of the suspense feeling at t is*

$$s_B^p(\varphi, t) = \begin{cases} 0 & \text{if } t < t_0 \\ \frac{SMax - c}{\alpha}(t - t_0) + c & \text{if } t \in [t_0, t_0 + \alpha] \\ SMax & \text{if } t \in [t_0 + \alpha, t_0 + \alpha + \beta] \\ -\frac{SMax}{\gamma}(t - t_0 - \alpha - \beta) + SMax & \text{if } t \in [t_0 + \alpha + \beta, t_0 + \alpha + \beta + \gamma] \\ 0 & \text{if } t \geq t_0 + \alpha + \beta + \gamma \end{cases}$$

where t_0 is the earliest time where the agent was curious about φ and $c = c_B(\varphi, t_0)$ is the curiosity intensity at t_0 .

Note that in this definition, the suspense intensity may only vary according to the profile of the agent and the duration. A more refined way to handle this would be to define a decreasing persistence of awareness, enabling the agent to forget a variable after some delay, it would be in accordance with the common knowledge that the suspense should be revived from time to time.

Concerning surprise intensity about a formula φ , we propose to adopt the point of view of Shackle [48] as done in [20], by assimilating it to the degree of impossibility of φ (or equivalently the possibility degree of $\neg\varphi$). It amounts to finding the most specific rule that is violated by $\varphi \cup F \cup B_{\mathcal{L}}$, the more specific this rule, the more surprising φ becomes⁵.

► **Definition 11** (surprise intensity). *Given an epistemic state $B = (F, B_{\mathcal{L}}, B_{\Delta} = \Delta_1 \dots \Delta_n)$ where there is a surprise at time t , the surprise intensity is $surp_B(\varphi, t) = n - i$, where i is the most specific strata level, s.t. there is a rule $\alpha \rightsquigarrow \beta \in \Delta_i$ with $\{\varphi \wedge \alpha\} \cup F \cup B_{\mathcal{L}} \models \neg\beta$.*

The definitions of this section are a first step towards being able to compare stories with respect to the intensity of emotions they generate in the agent listening to them.

⁵ Such a definition is classical in possibility theory, and more specifically in the context where a default rule $\alpha \rightsquigarrow \beta$ is interpreted as a constraint on a possibility measure Π see e.g., [7]. This constraint being $\Pi(\alpha \wedge \beta) > \Pi(\alpha \wedge \neg\beta)$ expressing that when α is true, having β true is more possible than β false.

5 Conclusion

This paper aims at providing a unified framework in which the three emotions at the heart of narrative tension, namely curiosity, surprise, and suspense are formalized and their relationships clarified. This framework is built on non-monotonic reasoning for representing compactly the default behavior of the world and also for simulating the reasoning of an agent in front of a story. The use of non-monotonic reasoning induces a cost in complexity: the detection problems associated with the three emotions are in P^{NP} (due to the use of lexicographic inference). For each of the three emotions, we describe methods to evaluate their intensity.

While we illustrated our formalization by adopting the point of view of a single agent in a chronological story for the sake of clarity, it does not preclude its adaptability for storytelling using other points of views such as an extradiegetic narrator disclosing knowledge to the listener through a discourse that does not reflect the timeline of the story.

To operationalize this model, we plan to investigate different frameworks that are equipped with solvers namely PDDL planning, Linear logic with Ceptre [35] and propositional default logic with TouIST [49]. Moreover, the inherent growing complexity of this problem for scaling to complex narratives requires further study about the granularity of story events, for instance inspired by discussions about the representation of causality [36].

References

- 1 Carole Adam, Andreas Herzig, and Dominique Longin. A logical formalization of the OCC theory of emotions. *Synthese*, 168(2):201–248, May 2009. doi:10.1007/s11229-009-9460-9.
- 2 Carlos E Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The journal of symbolic logic*, 50(2):510–530, 1985. doi:10.2307/2274239.
- 3 Guillaume Aucher and Thomas Bolander. Undecidability in epistemic planning. In F. Rossi, editor, *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI'2013)*, pages 27–33, Beijing, China, 2013. URL: <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6903>.
- 4 Heather Barber and Daniel Kudenko. Generation of dilemma-based interactive narratives with a changeable story goal. In *2nd International Conference on INtelligent TEchnologies for interactive enterTAINment*. ICST, May 2010. doi:10.4108/ICST.INTETAIN2008.2477.
- 5 Raphaël Baroni. *La tension narrative: suspense, curiosité et surprise*. Poétique. Éd. du Seuil, Paris, 2007.
- 6 Salem Benferhat, Claudette Cayrol, Didier Dubois, Jérôme Lang, and Henri Prade. Inconsistency management and prioritized syntax-based entailment. In *Proc. of Int. Joint. Conf. on Artificial Intelligence (IJCAI'93)*, volume 93, pages 640–645, 1993.
- 7 Salem Benferhat, Didier Dubois, and Henri Prade. Nonmonotonic reasoning, conditional objects and possibility theory. *Artif. Intell.*, 92(1-2):259–276, 1997. doi:10.1016/S0004-3702(97)00012-X.
- 8 Thomas Bolander and Mikkel Birkegaard Andersen. Epistemic planning for single-and multi-agent systems. *Journal of Applied Non-Classical Logics*, 21(1):9–34, 2011. doi:10.3166/JANCL.21.9-34.
- 9 Lorenzo Bonoli. Raphaël Baroni, La tension narrative. Suspense, curiosité, surprise, Paris, Seuil, 2007. *Cahiers de Narratologie. Analyse et théorie narratives*, 14, February 2008. doi:10.4000/narratologie.608.
- 10 Anne-Gwenn Bosser, Marc Cavazza, and Ronan Champagnat. Linear Logic for Non-Linear Storytelling. *ECAI 2010*, pages 713–718, 2010. Publisher: IOS Press. doi:10.3233/978-1-60750-606-5-713.

10:14 What Killed the Cat? Towards a Logical Formalization of Curiosity in Narratives.

- 11 Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994. doi:10.1016/0004-3702(94)90081-7.
- 12 Rogelio E. Cardona-Rivera, Bradley A. Cassell, Stephen G. Ware, and R. Michael Young. *Indexter : A Computational Model of the Event-Indexing Situation Model for Characterizing Narratives*, 2012.
- 13 Rogelio E. Cardona-Rivera, Arnav Jhala, Julie Porteous, and R. Michael Young. The story so far on narrative planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34(1):489–499, May 2024. doi:10.1609/icaps.v34i1.31509.
- 14 Noël Carroll. Narrative closure. *Philosophical Studies*, 135(1):1–15, August 2007. doi:10.1007/s11098-007-9097-9.
- 15 Yun-Gyung Cheong and R. Michael Young. Suspenser: A Story Generation System for Suspense. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(1):39–52, March 2015. doi:10.1109/TCIAIG.2014.2323894.
- 16 Christophe Dousson and Pierre Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *IJCAI*, volume 7, pages 324–329. Citeseer, 2007. URL: <http://ijcai.org/Proceedings/07/Papers/050.pdf>.
- 17 Florence Dupin de Saint-Cyr. Scenario Update Applied to Causal Reasoning. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 11th International Conference, KR 2008*, pages 188–197, January 2008. URL: <http://www.aaai.org/Library/KR/2008/kr08-019.php>.
- 18 Florence Dupin de Saint-Cyr, Andreas Herzig, Jérôme Lang, and Pierre Marquis. Reasoning About Action and Change. In Pierre Marquis, Odile Papini, and Henri Prade, editors, *A Guided Tour of Artificial Intelligence Research*, volume 1 / 3 of *Knowledge Representation, Reasoning and Learning*, pages 487–518. Springer International Publishing, May 2020. doi:10.1007/978-3-030-06164-7_15.
- 19 Florence Dupin De Saint-Cyr and Jérôme Lang. Belief extrapolation (or how to reason about observations and unpredicted change). *Artificial Intelligence*, 175(2):760–790, February 2011. doi:10.1016/j.artint.2010.11.002.
- 20 Florence Dupin de Saint-Cyr and Henri Prade. Belief revision and incongruity: Is it a joke? *Journal of Applied Non-Classical Logics*, 33(3-4):467–494, October 2023. doi:10.1080/11663081.2023.2244379.
- 21 Thomas Eiter and Thomas Lukasiewicz. Default reasoning from conditional knowledge bases: Complexity and tractable cases. *Artificial Intelligence*, 124(2):169–241, 2000. doi:10.1016/S0004-3702(00)00073-4.
- 22 Paul Ekman. An argument for basic emotions. *Cognition and Emotion*, 6(3-4):169–200, May 1992. doi:10.1080/02699939208411068.
- 23 Jeffrey Ely, Alexander Frankel, and Emir Kamenica. Suspense and Surprise. *Journal of Political Economy*, 123(1):215–260, February 2015. doi:10.1086/677350.
- 24 Joseph Jeffrey Finger. *Exploiting constraints in design synthesis*. PhD thesis, Stanford University, Stanford, CA, 1987.
- 25 Gerard Genette. *Nouveau Discours du Récit*. Seuil, 1983.
- 26 Moisés Goldszmidt and Judea Pearl. On the consistency of defeasible databases. *Artificial Intelligence*, 52(2):121–149, 1991. doi:10.1016/0004-3702(91)90039-M.
- 27 Melanie Green and Timothy Brock. The Role of Transportation in the Persuasiveness of Public Narrative. *Journal of personality and social psychology*, 79:701–21, November 2000. doi:10.1037/0022-3514.79.5.701.
- 28 Joseph Y. Halpern. Alternative Semantics for Unawareness. *Games and Economic Behavior*, 37(2):321–339, November 2001. doi:10.1006/game.2000.0832.
- 29 Kaarlo Jaakko Juhani Hintikka. *Knowledge and belief: An introduction to the logic of the two notions*. Cornell University Press, Ithaca and London, 1962.
- 30 Hirofumi Katsuno and Alberto O Mendelzon. On the difference between updating a knowledge base and revising it. *KR*, 91:387–394, 1991.

- 31 Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial intelligence*, 44(1-2):167–207, 1990. doi:10.1016/0004-3702(90)90101-5.
- 32 Daniel Lehmann. Another perspective on default reasoning. *Annals of mathematics and artificial intelligence*, 15:61–82, 1995. doi:10.1007/BF01535841.
- 33 Emiliano Lorini and Cristiano Castelfranchi. The cognitive structure of surprise: looking for basic principles. *Topoi*, 26(1):133–149, 2007.
- 34 Emiliano Lorini and François Schwarzentruher. A logic for reasoning about counterfactual emotions. *Artificial Intelligence*, 175(3):814–847, March 2011. doi:10.1016/j.artint.2010.11.022.
- 35 Chris Martens. Ceptre: A language for modeling generative interactive systems. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11, pages 51–57, 2015. URL: <http://www.aaai.org/ocs/index.php/AIIDE/AIIDE15/paper/view/11536>.
- 36 Lawrence J. Mazlack. Granular causality speculations. In *IEEE Annual Meeting of the Fuzzy Information, 2004. Processing NAFIPS '04.*, volume 2, pages 690–695 Vol.2, 2004. doi:10.1109/NAFIPS.2004.1337385.
- 37 John McCarthy. Epistemological problems of artificial intelligence. In *Proc. Int. Joint Conf on Artificial Intelligence (IJCAI'77)*, pages 1038–1044. Elsevier, 1977. URL: <http://ijcai.org/Proceedings/77-2/Papers/094.pdf>.
- 38 John McCarthy and Patrick J Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.
- 39 Salvatore Modica and Aldo Rustichini. Awareness and partitionial information structures. *Theory and decision*, 37:107–124, 1994.
- 40 Emma Norling. Capturing the quake player: Using a BDI agent to model human behaviour. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1080–1081, Melbourne Australia, July 2003. ACM. doi:10.1145/860575.860805.
- 41 Andrew Ortony, Gerald L Clore, and Allan Collins. *The cognitive structure of emotions*. Cambridge university press, 2022.
- 42 Judea Pearl. System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In *Proc. 3rd Conf. on Theoretical Aspects of Reasoning about Knowledge*, pages 121–135, 1990.
- 43 David Pizzi, Fred Charles, Jean-Luc Lugrin, and Marc Cavazza. Interactive Storytelling with Literary Feelings, April 2007. doi:10.1007/978-3-540-74889-2_55.
- 44 Robert Plutchik. A general psychoevolutionary theory of emotion. In *Theories of emotion*, pages 3–33. Elsevier, 1980.
- 45 Vladimir Propp. *Morphology of the Folktale: Second Edition*. University of Texas Press, 1968. doi:10.7560/783911.
- 46 Raymond Reiter. A logic for default reasoning. *Artificial intelligence*, 13(1-2):81–132, 1980. doi:10.1016/0004-3702(80)90014-4.
- 47 Jessica Rivera-Villicana, Fabio Zambetta, James Harland, and Marsha Berry. Using BDI to Model Players Behaviour in an Interactive Fiction Game. In Frank Nack and Andrew S. Gordon, editors, *Interactive Storytelling*, volume 10045, pages 209–220. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-48279-8_19.
- 48 George Lennox Sharman Shackle. *Decision, Order and Time in Human Affairs*. (2nd edition), Cambridge University Press, UK, 1961.
- 49 Khaled Skander Ben Slimane, Alexis Comte, Olivier Gasquet, Abdelwahab Heba, Olivier Lezaud, Frederic Maris, and Maël Valais. Twist your logic with touist. *CoRR*, abs/1507.03663, 2015. arXiv:1507.03663.

10:16 What Killed the Cat? Towards a Logical Formalization of Curiosity in Narratives.

- 50 Tran Cao Son, Enrico Pontelli, Marcello Balduccini, and Torsten Schaub. Answer set planning: a survey. *Theory and Practice of Logic Programming*, 23(1):226–298, 2023. doi:10.1017/S1471068422000072.
- 51 Meir Sternberg. How Narrativity Makes a Difference. *Narrative*, 9(2):115–122, 2001. arXiv:20107236.
- 52 David Thue and Vadim Bulitko. Modelling goal-directed players in digital games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 2, pages 86–91, 2006. URL: <http://www.aaai.org/Library/AIIDE/2006/aiide06-018.php>.
- 53 Tom Trabasso and Linda L Sperry. Causal relatedness and importance of story events. *Journal of Memory and Language*, 24(5):595–611, October 1985. doi:10.1016/0749-596X(85)90048-8.
- 54 Johan Van Benthem and Fernando R Velázquez-Quesada. The dynamics of awareness. *Synthese*, 177:5–27, 2010. doi:10.1007/S11229-010-9764-9.
- 55 Marianne Winslett. *Updating Logical Databases*. Cambridge University Press, 1990.

Faster Algorithm for Converting an STNU into Minimal Dispatchable Form

Luke Hunsberger   

Vassar College, Poughkeepsie, NY, USA

Roberto Posenato   

University of Verona, Italy

Abstract

A Simple Temporal Network with Uncertainty (STNU) is a data structure for representing and reasoning about temporal constraints on activities, including those with uncertain durations. An STNU is dispatchable if it can be flexibly and efficiently executed in real time while guaranteeing that all relevant constraints are satisfied. The number of edges in a dispatchable network affects the computational work that must be done during real-time execution. Recent work presented an $O(kn^3)$ -time algorithm for converting a dispatchable STNU into an equivalent dispatchable network having a minimal number of edges, where n is the number of timepoints and k is the number of actions with uncertain durations. This paper presents a modification of that algorithm, making it an order of magnitude faster, down to $O(n^3)$. Given that in typical applications $k = O(n)$, this represents an effective order-of-magnitude reduction from $O(n^4)$ to $O(n^3)$.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning; Theory of computation → Dynamic graph algorithms

Keywords and phrases Temporal constraint networks, dispatchable networks

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.11

1 Background

Temporal constraint networks facilitate representing and reasoning about temporal constraints on activities. *Simple Temporal Networks with Uncertainty* (STNUs) are one of the most important kinds of temporal networks because they allow the explicit representation of actions with uncertain durations [13]. An STNU is *dispatchable* if it can be executed by a flexible and efficient real-time execution algorithm while guaranteeing that all of its constraints will be satisfied. This paper modifies an existing algorithm for converting a dispatchable network into an equivalent dispatchable network having a minimal number of edges, making it an order of magnitude faster.

1.1 Simple Temporal Networks

A *Simple Temporal Network* (STN) is a pair $(\mathcal{T}, \mathcal{C})$ where \mathcal{T} is a set of real-valued variables called timepoints; and \mathcal{C} is a set of *ordinary* constraints, each of the form $(Y - X \leq \delta)$ for $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$ [3]. An STN is *consistent* if it has a solution as a constraint satisfaction problem (CSP). Each STN has a corresponding graph where the timepoints serve as nodes and the constraints correspond to labeled, directed edges. In particular, each constraint $(Y - X \leq \delta)$ corresponds to an edge $X \xrightarrow{\delta} Y$ in the graph. For convenience, such edges may be notated as (X, δ, Y) or, if the weight is not being considered, simply XY . Similarly, a path from X to Y may be notated by listing the timepoints visited by the path (e.g., $XUVWY$) or, if the context is clear, simply XY .

A flexible and efficient *real-time execution* (RTE) algorithm has been defined for STNs that maintains time windows for each timepoint and, as each timepoint X is executed, only propagates constraints *locally*, to *neighbors* of X in the STN graph [16, 14]. An STN is called



© Luke Hunsberger and Roberto Posenato;

licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 11; pp. 11:1–11:14

Leibniz International Proceedings in Informatics



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

dispatchable if that RTE algorithm is guaranteed to satisfy all of the STN’s constraints no matter how the flexibility afforded by the algorithm is exploited during execution. Morris [12] proved that a consistent STN is dispatchable if and only if every pair of timepoints that are connected by a path in the STN graph are connected by a shortest *vee-path* (i.e., a shortest path comprising zero or more negative edges followed by zero or more non-negative edges). Algorithms for generating equivalent dispatchable STNs having a *minimal number of edges* have been presented [16, 14]. Minimizing the number of edges is important since it directly impacts the real-time computations required during execution.

1.2 Simple Temporal Networks with Uncertainty

A *Simple Temporal Network with Uncertainty* (STNU) augments an STN to include *contingent links* that represent actions with uncertain, but bounded durations [13]. An STNU is a triple $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ where $(\mathcal{T}, \mathcal{C})$ is an STN, and \mathcal{L} is a set of contingent links, each of the form (A, x, y, C) , where $A, C \in \mathcal{T}$ and $0 < x < y < \infty$. The semantics of STNU execution ensures that regardless of when the *activation timepoint* A is executed, the *contingent timepoint* C will occur such that $C - A \in [x, y]$. Thus, the duration $C - A$ is uncontrollable, but bounded. Each STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ has a corresponding graph $\mathcal{G} = (\mathcal{T}, \mathcal{E}_o, \mathcal{E}_{lc}, \mathcal{E}_{uc})$, where $(\mathcal{T}, \mathcal{E}_o)$ is the graph for the STN $(\mathcal{T}, \mathcal{C})$, and \mathcal{E}_{lc} and \mathcal{E}_{uc} are sets of *labeled* edges corresponding to the contingent durations in \mathcal{L} . In particular, each contingent link (A, x, y, C) in \mathcal{L} has a *lower-case* (LC) edge $A \xrightarrow{c:x} C$ in \mathcal{E}_{lc} that represents the uncontrollable possibility that the duration might take on its minimum value x ; and an *upper-case* (UC) edge $C \xrightarrow{C:-y} A$ in \mathcal{E}_{uc} that represents the possibility that it might take on its maximum value y . For convenience, edges such as $A \xrightarrow{c:x} C$ and $C \xrightarrow{C:-y} A$ may be notated as $(A, c:x, C)$ and $(C, C:-y, A)$, respectively.

An STNU is *dynamically controllable* (DC) if there exists a dynamic, real-time execution strategy that guarantees that all constraints in \mathcal{C} will be satisfied no matter how the contingent durations turn out [13, 4]. A strategy is dynamic in that its execution decisions can react to observations of contingent executions, but with no advance knowledge of future events. Morris [10] proved that an STNU is DC if and only if it does not include any *semi-reducible* negative cycles (SRN cycles). (A path \mathcal{P} is semi-reducible if certain constraint-propagation rules can be used to provide new edges that effectively *bypass* each occurrence of an LC edge in \mathcal{P} .) In 2014, Morris [11] presented the first $O(n^3)$ -time DC-checking algorithm.¹ In 2018, Cairo *et al.* [1] presented their $O(mn + k^2n + kn \log n)$ -time RUL^- DC-checking algorithm. Hunsberger and Posenato [6] subsequently presented a faster version, called $\text{RUL}2021$, that has the same worst-case complexity but achieves an order-of-magnitude speedup in practice by restricting the edges it inserts into the network during constraint propagation.

1.3 Flexible and Efficient Real-time Execution

Most DC-checking algorithms generate conditional *wait* constraints that must be satisfied by any valid execution strategy. Each wait is represented by a labeled edge of the form $W \xrightarrow{C:-w} A$, which may be notated as $(W, C:-w, A)$. (Despite the similar notation, a wait is distinguishable from the original UC edge since its source timepoint is *not* the contingent timepoint C .) Such a wait can be glossed as: “While C remains unexecuted, W must wait at least w after A .” Morris [11] defined an *Extended STNU* (ESTNU) to be an STNU augmented with such waits. Thus, the graph for an ESTNU includes a set \mathcal{E}_{ucg} of generated wait edges. For convenience, we intentionally blur the distinction between an ESTNU and its graph.

¹ As is common in the literature, we use n for the number of timepoints, m for the number of ordinary constraints; and k for the number of contingent links.

Morris then extended the notion of dispatchability to ESTNUs, defining it in terms of the ESTNU's STN projections. A *projection* of an ESTNU is the STN derived from assigning fixed values to the contingent durations. In any projection, each edge from the ESTNU projects onto an ordinary edge [12, 9]. For example, consider the contingent link $(A, 1, 10, C)$ and the projection where its duration $C - A$ equals 4. In that projection, the LC and UC edges, $(A, c:1, C)$ and $(C, C:-10, A)$, project onto the respective ordinary edges, $(A, 4, C)$ and $(C, -4, A)$, representing that $C - A = 4$. Meanwhile, the wait edges, $(W, C:-7, A)$ and $(V, C:-3, A)$, project onto $(W, -4, A)$ and $(V, C:-3, A)$, respectively, since the wait on W expires when C executes at $A + 4$, and the wait on V is satisfied at time $A + 3$.

Morris defined an ESTNU to be dispatchable if all of its STN projections are dispatchable (as STNs). He then argued that a dispatchable ESTNU would necessarily provide a guarantee of flexible and efficient real-time execution. Hunsberger and Posenato [9] later:

1. formally defined a flexible and efficient real-time execution algorithm for ESTNUs, called RTE*;
2. defined an ESTNU to be dispatchable if every run of RTE* necessarily satisfies all of the ESTNU's constraints; and
3. proved that an ESTNU satisfying their definition of dispatchability necessarily satisfies Morris' definition (i.e., all of its STN projections are STN-dispatchable).

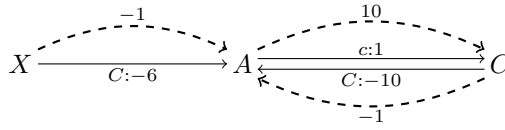
The RTE* algorithm provides maximum flexibility during execution, unlike the *earliest-first* strategy used for non-dispatchable networks [5].

Most DC-checking algorithms do not generate dispatchable ESTNUs. However, Morris [11] argued that his $O(n^3)$ -time DC-checking algorithm could be modified, without impacting its complexity, to generate a dispatchable output. In 2023, Hunsberger and Posenato [7] presented a faster, $O(mn + kn^2 + n^2 \log n)$ -time ESTNU-dispatchability algorithm called FD_{ESTNU} . However, neither of these algorithms provides any guarantees about the number of edges in the dispatchable output. Since the number of edges in the network directly impacts the real-time computations required to execute the network, it is important to minimize that number. Hunsberger and Posenato [8] subsequently presented the first ESTNU-dispatchability algorithm, called $\text{minDisp}_{\text{ESTNU}}$, that, in $O(kn^3)$ time, generates an equivalent dispatchable ESTNU *having a minimal number of edges*. To date, it is the only such algorithm. The main contribution of this paper is to modify $\text{minDisp}_{\text{ESTNU}}$ so that it solves the same problem in $O(n^3)$ -time, an order of magnitude faster, especially since it is common that $k = O(n)$, meaning the reduction in complexity is effectively from $O(n^4)$ to $O(n^3)$.

2 Overview of the Existing $\text{minDisp}_{\text{ESTNU}}$ Algorithm

The $\text{minDisp}_{\text{ESTNU}}$ algorithm [8] takes a dispatchable ESTNU $\mathcal{E} = (\mathcal{T}, \mathcal{E}_o, \mathcal{E}_{lc}, \mathcal{E}_{uc}, \mathcal{E}_{ucg})$ as its only input and generates as its output an equivalent dispatchable ESTNU having a minimal number of edges. (Such an ESTNU is called a μESTNU for \mathcal{S} .) It has four steps:

1. Compute the set $\mathcal{E}_o^{\text{si}}$ of so-called *stand-in* edges: ordinary edges that are entailed by various combinations ordinary, LC, UC, and wait edges from the ESTNU.
2. Apply the STN-dispatchability algorithm from Tsamardinos *et al.* [16] to the resulting set of ordinary edges, thereby generating a dispatchable STN subgraph, $(\mathcal{T}, \mathcal{E}_o^*)$.
3. Let $\hat{\mathcal{E}}_o^* = \mathcal{E}_o^* \setminus \mathcal{E}_o^{\text{si}}$ be the result of removing any remaining stand-in edges from \mathcal{E}_o^* .
4. Compute the set of wait edges that are not needed for dispatchability and remove them from \mathcal{E}_{ucg} ; call the resulting set $\hat{\mathcal{E}}_{ucg}$; then return the μESTNU $(\mathcal{T}, \hat{\mathcal{E}}_o^*, \mathcal{E}_{lc}, \mathcal{E}_{uc}, \hat{\mathcal{E}}_{ucg})$.



■ **Figure 1** (Dashed) stand-in edges entailed by individual labeled edges.

The worst-case time complexity of the $\text{minDisp}_{\text{ESTNU}}$ algorithm is dominated by the first step: finding the set $\mathcal{E}_o^{\text{si}}$ of so-called *stand-in* edges. Therefore, our new, faster algorithm modifies only that step, achieving an order-of-magnitude reduction in the overall worst-case time complexity. The rest of this section gives an overview of Step 1 of the existing $\text{minDisp}_{\text{ESTNU}}$ algorithm, as implemented by its genStandIns helper algorithm.

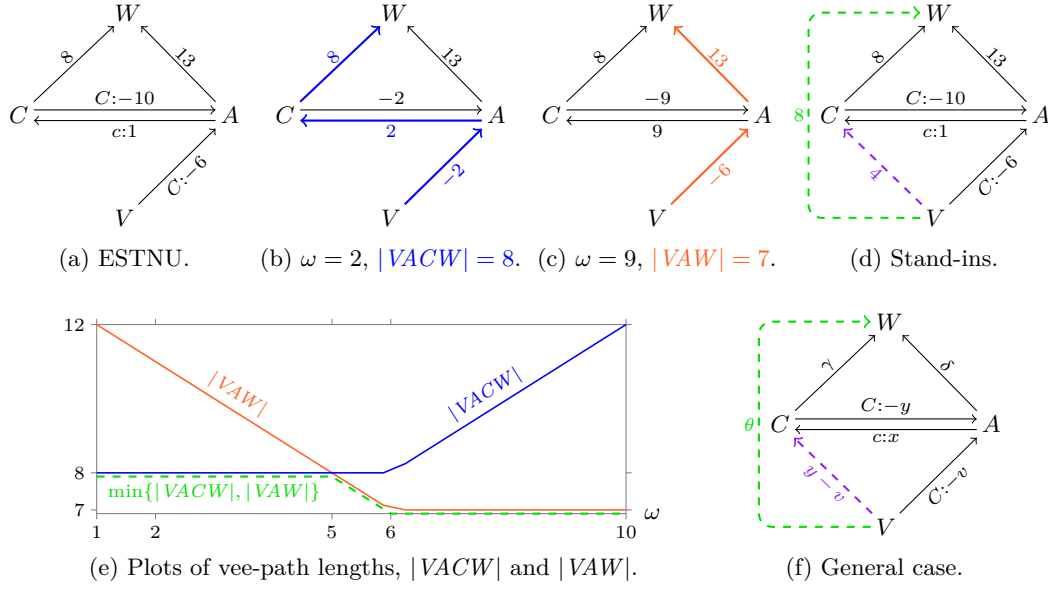
2.1 Generating Stand-in Edges

Following Morris [11, 12], an ESTNU is dispatchable if all of its STN projections are dispatchable (as STNs). That, in turn, requires that in each STN projection, each pair of timepoints V and W that are connected by a path be connected by a *shortest vee-path* (i.e., a path comprising zero or more negative edges followed by zero or more non-negative edges). A key insight behind the $\text{minDisp}_{\text{ESTNU}}$ algorithm is that in different projections, the shortest vee-paths from V to W may take different routes and may have different lengths.

Before addressing more complex cases, genStandIns generates stand-in edges entailed by individual labeled edges. For example, given a contingent link (A, x, y, C) , the LC edge $(A, c:x, C)$ entails a stand-in edge (A, y, C) because in any projection where $\omega = C - A \in [x, y]$, the LC edge projects onto the ordinary edge (A, ω, C) , whose length is $\omega \leq y$. Similarly, the UC edge $(C, C:-y, A)$ entails a stand-in edge $(C, -x, A)$ since in any projection the UC edge projects onto the ordinary edge $(C, -\omega, A)$, whose length is $-\omega \leq -x$. Finally, a wait edge $(V, C:-v, A)$, where $-v < -x$, projects onto the ordinary edge $(V, \max\{-\omega, -v\}, A)$ and hence entails a stand-in edge $(V, -x, A)$, since $-\omega \leq -x$ and $-v < -x$.² Figure 1 shows an example of the stand-in edges entailed by individual labeled edges.

The most computationally costly part of the genStandIns algorithm is its computation of stand-in edges entailed by different combinations of ESTNU edges. For example, consider the ESTNU in Figure 2a, commonly referred to as a *diamond structure*. In the projection where $\omega = C - A = 2$, the projected path $VACW$, shown in blue in Figure 2b, is the shortest vee-path from V to W : its length is 8. But in the projection where $\omega = C - A = 9$, the projected path VAW , shown in orange in Figure 2c, is the shortest vee-path from V to W : its length is 7. The plots of the lengths, $|VACW|$ and $|VAW|$, in Figure 2e, show that across *all* projections the *maximum* length of the *shortest* vee-path from V to W , indicated by the dashed green line, is 8. In other words, the combination of edges in the diamond structure entails the stand-in edge $(V, 8, W)$, shown as dashed and green in Figure 2d. Since the constraint, $W - V \leq 8$, must be satisfied in all projections, it must also be satisfied by any dynamic execution strategy for the ESTNU. Similarly, the path VAC satisfies $|VAC| = \max\{-\omega, -6\} + \omega = \max\{0, \omega - 6\} \leq 4$, for all $\omega \in [1, 10]$. Thus, that path entails the stand-in edge $(V, 4, C)$, shown as dashed and purple in Figure 2d. Like all stand-in edges, it must be satisfied by any dynamic execution strategy. The purpose of

² As a first step, genStandIns replaces *weak* waits (i.e., those where $-v \geq -x$) by ordinary edges and adjusts *misleading* waits (i.e., those where $-v < -y$). But those details are not important for this paper.



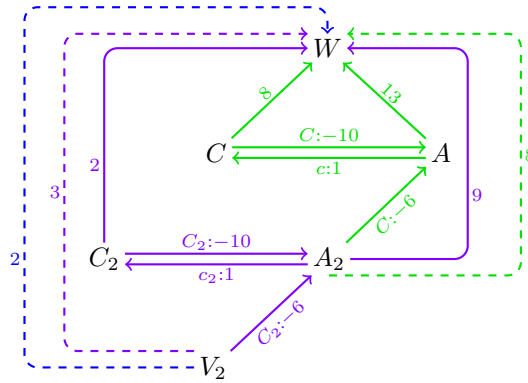
■ **Figure 2** (a) Sample ESTNU, (b) and (c) two of its projections with (colored) shortest *vee*-paths, (d) entailed (dashed) stand-in edges, (e) plots of vee-path lengths, and (f) the general case.

the `genStandIns` helper algorithm is to make all such constraints temporarily explicit so that Step 2 of `minDispESTNU` can determine which ordinary edges can be removed without threatening the dispatchability of the ESTNU.

Each iteration of the `genStandIns` algorithm's main loop explores $O(n^2k)$ diamond structures (n choices for V , n choices for W , and k choices for the contingent link), as illustrated in Figure 2f, where the distances δ and γ are provided by the all-pairs shortest-paths (APSP) matrix for the ordinary edges in the ESTNU. (The APSP matrix for the ordinary edges is commonly called the *distance matrix*, denoted by \mathcal{D} .) The lengths of the alternative vee-paths, $VACW$ and VAW , are given by $|VACW| = \max\{-\omega, -v\} + \omega + \gamma$ and $|VAW| = \max\{-\omega, -v\} + \delta$. Their intersection occurs where $\omega = \delta - \gamma$. If that value falls within the interval (x, y) , it is not hard to show that the maximum length of any shortest vee-path from V to W across *all* projections is $\theta = \max\{\gamma, \delta - v\}$, represented by the stand-in edge (V, θ, W) , shown as dashed in Figure 2f. The other stand-in edge $(V, y - v, C)$ derives from the two-edge path, VAC , whose length in the projection where $\omega = C - A$ is given by: $|VAC| = \max\{-\omega, -v\} + \omega = \max\{0, \omega - v\} \leq y - v$. After exploring all such diamond structures, Johnson's algorithm [2] is called to update the APSP matrix.

2.2 Stand-in edges arising from nested diamond structures

Because the distances involved in the analysis of diamond structures depend on shortest paths in the subgraph of ordinary edges (e.g., $\gamma = \mathcal{D}(C, W)$ and $\delta = \mathcal{D}(A, W)$ in Figure 2f), which can be affected by inserting (ordinary) stand-in edges into the ESTNU, it follows that stand-in edges can derive from *nested* diamond structures, for example, as illustrated in Figure 3. That figure shows a more complicated ESTNU, where the diamond structure involving the solid green edges is nested inside the diamond structure involving the solid purple edges. Ignoring the green edges, for now, the solid purple edges can be shown to entail the (purple, dashed) stand-in edge $(V_2, 3, W)$. In particular, in projections where $\omega_2 = C_2 - A_2 \leq 7$, the length of the path $V_2A_2C_2W$ is: $\max\{-\omega_2, -6\} + \omega_2 + 2 = \max\{2, \omega_2 - 4\} \leq 3$. In contrast, if $\omega_2 \geq 7$, the length of the alternative path V_2A_2W is: $\max\{-\omega_2, -6\} + 9 = \max\{9 - \omega_2, 3\} \leq 3$.



■ **Figure 3** Deriving stand-in edges from nested diamond structures.

Next, since the green diamond is isomorphic to the diamond from Figure 2d, it entails the (green, dashed) stand-in edge $(A_2, 8, W)$. But now, using that stand-in edge instead of the purple edge $(A_2, 9, W)$, a new analysis of the purple structure shows that it entails a stronger (blue, dashed) stand-in edge $(V_2, 2, W)$. In other words, nested diamond structures can sometimes combine to entail stronger stand-in edges.

Hunsberger and Posenato [8] proved that it suffices to explore nested diamond structures up to a maximum depth of k . Thus, the `genStandIns` algorithm does up to k iterations of its main loop. Since each iteration ends by calling Johnson’s algorithm on up to n^2 edges, the overall complexity of `genStandIns` is $O(kn^3)$.

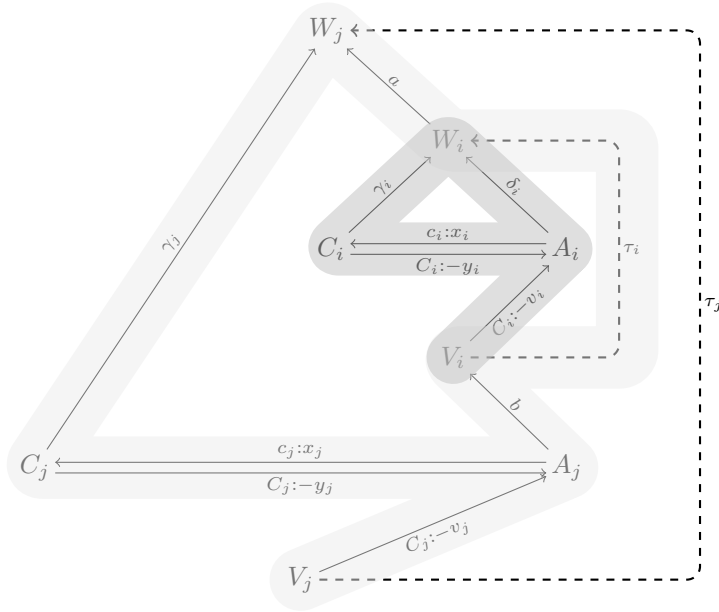
3 Speeding up the `minDispESTNU` Algorithm

The complexity of the `minDispESTNU` algorithm is driven by the $O(kn^3)$ -time complexity of `genStandIns`. Our modification of `minDispESTNU` replaces `genStandIns` with `newGenStandIns`, which, taking a more focused and efficient approach to dealing with nested diamond structures, works in $O(n^3)$ time. Since $k = O(n)$ is common in applications (e.g., $k \approx n/10$ in some benchmarks [15]), the reduction in worst-case time-complexity is effectively from $O(n^4)$ to $O(n^3)$.

3.1 Stand-in Edges Derived from Nested Diamond Structures

Figure 4 illustrates the nested relationship between an inner diamond D_i (involving timepoints V_i, A_i, C_i and W_i , shaded dark gray) and an outer diamond D_j (involving timepoints V_j, A_j, C_j and W_j , shaded light gray), where the arrows labeled by a, b, δ_i, γ_i and γ_j represent ordinary edges or paths, and the dashed arrows represent the stand-in edges (V_i, τ_i, W_i) and (V_j, τ_j, W_j) entailed by the diamonds.³ Lemma 1, below, ensures that in any such nesting, there must be a path from A_j to A_i that comprises zero or more negative ordinary edges followed by one (negative) wait edge, which for convenience we call a *negOrdWait* path. This implies that the activation timepoints involved in nested structures can be put into a strict partial order which, in turn, implies that generating the stand-in zedges associated

³ Hunsberger and Posenato [8] proved that when considering vee-paths from A_j to W_j , the only relevant nesting of diamonds occurs if the inner diamond D_i resides along the path from A_j to W_j in the outer diamond D_j , as shown in the figure. Since the inner diamond begins with a negative wait edge, any path from A_j to W_j that included D_i between C_j and W_j could not be a vee-path.



■ **Figure 4** Nested diamond structures (one shaded light, one shaded dark) considered in Lemma 1.

with nested diamonds can be done in just one pass, instead of the k passes through the main loop of `genStandIns`. Furthermore, to determine the length of the stand-in edge from V_j to any W_j , taking advantage of the nesting of D_i within D_j , it suffices to know the length of the shortest ordinary path from A_j to W_j . (Recall that the length of the entailed stand-in edge depends only on the values of $\mathcal{D}(C_j, W_j)$, $\mathcal{D}(A_j, W_j)$ and $-v_j$.) In other words, when generating stand-in edges derived from diamonds involving the labeled edges $(A_i, c_i:x_i, C_i)$ and $(C_i, C_i:-y_i, A_i)$, it is not necessary to find all ordinary distances affected by those stand-in edges (which is what the existing `genStandIns` algorithm uses Johnson's algorithm to do – *on each of up to k passes*); instead, it suffices to focus on the distances of ordinary paths emanating from A_j that are affected by those stand-in edges. In the case of A_j shown in the figure, it suffices to record distances of the form, $\mathcal{D}(A_j, W_i) = b + \tau_i$, resulting from new stand-in edges. Crucially, all of these distances correspond to paths emanating from a single source, A_j . After exploring all inner diamonds D_i and recording the new distances, $\mathcal{D}(A_j, W_i)$, then *all* values $\mathcal{D}(A_j, \cdot)$ can be updated using Dijkstra's single-source shortest-paths algorithm, guided by a potential function [2]. These observations enable the `newGenStandIns` algorithm, presented later in this section, to call Dijkstra's algorithm k times, instead of calling Johnson's algorithm k times, leading to an order-of-magnitude reduction in worst-case time complexity, from $O(kn^3)$ down to $O(n^3)$.

► **Lemma 1.** *Let \mathcal{S} be any dispatchable ESTNU. Suppose that E is a stand-in edge derived from nested diamond structures in which the diamond structure D_i associated with the contingent link (A_i, x_i, y_i, C_i) is nested directly inside the diamond structure D_j associated with the contingent link (A_j, x_j, y_j, C_j) . Furthermore, suppose that the labeled edges from these contingent links are needed for E (i.e., without their labeled edges, E would not be entailed by the remaining edges in \mathcal{S}). Then there must be a path from A_j to A_i in \mathcal{S} that consists of zero or more negative ordinary edges, followed by a single wait edge of the form $(V_i, C_i:-v_i, A_i)$ (i.e., a `negOrdWait` path).*

Proof. Suppose that E is the stand-in edge (V_j, τ_j, W_j) . Since the labeled edges from these contingent links are needed for E , it follows that in at least one STN projection, the shortest vee-path from V_j to W_j must include the path from A_j to V_i to A_i . Since any subpath of a vee-path is also a vee-path and the wait edge $(V_i, C_i: -v_i, A_i)$ has negative length, it follows that all of the ordinary edges represented in the figure by (A_j, b, V_i) must be negative. ◀

Given Lemma 1, the activation timepoints participating in a nested diamond structure must be linked by a chain of *negOrdWait* paths. In addition, for a DC STNU, there can be no cycles of such paths because they would constitute a negative cycle in the OU-graph, i.e., $\mathcal{G}_{ou} = (\mathcal{T}, \mathcal{E}_o \cup \mathcal{E}_{uc} \cup \mathcal{E}_{ucg})$, the graph containing all the original and derived edges but the lower-case ones. However, a single activation timepoint may participate in multiple nested structures. Hence, the set of all *negOrdWait* paths among the activation timepoints necessarily forms a strict partial order (equivalently, a forest of one or more directed acyclic graphs in the OU-graph).

For each pair of activation timepoints, A_j and A_i , for which there is a *negOrdWait* path from A_j to A_i , we say that A_j is a *parent* of A_i and that A_i is a *child* of A_j . The relevant information for determining the stand-in edges emanating from A_j and passing through a diamond structure involving labeled edges from (A_i, x_i, y_i, C_i) is: (1) ℓ , the (negative) length of the *negOrdWait* path from A_j to A_i ; and (2) $-v_i$, the (negative) length of the wait edge, $(V_i, C_i: -v_i, A_i)$, terminating that *negOrdWait* path. These lengths are shown in Figure 4, where $b = \ell + v_i$ is the length of the prefix of the *negOrdWait* path that includes only the ordinary edges (i.e., everything except the terminal wait edge). Then, as shown by Hunsberger and Posenato [8], for any timepoint $W_i \in \mathcal{T} \setminus \{A_i, C_i, A_j, C_j\}$, the length of the potential stand-in edge from A_j to W_i is given by $b + \max\{\gamma_i, \delta_i - v_i\} = \max\{b + \gamma_i, \ell + \delta_i\}$, where $\gamma_i = \mathcal{D}(C_i, W_i)$ and $\delta_i = \mathcal{D}(A_i, W_i)$, also shown in the figure. Then, for any W_j , the ordinary distance $\mathcal{D}(A_j, W_j)$ affected by such a stand-in edge can be determined by the previously mentioned call to Dijkstra's algorithm, guided by a potential function.

3.2 The getPCinfo (get parent/child info) Algorithm

The `getPCinfo` algorithm (Algorithm 1) efficiently computes the relevant parent/child information, returning a pair of vectors of hash tables, called *parent* and *child*. For each activation timepoint A_i , `parent`[A_i] is a hash table containing entries where some A_j is the key and $(\ell, -v_i)$ is the value (i.e., A_j is the parent, ℓ is the length of the *negOrdWait* path from A_j to A_i , and $-v_i$ is the length of its terminating wait edge). Similarly, for each activation timepoint A_j , `child`[A_j] is a hash table containing entries linking some child A_i to the corresponding pair $(\ell, -v_i)$, where ℓ is the length of the *negOrdWait* path from A_j to A_i , and $-v_i$ is the length of its terminal wait edge.

An important factor is that if two *negOrdWait* paths from A_j to A_i have the same length, but one has a stronger (i.e., more negative) terminating wait edge, then the *negOrdWait* path terminated by the *weaker* wait *dominates* the one with the stronger wait because in any projection the projected length of the one with the weaker wait will be shorter than (or the same as) that of the one with the stronger wait. For example, if ℓ is the length of two *negOrdWait* paths from A_j to A_i , but $-v_1 > -v_2$, where the corresponding terminal wait edges are $(V_1, C_i: -v_1, A_i)$ and $(V_2, C_i: -v_2, A_i)$, then $|A_j V_1 A_i| = (\ell + v_1) + \max\{-\omega, -v_1\} = \max\{\ell + v_1 - \omega, \ell\} \leq \max\{\ell + v_2 - \omega, \ell\} = (\ell + v_2) + \max\{-\omega, -v_2\} = |A_j V_2 A_i|$. Another important factor involves *negOrd* paths (i.e., paths comprising solely negative ordinary edges). If a *negOrd* path has the same length as a *negOrdWait* path, then the *negOrd* path dominates the *negOrdWait* path since in every projection the length of the *negOrd* path will be the same as or shorter than the length of the projected *negOrdWait* path.

■ **Algorithm 1** `getPCinfo`: find *negOrdWait* paths between pairs of activation timepoints.

```

Input:  $\mathcal{G} = (\mathcal{T}, \mathcal{E}_o, \mathcal{E}_{lc}, \mathcal{E}_{uc}, \mathcal{E}_{ucg})$ , an ESTNU graph
Output:  $(parent, child)$ , where parent and child are  $k$ -vectors of hash tables signaling the
presence of negOrdWait paths between pairs of activation timepoints
1  $f := \text{bellmanFord}(\mathcal{G}_{ou})$  // A potential function for  $\mathcal{G}_{ou} = (\mathcal{T}, \mathcal{E}_o \cup \mathcal{E}_{uc} \cup \mathcal{E}_{ucg})$ 
2  $parent := (\emptyset, \dots, \emptyset)$ 
3  $child := (\emptyset, \dots, \emptyset)$  //  $k$ -vectors of hash tables
4 foreach  $(A, x, y, C) \in \mathcal{L}$  do // Back-propagate from  $A$  along negOrdWait paths
5    $negLen := (\infty, \dots, \infty)$  // An  $n$ -vector of accum. lengths of negOrdWait paths ending in  $A$ 
6    $negWait := (\perp, \dots, \perp)$  // An  $n$ -vector of corresp. neg. wait values (or  $\perp$  for ord paths)
// Initialize min priority queue  $\mathcal{Q}$  with entries for negative ord and wait edges incoming to  $A$ 
// Element =  $U$ , a timepoint
// Key = Non-negative accumulated length adjusted by potential function,  $f$ 
7    $\mathcal{Q} :=$  new priority queue
8   foreach  $(U, \delta, A)$  with  $\delta < 0$  do // Negative ordinary edges incoming to  $A$ 
9      $\mathcal{Q}.insert(U, \delta - f(A) + f(U))$  //  $f(A) - f(U) \leq \delta \iff \delta - f(A) + f(U) \geq 0$ 
10     $negLen[U] := \delta$ 
11   foreach  $(V, C:-v, A) \in \mathcal{E}_{ucg}$  do // (Negative) wait edges incoming to  $A$ 
12      $\mathcal{Q}.insert(V, -v - f(A) + f(V))$  //  $f(A) - f(V) \leq -v \iff -v - f(A) + f(V) \geq 0$ 
13      $negLen[V] := -v$ ;  $negWait[V] := -v$ 
// Use back-propagation to find shortest negOrd or negOrdWait paths terminating at  $A$ 
14   while  $\neg \mathcal{Q}.empty()$  do
15      $U := \mathcal{Q}.extractMin()$ 
16     if  $U = A'$  is an activation timepoint and  $negWait[A'] \neq \perp$  then
17       // Record negOrdWait path found from  $A'$  to  $A$ 
18        $parent[A].insert(A', (negLen[A'], negWait[A']))$ 
19        $child[A'].insert(A, (negLen[A'], negWait[A']))$ 
// Continue back-propagating along negative ordinary edges
19     foreach  $(V, v, U) \in \mathcal{E}_o \mid v < 0$  do
20        $newLen := v + negLen[U]$ 
21       if  $newLen < negLen[V]$  or  $((newLen == negLen[V])$  and
22          $((negWait[U] == \perp)$  or  $(negWait[U] > negWait[V])))$  then
23         // Record new shortest negOrd or negOrdWait path from  $V$  to  $A$  (via  $U$ )
24         if  $negLen[V] == \infty$  then  $\mathcal{Q}.insert(V, newLen - f(A) + f(V))$ 
25         else  $\mathcal{Q}.decreaseKey(V, newLen - f(A) + f(V))$ 
26          $negLen[V] := newLen$ 
27          $negWait[V] := negWait[U]$ 
26 return  $(parent, child)$  // Return the vectors of parent/child hash tables

```

At Line 1, `getPCinfo` calls the Bellman-Ford algorithm [2] to generate a solution to the OU-graph that will be used as a potential function to guide the traversal of *negOrd* and *negOrdWait* paths. Line 2 initializes the *parent* and *child* vectors of hash tables.

Each iteration of the for loop at Lines 4–25 processes one activation timepoint A , looking for shortest *negOrd* or *negOrdWait* paths from A backward to other activation timepoints. Lines 5–6 initialize the *negLen* and *negWait* vectors. For each X , $negLen[X]$ specifies the length of the shortest *negOrd* or *negOrdWait* path from X to A that has been found so far (or ∞). If a shortest *negOrdWait* path from X to A has been found that is not dominated by a *negOrd* path, then $negWait[X]$ specifies the length of its terminating wait edge.

Lines 7–13 initialize a min priority queue [2] to include an entry for each negative ordinary edge and each wait edge incoming to A . Like in Johnson’s algorithm, the potential function f is used to adjust the distances in the OU-graph to be non-negative to enable the use of Dijkstra’s algorithm to guide the exploration of *negOrd* and *negOrdWait* paths.

Each iteration of the **while** loop (Lines 14–25) pops a timepoint U off the queue. If U happens to be an activation timepoint A' for which an undominated *negOrdWait* path has been found, then entries linking A (the child) to A' (the parent) are inserted into the relevant hash tables (Lines 16–18). Next, back-propagation along negative ordinary edges continues at Lines 19–25. The complicated **if** condition at Line 21 covers cases where a new shortest *negOrd* or *negOrdWait* path from V to A (via U) has been found. First, if $newLen < negLen[V]$ (which includes $negLen[V] = \infty$), then the path via U is a new shortest path. Second, if $newLen = negLen[V]$, then the path via U dominates a pre-existing path from V to A if: (1) the path via U is a *negOrd* path (whence $negWait[U] = \perp$); or (2) the wait terminating the path via U is weaker than the terminal wait in the pre-existing path (i.e., $negWait[U] > negWait[V]$). In any of these cases, the values of $negLen[V]$ and $negWait[V]$ are updated, and V is either newly inserted into the queue or its key is updated (Lines 22–25). After the main **for** loop is completed, the *parent* and *child* vectors of hash tables are returned at Line 26.

3.3 The newGenStandIns Algorithm

The section presents our **newGenStandIns** algorithm (Algorithm 2). It uses the *parent* and *child* hash tables computed by **getPCinfo** to more efficiently generate all of the stand-in edges arising from nested diamond structures. Its time-complexity is $O(n^3)$, an order-of-magnitude improvement over the $O(kn^3)$ -time complexity of **genStandIns**.

For simplicity, we assume that all stand-in edges entailed by *individual* labeled edges have already been computed and have been passed as an input \mathcal{E}_{isi} into **newGenStandIns**.

At Line 1, **newGenStandIns** calls the Bellman-Ford algorithm on the subgraph of ordinary edges which will be used as a potential function to enable the use of Dijkstra’s single-source shortest-paths algorithm to update distance-matrix entries. At Line 2, \mathcal{E}_o^t is initialized; it will accumulate changes to $\mathcal{D}(A_j, \cdot)$ values, stored as *temporary edges*, that are derived directly from nested stand-in edges. Next, at Lines 3–7, the list, *readyToGo*, of activation timepoints that are ready to process is initialized. Since the activation timepoints form a strict partial order, this list is initially populated by those having no children. The vector, *numUnprocd*, keeps track of how many unprocessed children each activation timepoint has. Later on, as each activation timepoint is processed, its parent’s entry in *numUnprocd* will be decremented.

Each iteration of the **while** loop (Lines 8–28) pops one activation timepoint A_j off the *readyToGo* list and, at Lines 12–19, for each child A_i and each timepoint W_i , explores diamond structures involving the labeled edges from the contingent link (A_i, x_i, y_i, C_i) , to determine whether the distance $\mathcal{D}(A_j, W_i)$ can be affected by a nested diamond. (Recall Figure 3.) Instead of explicitly dealing with the wait edge $(V_i, C_i: -v_i, A_i)$ shown in the figure, **newGenStandIns** uses the ℓ_i and $-v_i$ values retrieved from the *child*[A_j] hash table at Line 12 (where b in the figure equals $\ell_i + v_i$), along with the distances, $\gamma_i = \mathcal{D}(C_i, W)$ and $\delta_i = \mathcal{D}(A_i, W_i)$, obtained from the distance matrix at Line 14. This information is sufficient to determine whether the paths $V_i A_i C_i W_i$ and $V_i A_i W_i$ combine to entail a new stand-in edge, (V_i, τ_i, W_i) , where $\tau_i = \max\{\gamma_i, \delta_i - v_i\}$. In particular, as in **genStandIns**, $\omega_i = \delta_i - \gamma_i$ (at Line 15) specifies the projection where $|V_i A_i C_i W_i| = |V_i A_i W_i|$; and a new stand-in edge from V_i to W_i is entailed if $\omega_i \in (x_i, y_i)$ and if that new stand-in edge is at least as strong as any existing ordinary path from V_i to W_i . However, here, the goal is not to generate

■ **Algorithm 2** `newGenStandIns`: Compute the stand-in edges arising from nested diamonds.

Input: $(\mathcal{T}, \mathcal{E}_o, \mathcal{E}_{lc}, \mathcal{E}_{uc}, \mathcal{E}_{ucg})$, dispatchable ESTNU; *parent*, *child*, vectors of hash tables computed by `getPCInfo`; \mathcal{D} , distance matrix for $\mathcal{G}_o = (\mathcal{T}, \mathcal{E}_o)$; $\mathcal{E}_{isi} \subseteq \mathcal{E}_o$, stand-in edges entailed by *individual* labeled edges

Output: \mathcal{E}_{si} , the set of *all* stand-in edges (including \mathcal{E}_{isi}); and \mathcal{D} , the updated distance matrix.

```

1  $f := \text{bellmanFord}(\mathcal{G}_o)$  // Initialize a potential function  $f$  on the ordinary subgraph  $\mathcal{G}_o$ 
2  $\mathcal{E}_o^t := \emptyset$  // Used to collect all temporary (ordinary) edges
3  $\text{readyToGo} := \emptyset$  // A list of activation timepoints ready for processing
4  $\text{numUnprocd} := (0, \dots, 0)$  // For each activ'n. timepoint, the num of its unprocessed children
5 foreach  $(A, x, y, C) \in \mathcal{L}$  do
6    $\text{numUnprocd}[A] := \text{child}[A].\text{count}()$  // Fetch the number of  $A$ 's children
7   if  $\text{numUnprocd}[A] == 0$  then  $\text{readyToGo}.\text{push}(A)$  // If no children, then ready to process
8 while  $\text{readyToGo} \neq \emptyset$  do
9    $A_j := \text{readyToGo}.\text{pop}()$  // Contingent link for  $A_j$  is  $(A_j, x_j, y_j, C_j)$ 
10   $\text{anyChange} := \perp$ 
11   $\text{newLengths} := \text{empty hash table}$  // For collecting new  $\mathcal{D}(A_j, \cdot)$  values
12  foreach  $(A_i, (\ell_i, -v_i)) \in \text{child}[A_j]$  do // Contingent link for  $A_i$  is  $(A_i, x_i, y_i, C_i)$ 
13    foreach  $W_i \in \mathcal{T} \setminus \{A_i, C_i, A_j, C_j\}$  do
14       $\gamma_i = \mathcal{D}(C_i, W_i)$ ;  $\delta_i = \mathcal{D}(A_i, W_i)$ ;  $\omega_i := \delta_i - \gamma_i$ 
15      if  $\omega_i \in (x_i, y_i)$  then //  $\omega_i$  specifies proj'n. where max shortest vee-path occurs
16         $\text{newVal} := \max\{\ell_i + v_i + \gamma_i, \ell_i + \delta_i\}$  // Length of potential new  $\mathcal{D}(A_j, W_i)$  value
17        if  $\text{newVal} < \mathcal{D}(A_j, W_i)$  then
18           $\text{newLengths}.\text{insert}(W_i, \text{newVal})$  // Record new  $\mathcal{D}(A_j, W_i)$  value
19           $\text{anyChange} := \top$ 
20  if  $\text{anyChange} == \top$  then // Need to update potential function and  $\mathcal{D}(A_j, \cdot)$  values
21     $\mathcal{E}_o^+ := \emptyset$  // Collect set of changed  $\mathcal{D}(A_j, \cdot)$  values as temporary edges
22    foreach  $(W_i, \text{newVal}) \in \text{newLengths}$  do  $\mathcal{E}_o^+ := \mathcal{E}_o^+ \cup \{(A_j, \text{newVal}, W_i)\}$ 
23     $f := \text{updatePotFn}(\mathcal{T}, \mathcal{E}_o \cup \mathcal{E}_o^+, f)$  // Update pot'l. fn. to accommodate temp edges
24     $\mathcal{D}(A_j, \cdot) := \text{dijkstra}(A_j, \mathcal{E}_o \cup \mathcal{E}_o^+, f)$  // Update  $\mathcal{D}(A_j, \cdot)$  values for next iteration
25     $\mathcal{E}_o^t := \mathcal{E}_o^t \cup \mathcal{E}_o^+$  // Accumulate temp edges RE:  $A_j$  in global set  $\mathcal{E}_o^t$ 
26  foreach  $A \in \text{parent}[A_j]$  do // Update info for  $A_j$ 's parents now that  $A_j$  is done
27     $\text{numUnprocd}[A] := \text{numUnprocd}[A] - 1$ 
28    if  $\text{numUnprocd}[A] == 0$  then  $\text{readyToGo}.\text{push}(A)$ 

```

// Fully updated \mathcal{D} ensures that *one* iteration of `genStandIns` will generate *all* stand-in edges

```

29  $\mathcal{D} := \text{johnson}(\mathcal{T}, \mathcal{E}_o \cup \mathcal{E}_o^t)$  // After this, temp edges are discarded
30  $\mathcal{E}_{si} := \text{genStandInsOnce}(\mathcal{T}, \mathcal{E}_o, \mathcal{E}_{lc}, \mathcal{E}_{uc}, \mathcal{E}_{ucg}, \mathcal{E}_{isi}, \mathcal{D})$ 
31  $\mathcal{D} := \text{johnson}(\mathcal{T}, \mathcal{E}_o \cup \mathcal{E}_{si})$  // Final update of  $\mathcal{D}$  to accommodate the generated stand-in edges
32 return  $(\mathcal{E}_{si}, \mathcal{D})$ 

```

that stand-in edge, but instead to provide the $\mathcal{D}(A_j, W_i)$ value affected by it. Therefore, the only information accumulated in the *newLengths* hash table is the pair (W_i, newVal) , where $\text{newVal} = b + \tau_i = \ell_i + v_i + \tau_i = \max\{\ell_i + v_i + \gamma_i, \ell_i + \delta_i\}$ (at Lines 16–18).

Afterward, at Line 20, if processing A_j led to changes in any $\mathcal{D}(A_j, \cdot)$ values, then `newGenStandIns` collects all of the changes as a set \mathcal{E}_o^+ of *temporary* edges (Lines 21–22) that it then uses to (1) incrementally update the potential function f (at Line 23), and (2) propagate the new $\mathcal{D}(A_j, \cdot)$ values to update *all* affected $\mathcal{D}(A_j, \cdot)$ values (at Line 24). For updating the potential function, it calls the `updatePotFn`, which is a simplified version of the `UpdPF` algorithm from the `RUL2021` algorithm [6]; here, it explores paths emanating from A_j as long as changes to the potential function are needed. For updating $\mathcal{D}(A_j, \cdot)$ values, it calls

■ **Algorithm 3** The `updatePotFn` function.

Input: $\mathcal{G}_o = (\mathcal{T}, \mathcal{E}_o)$, STN; A , timepoint; h , pot'l. fn. for \mathcal{G}_o , excluding edges emanating from A
Output: A pot'l. fn. h' for \mathcal{G}_o (including edges emanating from A); or \perp if \mathcal{G}_o is inconsistent

```

1  $h' := \text{copy-vector}(h)$ 
2  $\mathcal{Q} := \text{new empty priority queue}$ 
3  $\mathcal{Q}.\text{insert}(A, 0)$  // Initialize queue for forward propagation from  $A$ 
4 while ( $!\mathcal{Q}.\text{empty}()$ ) do
5    $(V, \text{key}(V)) := \mathcal{Q}.\text{extractMinNode}()$ 
6   foreach  $((V, \delta, W) \in \mathcal{E}_o)$  do // Propagate along ordinary edges emanating from  $V$ 
7     if  $(h'(W) > h'(V) + \delta)$  then
8        $h'(W) := h'(V) + \delta$  // Update pot'l. fn.  $h'$  and insert  $W$  into  $\mathcal{Q}$  or decrease its key
9       if  $(\mathcal{Q}.\text{state}(W) == \text{notYetInQ})$  then  $\mathcal{Q}.\text{insert}(W, h(W) - h'(W))$ 
10      else  $\mathcal{Q}.\text{decreaseKey}(W, h(W) - h'(W))$ 
11 return  $h'$ 

```

Dijkstra's single-source shortest-paths algorithm using A_j as the source and f as a potential function to re-weight the edges to non-negative values. This use of Dijkstra is similar to its use in Johnson's algorithm [2]. Note that after these updates the temporary edges in \mathcal{E}_o^+ are *not* inserted into the ESTNU graph, but they are accumulated in \mathcal{E}_o^t for later use at Line 25.

The processing of A_j ends at Lines 26–28, where for each parent A of A_j , the number of A 's unprocessed children is decremented by 1 and, if that number reaches 0, then A is pushed onto the *readyToGo* list, indicating that it is ready for processing.

Once all activation timepoints have been processed, all distance values $\mathcal{D}(A_j, \cdot)$ needed to account for arbitrary nestings of diamond structures have been accumulated. All that remains is to *use* these values to generate all of the stand-in edges. For example, suppose that the diamond formed by V_j, A_j, C_j and W_j from Figure 3 is the *outermost* diamond in a nested sequence that entails a stand-in edge of the form, (V_j, τ_j, W_j) . Then the resulting $\mathcal{D}(A_j, W_j)$ value, determined by the inner levels of nesting, was computed when A_j was processed by the **while** loop at Lines 8–19. But the stand-in edge (V_j, τ_j, W_j) has not yet been generated. However, given all of the $\mathcal{D}(A_j, \cdot)$ values computed so far (for all A_j), generating *all* such stand-in edges, including those that are *not* involved in any nesting, can be accomplished by an $O(kn^2)$ -time exploration of diamond structures involving any timepoints, V, A, C, W , where A and C are timepoints associated with a contingent link (A, x, y, C) , and V and W are any timepoints other than A or C . This is precisely what a *single iteration* of the **for** loop at Lines 13–27 of `genStandIns` does. Here, it is called `genStandInsOnce`, at Line 30. Afterward, at Line 31, a final call to Johnson's algorithm computes the full distance matrix to accommodate all of the new stand-in edges, including those in \mathcal{E}_{isi} passed in as an input.

3.4 Complexity of `newGenStandIns`

Our modification of the `minDispESTNU` algorithm replaces the `genStandIns` helper by the `newGenStandIns` algorithm presented above. The complexity of `newGenStandIns` is determined as follows. Its k calls of Dijkstra's algorithm on at most $m + nk$ edges cost $O(mk + nk^2 + kn \log n)$ time. Its k calls of the `updatePotFn` function similarly require $O(mk + nk^2 + kn \log n)$ time. The call to `genStandInsOnce`, as reported by Hunsberger and Posenato [8], requires $O(kn^2)$ time (n choices for V , n choices for W , and k choices for (A, x, y, C)). The most costly computation, however, is the last one: the call to Johnson's algorithm on at most $m = n^2$ edges costs $O(n^3)$ time. Therefore, the overall complexity of

`newGenStandIns` is $O(n^3)$. This is an order-of-magnitude reduction compared to the $O(kn^3)$ complexity of `genStandIns`, especially since, for applications, $k = O(n)$ (e.g., $k \approx n/10$ in some benchmarks [15]), implying an effective reduction from $O(n^4)$ to $O(n^3)$.

The complexity of steps 2, 3 and 4 of `minDispESTNU`, which we do not change, is dominated by the call to the STN-dispatchability algorithm on at most n^2 edges, which is also $O(n^3)$. So the overall complexity of our modification of `minDispESTNU` is $O(n^3)$.

4 Conclusions

Generating an equivalent dispatchable ESTNU having a minimal number of edges is an important problem for applications involving actions with uncertain but bounded durations. The number of edges in the dispatchable network is important because it directly impacts the real-time computations that are necessary when executing the network. Therefore, for time-sensitive applications it is important to generate an equivalent dispatchable ESTNU having a minimal number of edges, called a μ ESTNU. This paper modified the only existing algorithm for generating a μ ESTNU, making it an order-of-magnitude faster. It reduced the worst-case time-complexity from $O(kn^3)$ to $O(n^3)$ which, given that in typical applications $k = O(n)$, implies an effective reduction from $O(n^4)$ to $O(n^3)$.

References

- 1 Massimo Cairo, Luke Hunsberger, and Romeo Rizzi. Faster Dynamic Controllability Checking for Simple Temporal Networks with Uncertainty. In *25th International Symposium on Temporal Representation and Reasoning (TIME-2018)*, volume 120 of *LIPICs*, pages 8:1–8:16, 2018. doi:10.4230/LIPICs.TIME.2018.8.
- 2 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 4th Edition*. MIT Press, 2022. URL: <https://mitpress.mit.edu/9780262046305/introduction-to-algorithms>.
- 3 Rina Dechter, Itay Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. doi:10.1016/0004-3702(91)90006-6.
- 4 Luke Hunsberger. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *16th International Symposium on Temporal Representation and Reasoning (TIME-2009)*, pages 155–162, 2009. doi:10.1109/TIME.2009.25.
- 5 Luke Hunsberger. Efficient execution of dynamically controllable simple temporal networks with uncertainty. *Acta Informatica*, 53(2):89–147, 2015. doi:10.1007/s00236-015-0227-0.
- 6 Luke Hunsberger and Roberto Posenato. Speeding up the RUL⁻ Dynamic-Controllability-Checking Algorithm for Simple Temporal Networks with Uncertainty. In *36th AAAI Conference on Artificial Intelligence (AAAI-22)*, volume 36-9, pages 9776–9785. AAAI Pres, 2022. doi:10.1609/aaai.v36i9.21213.
- 7 Luke Hunsberger and Roberto Posenato. A Faster Algorithm for Converting Simple Temporal Networks with Uncertainty into Dispatchable Form. *Information and Computation*, 293(105063):1–21, 2023. doi:10.1016/j.ic.2023.105063.
- 8 Luke Hunsberger and Roberto Posenato. Converting Simple Temporal Networks with Uncertainty into Minimal Equivalent Dispatchable Form. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, volume 34, pages 290–300, 2024. doi:10.1609/icaps.v34i1.31487.
- 9 Luke Hunsberger and Roberto Posenato. Foundations of Dispatchability for Simple Temporal Networks with Uncertainty. In *16th International Conference on Agents and Artificial Intelligence (ICAART 2024)*, volume 2, pages 253–263. SCITEPRESS, 2024. doi:10.5220/0012360000003636.

- 10 Paul Morris. A Structural Characterization of Temporal Dynamic Controllability. In *Principles and Practice of Constraint Programming (CP-2006)*, volume 4204, pages 375–389, 2006. doi:10.1007/11889205_28.
- 11 Paul Morris. Dynamic controllability and dispatchability relationships. In *Int. Conf. on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR-2014)*, volume 8451 of *LNCIS*, pages 464–479. Springer, 2014. doi:10.1007/978-3-319-07046-9_33.
- 12 Paul Morris. The Mathematics of Dispatchability Revisited. In *26th International Conference on Automated Planning and Scheduling (ICAPS-2016)*, pages 244–252, 2016. doi:10.1609/icaps.v26i1.13739.
- 13 Paul Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *17th Int. Joint Conf. on Artificial Intelligence (IJCAI-2001)*, volume 1, pages 494–499, 2001. URL: <https://www.ijcai.org/Proceedings/01/IJCAI-2001-e.pdf>.
- 14 Nicola Muscettola, Paul H. Morris, and Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning, KR'98*, pages 444–452, 1998.
- 15 Roberto Posenato. STNU Benchmark version 2020, 2020. Last access 2022-12-01. URL: <https://profs.scienze.univr.it/~posenato/software/cstnu/benchmarkWrapper.html>.
- 16 Ioannis Tsamardinos, Nicola Muscettola, and Paul Morris. Fast Transformation of Temporal Plans for Efficient Execution. In *15th National Conf. on Artificial Intelligence (AAAI-1998)*, pages 254–261, 1998. URL: <https://cdn.aaai.org/AAAI/1998/AAAI98-035.pdf>.

Robust Execution of Probabilistic STNs

Luke Hunsberger   

Vassar College, Poughkeepsie, NY, USA

Roberto Posenato   

University of Verona, Italy

Abstract

A Probabilistic Simple Temporal Network (PSTN) is a formalism for representing and reasoning about actions subject to temporal constraints, where some action durations may be uncontrollable, modeled using continuous probability density functions. Recent work aims to manage this kind of uncertainty during execution by approximating a PSTN by a Simple Temporal Network with Uncertainty (STNU) (for which well-known execution strategies exist) and using an STNU execution strategy to execute the PSTN, hoping that its probabilistic action durations will not cause any constraint violations.

This paper presents significant improvements to the robust execution of PSTNs. Our approach is based on a recent, faster algorithm for finding negative cycles in non-DC STNUs. We also formally prove that many of the constraints included in others' work are unnecessary and that our algorithm can take advantage of a flexible real-time execution algorithm to react to observations of contingent durations that may fall outside the fixed STNU bounds. The paper presents an empirical evaluation of our approach that provides evidence of its effectiveness in robustly executing PSTNs derived from a publicly available benchmark.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning; Theory of computation → Dynamic graph algorithms

Keywords and phrases Temporal constraint networks, probabilistic durations, dispatchable networks

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.12

Supplementary Material *Software (Source code)*: <https://profs.scienze.univr.it/~posenato/software/cstnu/> [26]

1 Introduction

In many sectors of real-world industry, it is necessary to plan and schedule tasks allocated to agents participating in complex processes [19, 1]. Temporal planning aims to schedule tasks while respecting temporal constraints such as release times, maximum durations, and deadlines, which requires quantitative temporal reasoning. Over the years, major application developers have highlighted the need for explicit representation of actions with uncertain durations; and efficient algorithms for checking whether plans involving such actions are controllable, and for converting such plans into forms that enable them to be executed in real time with minimal computation, while preserving maximum flexibility.

A Probabilistic Simple Temporal Network (PSTN) is a formalism for representing and reasoning about actions subject to temporal constraints, where some action durations may be uncontrollable, modeled using continuous probability density functions. Recent work aims to manage this kind of uncertainty during execution by:

1. computing a dynamically controllable (DC) Simple Temporal Network with Uncertainty (STNU) whose bounded action durations capture as much of the combined probability mass of the corresponding probabilistic durations as possible;
2. deriving a dynamic execution strategy for the approximating STNU; and
3. using that strategy to execute the PSTN, hoping that its probabilistic action durations will not cause any constraint violations.



© Luke Hunsberger and Roberto Posenato;
licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 12; pp. 12:1–12:19

Leibniz International Proceedings in Informatics



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

Since unlikely action durations may nonetheless occur, this approach incurs a non-zero risk of failure. The typical goal is to minimize this risk, although some have sought to optimize a different objective function while accepting a pre-determined bound on the risk of failure.

This paper presents significant improvements to this approach that derive from recent, faster algorithms for solving several closely related problems, as well as some new theoretical results:

1. Since the iterative process of computing a DC STNU to approximate a PSTN relies on efficiently finding negative cycles in non-DC STNUs so that they can be resolved (e.g., by tightening the bounds on participating contingent durations), this paper uses a recent, faster algorithm for finding such cycles (Algorithm `FindSRNC` [16]). Its compact representation of such cycles avoids exponential blow-up. Like some recent work, our approximating algorithm (Algorithm `genApproxSTNU`) uses a general-purpose non-linear optimization solver to aid in this process; however, `genApproxSTNU` explicitly aims to maximize the combined probability mass of the probabilistic durations captured by the STNU’s contingent durations. We also formally prove that many constraints included in others’ work are unnecessary.
2. Given an approximating DC STNU, we then propose to use a recent, fast algorithm (Algorithm `minDispESTNU` [17]) to compute an equivalent dispatchable STNU having a minimal number of edges. Doing so allows the use of a flexible and efficient real-time execution strategy, implemented by the algorithm `RTE*` [18], instead of, for example, the inflexible earliest-first strategy used by many researchers.
3. Hence, we propose to execute the PSTN using `RTE*` to exploit the strategy’s flexibility to react to observations of contingent durations that may fall outside the fixed STNU bounds.

The paper presents an empirical evaluation of our approach that provides evidence of its effectiveness in robustly executing PSTNs derived from a publicly available benchmark. In particular, it shows that taking advantage of a flexible real-time execution algorithm can increase the chances of successful executions.

2 Background

In this section, we recall the basic concepts and results about Simple Temporal Networks, Simple Temporal Networks with Uncertainty (STNUs), Probabilistic Simple Temporal Networks (PSTNs) and the known methods for approximating PSTNs by STNUs.

2.1 Simple Temporal Networks

A *Simple Temporal Network* (STN) is a pair $(\mathcal{T}, \mathcal{C})$ where \mathcal{T} is a set of real-valued variables called timepoints; and \mathcal{C} is a set of *ordinary* constraints, each of the form $(Y - X \leq \delta)$ for $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$ [5]. An STN is *consistent* if it has a solution as a constraint satisfaction problem (CSP). Each STN has a corresponding graph where the timepoints serve as nodes, and the constraints correspond to labeled, directed edges. In particular, each constraint $(Y - X \leq \delta)$ corresponds to an edge $X \xrightarrow{\delta} Y$ in the graph. Such edges may be notated as (X, δ, Y) for convenience.

A flexible and efficient *real-time execution* (RTE) algorithm has been defined for STNs that maintains time windows for each timepoint and, as each timepoint X is executed, only propagates constraints *locally*, to *neighbors* of X in the STN graph [28, 24]. An STN is called *dispatchable* if that RTE algorithm is guaranteed to satisfy all of the STN’s constraints no matter which execution decisions are made subject to the time-window constraints. Algorithms for generating equivalent dispatchable STNs have been presented [28, 24].



■ **Figure 1** A semi-reducible path (shaded gray on the left) and a Semi-Reducible Negative (SRN) cycle (shaded gray on the right).

2.2 Simple Temporal Networks with Uncertainty

A *Simple Temporal Network with Uncertainty* (STNU) augments an STN to include *contingent links* that represent actions with uncertain, but bounded durations [23]. An STNU is a triple $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ where $(\mathcal{T}, \mathcal{C})$ is an STN, and \mathcal{L} is a set of contingent links, each of the form (A, x, y, C) , where $A, C \in \mathcal{T}$ and $0 < x < y < \infty$. The semantics of STNU execution ensure that regardless of when the *activation timepoint* A is executed, the *contingent timepoint* C will occur such that $C - A \in [x, y]$. Thus, the duration $C - A$ is uncontrollable but bounded. The graph of an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is the graph of the STN $(\mathcal{T}, \mathcal{C})$ augmented to include *labeled* edges representing the contingent durations. In particular, each contingent link (A, x, y, C) has two corresponding edges in the STNU graph: a *lower-case* (LC) edge $A \xrightarrow{c:x} C$, notated as $(A, c:x, C)$, representing the uncontrollable possibility that the duration might take on its minimum value x ; and an *upper-case* (UC) edge $C \xrightarrow{C:-y} A$, notated as $(C, C:-y, A)$, representing the possibility that it might take on its maximum value y .

The most important property of an STNU is whether it is *dynamically controllable* (DC). An STNU is *dynamically controllable* (DC) if there exists a dynamic, real-time execution strategy that guarantees that all constraints in \mathcal{C} will be satisfied no matter how the contingent durations turn out [23, 10]. A strategy is dynamic because its execution decisions can react to observations of contingent executions without advance knowledge of future events. Morris [21] proved that an STNU is DC if and only if it does not include any *semi-reducible* negative cycles (SRN cycles). A path \mathcal{P} is semi-reducible if certain constraint-propagation rules can be used to provide new edges that effectively *bypass* each occurrence of an LC edge in \mathcal{P} . As an example of a semi-reducible path and an SRN cycle, consider Figure 1. In the left network, the path $\Pi = (A, c:1, C, -1, B)$ is semi-reducible because it is possible to combine constraints $(A, c:1, C)$ and $(C, -1, B)$ to create an equivalent constraint $(A, 0, B)$ (dashed red) that bypasses $(A, c:1, C)$ in Π . In the right network, the path (cycle) $\Pi = (A, c:1, C, -1, D, D:-10, B, 7, A)$ is an SRN cycle because as before, it is possible to bypass $(A, c:1, C)$ by constraint $(A, 0, D)$ (dashed red), and the value of the resulting cycle $(A, 0, D, D:-10, B, 7, A)$ (sum of constraint values discarding possible labels) is negative. Indeed, this network is not DC because A must be executed after or as soon as D occurs to satisfy $(A, 0, D)$, and in the case that the contingent link $(B, 1, -10, D)$ duration outcomes to be 10, the constraint $(B, 7, A)$ will be violated.

In 2014, Morris [22] presented the first $O(n^3)$ -time DC-checking algorithm.¹ In 2018, Cairo *et al.* [2] presented their $O(mn + k^2n + kn \log n)$ -time RUL^- algorithm. In 2022, Hunsberger and Posenato [14] subsequently presented a faster version, called RUL2021 , that has the same worst-case complexity but achieves an order-of-magnitude speedup in practice by restricting the edges it inserts into the network during constraint propagation.

¹ As is common in the literature, we use n for the number of timepoints, m for the number of ordinary constraints; and k for the number of contingent links.

Following the literature, we refer to ordinary or LC edges as LO-edges and ordinary or UC edges as OU-edges. An ESTNU graph has the form $(\mathcal{T}, \mathcal{E}_o \cup \mathcal{E}_{lc} \cup \mathcal{E}_{uc} \cup \mathcal{E}_{ucg})$, where \mathcal{E}_o is the set of ordinary edges, \mathcal{E}_{lc} and \mathcal{E}_{uc} are the sets of LC and UC edges, and \mathcal{E}_{ucg} is the set of generated *wait* edges (described later). The graphs, \mathcal{G}_{lo} and \mathcal{G}_{ou} , of the LO- and OU-edges, respectively, can be viewed as STNs by ignoring the alphabetic labels on LC or UC edges.

2.3 Probabilistic Simple Temporal Networks

A *Probabilistic Simple Temporal Network* (PSTN) is similar to an STNU, except that each contingent duration, $C - A$, is modeled as a random variable with a specified probability density function (pdf) p [27, 7]. This paper assumes that each probabilistic duration has a log-normal distribution.²

Since pdfs can have infinite tails, successfully executing a PSTN cannot be guaranteed in general. Instead, researchers have focused on approximating PSTNs by STNUs [7, 33, 30, 31]. *The approximating STNU differs from the PSTN only in representing the contingent durations; the ordinary constraints all stay the same.* The aim is to choose bounds for the approximating STNU's contingent links that capture as much probability mass of the probabilistic durations as possible while preserving the STNU's controllability. For example, if (A, x, y, C) is a contingent link approximating a probabilistic duration (A, C, p) , then the probability mass captured by the contingent link is $\int_x^y p(t)dt = F(y) - F(x)$, where F is the associated cumulative distribution function (cdf).

2.3.1 Approximating PSTNs by Strongly Controllable STNUs

Early work sought to approximate PSTNs by *strongly controllable* STNUs. (An STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is *strongly controllable* (SC) if there exists a fixed schedule for its controllable timepoints that guarantees that all constraints in \mathcal{C} will be satisfied no matter how the durations of the contingent links in \mathcal{L} turn out.) Tsamardinou [27] aimed to find a fixed schedule for a PSTN that maximized the probability that all of its constraints would be satisfied. However, his approach was too restrictive: it did not allow ordinary constraints between pairs of contingent timepoints.

Fang *et al.* [7] defined a similar problem, called the *chance-constrained probabilistic Simple Temporal Problem* (cc-pSTP). Instead of aiming to minimize the risk of failure, the cc-pSTP is the problem of finding a static schedule that optimizes a given objective function (e.g., complete all tasks as early as possible) while keeping the risk of failure below a given bound (e.g., less than 5 percent). In other words, the cc-pSTP accepts a bounded risk of failure (a.k.a. a *chance constraint*). To solve the cc-pSTP, they create an initial approximating STNU in which the bounds on each contingent link are *variables*, not constants. Their algorithm then applies constraint-propagation/edge-generation rules (a.k.a. reduction rules) to enforce the SC property. These rules are generalized from prior work on strong controllability [29, 27] to accommodate the bounds on the contingent links being variables instead of constants. The result is at most n^2 linear constraints, each involving the contingent link bounds-as-variables. In contrast, the chance constraint is non-linear since it depends on the cdfs for the probabilistic durations. They approximate the chance constraint using Boole's inequality, which does not require assuming independence of the probabilistic durations, as follows:

² Chen *et al.* [3] observed that "Existing experiments data ... showed that heavy-tailed distributions, such as lognormal, best fit the task uncertainty introduced by humans in collaborative tasks [6]. This is corroborated by work that showed the human reaction time is also best modeled as log-normal [32]."

(actual probability of failure) $\leq \sum_{i=1}^k (F_i(x_i) + (1 - F_i(y_i))) \leq \Delta$, where each F_i is the cdf for the i^{th} probabilistic link, and Δ is the given bound on the risk of failure. The objective function, which is provided as an input, can also be non-linear. After constructing their non-linear optimization problem, they solve it using an off-the-shelf solver, called SNOPT [9].

Wang and Williams [30] presented the *Rubato* algorithm, which tackles the cc-pSTP by *decoupling* the risk-allocation problem (i.e., assigning fixed bounds to the STNU's contingent links) from strong-controllability checking. In this way, the risk-allocation problem, solved by a non-linear solver, need not include the $O(n^2)$ constraints generated by the previously mentioned constraint-propagation rules, keeping the optimization problem small. Once risk allocation is done, the SC checker is run which, in negative instances, outputs a simple negative cycle. In such cases, they then accumulate a new constraint stipulating that that cycle must be made non-negative. They iteratively run this risk-allocation/SC-checking process until an SC STNU is found, which then yields a static schedule for the PSTN.

2.3.2 Approximating a PSTN by a Dynamically Controllable STNU

Wang [31] defined a dynamic version of the cc-pSTP that aims to approximate a PSTN by a DC STNU. Analogous to *Rubato*, Wang used an iterative approach that decouples risk-allocation from DC checking. For the first risk-allocation step, a non-linear optimization solver generates initial bounds for the STNU's contingent durations that capture as much of the probability mass of the PSTN's probabilistic durations as possible while also satisfying the ordinary constraints from the STNU. For the DC-checking step, Morris' $O(n^4)$ -time DC-checking algorithm is modified so that it outputs an SRN cycle for non-DC networks. Wang noted that such cycles may not be simple, but presented no details on how to compute or represent them. (In the worst case, SRN cycles can involve exponentially many edges [12].)³

If the candidate STNU happens to be non-DC, it must contain an SRN cycle, which can be resolved by making it non-negative or non-semi-reducible. Following Morris [21], Wang noted that semi-reducibility requires that each LC edge can be *reduced away* by a (negative-length) *extension subpath*.⁴ Thus, he argued that modifying any one of the participating extension sub-paths by making it non-negative would cause the entire cycle to be non-semi-reducible. (However, as shown below, this is often not the case.) Thus, Wang's approach to resolving an SRN cycle involved accumulating a *disjunction* of potentially very many new constraints, one for each participating extension subpath. Hence, his approach requires the use of a disjunctive linear program solver. Although he gives some empirical evaluations, only very high-level implementation details are provided, making the results difficult to evaluate.

3 Preliminary Steps

In this section, we introduce some preliminary results that allow the determination of a new algorithm for a robust execution of PSTNs.

³ Yu, Fang and Williams [33] addressed resolving a non-DC STNU by finding an SRN cycle within it and then tightening the bounds on participating contingent durations. However, unlike Wang, they failed to recognize that individual labeled edges can appear multiple times in an SRN cycle.

⁴ An *extension subpath* for an LC edge e in a path \mathcal{P} is a negative-length subpath \mathcal{P}_e that immediately follows e in \mathcal{P} and such that the constraint-propagation/edge-generation rules given by Morris [21] can be used to generate a new edge E that effectively bypasses e in \mathcal{P} .

3.1 Efficiently Finding and Representing SRN Cycles

Iteratively finding a DC STNU to approximate a PSTN typically requires numerous calls to an algorithm for finding SRN cycles in non-DC STNUs. For this, Wang used a modified version of Morris' $O(n^4)$ -time DC-checking algorithm. Instead, this paper takes advantage of a new, faster $O(mn + kn^2 + kn \log n)$ -time algorithm, **FindSRNC**, for finding *and compactly representing* SRN cycles [16]. Aside from its greater speed, there are two main features that are important for this paper. First, because an *indivisible* SRN cycle in a non-DC STNU can have, in the worst case, an *exponential number of occurrences* of LC and UC edges [12], the output of **FindSRNC** includes a hash table that compactly represents the repeating structures that necessarily occur in such cycles, while requiring only $O(mk + k^2n)$ space. Second, **FindSRNC**, like the RUL2021 algorithm [14] on which it is based, detects three different kinds of SRN cycles: (1) a negative cycle in the LO-graph; (2) a special kind of cycle, called a *CC loop*; and (3) a cycle arising from a cycle of interruptions of its recursive processing of UC edges. The following section recalls how Wang's approach to resolving SRN cycles introduces potentially very many disjunctive constraints and then rigorously addresses the different ways that each kind of SRN cycle returned by **FindSRNC** can be resolved, in one case without requiring any disjunctions, in another case requiring only a single disjunction, and in a third case requiring a bounded number of disjunctions.

3.2 More Efficient Resolution of SRN Cycles

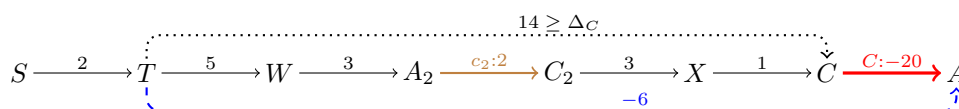
To resolve an SRN cycle L , Wang generates a *disjunctive* collection of linear constraints. The main constraint is to make $|L|$ non-negative. The other constraints, which can be numerous, aim to make L non-semi-reducible by, for each *occurrence* of an LC edge e in L , constraining its *extension subpath* \mathcal{P}_e to be non-negative. (Each occurrence of an LC edge in L can have a very different extension subpath.) The idea is that if *any* of these constraints are satisfied, then L will either be non-negative or non-semi-reducible (or both). However, while it is true that modifying an extension subpath \mathcal{P}_e by making it non-negative renders it unable to reduce away the LC edge, it does not necessarily make L non-semi-reducible. Why? Because other edges following \mathcal{P}_e in L might combine with \mathcal{P}_e to create a new extension subpath for e , as illustrated below.

$$\dots \quad A \xrightarrow{c:5} C \xrightarrow{1} A' \xrightarrow{c':4} C' \xrightarrow{-6} F \xrightarrow{-2} G \xrightarrow{-3} H \quad \dots$$

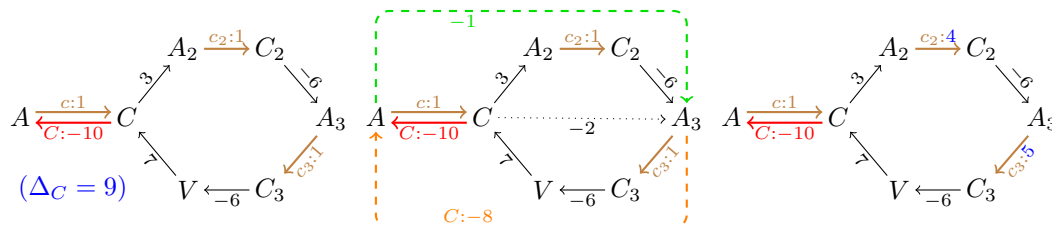
In this example, the extension subpath for the LC edge $e = (A, c:5, C)$ is the negative-length subpath from C to F , shaded dark gray. This subpath can be made non-negative by increasing the lower bound on the LC edge $(A', c':4, C')$ from 4 to 5. However, doing so would not make the overall path non-semi-reducible because the path from C to G , shaded light gray, would still be negative and hence could be used to reduce away e . As a result, a subsequent iteration of Wang's algorithm might return the very same SRN cycle, albeit with a slightly different length. Even worse, a chain of negative edges following an existing extension subpath for e might lead to numerous nearly identical iterations. Furthermore, a single SRN cycle might have many LC edges leading to numerous disjunctive constraints, thereby compounding the problem for the disjunctive optimization solver, making it expensive for larger networks.

3.3 Three Kinds of SRN Cycles Computed by FindSRNC

Before addressing how to resolve the SRN cycles output by **FindSRNC**, we must discuss how **FindSRNC** works. As shown in Figure 2, **FindSRNC** processes each UC edge $\mathbf{E} = (C, C:-y, A)$, propagating backward from C along LO-edges aiming to generate edges that effectively



■ **Figure 2** Generating a (blue, dashed) bypass edge for a (red) UC-edge, assuming that $\Delta_C = 12$.



■ **Figure 3** A CC loop (left); a CC-based SRN cycle (center); and resolving the SRN cycle (right).

bypass E. Back-propagation continues while the subpath being explored has length less than $\Delta_C = y - x$. If that distance ever becomes greater than or equal to Δ_C , as in the path from T to C in Figure 2, then a bypass edge, shown as blue and dashed, is generated, and back-propagation stops.

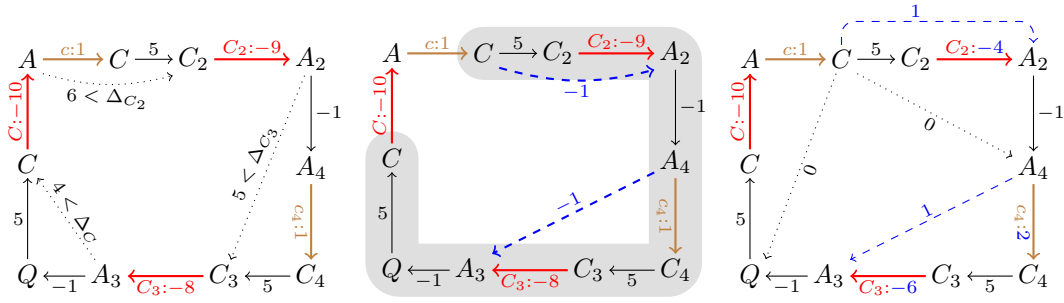
As in Johnson’s algorithm [4], the back-propagation is guided by a potential function that is a solution to the graph of LO-edges viewed as an STN. The potential function is initialized by a call to Bellman-Ford [4] and, after the processing of each UC edge, is updated to accommodate any newly generated edges. If the updating reveals a negative cycle in the LO-graph, then the STNU cannot be DC. Therefore, **FindSRNC** outputs that negative cycle.

There are two ways that **FindSRNC**’s back-propagation can be blocked: (1) by a *CC loop*, or (2) by bumping into another UC edge. A *CC loop* is where back-propagation from C cycles back to C with all encountered distances less than Δ_C , as illustrated on the lefthand side of Figure 3. A *CC loop* does not necessarily entail an SRN cycle, but it can: if there exists a negative-length LO-path emanating from C that can be used to reduce away the LC edge $(A, c:x, C)$ [14]. An example of this is shown in the center of Figure 3. Based on the edge-generation rules from Morris [21], the negative-length (dotted) path from C to A_3 can be used to generate the (dashed, green) bypass edge $(A, -1, A_3)$. Meanwhile, the path from A_3 to A can be used to generate the (dashed, orange) *wait* edge $(A_3, C:-8, A)$, thereby forming a negative cycle in the OU-graph, which implies that the network cannot be DC. In such a case, **FindSRNC** outputs the SRN cycle formed by the matching LC and UC edges together with the *CC loop*. We call such a cycle a *CC-based SRN cycle* for convenience.

Back-propagation from C can also be blocked by bumping into another UC edge, say E_2 , while encountered distances remain less than Δ_C . In such cases, **E**’s processing is interrupted until E_2 is fully processed. Once all edges bypassing E_2 have been generated, back-propagation from C continues. But if a *cycle* of such interruptions is found, all processing is blocked, and the network cannot be DC [2]. In that case, **FindSRNC** returns the SRN cycle formed by concatenating the interrupted subpaths, including the corresponding UC edges, as shown on the left of Figure 4, where it is assumed that the length of each LC edge is 1.

3.4 Resolving SRN Cycles Output by FindSRNC

SRN cycles are, by definition, *negative* and *semi-reducible*, so such cycles can be resolved by making them non-negative or non-semi-reducible. As in earlier work, we restrict attention to resolving an SRN cycle by increasing the lengths of LC or UC edges contained within it (i.e.,



■ **Figure 4** A cycle of interruptions (left); a weakened version with a (shaded) CC loop (center); making it non-semi-reducible by constraining subpaths emanating from C to be non-negative (right).

by tightening the bounds on the corresponding contingent links). Although the bypass edges computed by FindSRNC are invariably ordinary, the paths they bypass may have multiple LC and UC edges. Increasing the lengths of those LC or UC edges in turn increases the lengths of the bypass edges.

Since resolving an SRN cycle by making it non-negative is always an option, this section focuses on cases where an SRN cycle can be made non-semi-reducible without making it non-negative. The lemmas below address the three kinds of SRN cycles output by FindSRNC.

► **Lemma 1.** *If an SRN cycle comprises only LO-edges, then the only way to resolve the cycle is by making it non-negative.*

Proof. A negative cycle comprising only LO-edges is necessarily semi-reducible [13]. ◀

► **Lemma 2.** *Let L be a CC-based SRN cycle where $(C, C:-y, A)$ and $(A, c:x, C)$ are the relevant UC and LC edges. Then, the only way to make L non-semi-reducible is by making the length of each subpath emanating from C in the CC loop non-negative.*

The righthand side of Figure 3 shows an example of making a CC-based SRN cycle non-semi-reducible, in this case, by increasing the lengths of the LC edges A_2C_2 and A_3C_3 to ensure that every subpath emanating from C is non-negative. (The modified lengths are shown in blue.) Notice that the length of the entire CC-based cycle is still negative: -2 .

Proof. If any subpath emanating from C in the CC loop has negative length, then it can be used to reduce away (bypass) the LC edge $(A, c:x, C)$, preserving the SRN cycle [14]. ◀

Although each subpath emanating from C needs to be non-negative, that need not require an explicit constraint for each timepoint following C . First, since the only allowed modifications involve lengthening edges, any subpath emanating from C that is already non-negative in L does not need to be explicitly constrained. In addition, if a subpath from C to X is constrained to be non-negative and the path from X to Y is non-negative, then the subpath from C to Y will automatically be non-negative. A one-time traversal of the edges in L suffices to determine the conjunction of constraints needed to make L non-semi-reducible.

► **Lemma 3.** *Let L be an SRN cycle obtained from a cycle of interruptions of processings of UC edges (e.g., as shown on the lefthand side of Figure 4). If $\mathbf{E} = (C, C:-y, A)$ and $e = (A, c:x, C)$ are adjacent in L , then L can be made non-semi-reducible by making the length of each subpath emanating from C that does not include \mathbf{E} non-negative. Although there can be multiple pairs of adjacent labeled edges providing such opportunities for making L non-semi-reducible, there are no other ways of making L non-semi-reducible.*

Proof. A cycle of interruptions necessarily entails an SRN cycle [2], so resolving L requires breaking that cycle of interruptions. One way is to lengthen edges in L enough to enable the generation of bypass edges for all UC edges in L . But that would yield a cycle comprising only LO-edges and, since negative LO-cycles are invariably semi-reducible, resolving the SRN cycle in this way would still require making $|L|$ non-negative. The only other outcome that can arise from increasing the lengths of edges preceding a UC edge would be the creation of a CC loop, as illustrated in Figure 4 (center), where the UC edges C_2A_2 and C_3A_3 have been bypassed by dashed, blue edges, creating a CC loop from C back to C . Since a CC loop contains only LO-edges, a CC loop can only be created if all other UC edges have been bypassed.

▷ **Claim.** Constraining every subpath emanating from C that terminates at or before the UC edge $(C, C:-y, A)$, as illustrated on the righthand side of Figure 4, will ensure that L is non-semi-reducible. (In the figure, constraining the subpath from C to A_4 to be non-negative automatically ensures that the subpaths terminating at A_2 , C_4 and C_3 will also be non-negative, given the negative edge from A_2 to A_4 , and the non-negative paths from A_4 to C_4 and C_3 . Similarly, constraining the subpath from C to Q to be non-negative ensures that the subpaths terminating at A_3 and C will also be non-negative.)

Proof. If every subpath emanating from C is non-negative, then every UC edge other than $(C, C:-y, A)$ must be bypassable. For example, the first encountered UC edge $(C', C':-y', A')$ must be bypassable since the subpath from C to A' being non-negative implies that the subpath from C to C' must be at least $y' > \Delta_{C'}$. An inductive argument ensures that all following UC edges are bypassable. But then Lemma 2 ensures that the CC-based cycle formed using those bypass edges is non-semi-reducible. ◁

Finally, if any subpath emanating from C is negative, then the LC edge $(A, c:x, C)$ can be bypassed, yielding a cycle of interruptions that cannot be resolved via a CC loop involving $(C, C:-y, A)$ and $(A, c:x, C)$; hence the only options for making L non-semi-reducible must involve forming a CC loop using a different pair of adjacent, matching UC and LC edges. ◀

Summary. All three types of SRN cycles L returned by `FindSRNC` can be resolved by making L non-negative. Alternatively, L can be made non-semi-reducible if: (1) it is a CC-based SRN cycle for a contingent timepoint C , where each subpath emanating from C is non-negative; or (2) L arises from a cycle of interruptions and L includes at least one adjacent pair of matching UC and LC edges. This analysis of SRN cycles greatly reduces the need for disjunctive constraints as compared to the approach of Wang. It also avoids the problem of repeatedly revisiting the same SRN cycle, when making the length of an extension subpath non-negative, fails to make it non-semi-reducible. Finally, we conjecture that occurrences of CC loops and (especially) cycles of interruptions that can be weakened to reveal a CC loop will occur only rarely in practice and, therefore, our new algorithm, presented in Section 4, focuses exclusively on constraining the SRN cycle itself to be non-negative (i.e., a single constraint).

4 New Algorithm for Robustly Executing PSTNs

Given any PSTN, our new algorithm for robustly executing PSTNs: (1) computes an approximating STNU that is DC, using the `FindSRNC` algorithm to efficiently compute and compactly represent SRN cycles in non-DC STNUs; (2) converts that STNU into an equivalent dispatchable ESTNU; and (3) executes the original PSTN using the RTE* algorithm, leveraging its flexibility to react to possibly extreme contingent durations.

■ **Algorithm 1** `genApproxSTNU`: generate a DC STNU that approximates a given PSTN.

Input: $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{M})$: a PSTN where $\mathcal{M} = \{(A_i, C_i, \text{Lognormal}(\mu_i, \sigma_i)) \mid i \in \{1, \dots, k\}\}$
Output: (\mathcal{S}_u, F) , where $\mathcal{S}_u = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is an approximating DC STNU for \mathcal{S} , and F is the joint probability mass of the durations in \mathcal{M} captured by the links in \mathcal{L} . Or \perp if unable.

// Initialize the approximating STNU
1 $\mathcal{S}_u := (\mathcal{T}, \mathcal{C}, \mathcal{L})$, where $\mathcal{L} = \{(A_i, x_i = e^{\mu_i - 3.3\sigma_i}, y_i = e^{\mu_i + 3.3\sigma_i}, C_i) \mid i \in \{1, \dots, k\}\}$
2 $(L, \mathcal{H}) := \text{FindSRNC}(\text{copy}(\mathcal{S}_u))$ // L = SRN cycle; \mathcal{H} = edge-annotation hash table
3 **while** L **do**
4 // Below, $\text{len} = |L|$; a_i, b_i = num. occurrences of i th LC, UC edges in (fully expanded) L
5 $(\text{len}, (a_1, \dots, a_n), (b_1, \dots, b_n)) := \text{fetchEdgeInfo}(\text{negCycle}, \text{edgeAnnHash})$
6 **if** $\sum_{i=1}^k (a_i + b_i) == 0$ **then return** \perp // No labeled edges in expanded SRN cycle
7 $\mathcal{A} := \{i \mid a_i > 0 \text{ or } b_i > 0\}$ // Collect indices of contingent links participating in SRN cycle
8 $\kappa := |\mathcal{A}|$ // $\kappa \leq k$ is num. contingent links participating in SRN cycle
9 Let $\pi: \{1, \dots, \kappa\} \mapsto \mathcal{A}$ be a re-ordering of the indices of \mathcal{A} from 1 to κ
10 $\text{bounds} := (x_{\pi(1)}, y_{\pi(1)}, \dots, x_{\pi(\kappa)}, y_{\pi(\kappa)})$
11 $\text{muVec} := (\mu_{\pi(1)}, \dots, \mu_{\pi(\kappa)})$; $\text{sigVec} := (\sigma_{\pi(1)}, \dots, \sigma_{\pi(\kappa)})$
12 $\text{coeffs} := (a_{\pi(1)}, -b_{\pi(1)}, \dots, a_{\pi(\kappa)}, -b_{\pi(\kappa)})$
13 $\text{const} := -\text{len} + \sum_{1 \leq i \leq \kappa} (a_{\pi(i)} x_{\pi(i)} - b_{\pi(i)} y_{\pi(i)})$
14 $(\hat{x}_{\pi(1)}, \hat{y}_{\pi(1)}, \dots, \hat{x}_{\pi(\kappa)}, \hat{y}_{\pi(\kappa)}, \hat{F}) := \text{nlpOpt}(\kappa, \text{muVec}, \text{sigVec}, \text{coeffs}, \text{const}, \text{bounds})$
15 **if** $\hat{F} == \perp$ **then return** \perp
16 **foreach** $i \in \{1, \dots, \kappa\}$ **do** $x_{\pi(i)} := \hat{x}_{\pi(i)}$ **and** $y_{\pi(i)} := \hat{y}_{\pi(i)}$ // Update bounds
17 $(\text{negCycle}, \text{edgeAnnHash}) := \text{FindSRNC}(\text{copy}(\mathcal{S}_u))$ // Prepare for next iteration
18 // \mathcal{S}_u is dynamically controllable. Re-compute objective function over all contingent links.
19 $F := \prod_{1 \leq i \leq k} (\text{lnCDF}(y_i, \mu_i, \sigma_i) - \text{lnCDF}(x_i, \mu_i, \sigma_i))$
20 **return** (\mathcal{S}_u, F) , where \mathcal{S}_u has updated bounds $(x_1, y_1, \dots, x_k, y_k)$

4.1 Generating a DC STNU to Approximate a PSTN

The `genApproxSTNU` algorithm (Algorithm 1) takes as its input a PSTN \mathcal{S} with k probabilistic durations of the form $(A_i, C_i, \text{Lognormal}(\mu_i, \sigma_i))$.⁵ It aims to generate an approximating STNU for \mathcal{S} that is DC by providing bounds for the contingent links that maximize the joint probability mass of the probabilistic durations they capture while preserving the DC property.

At Line 1, the approximating STNU is initialized by setting the bounds for each contingent link (A_i, x_i, y_i, C_i) to $x_i = e^{\mu_i - 3.3\sigma_i}$ and $y_i = e^{\mu_i + 3.3\sigma_i}$, which represent ± 3.3 standard deviations for the underlying normal distribution, which ensures capturing approximately 99.96% of the probability mass. As a result, we expect that the initial STNU will *not* be DC.

Next, at Line 2, it calls the `FindSRNC` algorithm on a *copy* of the STNU. (`FindSRNC` destructively modifies its input.) For non-DC STNUs, `FindSRNC` outputs a compact representation of an SRN cycle as a pair, (L, \mathcal{H}) , where L is a list of ordinary, LC and UC edges with no repeats, and \mathcal{H} is an *edge-annotation* hash table [16]. Although L has no repeat edges, some of its ordinary edges may be *bypass* edges. Each bypass edge E in L has an entry in the hash table \mathcal{H} that identifies the path \mathcal{P}_E bypassed by E . In addition, the bypassed paths may recursively include other bypass edges. In the worst case, fully expanding L by recursively replacing each occurrence of a bypass edge by the path it bypassed can lead to an exponential number of edges due to the presence of repeated structures [11]. In contrast, the edge-annotation hash table uses only $O(k^2n)$ space to store the relevant information [16].

⁵ In other words, $\text{Lognormal}(\mu_i, \sigma_i) = e^{\mu_i + \sigma_i Z}$, where Z is a standard normal random variable.

As long as `FindSRNC` returns an SRN cycle, the `while` loop at Lines 3–16 aims to resolve the cycle by tightening the bounds on the participating contingent links while retaining as much of the probability mass from the corresponding probabilistic durations as possible. Each iteration begins, at Line 4, by calling the `fetchEdgeInfo` algorithm (Algorithm 2) which returns the following information: `len`, the length of (one traversal of) the SRN cycle; and two vectors (a_1, \dots, a_k) and (b_1, \dots, b_k) , where each a_i specifies the number of occurrences of the LC edge $(A_i, c_i: x_i, C_i)$ in the (fully expanded) SRN cycle, and each b_i the number of occurrences of the UC edge $(C_i, C_i: -y_i, A_i)$. Crucially, as will be seen later, this can be done in $O(nk^2)$ time, even if the (fully expanded) cycle contains an exponential number of edges.

At Line 5, if there are no labeled edges in the (fully expanded version of) the SRN cycle, `genApproxSTNU` returns \perp , since such a cycle cannot be resolved by adjusting the bounds on contingent links. Otherwise, at Lines 6–12, it prepares data for the the constraint optimization problem of finding new bounds for the contingent links that maximize the captured joint probability mass subject to the constraint of making the SRN cycle non-negative.

At Line 6, the set \mathcal{A} collects the indices i for the contingent links whose labeled edges participate in the SRN cycle L . At Line 7, $\kappa = |\mathcal{A}| \leq k$ denotes the number of contingent links participating in L . Since resolving the SRN cycle only requires dealing with those κ contingent links, Line 8 specifies a bijection π from $\{1, 2, \dots, \kappa\}$ to \mathcal{A} that facilitates preparing data for the non-linear solver, focusing only on the participating contingent links.

Lines 9–10 collect, for each participating contingent link, the current values of the bounds, $x_{\pi(i)}$ and $y_{\pi(i)}$, and the μ_i and σ_i values of the associated log-normal distributions. Lines 11–12 collect information needed to specify the constraint, $|L| \geq 0$. First, `coeffs` collects the number of occurrences of the labeled edges from participating contingent links. These counts are important because, for example, increasing the value of some x_i to \hat{x}_i increases $|L|$ by $a_i(\hat{x}_i - x_i)$, while decreasing y_i to \hat{y}_i increases $|L|$ by $b_i(y_i - \hat{y}_i)$. Overall, changing the values in $(x_{\pi(1)}, y_{\pi(1)}, \dots, x_{\pi(\kappa)}, y_{\pi(\kappa)})$ to $(\hat{x}_{\pi(1)}, \hat{y}_{\pi(1)}, \dots, \hat{x}_{\pi(\kappa)}, \hat{y}_{\pi(\kappa)})$ increases $|L|$ by:

$$\sum_{i=1}^{\kappa} (a_{\pi(i)}(\hat{x}_{\pi(i)} - x_{\pi(i)}) + b_{\pi(i)}(y_{\pi(i)} - \hat{y}_{\pi(i)}))$$

Therefore, satisfying $|L| \geq 0$ requires choosing values, $\hat{x}_{\pi(i)}$ and $\hat{y}_{\pi(i)}$, such that:

$$\sum_{i=1}^{\kappa} (a_{\pi(i)}\hat{x}_{\pi(i)} - b_{\pi(i)}\hat{y}_{\pi(i)}) \geq -|L| + \sum_{i=1}^{\kappa} (a_{\pi(i)}x_{\pi(i)} - b_{\pi(i)}y_{\pi(i)})$$

The lefthand sum is a linear combination of the *variables*, $\hat{x}_{\pi(i)}$ and $\hat{y}_{\pi(i)}$, while the quantity on the righthand side is a constant. That constant is assigned to `const` at Line 12.

Line 13 calls a non-linear optimization solver, here called `nlpOpt`. Currently, our algorithm uses the `fmincon` solver provided by Matlab; others have used the SNOPT solver. If the solver is unable to find a new set of bounds for the contingent links to resolve the SRN cycle, then the entire algorithm fails. However, if successful, it returns a vector of the new bounds, \hat{x}_i and \hat{y}_i , and the value of the objective function F . Line 15 updates the bounds in the STNU to reflect the new values. Line 16 calls `FindSRNC` in preparation for the next iteration of the `while` loop. If Line 18 is reached, then the STNU \mathcal{S}_u has been made DC. It is returned by the algorithm, along with the updated value of the objective function.

The `fetchEdgeInfo` Algorithm

The `fetchEdgeInfo` algorithm (Algorithm 2) accumulates the numbers of occurrences of LC and UC edges in the SRN cycle L . Crucially, it does not need to expand L fully. Instead, it uses a hash table, `infoHash`, to keep track of the numbers of occurrences of labeled edges recursively hiding within each encountered bypass edge. When it first sees a bypass edge

Algorithm 2 `fetchEdgeInfo`.

```

Input:  $k$ , the number of contingent links;  $\mathcal{P}$ , a path in an STNU graph; edgeAnnHash, a
hash-table of  $(E, \mathcal{P}_E)$  pairs where  $\mathcal{P}_E$  is the path bypassed by the edge  $E$ 
Output:  $(\text{len}, (a_1, \dots, a_k), (b_1, \dots, b_k))$ , where  $\text{len} = |\mathcal{P}|$ , and  $a_i$  and  $b_i$  are the numbers of
times  $(A_i, c_i : x_i, C_i)$  and  $(C_i, C_i : -y_i, A_i)$  appear in the fully unwound version of  $\mathcal{P}$ 
1 infoHash := new hash table; len := 0
2 lcCounts :=  $(0, \dots, 0)$ ; ucCounts :=  $(0, \dots, 0)$  // Counts of occurrences of LC/UC edges
3 foreach  $E \in \mathcal{P}$  do
4   if  $E = (A_i, c_i : x_i, C_i)$  is an LC edge for some  $i$  then
5     len := len +  $x_i$ ; lcCounts[ $i$ ] := lcCounts[ $i$ ] + 1
6   else if  $E = (C_i, C_i : -y_i, A_i)$  is a UC edge for some  $i$  then
7     len := len -  $y_i$ ; ucCounts[ $i$ ] := ucCounts[ $i$ ] + 1
8   else if  $\exists (E, \mathcal{P}_E) \in \text{edgeAnnHash}$  then //  $E$  is a bypass edge for path  $\mathcal{P}_E$ 
9     if  $\exists (E, \cdot) \in \text{infoHash}$  then //  $E$  has already been processed by fetchEdgeInfo
10      (len', lcCounts', ucCounts') := infoHash.getValue(E)
11     else
12       (len', lcCounts', ucCounts') := fetchEdgeInfo(k, \mathcal{P}_E) // Recursively process  $\mathcal{P}_E$ 
13       infoHash.setValue(E, (len', lcCounts', ucCounts')) // Store results in infoHash
14       len := len + len'
15       foreach  $i \in \{1, 2, \dots, k\}$  do
16         lcCounts[ $i$ ] := lcCounts[ $i$ ] + lcCounts'[ $i$ ]; ucCounts[ $i$ ] := ucCounts[ $i$ ] + ucCounts'[ $i$ ]
17     else len = len +  $|E|$  //  $E$  is an ordinary edge from the original STNU
18 return  $(\text{len}, \text{lcCounts}, \text{ucCounts})$ 

```

E , it recursively processes it, then stores the vectors of counts in the `infoHash` hash table. Subsequent encounters with E only need to do a constant-time look-up in the hash table (cf. Lines 9–13 in Algorithm 2). `fetchEdgeInfo` requires $O(nk^2)$ space due to at most $O(kn)$ entries stored in the `infoHash` hash table, each of size $O(k)$. This is less than the $O(n^2k)$ size of the edge-annotation hash table, \mathcal{H} , passed in as an input.

4.2 Flexible and Efficient Real-time Execution

Most DC-checking algorithms generate conditional *wait* constraints that must be satisfied by any valid execution strategy. Each wait is represented by a labeled edge of the form $(W, C : -w, A)$, which can be glossed as: “While C remains unexecuted, W must wait at least w after A .” (Despite the similar notation, a wait is distinguishable from the original UC edge since its source timepoint is *not* the contingent timepoint C .) Morris [22] defined an *Extended STNU* (ESTNU) to be an STNU augmented with such waits. He then extended the notion of dispatchability to ESTNUs, defining an ESTNU to be dispatchable if all of its *STN projections* are STN-dispatchable.⁶ He then argued that a dispatchable ESTNU would necessarily provide a guarantee of flexible and efficient real-time execution.

⁶ A projection of an ESTNU is the STN derived from forcing its contingent durations to take on fixed values. Each edge in an ESTNU projects onto an ordinary STN edge. For example, in the projection where $C - A = 4$, the edges $(A, c : 2, C)$, $(C, C : -9, A)$, $(W, C : -7, A)$ and $(V, C : -3, A)$ project onto the ordinary edges $(A, 4, C)$, $(C, -4, A)$, $(W, -4, A)$ and $(V, C : -3, A)$, respectively [18].

Hunsberger and Posenato [18] later:

1. formally defined a flexible and efficient real-time execution algorithm for ESTNUs, called RTE*;
2. defined an ESTNU to be dispatchable if every run of RTE* necessarily satisfies all of the ESTNU’s constraints; and
3. proved that an ESTNU satisfying their definition of dispatchability necessarily satisfies Morris’ definition (i.e., all of its STN projections are STN-dispatchable).

The RTE* algorithm provides maximum flexibility during execution, unlike the *earliest-first* strategy used for non-dispatchable networks.

Most DC-checking algorithms do not generate dispatchable ESTNUs. However, Morris [22] argued that his $O(n^3)$ -time DC-checking algorithm could be modified, without impacting its complexity, to generate a dispatchable output. In 2023, Hunsberger and Posenato [15] presented a faster, $O(mn + kn^2 + n^2 \log n)$ -time ESTNU-dispatchability algorithm. However, neither of these algorithms provides any guarantees about the number of edges in the dispatchable output. More recently, Hunsberger and Posenato [17] presented `minDispESTNU`, the first ESTNU-dispatchability algorithm that, in $O(kn^3)$ time, generates an equivalent dispatchable ESTNU *having a minimal number of edges*, which is important since it directly affects the real-time computations of the RTE* algorithm.

Our new approach to executing PSTNs in real time is the first to explore the use of the flexible and efficient RTE* algorithm. To enable this, we first use the `minDispESTNU` algorithm to convert the DC STNU output by `genApproxSTNU` into an equivalent, dispatchable ESTNU having a minimal number of edges. Then, we execute the PSTN using the RTE* algorithm as if it were being applied to the dispatchable ESTNU. In other words, the time-windows and wait constraints maintained by RTE* are determined by the ESTNU’s edges. In addition, to increase the chances of successful execution, RTE* is run not with the needlessly inflexible *earliest-first* strategy that has been used by others [3, 31, 8], but with a more flexible *midpoint* strategy made available by RTE*. In particular, if a currently enabled timepoint X has a time-window $[a, b]$, then instead of executing X at a , we execute it at $\frac{a+b}{2}$. This enables RTE* to adapt to unexpected durations that fall outside the STNU’s fixed bounds.

5 Empirical Evaluation

We evaluated the robust execution of PSTNs by generating random PSTN instances, then executing them using the RTE* algorithm based on the approximating STNU, converted to a dispatchable ESTNU. We randomly generated durations for the probabilistic links according to their distributions. Since the probabilistic durations could fall outside the contingent bounds of the ESTNU, RTE* might not succeed in all instances, but the percentage of successful executions across random trials provides a measure of the PSTN’s robustness.

We wanted to evaluate whether (1) creating a dynamically controllable STNU to approximate a PSTN; and (2) taking advantage of the flexibility offered by the RTE* execution algorithm might lead to a greater percentage of successful PSTN executions, even in cases where the sampled durations fall outside the STNU’s fixed bounds. Toward that end, we took *non-DC* STNUs from a published benchmark [25] and converted them into PSTNs as described in the Appendix (cf. the `GenPSTN` algorithm, Algorithm 4). The results of this phase are summarized in Table 1, where n , k , and m are the numbers of timepoints, contingent durations, and constraints; “exTime” is the average time to execute `genApproxSTNU`; “optTime” is the average time spent running the non-linear optimization solver; “#NLOprobs” is the average number of calls to the non-linear optimization solver; “%probMass” is the average probability mass of the probabilistic links captured by the approximating STNU; and

■ **Table 1** Results using `genApproxSTNU` to generate DC approximating STNUs for PSTNs.

#PSTNs	n	k	\bar{m}	exTime [s]	optTime [s]	#NLOprobs	%probMass	#RCs
24	500	50	1558	0.191	0.141	0.96	77	11
24	1000	100	3136	0.223	0.042	1.00	67	17
14	1500	150	4713	0.573	0.100	1.21	43	10
17	2000	200	6289	0.914	0.046	1.11	53	16

■ **Table 2** Results of RTE* execution algorithm on PSTNs: Earliest-First (EF) vs. *Midpoint (MP)*.

#PSTNs	n	k	\bar{m}	execTP (μ s)	%trials-in succ		%trials-out succ		%trials-out fail		num out if succ		num out if fail	
					EF	<i>MP</i>	EF	<i>MP</i>	EF	<i>MP</i>	EF	<i>MP</i>	EF	<i>MP</i>
24	500	50	2500	9.16	73	<i>73</i>	5	<i>5</i>	22	<i>22</i>	1.08	<i>1.09</i>	1.06	<i>1.06</i>
24	1000	100	5119	14.98	66	<i>66</i>	8	<i>6</i>	26	<i>28</i>	1.03	<i>1.03</i>	1.10	<i>1.12</i>
14	1500	150	7883	26.14	58	<i>58</i>	4	<i>7</i>	38	<i>35</i>	1.07	<i>1.08</i>	1.16	<i>1.15</i>
17	2000	200	106522	31.05	53	<i>53</i>	8	<i>8</i>	39	<i>39</i>	1.10	<i>1.11</i>	1.21	<i>1.23</i>

“#RCs” is the number of approximating STNUs having one or more activation timepoints participating in rigid components. As expected, the percentage of the probability mass captured by the approximating STNU fell as the number of contingent durations increased since, for example, $.995^{50} \approx .778$, whereas $.995^{200} \approx .367$. In addition, since the initial STNU was non-DC, making it DC could require reducing contingent ranges significantly.

After converting the STNU instances into their minimal dispatchable form [17], we ran the RTE* algorithm 200 times on each dispatchable ESTNU, where the contingent durations were obtained by randomly sampling the associated log-normal distributions (15800 executions in total). To test the impact of the execution strategy on the rate of successful execution, the execution of each network in the same situation (i.e., in the same projection) was run twice: once with the earliest-first strategy, which executes timepoints as soon as possible, and once with the midpoint strategy, which executes timepoints at the midpoints of their time-windows. Table 2 summarizes our results, where: “execTP” reports the average time (in μ secs) to schedule each timepoint; “%trials-in succ”, the percentage of executions/trials where all sampled durations fell within the respective contingent bounds of the ESTNU. For such cases, the execution strategy (earliest-first results in plain text, midpoint in italic) is irrelevant because any RTE* execution is guaranteed to succeed for dispatchable ESTNUs. Column “%trials-out succ” reports the percentage of trials where one or more contingent durations fell outside the ESTNU’s contingent bounds (called *outlier trials*), but the execution succeeded anyway due to the flexibility of RTE* (higher value represents the best performance); while “%trials-out fail” reports the average number of outlier trials where the execution failed (lower value represents the best performance). Column “num out if fail” reports the average number of outlier durations in failed executions; while “num out if succ” reports the average number of outlier durations in successful executions. The comparison of the “%probMass” values from Table 1 and “%trials-in succ” from Table 2 confirms that the probability mass captured by the ESTNU’s contingent links corresponds to situations that always generate successful executions. It is not clear if the execution strategy for the controllable timepoints (earliest-first or midpoint) can increase the rate of successful executions, given that the number of different PSTNs is limited. Further investigation is necessary, including on PSTNs from real-world applications. Nonetheless, our results provide evidence that the RTE* algorithm makes it possible to have successful executions even when one or more contingent durations are outside the ESTNU’s bounds.

Our implementations are publicly available [26].

6 Conclusions

The paper presented a new approach to the robust execution of PSTNs that takes advantage of several recent efficient algorithms for:

1. finding and compactly representing SRN cycles in non-DC STNUs;
2. converting DC STNUs into equivalent, dispatchable ESTNUs having a minimal number of edges; and
3. flexibly and efficiently executing ESTNUs in real time.

We presented a new algorithm to generate an approximating STNU that aims to maximize the combined probability mass of the PSTN's probabilistic durations while maintaining the dynamic controllability of the STNU; and a formal analysis of SRN cycles that provided new insights into how to efficiently resolve them while avoiding issues arising in past approaches. Our empirical evaluation of our approach provides evidence of its effectiveness on robustly executing PSTNs derived from a publicly available benchmark. In particular, it shows that approximating a PSTN by a dispatchable ESTNU and taking advantage of a flexible real-time execution algorithm can increase the chances for a successful execution of that PSTN.

References

- 1 Nikhil Bhargava. Multi-Agent Coordination under Uncertain Communication. *33rd AAAI Conference on Artificial Intelligence (AAAI-19)*, 33(1):9878–9879, 2019. doi:10.1609/aaai.v33i01.33019878.
- 2 Massimo Cairo, Luke Hunsberger, and Romeo Rizzi. Faster Dynamic Controllability Checking for Simple Temporal Networks with Uncertainty. In *25th International Symposium on Temporal Representation and Reasoning (TIME-2018)*, volume 120 of *LIPICs*, pages 8:1–8:16, 2018. doi:10.4230/LIPICs.TIME.2018.8.
- 3 Rosy Chen, Yiran Ma, Siqi Wu, and James C. Boerkoel, Jr. Sensitivity analysis for dynamic control of pstns with skewed distributions. In *33rd International Conference on Automated Planning and Scheduling (ICAPS 2023)*, volume 33, pages 95–99, 2023. doi:10.1609/icaps.v33i1.27183.
- 4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 4th Edition*. MIT Press, 2022. URL: <https://mitpress.mit.edu/9780262046305/introduction-to-algorithms>.
- 5 Rina Dechter, Itay Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. doi:10.1016/0004-3702(91)90006-6.
- 6 Maya Abo Dominguez, William La, and James C. Boerkoel Jr. Modeling human temporal uncertainty in human-agent teams. *CoRR*, abs/2010.04849, 2020. URL: <https://arxiv.org/abs/2010.04849>, arXiv:2010.04849.
- 7 Cheng Fang, Peng Yu, and Brian C. Williams. Chance-constrained probabilistic simple temporal problems. In *28th AAAI Conference on Artificial Intelligence (AAAI-2014)*, volume 3, pages 2264–2270, 2014. doi:10.1609/aaai.v28i1.9048.
- 8 Michael Gao, Lindsay Popowski, and James C. Boerkoel, Jr. Dynamic Control of Probabilistic Simple Temporal Networks. In *34th AAAI Conference on Artificial Intelligence (AAAI-20)*, volume 34, pages 9851–9858, 2020. doi:10.1609/aaai.v34i06.6538.
- 9 Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. doi:10.1137/S0036144504446096.
- 10 Luke Hunsberger. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *16th International Symposium on Temporal Representation and Reasoning (TIME-2009)*, pages 155–162, 2009. doi:10.1109/TIME.2009.25.

- 11 Luke Hunsberger. Magic Loops in Simple Temporal Networks with Uncertainty—Exploiting Structure to Speed Up Dynamic Controllability Checking. In *5th International Conference on Agents and Artificial Intelligence (ICAART-2013)*, volume 2, pages 157–170, 2013. doi:10.5220/0004260501570170.
- 12 Luke Hunsberger. Magic Loops and the Dynamic Controllability of Simple Temporal Networks with Uncertainty. In Joaquim Filipe and Ana Fred, editors, *Agents and Artificial Intelligence*, volume 449 of *Communications in Computer and Information Science (CCIS)*, pages 332–350, 2014. doi:10.1007/978-3-662-44440-5_20.
- 13 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.
- 14 Luke Hunsberger and Roberto Posenato. Speeding up the RUL⁻ Dynamic-Controllability-Checking Algorithm for Simple Temporal Networks with Uncertainty. In *36th AAAI Conference on Artificial Intelligence (AAAI-22)*, volume 36-9, pages 9776–9785. AAAI Pres, 2022. doi:10.1609/aaai.v36i9.21213.
- 15 Luke Hunsberger and Roberto Posenato. A Faster Algorithm for Converting Simple Temporal Networks with Uncertainty into Dispatchable Form. *Information and Computation*, 293(105063):1–21, 2023. doi:10.1016/j.ic.2023.105063.
- 16 Luke Hunsberger and Roberto Posenato. A Faster Algorithm for Finding Negative Cycles in Simple Temporal Networks with Uncertainty. In *The 31st International Symposium on Temporal Representation and Reasoning (TIME-2024)*, volume 318 of *LIPICs*, 2024. doi:10.4230/LIPICs.TIME.2024.8.
- 17 Luke Hunsberger and Roberto Posenato. Converting Simple Temporal Networks with Uncertainty into Minimal Equivalent Dispatchable Form. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, volume 34, pages 290–300, 2024. doi:10.1609/icaps.v34i1.31487.
- 18 Luke Hunsberger and Roberto Posenato. Foundations of Dispatchability for Simple Temporal Networks with Uncertainty. In *16th International Conference on Agents and Artificial Intelligence (ICAART 2024)*, volume 2, pages 253–263. SCITEPRESS, 2024. doi:10.5220/0012360000003636.
- 19 Erez Karpas, Steven J. Levine, Peng Yu, and Brian C. Williams. Robust Execution of Plans for Human-Robot Teams. In *25th Int. Conf. on Automated Planning and Scheduling (ICAPS-15)*, volume 25, pages 342–346, 2015. doi:10.1609/icaps.v25i1.13698.
- 20 Dimitri Kececioglu et al. *Reliability Engineering Handbook*, volume 1. DEStech Publications, Inc, 2002.
- 21 Paul Morris. A Structural Characterization of Temporal Dynamic Controllability. In *Principles and Practice of Constraint Programming (CP-2006)*, volume 4204, pages 375–389, 2006. doi:10.1007/11889205_28.
- 22 Paul Morris. Dynamic controllability and dispatchability relationships. In *Int. Conf. on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR-2014)*, volume 8451 of *LNCS*, pages 464–479. Springer, 2014. doi:10.1007/978-3-319-07046-9_33.
- 23 Paul Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *17th Int. Joint Conf. on Artificial Intelligence (IJCAI-2001)*, volume 1, pages 494–499, 2001. URL: <https://www.ijcai.org/Proceedings/01/IJCAI-2001-e.pdf>.
- 24 Nicola Muscettola, Paul H. Morris, and Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning, KR'98*, pages 444–452, 1998.
- 25 Roberto Posenato. STNU Benchmark version 2020, 2020. Last access 2022-12-01. URL: <https://profs.scienze.univr.it/~posenato/software/cstnu/benchmarkWrapper.html>.
- 26 Roberto Posenato. CSTNU Tool: A Java library for checking temporal networks. *SoftwareX*, 17:100905, 2022. doi:10.1016/j.softx.2021.100905.

- 27 Ioannis Tsamardinos. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence (SETN 2002)*, volume 2308 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 97–108, 2002. doi:10.1007/3-540-46014-4_10.
- 28 Ioannis Tsamardinos, Nicola Muscettola, and Paul Morris. Fast Transformation of Temporal Plans for Efficient Execution. In *15th National Conf. on Artificial Intelligence (AAAI-1998)*, pages 254–261, 1998. URL: <https://cdn.aaai.org/AAAI/1998/AAAI98-035.pdf>.
- 29 Thierry Vidal and H el ene Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999. doi:10.1080/095281399146607.
- 30 Andrew Wang and Brian C. Williams. Chance-Constrained Scheduling via Conflict-Directed Risk Allocation. In *29th Conference on Artificial Intelligence (AAAI-2015)*, volume 29, 2015. doi:10.1609/aaai.v29i1.9693.
- 31 Andrew J. Wang. *Risk-bounded Dynamic Scheduling of Temporal Plans*. PhD thesis, Massachusetts Institute of Technology, 2022. URL: <https://hdl.handle.net/1721.1/147542>.
- 32 Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. Silence is also evidence: interpreting dwell time for recommendation from psychological perspective. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13*, pages 989–997, 2013. doi:10.1145/2487575.2487663.
- 33 Peng Yu, Cheng Fang, and Brian Charles Williams. Resolving uncontrollable conditional temporal problems using continuous relaxations. In *24th International Conference on Automated Planning and Scheduling, ICAPS 2014*. AAAI, 2014. URL: <http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7895>, doi:10.1609/icaps.v24i1.13623.

Appendix

A Procedure for Tighten Contingent Bounds to Resolve an SRN

In this section, we propose `nlpOpt`, a possible algorithm that tightens contingent bounds to resolve an SRN cycle using “Sparse Nonlinear OPTimizer” (SNOPT) library [9]. SNOPT is a software package for solving large-scale optimization problems (linear and nonlinear programs). It employs a sparse Sequential quadratic programming (SQP) algorithm with limited-memory quasi-Newton approximations to the Hessian of Lagrangian.

In `nlpOpt` we assume that the bounds on contingent links are monotonically tightened, using only a single linear constraint per iteration. A different possibility is to collect the linear constraints from each iteration and run the optimization solver on all of the accumulated constraints.

In the experimental evaluation, it was not possible to use the SNOPT library due to a compatibility problem. MatLab-Optimization Toolbox library offers the `fmincon` function to solve *minimization constrained nonlinear problems* using a sparse Sequential quadratic programming (SQP) algorithm, the same technique used by SNOPT. Therefore, we adapted the `nlpOpt` algorithm, reformulating the optimization problem as a minimization one and using a MatLab script to represent the non-linear objective function.

B PSTN Generation

To generate a set of PSTN instances for our benchmark, we considered the set of random *non-DC* STNUs from a published benchmark [25]. Such instances aim to represent the temporal representation of business processes organized in worker lanes. Contingent links represent tasks and ordinary links represent temporal deadlines or release times of such tasks.

■ **Algorithm 3** The `nlpOpt` algorithm: tighten contingent bounds to resolve an SRN cycle.

Input: k : the number of contingent durations; (μ_1, \dots, μ_k) and $(\sigma_1, \dots, \sigma_k)$: k -vectors of μ and σ parameters for log-normal distributions; **coeffs** and **const**: a matrix of coefficients and a corresponding vector of lower bounds for one or more linear constraints;
 $(x_1, y_1, \dots, x_k, y_k)$: a vector of initial bounds for the contingent durations

Output: (\mathbf{v}, \mathbf{F}) , where \mathbf{v} contains optimized bounds for the k contingent durations, and \mathbf{F} is the corresponding value of the objective function

```

1  snN := 2k
2  numCs := numRows(coeffs)           // numCs is the number of linear constraints
3  snNF := 1 + numCs                  // snNF includes 1 for the objective function
4  F := a new vector with snNF slots // F will hold values of objective function and linear constraints
   // Initialize v, the vector of variables
5  v := (x1, y1, ..., xk, yk)
   // Set lower and upper bounds for the variables in v
6  vlow := (x1, eμ1, x2, eμ2, ..., xk, eμk)
7  vupp := (eμ1, y1, eμ2, y2, ..., eμk, yk)
   // Set lower and upper bounds for the objective function (in [-1, 0]) and the linear constraints
8  Flow := (-1, const[1], const[2], ..., const[numCs])
9  Fupp := (0, ∞, ∞, ..., ∞)
   // Local function that SNOPT uses to compute the objective function and linear constraints
10 Function stnuUsrFun(v):           // v = (ℓ1, u1, ..., ℓk, uk)
    // Store the value of the objective function in F[1]
11    F[1] := Π1 ≤ i ≤ k (lnCDF(ui, μi, σi) - lnCDF(ℓi, μi, σi)) // lnCDF = log-normal CDF
    // Store the values of the lefthand sides of the linear constraints in F[2], ..., F[snNF]
12    foreach j ∈ {1, ..., numRows(coeffs)} do
13    | F[j + 1] := coeffs[j][1] * v[1] + ... coeffs[j][2k] * v[2k]
    // Call the SNOPT solver, which destructively modifies v
14 (v, F, ...) := snSolveA(v, vlow, vupp, Flow, Fupp, &stnuUsrFun)
15 return (v, F)

```

■ **Algorithm 4** The `GenPSTN` algorithm: generation of a PSTN candidate from an STNU.

Input: $\mathcal{N} = (\mathcal{T}_N, \mathcal{C}_N, \mathcal{L})$: an STNU where \mathcal{L} is a set of k contingent links, each of the form (A_i, x_i, y_i, C_i) , where $A, C \in \mathcal{T}$ and $0 < x < y < \infty$.

Output: $\mathcal{S} = (\mathcal{T}_S, \mathcal{C}_S, \mathcal{M})$: a PSTN where $\mathcal{M} = \{(A_i, C_i, \text{Lognormal}(\mu_i, \sigma_i)) \mid i \in \{1, \dots, k\}\}$

```

1  TS := TN
2  CS := CN
3  M := ∅
4  σf := 0.3 // Factor to limit the final σ value
5  foreach (A, x, y, C) ∈ L do
6  | M = (x + y)/2
7  | S = σf(y - x)/2
8  | μ = ln(M2/√(M2 + S2))
9  | σ = √(ln(1 + S2/M2))
10 | M := M ∪ {(A, C, Lognormal(μ, σ))}
11 return (TS, CS, M)

```

Each random STNU was converted into a PSTN using the `GenPSTN` algorithm described in Algorithm 4. For each contingent link (A, x, y, C) in the STNU, `GenPSTN` creates a probabilistic duration with a log-normal distribution with parameters μ and σ chosen to ensure that the mean of the distribution is $(x + y)/2$, and three standard deviations captures

the entire range $[x, y]$ [20]. Starting with a non-DC STNU guarantees that the initial STNU candidate generated by `genApproxSTNU` would not be DC and, hence, would require multiple iterations to find an approximating STNU that was DC. However, because some non-DC STNUs have negative cycles comprising only ordinary edges and, hence, cannot be made DC by restricting their contingent ranges, only the PSTNs for which DC approximating STNUs can be created were kept.

Introducing Interdependent Simple Temporal Networks with Uncertainty for Multi-Agent Temporal Planning

Ajdin Sumic ✉

Technological University of Tarbes, France

Thierry Vidal ✉ 

Technological University of Tarbes, France

Andrea Micheli ✉ 

Fundazione Bruno Kessler, Trento, Italy

Alessandro Cimatti ✉ 

Fundazione Bruno Kessler, Trento, Italy

Abstract

Simple Temporal Networks with Uncertainty are a powerful and widely used formalism for representing and reasoning over convex temporal constraints in the presence of uncertainty called *contingent* constraints. Since their introduction, they have been used in planning and scheduling applications to model situations where the scheduling agent does not control some activity durations or event timings. What needs to be checked is then the *controllability* of the network, i.e., that there is a valid execution strategy whatever the values of the contingents. This paper considers a new type of multi-agent extension, where, as opposed to previous works, each agent manages its own separate STNU, and the control over activity durations is *shared* among the agents: what is called here a *contract* is a mutual constraint controllable for some agent and contingent for others. We will propose a semantically enriched version of STNUs that will be composed into a global *Multi-agent Interdependent STNUs* model. Then, controllability issues will be revisited, and we will focus on the repair problem, i.e., how to regain failed controllability by shrinking some of the shared contract durations, here in a centralized manner.

2012 ACM Subject Classification Computing methodologies

Keywords and phrases Temporal constraints satisfaction, uncertainty, STNU, Controllability checking, Explainable inconsistency, Multi-agent planning

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.13

1 Introduction

In many domains, such as planning and scheduling, systems diagnosis and control, etc, one needs to explicitly represent activities that may or must not overlap in time, last over some duration, and synchronize with timestamped expected events [4, 16, 15, 1]. The most commonly used model is Temporal Constraint Networks (TCN) [5]: nodes are time-points, and edges express sets of possible durations relating them. A key issue is the ability to check the temporal satisfiability of the plan/system/process through the *consistency* of the TCN. The simplest class of TCN, called the Simple Temporal Network (STN), arises when they have only binary constraints with convex intervals of values. Consistency checking is made through polynomial-time propagation algorithms.

A well-known extension of STNs that handles uncertainties, called STNU (Simple Temporal Network with Uncertainty), has been proposed by [17]. An STNU contains uncertain (*contingent*) durations between time-points, which means the effective duration is not under the control of the agent executing the plan, which is useful for addressing realistic dynamic and stochastic domains.



© Ajdin Sumic, Thierry Vidal, Andrea Micheli, and Alessandro Cimatti;
licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 13; pp. 13:1–13:14

Leibniz International Proceedings in Informatics



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

In STNUs, temporal consistency has been redefined as *controllability*: an STNU is controllable if a strategy exists for executing the schedule, whatever values the contingent durations take. In [17], the authors introduce three levels of controllability: *Weak Controllability* (WC), *Dynamic Controllability* (DC), and *Strong Controllability* (SC). These levels depend on how and when the uncertainties are resolved, i.e., the actual durations are observed/known. Different checking approaches have been proposed, widely discussed, and improved [17, 11, 9].

Considering Multi-agent agents interacting in a common environment, each with its own set of temporal activities, have been studied but only for multiple STNs [7] or for a global multi-agent STNU, all agents considering the same kind of contingent durations set by Nature [3] which hence cannot be modified in any way.

But there has been no work addressing the case where temporal coordination is needed due to the uncertainty for one agent coming from decisions made by another one: the duration of a shared activity is controllable (hence *flexible*) for say agent A_1 and contingent (only observed) for agent A_2 .

After exposing the most relevant related work in section 2, the paper focuses on our main contributions: (1) it revisits in section 3 the STNU model by formally defining the execution and observation semantics; (2) it proposes in section 4 a new global model called *Multi-agent Interdependent STNUs* (MISTNU), in which one can represent activity durations whose status differs among distinct agents, and it characterizes the three levels of controllability at the MISTNU level. Last, section 5 focuses on the *repair* problem, i.e., how to tighten the negotiable contingent constraints to recover controllability when checking has failed, proposing a first approach and experiments through a SMT-based encoding to synthesize valid repairs for Weak Controllability, before concluding our contributions with a few prospects.

2 Related work

In the literature, some works have tackled the problem of Multi-agent Simple Temporal Networks (MaSTN) in a centralized manner by decoupling a global STN into sub-networks, to distribute the control of a temporal plan among a group of agents in real-time execution scenarios [7, 8]. A fully distributed approach with the notion of privacy between agents is given in [2]. Still, this approach is incomplete in the sense that agents must agree in advance on some fixed durations, which prevents more dynamic solutions from being found. STNUs have received less attention in multi-agent settings, except for the MaSTNU model [3], which proposes a centralized approach to manage a multi-agent plan. This work decouples an STNU into sub-networks, ensuring all of them are dynamically controllable, using a Mixed Integer Linear Programming approach. A more decentralized approach that assumes a central agent that generates candidate decoupling solutions using the limited shared information of the agents is proposed in [18]. The authors ensure each agent independently tests the candidate on its local network and reports conflicts to guide the decoupling solution generation process until all networks are DC. However, the proposed decoupling algorithm may still prune some solutions. Nevertheless, both assume exogenous contingent constraints, i.e., uncertainties coming from outside the system and contingent for everyone.

To the best of our knowledge, no work in the literature has tackled the problem of a multi-agent system with non-exogenous durations (contingents because they are controlled by another agent). Contrary to previous approaches, this problem requires redefining and extending the expressiveness of STNUs. It also requires, as it will be explained, checking if the networks provided at the planning step are temporally well-formed, another issue that was not considered in previous works.

3 Definitions with Execution and Observation semantics

3.1 A revisited model for the single agent case

A Simple Temporal Network (*STN*) is a pair, (V, E) , where V is a set of time-points v_i representing event occurrence times, and E a set of temporal constraints between these time-points, in the form of convex intervals of possible durations [5], in the form $v_j - v_i \in [l_{ij}, u_{ij}]$, with lower bounds $l_{ij} \in \mathbb{R} \cup \{-\infty\}$ and upper bounds $u_{ij} \in \mathbb{R} \cup \{+\infty\}$. Interestingly enough, this model encompasses the qualitative precedence constraint, since v_i *precedes* v_j , noted $v_i \leq v_j$, iff $l_{ij} \geq 0$. A reference time-point v_0 is usually added to V , which is the “origin of time”, depending on the application (might be, e.g., the current day at 0:00). The goal is to assign values to time-points such that all constraints are satisfied, i.e., to assign a value to each constraint in its interval domain.

An *STN* with Uncertainty (*STNU*) is an extension in which one distinguishes a subset of constraints whose values are parameters that cannot be assigned but will be observed [17].

As for the global planning/execution framework, we first recall that for a single agent, one usually reasons upon two phases: plan generation and execution. Considering specific constraints (resources, time, uncertainties) often requires an additional constraint satisfaction phase to validate the generated plan. Here, we focus on the problem of checking the satisfiability of a plan under temporal uncertainty. So we start the definition of our framework with a *planning*, a *validation*, and an *execution* phase.

► **Definition 1** (STNU). *An STNU is a tuple (V, E, C) such that:*

- V is a set of time-points $\{v_0, v_1, \dots, v_n\}$, partitioned into controllable (V_c) and uncontrollable (V_u) with v_0 the reference time-point such that $\forall i, v_0 \leq v_i, v_0 \in V_c$;
- E is a set of requirement constraints $\{e_1, \dots, e_{|E|}\}$, where each e_k relates two time-points $e_k = v_j - v_i \in [l_{ij}, u_{ij}]$ with, $v_i, v_j \in V$.
- C is a set of contingent constraints $\{c_1, \dots, c_{|C|}\}$, where each c_k relates two time-points $c_k = v_j - v_i \in [l_{ij}, u_{ij}]$ with, $v_i \in V_c, v_j \in V_u$, and we have $v_i \leq v_j$ ¹. We will denote $\text{end}(c_k)$ as v_j .

► **Definition 2** (Schedule). *A **schedule** δ of an STNU $\mathcal{X} = (V, E, C)$ is a mapping δ from all the controllable time-points to real values where: $\delta = \{\delta(v_1), \dots, \delta(v_{|V_c|})\}$ with $\forall i, v_i \in V_c, \delta : v_i \rightarrow \mathbb{R}$*

► **Definition 3** (Situation and Projection). *Given an STNU $\mathcal{X} = (V, E, C)$, the **situations** of \mathcal{X} is a set of tuples Ω defined as the cartesian product of contingent domains:*

$$\Omega = \prod_{c \in C} [l_c, u_c]$$

A **situation** $\omega \in \Omega$ is composed of values noted $\omega_k \in [l_{ij}, u_{ij}]$ for each $c_k = [l_{ij}, u_{ij}] \in C$. A **projection** $\mathcal{X}_\omega = (V, E \cup C_\omega)$ of \mathcal{X} is an *STN* where C_ω is obtained by replacing each c_k in C by $c'_k = v_j - v_i \in [\omega_k, \omega_k]$. A schedule δ_ω is a **solution** of \mathcal{X}_ω if it satisfies all the constraints in \mathcal{X}_ω .

¹ Since here contingent durations are semantically linked to activities owned by some agent, with a start and end time-points, it is not possible to have a contingent duration between two unordered time-points.

13:4 Introducing MISTNU for Multi-Agent Temporal Planning

Intuitively, the set of situations defines the space of uncertainty, i.e., the possible values of contingent constraints; a projection substitutes all contingent links with a singleton, forcing its duration to the value appearing in ω . Now, a network shall be deemed controllable if it is possible to schedule the controllable time points to satisfy all requirement constraints in any possible projection. However, that depends on how and when the contingent durations are observed/known by the execution supervisor. Then, to reach a semantically sound definition of the controllability properties, we need to express not only at which time a controllable time-point (resp. a contingent duration) is executed by the agent (resp. set by the owner) but also at which time that value is decided (resp. observed/known) by the execution controller in charge of the agent plan execution.

► **Definition 4** (Decisions and Observations). $\forall v_i \in V_c$, $dec(v_i)$ is the time-point at which $\delta(v_i)$ is **decided** by the execution controller.

$\forall \omega_k \in C$, $obs(\omega_k)$ is the time-point at which ω_k is **observed** by the execution controller.

► **Definition 5** (Weak Controllability (WC)). An STNU \mathcal{X} is **weakly controllable** iff $\forall \omega \in \Omega$, $\exists \delta_\omega$ such that δ_ω is a solution of \mathcal{X}_ω .

Execution semantics: $\forall \omega_k \in \omega$, $obs(\omega_k) = v_0$, and the decision policy is free: $\forall v_i \in V_c$, $dec(v_i) \leq v_i$

► **Definition 6** (Strong Controllability (SC)). An STNU \mathcal{X} is **strongly controllable** iff $\exists \delta$ such that $\forall \omega \in \Omega$, δ is a solution of \mathcal{X}_ω .

Execution semantics: $\forall v_i \in V_c$, $dec(v_i) = v_0$, and the observations are free: possibly no observation ($\forall \omega_k \in \omega$, $obs(\omega_k) = \emptyset$) or observations during execution that will just update the bounds of the constraints in the network.

In other words, WC assumes that values of contingent durations will be known **after** plan *validation*, but **before** the *execution* starts. Without any loss of generality, we will consider that all values are set at once exactly at the beginning of the execution: we call this process the *initialization* phase. Then, the schedule can be assigned at the beginning (fixed schedule) or during execution (flexible schedule). For SC, values of contingent durations may be known (or not) at any time since one demands a fixed schedule, which must be set before execution starts, for instance, because users or other agents need to know the precise timing in advance. So, that schedule must be *conformant* to any possible contingent values. Thus, the *initialization* phase will be devoted to schedule assignment.

► **Definition 7** (Dynamic Controllability (DC)). An STNU \mathcal{X} is **Dynamically controllable** iff it is Weakly controllable and $\forall v_i \in V_c$, $\forall \omega, \omega' \in \Omega$, $\omega^{\leq v_i} = \omega'^{\leq v_i} \implies \delta_\omega(v_i) = \delta_{\omega'}(v_i)$ where $\omega^{\leq v} = \{\omega_k \in \omega \text{ s.t. } obs(\omega_k) \leq dec(v)\}$ is the part of the situation ω in which contingent constraints values are observed before executing v .

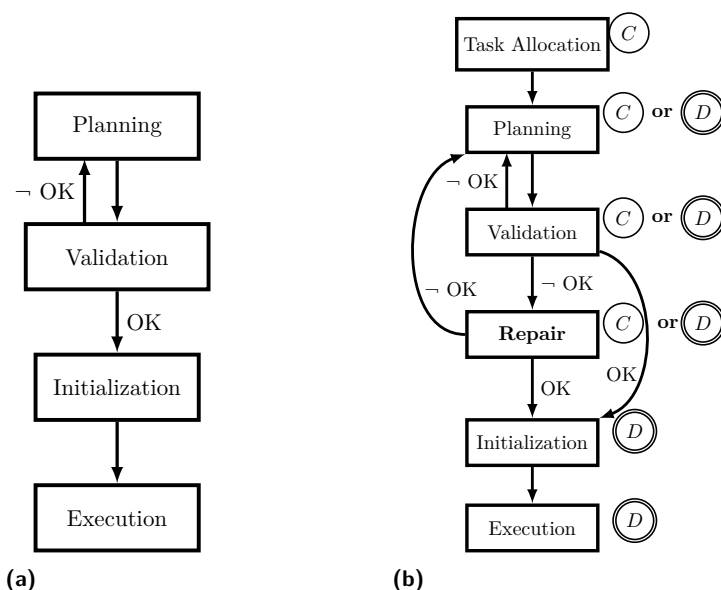
Execution semantics: $\forall \omega_k \in \omega$, $obs(\omega_k) = end(c_k)$, and $\forall v_i \in V_c$, $dec(v_i) = v_i$

In other words, DC assumes that values of contingent durations will be known **during** *execution*, and exactly at the time of occurrence of the ending time-point of the contingent constraint. The schedule is also assigned during execution (flexible schedule), deciding the time of activation of some activity only when all preceding time-points have occurred. Hence, there is no *initialization* phase.

► **Example.** A medical vehicle must visit several villages to offer free COVID testing to the population. The number of people to show off and, hence, the duration of the stay in each village is uncertain. A valid flexible strategy needs to be designed and checked in advance anyway (planning and validation), knowing that the precise information will be known and

sent to the agent by all the villages one hour before the route begins (initialization). Thus, the agent will know exactly the durations its activities will take and can start executing the plan, which implies WC. If the agent cannot know the number of people waiting in each village in advance, the duration of their testing activities will be known only when the agent arrives, which implies DC. Let's suppose now a rigid valid strategy is required, with village visiting times fixed in advance (hence at the initialization phase at the latest) and no prior knowledge of the number of people in each village. Then SC must be satisfied.

As a matter of conclusion for the single agent context, we have designed a general framework to deal with temporal uncertainty in planning, including 4 phases: *planning*, *validation*, *initialization*, and *execution*. There is only one possible backtrack: if the validation phase fails (controllability checking fails), the only thing to do is backtrack to the planning engine to find an alternative plan. We show these steps in Figure 1a.



■ **Figure 1** shows the global framework for a single agent (1a) and multiple agents (1b). Node C and double-circled node D refer to the possibility of that step being either centralized (C) or distributed (D). Please note that the initialization phase only exists for WC and SC as DC implies to decide/observe during execution.

3.2 A new temporal multi-agent framework

For a single agent, the contingent duration assignment is exogenous; hence, it is assumed that there is no way to influence them. Saying that “Nature” will assign those values is a usual way to capture that.

However, in a multi-agent environment, a contingent duration may be decided by another agent. So even though the agent that depends on this contingent cannot decide its value, it might be possible to communicate with the *owner* agent to change the possible values. Intuitively, that *owner* agent will decide the duration but commits to assign it within the lower and upper bounds. Some other *compliant* agents depend on that constraint, which is contingent on their network. The former agent (*owner*) communicates its commitment (lower and upper bound) to the compliant agents at some point before the agents check the

13:6 Introducing MISTNU for Multi-Agent Temporal Planning

controllability of their networks. We call this kind of commitment a *contract* between the owner and the compliant agents, where a compliant agent has the right to request new bounds as long as it guarantees the satisfaction of both agents, i.e., to ensure each is controllable.

First of all, to get a complete picture, we must recall that in multi-agent planning, there might be a first phase of *task allocation* to distribute the goals to achieve to the agents. That phase is usually centralized or devoted to specialized agents. Then the *planning* may be centralized, the global plan being decoupled into separate agent plans or distributed, each agent building its own plan individually. In both cases anyway, dependencies and synchronizations must be considered, calling for some way to share activities controlled by one agent but which outcome is needed by another.

In the end, it is assumed that *execution* will be launched concurrently by all the agents. But before that, after the planning is completed, there is still the need to *validate* the individual plans through, in our case, temporal controllability checking algorithms. Once again, it can be done by a central agent having a view of all the plans or in a distributed way by each agent.

The way that can be done then depends on the observation and decision semantics that have been introduced in the previous section. First, if the application requires that all schedules must be fixed in advance, that means one needs to consider a common *initialization* phase to fix those schedules, which means all agents must ensure SC. Second, if flexible schedules are allowed, then WC or DC apply. The difference depends on how and when the “owner” agents set and communicate the values of activity durations on which other agents are dependent. If that is done before the execution, they must consider a common *initialization* phase when all agents will decide and exchange the shared activities durations, which is consistent with the definition of WC. If such decisions are to be taken during the execution and communicated as soon as they occur and with no delay, then DC applies.

► **Example.** *In a hospital environment, a patient has to follow a path through several services that manage their timetabling separately. The path has to satisfy partially ordered constraints between the different services. Now consider that the durations of activities in each service depend, for instance, on the patient features that will only be assessed at the time each activity begins. Then, if all services require a rigid timetable where each operation has a unique starting time that is fixed in advance and appears in the calendar, then SC applies for all; if flexibility is allowed in the sense that operations start times might be decided on the fly, then all services must account for some global DC. Now, consider that each service does not know in advance how many people will be working that day (due to last-minute staff allocation and potential sick leaves), which affects the duration of the patient care. In that case, a plan must be proven valid in advance without such information, but all services will know and exchange them through a common initialization phase when the day starts, which implies WC.*

Of course, this framework is only relevant in homogeneous multi-agent problems when all agents have the same behavior, which we assume here. If not, the classical controllability checking algorithms will not be applicable, which is not our focus. For instance, when agents aim to satisfy different levels of controllability, it results in sharing the decisions of the contracts at different times (before, online, or never). This semantic is different from the one classical controllability checking algorithms assume, as the uncontrollable duration will either be known before execution (WC), online (DC), or never known (SC). This framework also assumes that the initialization phase is synchronous, i.e., all decisions must be taken by all agents at the same time, without communication nor hierarchy between them; otherwise, the semantics of WC (or SC) will not be met, and what needs to be checked will also be

something different that is out of the scope of our current study. Hence, extending the well-defined semantics of WC, SC, and somehow DC to a multi-agent setting aligns with specific and somehow restricted semantics of the behavior of the team of agents.

Then, going back to the *validation* phase, if at least one agent is not controllable, then it is still possible to backtrack to the planning phase to find an alternative. Still, it is possible to negotiate with other agents to change the values of some contracts they control. If the “owner” agent agrees to change the bounds of the contract controls so that both agents are now controllable, the problem is solved without needing a more complex replanning stage. This new phase, either centralized or distributed, is the *repair* phase and may require controllability checking algorithms capable of diagnosing the source of uncontrollability [10]. This new phase may also succeed or fail (no solution exists) with no other choice but to backtrack to the planning phase. Figure 1b synthesizes this new global framework.

4 The MISTNU model

4.1 Definitions

The concept of negotiable contingent constraints arises in a multi-agent context when such a constraint is not controlled by Nature but by one agent of the system. Hence, we slightly modify the definition of an STNU in the form of a *Contracting STNU* (cSTNU) by explicitly considering some constraints as *owned* by the agent and relating the contingent constraints to so-called *contracts*, the bounds and the owner of such contracts being now defined outside the model, to be shared by several agents².

► **Definition 8** (cSTNU). *A Contracting STNU (cSTNU) is an STNU where links representing contracts are labeled. A cSTNU is a tuple $\mathcal{S} = \langle V, R, W, E, C, O \rangle$ such that:*

- *V is a set of time points, partitioned into V_c and V_u (Definition 1)*
- *R and W are sets of contracts owned (W) and observed (R), such that $R \cap W = \emptyset$*
- *E is a set of requirement links of the form $v_i \xrightarrow{[l,u]} v_j$;*
- *C is a set of labeled contingent links of the form $v_i \xrightarrow{-p} v_j$ where $p \in R$.*
- *O is a set of owned contract links of the form either $v_i \xrightarrow{p} v_j$ or $v_i \xrightarrow{-p} v_j$, one for each contract $p \in W$.*

In addition, we require that $\forall v_j \in V_u$, there exists a unique labeled contingent link of the form $v_i \xrightarrow{-p} v_j$ in $C \cup O$.

One can notice that a contract is a labeled link where an agent may consider its owned contracts (in W) as contingent or requirement constraints, depending on its policy (represented by O). Forcing that constraint to be contingent prevents the agent from shrinking it when running some local propagation algorithm to respect its commitment to others or to retain the contract’s maximal flexibility at execution time (i.e., it refuses any reduction to allow other agents to regain control). That shall enable us to tune our global model accordingly in settings where agents are more or less selfish or cooperative. An expressiveness is not allowed by the STNU model. In addition, in a multi-agent scenario, it’s important to note that requirement constraints in E are private.

Then, in Definition 9, we define the model of Multi-agent Interdependent Simple Temporal Networks under Uncertainty (MISTNU).

² As already mentioned, an exogenous contingent constraint can still be modeled here, being related to a contract without any owner; which will be allowed in the global model.

13:8 Introducing MISTNU for Multi-Agent Temporal Planning

- **Definition 9** (MISTNU). A MISTNU is a tuple $\mathcal{G} = \langle A, \Sigma, B \rangle$ such that:
- A is a set of agents $\{a_1, a_2, \dots, a_n\}$;
 - Σ is a set of cSTNUs $\mathcal{S}_a = \langle V_a, R_a, W_a, E_a, C_a, O_a \rangle$, one for each $a \in A$, such that
 - $\forall a \in A, v_z \in V_a$, where v_z is the mutual reference time point: $\forall v_i \in V_a, v_z \leq v_i$;
 - for every pair of agents $a, b \in A$, $W_a \cap W_b = \emptyset$
 - B is a map from contracts to bounds $B : \bigcup_{a \in A} (R_a \cup W_a) \rightarrow \mathbb{R}^2$.

A MISTNU is a model \mathcal{G} composed of a set of agents, where each agent has its own cSTNU and might own or read contracts; some of them are shared and might be negotiated. Then, \mathcal{G} is also composed of a map of contracts to bounds B that indicates for every contract p its lower/upper bound denoted as a pair $\langle l_p, u_p \rangle$ with l_p and u_p respectively the lower and upper bound of the interval of possible durations. This allows us to reduce a cSTNU into an STNU by applying B :

- **Definition 10** (cSTNU reduction). Given a cSTNU $\mathcal{S} = \langle V, R, W, E, C, O \rangle$ and a map $B : W \cup R \rightarrow \mathbb{R}^2$ giving bounds to contracts, \mathcal{S} can be reduced to an STNU $\mathcal{S}^{\mathcal{G}} = \langle V, E', C' \rangle$ with:
- $E' = E \cup \{v_i \xrightarrow{[l_p, u_p]} v_j \mid v_i \xrightarrow{p} v_j \in O, B(p) = \langle l_p, u_p \rangle\}$
 - $C' = \{v_i \xrightarrow{[l_p, u_p]} v_j \mid v_i \xrightarrow{p} v_j \in C \cup O, B(p) = \langle l_p, u_p \rangle\}$

As a cSTNU can be reduced to an STNU with Definition 10, the definitions of its situations and projections directly come from Definition 3. However, for the global MISTNU model, things are a little more complex. We hence first provide further definitions:

- for any agent a , $P_a = R_a \cup W_a$ is the set of all its contracts;
- $P = \bigcup_a P_a$ gathers the contracts of all the agents; $W = \bigcup_a W_a$ the ones having owners.
- for any cSTNU \mathcal{S} , $\sigma(\mathcal{S}, p) = v_i$ s.t. $\exists v_j, v_i \xrightarrow{p} v_j \in C \cup O$ or $v_i \xrightarrow{p} v_j \in O$ denotes the starting time point of contract p in \mathcal{S} .

- **Definition 11** (MISTNU situation). Given a MISTNU $\mathcal{G} = \langle A, \Sigma, B \rangle$, the *situations* of \mathcal{G} is a set of tuples of reals $\Omega_{\mathcal{G}}$ defined as the cartesian product of:

$$\Omega_{\mathcal{G}} = \times_{p \in P} [l_p, u_p].$$

A *situation* is an element ω of $\Omega_{\mathcal{G}}$ and we write $\omega(p)$ with $p \in P$ to indicate the element in ω associated with p in the cross product.

- **Definition 12** (MISTNU projection). Given a MISTNU $\mathcal{G} = \langle A, \Sigma, B \rangle$, and a situation ω , the *projection* \mathcal{G}^{ω} is a model $\langle A, \Sigma^{\omega}, B^{\omega} \rangle$ where:

- B^{ω} is a map from contracts to fixed values $B^{\omega} : P \rightarrow \mathbb{R}^2$ such that $B^{\omega} = \{\langle \omega(p), \omega(p) \rangle \mid p \in B\}$
- Σ^{ω} is a set of STNs $\mathcal{X}_a^{\omega} = \langle V_a, E'_a \rangle$, one per $a \in A$, s.t for $\mathcal{S}_a = \langle V_a, R_a, W_a, E_a, C_a, O_a \rangle$ in Σ :

$$E'_a = E_a \cup \{v_i \xrightarrow{B^{\omega}(p)} v_j \mid v_i \xrightarrow{p} v_j \in O_a\} \cup \{v_i \xrightarrow{B^{\omega}(p)} v_j \mid v_i \xrightarrow{p} v_j \in C_a \cup O_a\}$$

It is important to point out that as the system considers temporal networks created independently, the model must ensure that all the temporal networks in Σ are temporally well-formed. This means that for any contract of the form $v_i \xrightarrow{p} v_j$ or $v_i \xrightarrow{p} v_j$ shared among a set of agents, the date in time on which its execution starts (v_i) and finishes (v_j) must be the same in each of the temporal networks where the contract is involved. As the contract duration between v_i and v_j is guaranteed to be the same by B , we need to ensure that it is also the case for the start time-point v_i .

► **Definition 13** (Temporally well-formed). A MISTNU $\mathcal{G} = \langle A, \Sigma, B \rangle$ is **temporally well-formed** if for every projection $\omega \in \Omega_{\mathcal{G}}$, for every pair of distinct agents a_1 and a_2 and for every contract $p \in P_1 \cap P_2$, all solutions δ_1 of \mathcal{X}_1^ω and δ_2 of \mathcal{X}_2^ω are such that $\delta_1(\sigma(\mathcal{S}_1, p)) = \delta_2(\sigma(\mathcal{S}_2, p))$.

► **Theorem 14.** Let T be a map from a contract p to its unique predecessor p' or v_z , $T : P \rightarrow P \cup \{v_z\}$, such that $\forall p$, there is no sequence of the form $T(T(\dots T(p))) = p$.

A MISTNU is well-formed if for every agent $a \in A$, $\forall p \in (W_a \cup R_a)$ we have:

- $T(p) = v_z \iff v_z \xrightarrow{p} v_j \in O_a \text{ or } v_z \xrightarrow{-p} v_j \in O_a \cup C_a$
- $T(p) = p' \iff (v_i \xrightarrow{p} v_j \in O_a \text{ or } v_i \xrightarrow{-p} v_j \in O_a \cup C_a) \wedge (v_k \xrightarrow{p'} v_i \in O_a \text{ or } v_k \xrightarrow{-p'} v_i \in O_a \cup C_a)$

Proof. Let's suppose, for the sake of contradiction, that map T exists and follows the constraints of the theorem. Still, the MISTNU \mathcal{G} is not well-formed, then there exists a projection ω , a pair of agents a_1 and a_2 and a contract p such that $\delta_1(\sigma(\mathcal{S}_1, p)) \neq \delta_2(\sigma(\mathcal{S}_2, p))$ as per Definition 13. But we observed from T that $T(p) = v_z$ or $T(p) = p'$. Therefore, by induction over T , we have:

- the base case where $T(p) = v_z$: obviously we have $\delta_1(\sigma(\mathcal{S}_1, p)) = \delta_2(\sigma(\mathcal{S}_2, p)) = 0$ as v_z is the common reference time-point shared among all agents;
- the inductive case where $T(p) = p'$: we assume that $\delta_1(\sigma(\mathcal{S}_1, p')) = \delta_2(\sigma(\mathcal{S}_2, p')) = k$, then $\delta_1(\sigma(\mathcal{S}_1, p)) = \delta_2(\sigma(\mathcal{S}_2, p)) = k + \omega_{p'}$ because in both agents p is started by the end of p' and p' has the same duration for all agents.

Consequently, it's impossible, from the induction, to have $\delta_1(\sigma(\mathcal{S}_1, p)) \neq \delta_2(\sigma(\mathcal{S}_2, p))$ if \mathcal{G} follows T , and hence \mathcal{G} is guaranteed to be well-formed. ◀

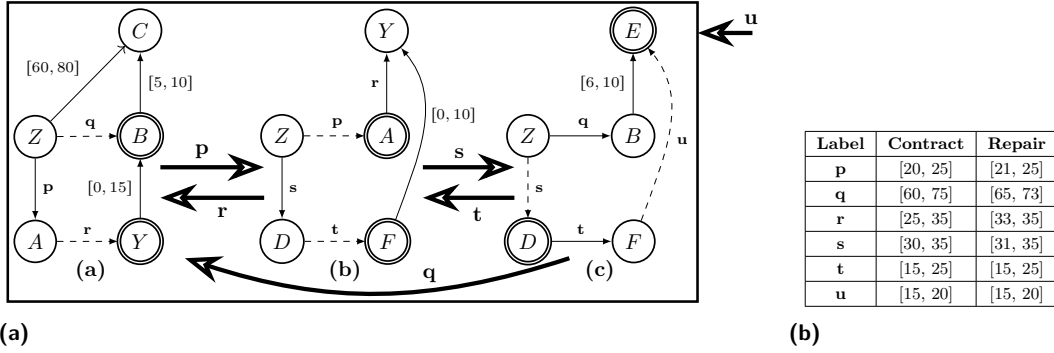
However, Theorem 14 provides a sufficient but not necessary condition for a MISTNU to be well-formed: one could simply require agents to agree on the starting date for a contract or a fixed duration between two contracts. Figure 2 shows an example of a MISTNU with three agents and their networks ensured through the map T to be temporally well-formed with respect to the contracts.

In this example, the contract u comes from an *external* agent, i.e., an agent with which **negotiation is not possible**, which can allow to express several semantic situations: a lack of communication with this agent during the repair, the agent being selfish, etc. In the model, we simply represent it by the contract having no owner, and we must ensure that such contract bounds cannot be shrunk. Nonetheless, such agents shall conform to the semantics and behave in the same way as others: e.g., if WC is considered, that means the duration of the contract u will be shared with its readers during the initialization phase. One can notice that a MISTNU with only one agent is equivalent to a single STNU, with all contracts read by this agent coming from outside the system (not negotiable).

4.2 Controllability

In previous work, the controllability of MaSTNU was defined as having all STNUs controllable [3, 18]. Thus, we define the controllability of MISTNU in the same manner, i.e., to be Dynamically controllable, the system would impose all the networks (cSTNU) to be Dynamically controllable. Then, with Definition 10, we check the controllability of a cSTNU through an STNU reduction, and checking the controllability of STNUs has already been tackled in the literature [12, 9].

13:10 Introducing MISTNU for Multi-Agent Temporal Planning



■ **Figure 2** Example of a MISTNU with agents a, b, and c and their networks. Nodes are time-points, uncontrollable ones being double circled, solid/dashed edges are requirement/contingent constraints. The contracts are: p owned by a, r, and s by b, q, t by c, and u by an external agent; communication is represented through larger edges, e.g., a sends p to b. In (b), we show the bounds and repairs of the contracts.

► **Definition 15** (Controllability). *Given a MISTNU $\mathcal{G} = \langle A, \Sigma, B \rangle$, we define the τ -controllability L_τ of \mathcal{G} with $\tau = \{Weak, Dynamic, Strong\}$ as:*

$$L_\tau \equiv \forall S_a \in \Sigma, S_a^{\mathcal{G}} \text{ is } \tau\text{-controllable.}$$

where $S_a^{\mathcal{G}}$ is the STNU obtained from S_a by the cSTNU reduction of Definition 10.

In Section 3.2, we argue that WC in a multi-agent system considers a common initialization phase where all agents will decide and exchange the shared activities durations just before execution. The fact that each agent independently decides the duration of the contracts it owns means that it must ensure that this duration is consistent with the choice over the contract's duration owned by the other agents. In other words, whatever the contract duration an agent decides, it must ensure that there always exists a consistent schedule with this duration, whatever the duration of the contract decided by others it receives, which is related to the definition of Weak controllability. Therefore, the semantics of Weak controllability is imposed to guarantee all possible combinations between the contracts owned by an agent and the ones it does not own to form a consistent STN. Hence, for the validation phase (controllability checking), all the contracts must be considered as contingent constraints to guarantee WC is well-checked. This is not the case for the other two types of controllability: DC implies the agent will decide after observing the decisions of the other agents, and SC supposes the agent will fix a schedule before receiving any information about the contracts owned by the others. Thus, it does not require all the owned contracts to be considered contingents for the validation phase.

5 Defining and solving the repair problem

5.1 The repair problem: definitions

The concept of repair for MISTNU arises when the system is not controllable. We focus on local controllability by finding a tightening (if it exists) of the bounds of an agent's contracts so that local controllability is recovered. In the following, we formalize the repair problem.

► **Definition 16 (Repair).** Given a global model $\mathcal{G} = (A, \Sigma, B)$ such that for some agent $a \in A$, \mathcal{S}_a is not τ -controllable with $\tau = \{\text{Weak, Dynamic, Strong}\}$. The **L_τ -repair problem** consists in finding new bounds B' for a global model $\mathcal{G}' = (A, \Sigma, B')$ such that:

- $\forall p \in W$ ³ let $\langle l_p, u_p \rangle = B(p)$ and $\langle l'_p, u'_p \rangle = B'(p)$ where $l'_p \geq l_p$, $u'_p \leq u_p$;
- \mathcal{G}' is L_τ -controllable.

In addition, we are interested in repair solutions that minimize the reduction in the size of the contract bounds, as done in [14] for DTNU and STNU. This intuitively corresponds to minimizing the amount of flexibility removed for each agent concerned by the repair. A unique solution might not exist for this optimal repair, and some general policies might also require fairness in finding an optimal solution, i.e., an optimal repair that equally shrinks the contracts among the agents. In the following, we define the optimal repair and the fair-optimal repair, the latter equally reducing as many contracts as possible by considering the reduction percentage. Somehow, this amounts to finding some optimal equity between the agents. For that, we write $C_{|P|}^2$ as the number of distinct pairs $\langle p_1, p_2 \rangle$ in W .

► **Definition 17 (Optimal Repair).** Let $\mathcal{G} = (A, \Sigma, B)$, be a non L_τ controllable MISTNU and let $R_{\mathcal{G}}$ be the set of all the solutions to the L_τ -repair problem for \mathcal{G} . An **optimal L_τ -repair** for \mathcal{G} is defined as:

$$\operatorname{argmin}_{\mathcal{G}' \in R_{\mathcal{G}}} \left(\sum_{p \rightarrow \langle l'_p, u'_p \rangle \in B'} ((l'_p - l_p) + (u_p - u'_p)) \mid \langle l_p, u_p \rangle = B(p) \right)$$

► **Definition 18 (Fair-Optimal Repair).** Let $\mathcal{G} = (A, \Sigma, B)$, be a non L_τ -controllable MISTNU and let $R_{\mathcal{G}}^{\text{opt}}$ be the set of all the solutions to the optimal L_τ -repair problem for \mathcal{G} . A **fair-optimal L_τ -repair** for \mathcal{G} is defined as:

$$\operatorname{argmax}_{\mathcal{G}' \in R_{\mathcal{G}}^{\text{opt}}} \left(\left| \left\{ \langle p_1, p_2 \rangle \in C_{|P|}^2 \mid \frac{((l'_{p_1} - l_{p_1}) + (u_{p_1} - u'_{p_1}))}{u_{p_1} - l_{p_1}} = \frac{((l'_{p_2} - l_{p_2}) + (u_{p_2} - u'_{p_2}))}{u_{p_2} - l_{p_2}} \right\} \right| \right)$$

Intuitively, we aim at maximizing the number of pairs of contracts that are shrunk by the same amount. This function selects *among the optimal repairs*, as per Definition 17, solutions in which the reduction of flexibility is shared as much as possible among the contracts. Table 2b shows the WC fair-optimal repair of the MISTNU of Figure 2 with the contracts s and p both being reduced to 20%.

5.2 Encoding of the repair

In this section, we present a centralized encoding for the WC-repair problem of MISTNU based on the encoding for the weak repair of classical STNUs presented in [14]. First, we remind the readers that WC implies all contracts to be contingents. Then, we exploit the convexity of the problem by considering that all combinations of the lower and upper bounds of the contingents are enough to check the controllability of an STNU [17]. This allows us to consider the situations where the duration of a contract is fixed to either its lower-bound or upper-bound for all contract readers. In addition, we define two rational variables for each contract, l^p and u^p , respectively, representing the lower and upper bound of the revised contract p . Please note that a variable is represented with the index as an exponent. Then, we formalize the basic components for the encoding as follows:

³ This ensures a contract from an external agent, which has no owner, shall not be shrunk (see definition of W in 4.1)

13:12 Introducing MISTNU for Multi-Agent Temporal Planning

- for each \mathcal{S}_a , we define \vec{X}_a as the set of variables $\{v^0, \dots, v^i\}$ representing the time-points in V_a (agents have disjoint sets of variables).
- $\vec{B}_l = \{l^1, \dots, l^i\}$ and $\vec{B}_u = \{u^1, \dots, u^i\}$ the two sets of variables for the lower and upper bound variables of all the contracts.

In addition, for each $\mathcal{S}_a = \langle V_a, R_a, W_a, E_a, C_a, O_a \rangle$, we have the set of projections β_a , one for each possible combination of bounds $\langle l_p, u_p \rangle$ for each contract p of the form $v_i - \overset{P}{\rightarrow} v_j$ in $C_a \cup O_a$:

$$\beta_a = \{\omega \mid \omega_p \in \{l_p, u_p\}, v_i - \overset{P}{\rightarrow} v_j \in C_a \cup O_a\}.$$

Then, as ω corresponds to a projection, equivalently an STN $\mathcal{X}_\omega = (V_a, E_a \cup C'_a)$, we encode a cSTNU as the conjunction of each projection ω . In addition, from Definition 5, a cSTNU is Weakly controllable if each projection ω has at least one schedule δ_ω . This requires the variables in \vec{X}_a to be unique per projection ω . In this particular case, we denote $\vec{X}_a^\omega = \{v_\omega^0, \dots, v_\omega^i\}$ the unique set of variables representing the time-points of the projection ω . Then, the MISTNU model can also be encoded as the conjunction of the encoding of each agent's cSTNU (\mathcal{S}_a) of the system and the encoding of the contracts' constraints. In the following, we present the encoding of a projection denoted as $\Upsilon_{wc}^{\mathcal{X}_\omega}(\vec{X}_a^\omega)$, the encoding of a cSTNU denoted as $\Upsilon_{wc}^{\mathcal{S}_a}$, the encoding of the contracts denoted $\Psi(\vec{B}_l, \vec{B}_u)$, and the encoding of a MISTNU denoted Υ_{wc} .

$$\Upsilon_{wc}^{\mathcal{X}_\omega}(\vec{X}_a^\omega) = \bigwedge_{t_i \in E'_a} \begin{cases} v_\omega^j - v_\omega^i \in [l^p, u^p] \text{ iff } t_i = v_i \xrightarrow{[\omega_p, \omega_p]} v_j \\ v_\omega^j - v_\omega^i \in [l, u] \text{ iff } t_i = v_i \xrightarrow{[l, u]} v_j \end{cases} \quad (1)$$

$$\Upsilon_{wc}^{\mathcal{S}_a} = \bigwedge_{\omega \in \beta_a} \Upsilon_{wc}^{\mathcal{X}_\omega}(\vec{X}_a^\omega) \quad (2)$$

$$\Psi(\vec{B}_l, \vec{B}_u) = \bigwedge_{p \rightarrow \langle l_p, u_p \rangle \in B} \begin{cases} \{l_p \leq l^p \leq u^p \leq u_p\} \text{ iff } \exists a \mid p \in W_a \\ \{l^p = l_p; u^p = u_p\} \text{ iff } \nexists a \mid p \in W_a \end{cases} \quad (3)$$

$$\Upsilon_{wc}(\vec{B}_l, \vec{B}_u) = \left(\bigwedge_{\mathcal{S}_a \in S} \Upsilon_{wc}^{\mathcal{S}_a} \right) \wedge \Psi(\vec{B}_l, \vec{B}_u) \quad (4)$$

Please note that Equation 3 also avoids shrinking contracts from external agents by fixing the variables l^p to l and u^p to u . Then, we solved the optimal WC-repair denoted $\chi_{wc}^{opt}(\vec{B}_l, \vec{B}_u)$ and the fair-optimal WC-repair χ_{wc}^{fair} with a lexicographic optimization process (multi-objective optimization supported by modern Optimization Modulo Theory Solvers [13]) with the optimal WC-repair optimization being the first one to be solved:

$$\chi_{wc}^{opt}(\vec{B}_l, \vec{B}_u) = \text{Minimize} \sum_{p \rightarrow \langle l_p, u_p \rangle \in B} ((l^p - l_p) + (u_p - u^p)) \text{ s.t. } \Upsilon_{wc}(\vec{B}_l, \vec{B}_u) \quad (5)$$

$$\chi_{wc}^{fair}(\vec{B}_l, \vec{B}_u) = \text{Maximize} \left(\left| \{ \langle \rho_1, \rho_2 \rangle \in C_{|P|}^2 \mid \rho_1 = \rho_2 \} \right| \right) \text{ s.t. } \chi_{wc}^{opt}(\vec{B}_l, \vec{B}_u) \quad (6)$$

where for each distinct pairs $\langle p_1, p_2 \rangle$ in P , we create the variables ρ_1 and ρ_2 such that $\forall p_k \in \{p_1, \dots, p_{|P|}\}$ we have:

$$\rho_k = \frac{((l^{p_k} - l_{p_k}) + (u_{p_k} - u^{p_k}))}{u_{p_k} - l_{p_k}}$$

5.3 Experiments

In this subsection, we simply show the effectiveness of the proposed approaches for WC repair and evaluate the experimental complexity. We implemented the encoding in Python using the pySMT framework [6]. For our experiments, we use the Z3 solver as the backend. We experimented on a large set of 400 MISTNU limited to four agents based on the T map of Theorem 14.

We randomly and safely generate MISTNUs following these parameters: the size of the networks growing from 10 to 200 nodes; the number of contracts (owned) per agent growing from 1 to 20 (according to the size of the network); the number of edges per network by setting at 3 the number of successors per *divergent* node (only v_z is allowed to have more successors), where a divergent node is a node with more than one successor. We considered the fair-optimal repair for the experiment and ran all the experiments on an Xeon E5-2620 2.10GHz with 3600s/10GB time/memory limits.

We solved more than 60 instances, which was expected as the number of projections (bounds) of all the networks grow exponentially (2^p) [17]. More precisely, the problem of checking Weak Controllability (WC) is expected to be co-NP-complete for a single network. Here, we are solving an optimization problem (repair with fairness), which is harder than the checking problem for multiple networks. This is why we did not solve many instances with a one-hour time limit. But that is only the first approach that calls for improvements.

6 Conclusion

This paper presents a new multi-agent model for temporal problems under uncertainty called MISTNU that considers independent networks where, for an agent network, the execution of some tasks might be controlled by other agents. Hence, an agent can negotiate the duration of such tasks. This paper formally defines the cSTNU model, which is an extension of the STNU model, the MISTNU model, and the problem of checking its controllability. In addition, the paper presents the repair problem for the MISTNU model by shrinking the contracts' bounds and proposes a repair encoding for Weak Controllability. Future work will focus on the repair problem for both SC and DC and on a more efficient one for WC. The proposed MISTNU model works well for homogeneous agents with distributed controllability checking, initialization, and execution. However, a heterogeneous system is more challenging as it implies that agents behave in different ways, which might result in a system with mixed controllability (since observation and execution semantics might be different among the agents). A similar study must be done to get a model capable of managing uncertainties from other agents and the classical uncertainty from the environment (Nature), which has its own semantics.

References

- 1 Arthur Bit-Monnot, Malik Ghallab, Félix Ingrand, and David E. Smith. FAPE: a constraint-based planner for generative and hierarchical temporal planning. *CoRR*, abs/2010.13121, 2020. [arXiv:2010.13121](https://arxiv.org/abs/2010.13121).
- 2 James C. Boerkoel and Edmund H. Durfee. Distributed reasoning for multiagent simple temporal problems. *J. Artif. Intell. Res.*, 47:95–156, 2013. [doi:10.1613/JAIR.3840](https://doi.org/10.1613/JAIR.3840).
- 3 Guillaume Casanova, Cédric Pralet, Charles Lesire, and Thierry Vidal. Solving dynamic controllability problem of multi-agent plans with uncertainty using mixed integer linear programming. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European*

- Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 930–938. IOS Press, 2016. doi:10.3233/978-1-61499-672-9-930.
- 4 Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. A constraint-based method for project scheduling with time windows. *J. Heuristics*, 8(1):109–136, 2002. doi:10.1023/A:1013617802515.
 - 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 Marco Gario and Andrea Micheli. pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop*, 2015.
 - 7 Luke Hunsberger. Algorithms for a temporal decoupling problem in multi-agent planning. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pages 468–475, 2002. URL: <http://www.aaai.org/Library/AAAI/2002/aaai02-071.php>.
 - 8 Luke Hunsberger. Distributing the control of a temporal network among multiple agents. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*, pages 899–906, 2003. doi:10.1145/860575.860720.
 - 9 Luke Hunsberger and Roberto Posenato. Speeding up the rule dynamic-controllability-checking algorithm for simple temporal networks with uncertainty. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 9776–9785. AAAI Press, 2022. doi:10.1609/AAAI.V36I9.21213.
 - 10 Josef Lubas, Marco Franceschetti, and Johann Eder. Resolving conflicts in process models with temporal constraints. In *Proceedings of the ER Forum and PhD Symposium*, 2022.
 - 11 Paul H Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *Aaai*, pages 1193–1198, 2005. URL: <http://www.aaai.org/Library/AAAI/2005/aaai05-189.php>.
 - 12 Roberto Posenato and Carlo Combi. Adding flexibility to uncertainty: Flexible simple temporal networks with uncertainty (ftnu). *Information Sciences*, 584:784–807, 2022. doi:10.1016/J.INS.2021.10.008.
 - 13 Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Trans. Comput. Log.*, 16(2):12:1–12:43, 2015. doi:10.1145/2699915.
 - 14 Ajdin Sumic, Alessandro Cimatti, Andrea Micheli, and Thierry Vidal. SMT-based repair of disjunctive temporal networks with uncertainty: Strong and weak controllability. In *Proceedings of the The 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2024)*, 2024.
 - 15 Alessandro Valentini, Andrea Micheli, and Alessandro Cimatti. Temporal planning with intermediate conditions and effects. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 9975–9982, 2020. doi:10.1609/AAAI.V34I06.6553.
 - 16 Gérard Verfaillie, Cédric Pralet, and Michel Lemaître. How to model planning and scheduling problems using constraint networks on timelines. *Knowl. Eng. Rev.*, 25(3):319–336, 2010. doi:10.1017/S0269888910000172.
 - 17 Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999. doi:10.1080/095281399146607.
 - 18 Yuening Zhang and Brian C. Williams. Privacy-preserving algorithm for decoupling of multi-agent plans with uncertainty. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*, pages 426–434. AAAI Press, 2021. URL: <https://ojs.aaai.org/index.php/ICAPS/article/view/15988>.

Learning Temporal Properties from Event Logs via Sequential Analysis

Francesco Chiariello   

IRIT, ANITI, University of Toulouse, France

Abstract

In this work, we present a novel approach to learning Linear Temporal Logic (LTL) formulae from event logs by leveraging statistical techniques from sequential analysis. In particular, we employ the Sequential Probability Ratio Test (SPRT), using Trace Alignment to quantify the discrepancy between a trace and a candidate LTL formula. We then test the proposed approach in a controlled experimental setting and highlight its advantages, which include robustness to noise and data efficiency.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Applied computing → Business process management

Keywords and phrases Process Mining, Declarative Process Discovery, Trace Alignment, Sequential Analysis

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.14

Funding This project has been funded by the French government as part of France 2030 (Grant agreement n°ANR-19-PI3A-0004) and by the European Union - Next Generation EU as part of the France Relance.

1 Introduction

Process Mining (PM) [36] is an interdisciplinary field at the intersection of Data Mining and Business Process Management (BPM) [40]. Its goal is to gain insight into operational processes by analyzing the associated event logs. An operational process, or process for short, corresponds to the series of activities an organization performs to accomplish its routine tasks, such as delivering a particular service or product. In enacting a process, an organization generates sequential data that an information system stores in the form of an event log. Therefore, an event log keeps track of all the activities performed during several task executions, also called traces. A fundamental problem of PM is the one of learning models of processes from event logs, also known as Process Discovery [35]. To achieve this, one may consider different formalisms to model the processes, including UML activity diagrams [16], Business Process Model and Notation [23], and Petri nets [34, 38]. Those formalisms are imperative in that they prescribe, at each step, the activities that can be performed and provide therefore an easy-to-follow recipe for process execution. However, they have known limitations due to the tendency of over-constraining the process [37] and the lack of interpretability. Declarative models [15], by contrast, consist of constraints over process executions. After that, every execution not violating such constraints is admitted. The most widespread declarative process specification language is Declare [28], which consists of a set of templates that allow to specify temporal constraints over the activities of the process. Declare semantics can be grounded into Linear Temporal Logic on process traces (LTL_p) [19], which allow us to exploit efficient automata-theoretic techniques [7]. For this reason, the use of LTL_p for the specification of processes is gaining increasing traction.

An obstacle to learning models of processes from event logs is that they often contain noise. Such noise may be caused by errors in the activities performed. Besides, it may also arise from logging errors, which result in activities being recorded in the wrong order,



© Francesco Chiariello;

licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 14; pp. 14:1–14:14

Leibniz International Proceedings in Informatics



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

duplicated, or omitted entirely (thus creating gaps). Finally, noise can also occur because events are inferred from low-level data using activity recognition techniques, rather than being directly recorded [2]. The presence of noise in the event logs makes it difficult to learn temporal patterns and calls for the use of statistical techniques able to handle such noise. Another challenge is that event logs are streams of data. This means that traces are not all available from the beginning, rather they are continuously collected as the process is executed. Ideally, the log should be processed sequentially, as new traces arrive, allowing conclusions about the satisfaction of temporal properties to be drawn immediately, and only waiting for new evidence if necessary.

In this paper, we address these issues by proposing a method based on sequential analysis to learn temporal properties of a process from its corresponding event log. Sequential analysis [21] consists of performing hypothesis tests where a stopping rule is used to halt the sampling process as soon as the collected evidence is sufficient to accept or reject the hypothesis under examination. In particular, we consider a test known as the Sequential Probability Ratio Test (SPRT) [39], originally developed within the domain of quality control in manufacturing. Given an input *lot*, the general idea of the test is to incrementally sample items from the lot and count the number of defects they have. If, at any step, the total number of defects falls below a specified acceptance threshold, the lot is accepted. Conversely, if it exceeds a rejection threshold (which is set higher than the acceptance threshold), the lot is rejected. Otherwise, sampling continues. Naturally, as the total number of defects increases with the number of items examined, both the acceptance and the rejection threshold progressively increase. In our approach, we put forward trace alignment as a way of quantifying the number of defects of a trace with respect to a given formula, after that SPRT can directly be applied to our learning setting. Trace alignment [12] refers to computing a minimal number of alignments, i.e., of modifications of the trace to make it satisfy the formula. In our approach, we compute alignments by framing the problem as cost-optimal planning [10] and solving it using an off-the-shelf AI planner.

Our approach has two main advantages. First, being based on a statistical approach, makes it robust to the noise naturally occurring in the event logs. Second, its ability to make a decision as soon as enough evidence arises, makes it very data-efficient.

The remainder of the paper is organized as follows: in Section 2 we provide the background and basic notation. Then, in Section 3 we describe the proposed method to discover temporal properties from event logs. In Section 4 we perform and discuss controlled experiments to demonstrate the methods’s application. In Section 5 we discuss related work. Finally, Section 6 concludes the paper and points out directions for future work.

2 Background

We start by recalling relevant notions of Linear Temporal Logic over Process Traces (LTL_p) [19]. Then, we describe Declare and show how its templates can be grounded into LTL_p . Finally, we describe Trace Alignment and Sequential Analysis.

2.1 Linear Temporal Logic on Process Traces

Let Σ be a set of propositional symbols, also called *activities*. A *process trace* is a non-empty sequence $\pi \in \Sigma^+$ of activities. An *event* is any occurrence of an activity in the trace. A process trace can be thought of as representing an execution of a process. An *event log* is then a sequence of process traces. Process traces are therefore ordered according to their occurrence (i.e., to the occurrence of their last activity, since multiple process instances may

run in parallel). The same trace may appear multiple times in a log. This is expected since they represent different executions of the same routine task. It is important to note that process traces differ from the traces encountered in Linear Temporal Logic (LTL) [29] in two ways: (i) they are finite, and (ii) exactly one activity occurs per each instant. This last characteristic makes process traces suitable models for logics that reason over actions, rather than over states.

Syntax

The syntax of LTL_p is the same as LTL. An LTL_p formula φ over Σ is defined according to the following grammar:

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid X\varphi \mid \varphi_1 U \varphi_2,$$

with $a \in \Sigma$. The intuitive meaning of the temporal operators “next” X and “until” U is as follows. The formula $X\varphi$ means that at the next time instant, φ holds. The formula $\varphi_1 U \varphi_2$ means that at a certain instant φ_2 holds and up to that point φ_1 holds. We assume common propositional and temporal abbreviations. In particular, for temporal operators, we define the “eventually” operator $F\varphi \equiv \top U \varphi$, the “always” operator $G\varphi \equiv \neg F \neg \varphi$, and the “weak until” operator $\varphi W \varphi' \equiv G\varphi \vee \varphi U \varphi'$.

Semantics

The semantics of LTL_p is defined on process traces. Let φ be an LTL_p formula, $\pi = \pi_1 \pi_2 \dots \pi_{|\pi|}$ a process trace, and $1 \leq i \leq |\pi|$ a time instant. We say that π satisfies φ at time i , and we write $\pi, i \models \varphi$, according to the following definition:

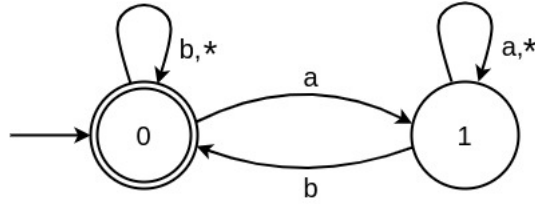
- $\pi, i \models a$ iff $a = \pi_i$;
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models X\varphi$ iff $i < |\pi|$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 U \varphi_2$ iff $\exists j, i \leq j \leq |\pi|$, s.t. $\pi, j \models \varphi_2$ and for $k = i, i+1, \dots, j-1$, $\pi, k \models \varphi_1$.

We say that π is a *model* for φ if $\pi, 1 \models \varphi$, denoted as $\pi \models \varphi$.

Note that, since LTL_p works on traces that are finite, and in analogy with the definitions in Linear Temporal Logic on Finite Trace (LTL_f) [17, 11] the “next” operator needs to explicitly require the existence of a next time instant.

Automata-Based Representation

Each LTL_p formula φ over Σ can be associated a finite-state automaton $\mathcal{A}(\varphi)$ over the same alphabet Σ such that for any trace π it holds that $\pi \models \varphi$ iff π , is accepted by $\mathcal{A}(\varphi)$ [7]. Fig 1 shows the minimal automaton associated to the formula $Response(a, b) = G(a \rightarrow Fb)$, saying that whenever a is performed, then b is performed afterward. Note that transitions are directly labeled with activities in Σ , resulting in an alphabet that is exponentially smaller compared to what we would have if we were interpreting the formula in LTL_f [11]. However, if we identify a process trace with a *simple finite trace* [18], i.e. a finite trace where each propositional interpretation is a singleton, one can use LTL_f to check properties of process traces. Additionally, one can also check whether a trace is a simple trace by suitably modifying the LTL_f formula [8], or by directly modifying the resulting automaton with the addition of a sink state [7].



■ **Figure 1** Automaton for the formula $Response(a, b) = G(a \rightarrow Fb)$. Here the asterisk stands for any activity other than the ones appearing in the formula.

2.2 Declare

Declare [28, 15] is the most common declarative process specification language. It consists of a set of templates that allow to easily specify the temporal constraints of the process. A constraint is any instantiation of the variables in the template with process activities. A process model is then a set of constraints. Table 1 shows how to write some Declare templates (in particular, the ones that will be used later in the experiments) as LTL_p formulae. Be aware that many errors in such encodings are present in the literature. This is due to the difficulty of working with temporal specifications [22]. These formulae have been carefully double-checked by exploiting the automata representation, which sometimes results easier to understand, as well as log generation techniques [5, 6].

■ **Table 1** Some Declare templates and their corresponding LTL_p formula.

Template	LTL_p
$Init(a)$	a
$Exactly2(a)$	$\neg aU(a \wedge X(\neg aU(a \wedge \neg X(Fa))))$
$Response(a, b)$	$G(a \rightarrow Fb)$
$RespondedExistence(a, b)$	$Fa \rightarrow Fb$
$AlternateResponse(a, b)$	$G(a \rightarrow X(\neg aUb))$
$Precedence(a, b)$	$(\neg b)Wa$
$ChainPrecedence(a, b)$	$G(Xb \rightarrow a) \wedge \neg b$
$Choice(a, b)$	$F(a \vee b)$
$ExclusiveChoice(a, b)$	$F(a \vee b) \wedge \neg(Fa \wedge Fb)$
$CoExistence(a, b)$	$Fa \leftrightarrow Fb$

$Init(a)$ says that any trace of the process must start with the activity a (here a stands for a generic activity and so we talk about templates). $Exactly2(a)$ says that exactly two occurrences of a must be present in the trace. This template can be generalized to consider any number of occurrences. $Init(a)$ and $Exactly2(a)$ are unary templates since they express constraints on one activity only. $Response(a, b)$ says that whenever a is executed, b must be executed afterwards. $RespondedExistence(a, b)$ says that whenever a is executed, then b must be executed (regardless of whether it appears before or after a). $AlternateResponse(a, b)$ says that every execution of a must be followed by b , without any other a in between. For those kinds of binary templates, a and b are sometimes referred to as the *activation* and *target* activity, since the occurrence of a triggers checking the occurrences of b . If no activation is present, then the constraints are automatically (vacuously) satisfied. $Precedence(a, b)$ says that b can be executed only if a has been executed before. $ChainPrecedence(a, b)$ says that a must be executed immediately before any execution of b . A common error in literature while encoding this template in temporal logics with only future operators is to forget that b

cannot be executed in the first instant. $Choice(a, b)$ says that eventually one among a and b must be executed. $Choice(a, b)$ further requires that it is not possible to execute them both. Finally, $CoExistence(a, b)$ says that either a and b are both executed, or none of them is executed.

2.3 Trace Alignment

The trace alignment problem is defined as follows. Given a trace π and an LTL_p formula φ , find a minimum number of alignments that makes π a model of φ [12]. An alignment refers to the removal or addition of an event in the trace. Such a problem can be solved by compiling it into cost-optimal planning [10] and then using any off-the-shelf planner supporting cost optimization. One can visualize the application of the alignments as a text cursor moving from the left to the right of π and that can add an event behind it (as is done by pressing a character key on a keyboard, where the cursor automatically moves forward) or remove one in front of it (as with the delete key). The goal of the planning problem is then to have the cursor reach the right side of the trace (possibly adding some other event at this extreme) and the resulting trace be accepted by $\mathcal{A}(\varphi)$. Note that we have here three kinds of actions since moving forward corresponds to an action of zero cost. This is the reason why we need to resort to cost-optimal planning.

2.4 Sequential Analysis

Sequential analysis [21] (not to be confused with sequence analysis [27]) involves conducting hypothesis tests using a stopping rule to halt sampling once there is sufficient evidence to either accept or reject the hypothesis being tested. One such test is the Sequential Probability Ratio Test (SPRT) [39], originally developed within the domain of quality control in manufacturing. Given a lot, we test the products one by one. For $n = 0, 1, 2, \dots$, let d_n be the total number of defects found after controlling the first n products (for $n = 0$ we have clearly $d_0 = 0$). Given a strictly increasing sequence $\{A_n\}$ of *acceptance numbers*, and a strictly increasing sequence $\{R_n\}$ of *rejection numbers*, with $A_n < R_n$ for all $n \in \mathbb{N}$, the test is as follows. If at any point n , we have $d_n < A_n$ then the number of errors is sufficiently small and we accept the lot. If $d_n > R_n$ too many errors have already been found in the lot and we reject it. If instead $A_n < d_n < R_n$ we don't have yet enough evidence to accept or reject the lot and we continue sampling. Let p denote the probability of a defect, p_0 a probability generating a tolerable level of noise, i.e., for which we accept the lot, and $p_1 > p_0$ a probability for which we reject the lot. $H_0 : p = p_0$ is then the null hypothesis, while $H_1 : p = p_1$ the alternative hypothesis. The values of A_n and R_n can be defined by:

$$A_n = \frac{\ln\left(\frac{\beta}{1-\alpha}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)} + n \frac{\ln\left(\frac{1-p_0}{1-p_1}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)}, \quad (1)$$

$$R_n = \frac{\ln\left(\frac{1-\beta}{\alpha}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)} + n \frac{\ln\left(\frac{1-p_0}{1-p_1}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)} \quad (2)$$

where the alpha level α is a parameter controlling type I errors (rejecting a true null hypothesis), and the beta level β is a parameter controlling type II errors (accepting a false alternative hypothesis).

3 SPRT-Based Learning of Temporal Formulae

In this section, we shall describe our approach to learning temporal formulae from event logs by applying SPRT. We begin by noting that equations (1),(2) involve four parameters: p_0 , p_1 , α , and β . However, they can be rewritten as

$$A_n = mn + c_A, \quad (3)$$

$$R_n = mn + c_R, \quad (4)$$

by posing

$$m = \frac{\ln\left(\frac{1-p_0}{1-p_1}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)}, \quad c_A = \frac{\ln\left(\frac{\beta}{1-\alpha}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)}, \quad c_R = \frac{\ln\left(\frac{1-\beta}{\alpha}\right)}{\ln\left(\frac{p_1}{p_0}\right) - \ln\left(\frac{1-p_1}{1-p_0}\right)}. \quad (5)$$

In other words, as a function of n , A_n and R_n are lines with the same slope m (i.e. they are parallel) and intercepts c_A and c_R , respectively. Note that for reasonably small significance levels α and β , i.e. such that $\alpha + \beta < 1$, (besides having $0 < p_0 < p_1 < 1$) it follows:

- $m_0 > 0$ (so that A_n and R_n are strictly increasing sequences).
- $c_A < 0 < c_R$ (otherwise one would end up accepting or rejecting a lot even before inspecting it).

Indeed, any value of m , c_A , and c_R satisfying the above inequalities is an admissible choice. Therefore in the following, we will directly work with these parameters, without explicitly modeling and reasoning about the noise. The method is provided in Algorithm 1. It takes as input a log L , a temporal formula φ , and suitable values for the parameters c_A , c_R , and c_R . First, the initial values d_0 , A_0 , R_0 are assigned (lines 1-3). Then, for each n , we do the following. If $d_n \leq A_n$ we return *Accept* (lines 5-6). If $d_n \geq R_n$ we return *Reject* (lines 7-8). Otherwise, we compute the next values d_{n+1} , A_{n+1} , R_0 (lines 9-12) and proceed. Note that the value of d_{n+1} is computed by first selecting a trace π from the log (line 11), which can be done according to the order of the traces in the log or by random sampling; then computing the number of defects of π , represented by the number of alignments required to make π a model of φ , and adding such quantity to the total number of defects (line 12). While the checks $A_0 < d_0$ and $d_0 < R_0$ could be avoided, treating $n = 0$ as all the other values turns out to be mathematically convenient for interpreting the parameters of the algorithm.

It is worth noting that what we have just described is actually a semi-algorithm. In fact, even assuming a new trace is always available, it could be the case that the total number of defects never satisfies any of the inequalities. To obviate this, one can simply return a default value of their choice (be it *Accept*, *Reject*, or, e.g., *Inconclusive*).

4 Experiments

In this section, we illustrate Algorithm 1 by performing some controlled experiments. We begin by detailing the process of constructing an appropriate log in Subsection 4.1. Subsequently, in Subsection 4.2, we employ this log to examine several Declare constraints.

4.1 Log Generation and Description

To perform controlled experiments, we resort to synthetic data. Fixed an alphabet $\Sigma = \{a, b, c, d, e\}$, we define 5 Declare constraints over it:

■ **Algorithm 1** SPRT-based Learning of Temporal Formulae.

Input: $L, \varphi, m > 0, c_A < 0$, and $c_R > 0$
Output: *Accept* or *Reject*

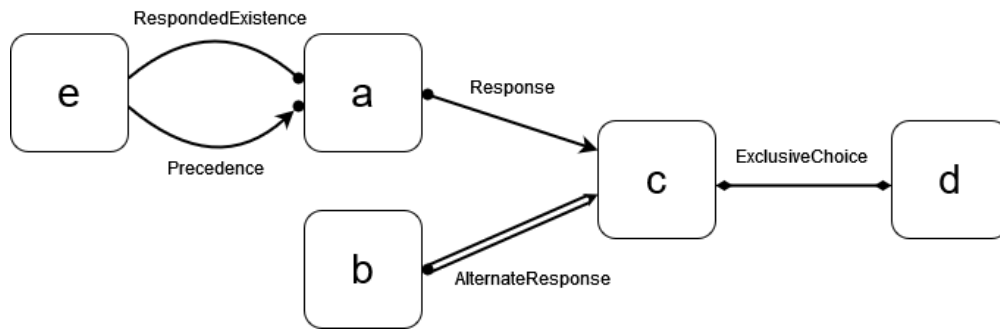
```

1  $defects \leftarrow 0$ 
2  $A \leftarrow c_A$ 
3  $R \leftarrow c_R$ 
4 while True do
5   if  $defects \leq A$  then
6      $\mid$  return Accept
7   if  $defects \geq R$  then
8      $\mid$  return Reject
9    $A \leftarrow A + m$ 
10   $R \leftarrow R + m$ 
11   $\pi \leftarrow Next(L)$  // Extract next trace
12   $defects \leftarrow defects + Align(\varphi, \pi)$  // add the alignments' cost

```

1. *ExclusiveChoice*(c, d),
2. *Response*(a, b),
3. *RespondedExistence*(a, e),
4. *Precedence*(e, a),
5. *AlternateResponse*(b, c).

The model \mathcal{M} represented by the conjunction of those five constraints is represented in the intuitive Declare graphical notation in Fig 2.



■ **Figure 2** The Declare model used for the experiments.

We then generate a log of 100 traces of length varying from 6 to 15. In particular, for each length, we generate 5 positive traces, i.e. satisfying (all the constraints of) the model, and 5 negative traces, i.e. violating at least one constraint. The generation is performed using ASP Log Generator [5, 6], which converts the constraints into their corresponding automata [7] and use Answer Set Programming [3] to find traces accepted or not by the automata (depending on whether we are looking for a positive or negative one). We generate negative traces by explicitly imposing the violation of the model since injecting some noise on a positive trace, e.g. by randomly adding and removing events, could result in the trace still satisfying the constraints. To improve the quality of the log, avoiding traces that are too repetitive and too similar among each other, we constrain each of the 5 activities to occur

■ **Table 2** Number of traces violating the constraints arranged according to the cost of repairs.

	1	2	3+	#traces	total cost
C1	19	13	6	38	68
C2	14	0	0	14	14
C3	9	0	0	9	9
C4	31	0	0	31	31
C5	16	15	2	33	52

■ **Table 3** Number of alignments per constraint as a function of the trace length.

	6	7	8	9	10	11	12	13	14	15	tot
C1	5	2	3	5	6	8	11	5	12	11	68
C2	2	2	1	1	0	1	2	2	0	3	14
C3	3	0	0	1	1	1	2	0	1	0	9
C4	4	2	4	4	4	2	3	2	2	4	31
C5	2	8	6	9	4	6	6	6	0	5	52
tot	16	14	14	20	15	18	24	15	15	23	174

no more than in $1/2$ of the instants. In addition, we modify the search strategy to perform random (rather than heuristic) decisions with probability 0.2 and run the log generator multiple times with different seeds asking each time for 1 solution. Note that how negative traces are generated greatly impacts the cost of the alignments. Table 2 reports, for each constraint, the number of traces violating it, arranged according to the cost of the alignments, together with the total cost of the alignments (traces with a repair cost of 3 or more are merged in one column, however, their different impact is reflected by the total cost). For example, constraint 1 *ExclusiveChoice*(c, d) (C1 for short in the tables) has 38 traces (out of the 50 negative ones) violating it of which 19 require 1 alignment, 13 require 2 alignments, and 6 require 3 or more alignments, for a total of 68 alignments. The costs are computed by compiling the problem into the Planning Domain Definition Language [9] and solving it with Fast Downward [24]. To better understand the noise of the log, in Table 3 we report, for each trace length, the number of alignments required to make the traces of that length conformant with the constraint. Recall that there are 5 negative traces per length.

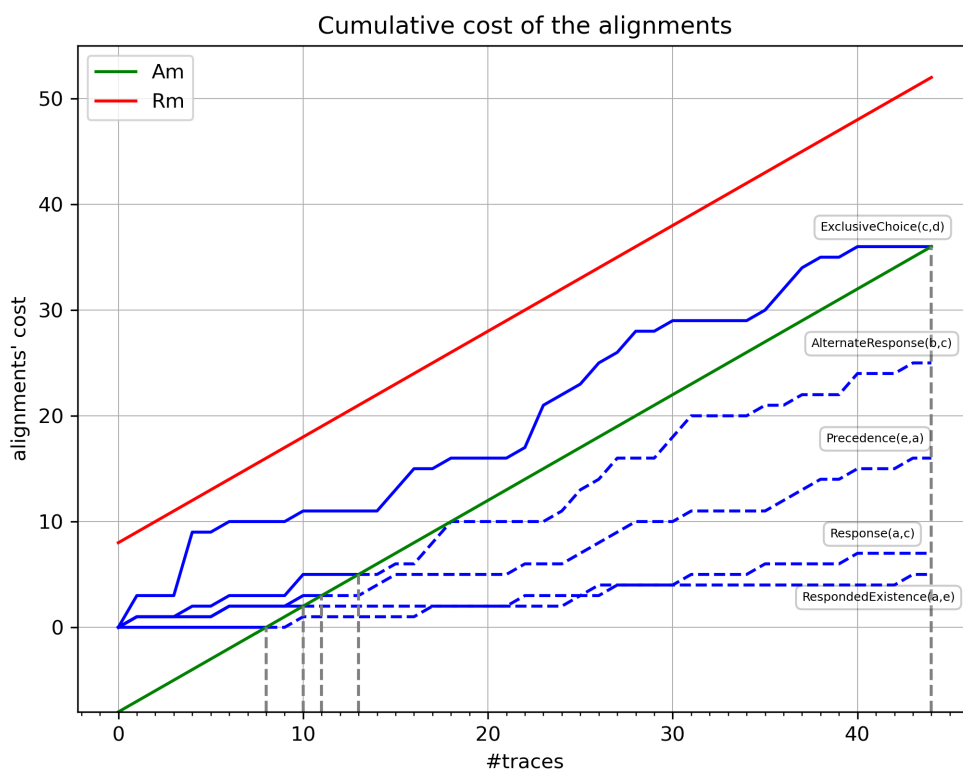
It is important to note that templates behave very differently. We can see for example that *ExclusiveChoice*(c, d) requires a total of 68 alignments over the whole log and that the cost is strongly influenced by the trace length. In fact, in case both c and d are present in the trace, to align it one has to determine which activity has fewer occurrences and remove them. Other templates such as *Response* always require at most 1 alignment, regardless of the trace length, since it is simply sufficient to add the target activity at the end of the trace.

This makes the choice of the values of the parameters m , c_A , and c_R in Algorithm 1 very important, particularly for the slope m . Such a choice, which must be made constraint by constraint, should therefore be undertaken after careful consideration of both the particular formula (where formulae intuitively easier to satisfy correspond to smaller values for m) and the expected noise.

4.2 Execution

In this subsection, we show and discuss the execution of the Algorithm 1 over the generated log. We start by considering the Declare model \mathcal{M} . We choose a value of $m = 1$, of $c_R = 8$, and of $c_A = -8$. In Figure 3, we plot the lines A_n and R_n in green and red, respectively.

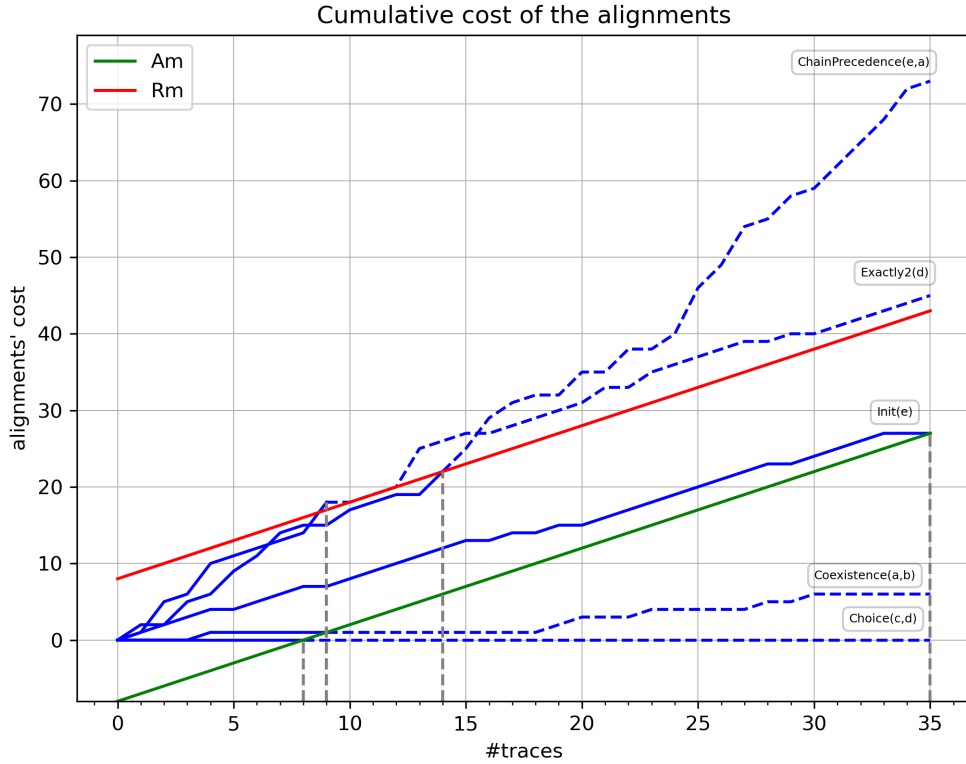
We then apply the algorithm to each of the five constraints of \mathcal{M} . Note that we use here the same parameters across the various constraints just for the sake of simplicity. We want to stress that the choice of such parameters must be done individually for each constraint after carefully pondering the expected noise of the log, the structure of the formula, as well as additional considerations about the risk of a wrong decision. We then plot, for each constraint, the curve representing the total number of defects as a function of the number of traces observed, selecting traces according to their order in the log. We observe that all the constraints are classified as satisfied by the model. The first constraint to be decided is $Response(a, c)$ at step 8. Note that, from $m = 1$ follows that $-c_A$ represents exactly the minimum number of steps before deciding on acceptance. The last constraint to be decided is instead $ExclusiveChoice(c, d)$ at step 44. One could stop plotting a curve after it intersects A_n , indeed the algorithm terminates and the new costs are not computed. In Figure 3 however we continue to show them (this time as dashed lines) to better visualize the behavior of the log with respect to the constraints.



■ **Figure 3** Results of the application of the algorithm to the constraints of the model.

In Figure 4 we consider instead a set of new constraints that were not used during the generation of the log. We again use the same parameters as before. We see in step 8 that the constraint $Choice(c, d)$ is accepted. This constraint is actually implied by one of the constraints used to generate the log, $ExclusiveChoice(c, d)$, that was accepted at a later step. This is a consequence of the fact that any alignment w.r.t. the latter formula is also an alignment w.r.t. to the former one, and holds indeed as a general result, that we state formally as follows:

► **Observation 1.** *If a formula φ is accepted in n_φ steps, and ψ is implied by φ then ψ is accepted as well and in a number of steps $n_\psi \leq n_\varphi$. Dual results hold for the rejection.*



■ **Figure 4** Results of the application of the algorithm to new constraints.

We note from the figure that $Choice(c, d)$ has indeed an alignments' cost of 0 throughout the steps and is therefore decided at step $-c_A = 8$. The fact that the costs are equal to 0 is a consequence of how we generated the log (i.e. by forcing the same activities not to appear too many times in trace). The constraints $CoExistence(a, b)$ and $Init(e)$ are accepted after 9 and 35 steps, respectively. By taking into consideration the meaning of the constraints is clear why the former were accepted: repairing a trace requires at most 1 alignment and therefore $m = 1$ is a high value compared with the expected defects of the traces. We have then the first two cases of rejected constraints: $Exactly2(D)$ rejected after 9 steps, and $ChainPrecedence(e, a)$ rejected after 14 steps. Both of those constraints require acting on the different occurrences of an activity and therefore for them, $m = 1$ may be a too-small value (again, depending on how much noise we expect in the log as well).

Finally, it is worthwhile to point out that, although Algorithm 1 stops after deciding to accept or reject, one could indeed continue to monitor a constraint to be able to change the decision when new evidence emerges. This is important, for example, in cases in which the distribution of the process executions substantially changes over time. In this respect, it is useful to note that, both in Figure 3 and 4 no decision has ever been overturned, not reconsidered. Once a curve enters one of the two half-plane defined by A_m and R_m it tends to stay there. This could however be due to the particular choice of the parameters. Value for m closer to the actual alignment cost, together with a smaller distance $c_R - c_A$ between A_m and R_m , could result in more oscillating situations.

5 Related Works

The problem of learning linear temporal formulae, sometimes known as *LTL (specification) mining* [25], has been approached by the Temporal Logic community mainly with exact methods, rather than relying on statistical ones. Camacho and McIlraith [4] reduce the problem of learning LTL_f formulae to proposition satisfiability. The approach is based on guessing the (alternating) automaton associated with a formula and then checking for conformance with the (positive and negative) example traces. Such automaton can then be converted in linear time into the corresponding formula. Note that no templates are used here; instead, they employ SAT solver to perform a search over the space of all possible formulae. Gaglione et al. [20] later modified the approach to handle noise using MaxSAT solvers. These approaches assume the availability of both positive and negative example traces. Roy et al. [32] consider the problem of learning from positive example only. This setting, which is the standard in Process Mining, is known in the Machine Learning literature as one-class (or unary) classification and is way more challenging than binary classification when considering minimality requirements about the discovered solutions [33, 1]. The approach by Roy et al. [32] is however limited in that it cannot handle noise.

Declarative Process Discovery algorithms [14, 26, 27] have been proposed that exploit statistical analysis, handle noise, and work with positive examples only. However, those algorithms usually consider only Declare constraints. Besides, they assume for simplicity the event log to be a fixed multiset of traces, rather than an ordered collection in continuous evolution, and do not reason about whether the evidence gathered up to some moment is enough to make a decision, as is done with the stopping rules in sequential analysis.

Sequential analysis has also inspired a constraint acquisition algorithm in the context of Constraint Programming [30, 31]. This algorithm is however a coarse simplification of SPRT, using no parameters other than constant acceptance and rejection thresholds, and does not quantify the discrepancy between a constraint and an example. Furthermore, our work differs in that it considers temporal properties of traces rather than constraints over the values of decision variables.

6 Conclusion

In this paper, we have shown how to use statistical techniques from sequential analysis for learning linear temporal formulae from events log. Our approach is based on analyzing the log incrementally and measuring how much a trace deviates from satisfying a given formula. As a measure of such deviation, we use the number of alignments, i.e., of modifications of the trace (in terms of additions and removals of activities) to make the trace a model of the formula. We have then implemented our approach and tested it with controlled experiments. The advantage of a statistical approach is its ability to handle the noise naturally occurring in the event logs. Sequential analysis, in particular, by exploiting only the strictly necessary number of samples, additionally provides early decision-making capabilities. This turns out to be useful both in the case one has to analyze a huge volume of data, alleviating the computational burden, as well as in the case where such data are scarce.

In future work, we intend to study how one can use Algorithm 1 to learn interpretable models, by combining it with a suitable procedure for selecting temporal formulae to be tested, taking into account subsumption-driven hierarchies [13] and their interaction with the learning phase. Furthermore, we intend to study how to suitably select the parameters m , c_A , and c_R of Algorithm 1 by considering both the structure of the formula and the noise

of the log generation process, which need to be explicitly modeled. Finally, we intend to consider other possible measures of the discrepancy between a formula and a trace, study their properties, and the impact they have on applying sequential analysis techniques.

References

- 1 Simone Agostinelli, Francesco Chiariello, Fabrizio Maria Maggi, Andrea Marrella, and Fabio Patrizi. Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis. *Inf. Syst.*, 114:102180, 2023. doi:10.1016/J.IS.2023.102180.
- 2 Iris Beerepoot, Claudio Di Ciccio, Hajo A. Reijers, Stefanie Rinderle-Ma, Wasana Bandara, Andrea Burattin, Diego Calvanese, Tianwa Chen, Izack Cohen, Benoît Depaïre, Gemma Di Federico, Marlon Dumas, Christopher G. J. van Dun, Tobias Fehrer, Dominik Andreas Fischer, Avigdor Gal, Marta Indulska, Vatche Isahagian, Christopher Klinkmüller, Wolfgang Kratsch, Henrik Leopold, Amy Van Looy, Hugo A. López, Sanja Lukumbuzya, Jan Mendling, Lara Meyers, Linda Moder, Marco Montali, Vinod Muthusamy, Manfred Reichert, Yara Rizk, Michael Rosemann, Maximilian Röglinger, Shazia Sadiq, Ronny Seiger, Tijs Slaats, Mantas Simkus, Ida Asadi Someh, Barbara Weber, Ingo Weber, Mathias Weske, and Francesca Zerbato. The biggest business process management problems to solve before we die. *Comput. Ind.*, 146:103837, 2023. doi:10.1016/J.COMPIND.2022.103837.
- 3 Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011. doi:10.1145/2043174.2043195.
- 4 Alberto Camacho and Sheila A. McIlraith. Learning interpretable models expressed in linear temporal logic. In *ICAPS*, pages 621–630. AAAI Press, 2019. URL: <https://ojs.aaai.org/index.php/ICAPS/article/view/3529>.
- 5 Francesco Chiariello, Fabrizio Maria Maggi, and Fabio Patrizi. Asp-based declarative process mining. In *AAAI*, pages 5539–5547. AAAI Press, 2022. doi:10.1609/AAAI.V36I5.20493.
- 6 Francesco Chiariello, Fabrizio Maria Maggi, and Fabio Patrizi. A tool for compiling declarative process mining problems in ASP. *Softw. Impacts*, 14:100435, 2022. doi:10.1016/J.SIMPA.2022.100435.
- 7 Francesco Chiariello, Fabrizio Maria Maggi, and Fabio Patrizi. From LTL on process traces to finite-state automata. In *BPM (Demos / Resources Forum)*, volume 3469 of *CEUR Workshop Proceedings*, pages 127–131. CEUR-WS.org, 2023. URL: <https://ceur-ws.org/Vol-3469/paper-23.pdf>.
- 8 Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *AAAI*, pages 1027–1033. AAAI Press, 2014. doi:10.1609/AAAI.V28I1.8872.
- 9 Giuseppe De Giacomo, Francesco Fuggitti, Fabrizio Maria Maggi, Andrea Marrella, and Fabio Patrizi. A tool for declarative trace alignment via automated planning. *Softw. Impacts*, 16:100505, 2023. doi:10.1016/J.SIMPA.2023.100505.
- 10 Giuseppe De Giacomo, Fabrizio Maria Maggi, Andrea Marrella, and Fabio Patrizi. On the disruptive effectiveness of automated planning for ltlf-based trace alignment. In *AAAI*, pages 3555–3561. AAAI Press, 2017. doi:10.1609/AAAI.V31I1.11020.
- 11 Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860. IJCAI/AAAI, 2013. URL: <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997>.
- 12 Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf. Syst.*, 47:258–277, 2015. doi:10.1016/J.IS.2013.12.005.
- 13 Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali, and Jan Mendling. Ensuring model consistency in declarative process discovery. In *BPM*, volume 9253 of *Lecture Notes in Computer Science*, pages 144–159. Springer, 2015. doi:10.1007/978-3-319-23063-4_9.

- 14 Claudio Di Ciccio and Massimo Mecella. On the discovery of declarative control flows for artful processes. *ACM Trans. Manag. Inf. Syst.*, 5(4):24:1–24:37, 2015. doi:10.1145/2629447.
- 15 Claudio Di Ciccio and Marco Montali. Declarative process specifications: Reasoning, discovery, monitoring. In *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 108–152. Springer, 2022. doi:10.1007/978-3-031-08848-3_4.
- 16 Marlon Dumas and Arthur H. M. ter Hofstede. UML activity diagrams as a workflow specification language. In *UML*, volume 2185 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2001. doi:10.1007/3-540-45441-1_7.
- 17 Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. *Form.Meth.Syst.Des.*, 24(2):101–127, 2004. doi:10.1023/B:FORM.0000017718.28096.48.
- 18 Valeria Fionda and Gianluigi Greco. The complexity of LTL on finite traces: Hard and easy fragments. In *AAAI*, pages 971–977. AAAI Press, 2016. doi:10.1609/AAAI.V30I1.10104.
- 19 Valeria Fionda and Gianluigi Greco. LTL on finite and process traces: Complexity results and a practical reasoner. *J. Artif. Intell. Res.*, 63:557–623, 2018. doi:10.1613/JAIR.1.11256.
- 20 Jean-Raphaël Gaglione, Daniel Neider, Rajarshi Roy, Ufuk Topcu, and Zhe Xu. Learning linear temporal properties from noisy data: A maxsat-based approach. In *ATVA*, volume 12971 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2021. doi:10.1007/978-3-030-88885-5_6.
- 21 Bhaskar Kumar Ghosh and Pranab Kumar Sen. *Handbook of sequential analysis*. CRC Press, 1991.
- 22 Ben Greenman, Sam Saarinen, Tim Nelson, and Shriram Krishnamurthi. Little tricky logic: Misconceptions in the understanding of LTL. *Art Sci. Eng. Program.*, 7(2), 2023. doi:10.22152/PROGRAMMING-JOURNAL.ORG/2023/7/7.
- 23 Object Management Group. Business process model and notation (bpmn), version 2.0.2, 2014. URL: www.omg.org/spec/BPMN.
- 24 Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006. doi:10.1613/JAIR.1705.
- 25 Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General LTL specification mining (T). In *ASE*, pages 81–92. IEEE Computer Society, 2015. doi:10.1109/ASE.2015.71.
- 26 Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *CAiSE*, volume 7328 of *Lecture Notes in Computer Science*, pages 270–285. Springer, 2012. doi:10.1007/978-3-642-31095-9_18.
- 27 Fabrizio Maria Maggi, Claudio Di Ciccio, Chiara Di Francescomarino, and Taavi Kala. Parallel algorithms for the automated discovery of declarative process models. *Inf. Syst.*, 74(Part):136–152, 2018. doi:10.1016/J.IS.2017.12.002.
- 28 Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. DECLARE: full support for loosely-structured processes. In *EDOC*, pages 287–300. IEEE Computer Society, 2007. doi:10.1109/EDOC.2007.14.
- 29 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.
- 30 Steve Prestwich. Robust constraint acquisition by sequential analysis. In *ECAI*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 355–362. IOS Press, 2020. doi:10.3233/FAIA200113.
- 31 Steven Prestwich and Nic Wilson. A statistical approach to learning constraints. *International Journal of Approximate Reasoning*, 2024.
- 32 Rajarshi Roy, Jean-Raphaël Gaglione, Nasim Baharisangari, Daniel Neider, Zhe Xu, and Ufuk Topcu. Learning interpretable temporal properties from positive examples only. In *AAAI*, pages 6507–6515. AAAI Press, 2023. doi:10.1609/AAAI.V37I5.25800.
- 33 Tijs Slaats, Søren Debois, and Christoffer Olling Back. Weighing the pros and cons: Process discovery with negative examples. In *BPM*, volume 12875 of *Lecture Notes in Computer Science*, pages 47–64. Springer, 2021. doi:10.1007/978-3-030-85469-0_6.

- 34 Wil M. P. van der Aalst. The application of petri nets to workflow management. *J. Circuits Syst. Comput.*, 8(1):21–66, 1998. doi:10.1142/S0218126698000043.
- 35 Wil M. P. van der Aalst. Foundations of process discovery. In *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 37–75. Springer, 2022. doi:10.1007/978-3-031-08848-3_2.
- 36 Wil M. P. van der Aalst. Process mining: A 360 degree overview. In *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 3–34. Springer, 2022. doi:10.1007/978-3-031-08848-3_1.
- 37 Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Comput. Sci. Res. Dev.*, 23(2):99–113, 2009. doi:10.1007/S00450-009-0057-9.
- 38 Wil M. P. van der Aalst and Christian Stahl. *Modeling Business Processes - A Petri Net-Oriented Approach*. Cooperative Information Systems series. MIT Press, 2011.
- 39 A Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.
- 40 Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, Third Edition*. Springer, 2019. doi:10.1007/978-3-662-59432-2.

A Framework for Assessing Inconsistency in Disjunctive Temporal Problems

Jean-François Condotta ✉ 🏠 

CRIL UMR 8188, Université d'Artois & CNRS, Lens, France

Yakoub Salhi ✉ 🏠 

CRIL UMR 8188, Université d'Artois & CNRS, Lens, France

Abstract

Inconsistency measures serve to quantify the level of contradiction present within a knowledge base. They can be used for both consistency restoration and information extraction. In this article, we specifically explore inconsistency measures applicable to Disjunctive Temporal Problems (DTPs). We present a framework that extends traditional propositional logic approaches to DTPs, incorporating both new postulates and adaptations of existing ones. We identify and elaborate on various properties that establish relationships among these postulates. Furthermore, we introduce multiple inconsistency measures, adopting both a conventional approach that particularly leverages Minimal Inconsistent Subsets and a DTP-specific strategy based on constraint relaxation. Finally, we show the applicability of the inconsistency measures in DTPs through two real-world applications.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning

Keywords and phrases Disjunctive Temporal Problems, Inconsistency Measures, Temporal Reasoning

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.15

1 Introduction

Numerous formalisms have been proposed in the literature for representing and reasoning about temporal information under constraints. The temporal constraints considered by these formalisms differ in two primary aspects. First, the types of temporal entities represented by their variables, which can include temporal points, intervals, or even durations. Second, their nature can be qualitative [1], quantitative, or a combination of both [11, 5]. Simple Temporal Problems (STPs) [7] belong to the temporal formalisms allowing to handle quantitative constraints. They represent temporal entities as points on a timeline and allow constraining distance between each pair of temporal entities using numeric values specified by an interval. To increase the expressiveness of the considered temporal constraints, STPs have been extended numerous times [12, 20]. In particular, Disjunctive Temporal Problems (DTPs) [18] extend STPs by employing disjunctions of STP constraints, thus providing a temporal framework highly useful in a wide range of applications.

In the literature, an inconsistency measure is defined as a function that assigns a non-negative value to each knowledge base. It quantifies the degree of conflict or contradiction present within the database, offering an interesting tool for evaluating and managing inconsistencies (e.g. see [10, 19, 13, 9]). In the realm of application, these measures are used in various analytical reasoning approaches. For instance, in the data mining task of clustering, inconsistency measures are utilized to enhance the quality of clusters by actively reducing contradictions [14]. Furthermore, these measures are used as a stepping stone for defining paraconsistent consequence relations, which allow for logical deduction in the presence of inconsistent information [15].

The literature presents a wide range of proposals for defining inconsistency measures, aimed at identifying and addressing various forms of conflict, highlighting the richness of this research field (e.g., see [10, 8, 3, 2, 4]). Numerous studies on inconsistency measures have adopted a



© Jean-François Condotta and Yakoub Salhi;

licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 15; pp. 15:1–15:18

Leibniz International Proceedings in Informatics



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

postulate-based approach by reasoning about various dimensions related to the measurement and management of inconsistency. This approach standardizes assessment criteria and facilitates a comprehensive understanding of the underlying causes of inconsistency.

Despite extensive research, there remains a significant gap in the exploration of inconsistency measurement within the realm of temporal reasoning. To the best of our knowledge, only a few studies have specifically focused on adapting inconsistency measures to this context, concentrating primarily on qualitative spatio-temporal reasoning and temporal logic (see [6, 17, 16]). In this paper, we introduce the first framework designed specifically for measuring inconsistency in DTPs.

Our first contribution consists in introducing a range of rationality postulates for defining inconsistency measures in DTPs. Some of these postulates are adaptations from those established in the propositional context, while others are uniquely tailored to DTPs. For example, one DTP-specific postulate asserts that applying an identical shift to all intervals within a DTP does not alter the amount of inconsistency. We also examine the relationships among these postulates. Our analysis particularly highlights that certain postulates are incompatible, and combining specific ones can yield an inconsistency measure that can only distinguish between consistent and inconsistent DTPs.

Our second contribution is the development of various inconsistency measures using different approaches. Specifically, we employ a traditional approach that involves Minimal Inconsistent Subsets, and we introduce a strategy specifically tailored for DTPs based on constraint relaxation. This relaxation is achieved by widening the temporal intervals.

Our third contribution details two applications of inconsistency measures within DTPs. The core principle of our approach involves using these measures to facilitate the selection of the most suitable solutions. Within a DTP framework, constraints may correspond to either the specific needs of an individual agent or the integrity constraints of a computational service. By applying inconsistency measures, we are able to identify the optimal service or achieve consensus among agents.

2 Preliminaries

2.1 Temporal problems

In the sequel we will denote by $\mathcal{I}^{\mathbb{Z}}$ the set of closed (possibly half-unbounded or unbounded) intervals over \mathbb{Z} with endpoints in $\mathbb{Z} \cup \{-\infty, +\infty\}$. Given $I \in \mathcal{I}^{\mathbb{Z}}$, I^{-1} will denote the interval of $\mathcal{I}^{\mathbb{Z}}$ containing the opposite values of I . We consider Disjunctive Temporal Problems (DTP) [18] on $\mathcal{I}^{\mathbb{Z}}$.

► **Definition 1** (Disjunctive Temporal Problem (DTP)).

- A temporal constraint c is a disjunction $x_1 - y_1 \in I_1 \vee \dots \vee x_k - y_k \in I_k$ where $k \geq 1$, $x_1, \dots, x_k, y_1, \dots, y_k$ are temporal variables with domain \mathbb{Z} and I_1, \dots, I_k are intervals belonging to $\mathcal{I}^{\mathbb{Z}}$.
- A DTP D is a pair (V, C) where $V = \{x_1, \dots, x_n\}$ is a finite set of temporal variables ranging over \mathbb{Z} and $C = \{c_1, \dots, c_m\}$ is a set of temporal constraints involving V where $n \geq 1$ and $m \geq 1$.
- A solution σ of a DTP $D = (V, C)$ is an assignment of integer numbers to the variables in V such that all constraints in C are satisfied. More formally, a solution σ of D is a function $\sigma : V \rightarrow \mathbb{Z}$ such that for each $c \in C$ there exists at least one disjunct $x - y \in I$ belonging to c such that the value $\sigma(x) - \sigma(y)$ belongs to the interval I .
- A DTP admitting at least one solution will be said consistent. In the contrary case it will be said inconsistent.

Let \mathcal{DTP} represent the set of all DTPs. In the sequel, a temporal constraint (resp. a disjunct of a temporal constraint) will also be called a temporal clause (resp. a temporal literal).

Given a DTP $D = (V, C)$ and a constraint $c \in C$, we sometimes use $D \setminus \{c\}$ to denote the DTP $(V, C \setminus \{c\})$.

A Simple Temporal Problem (STP) [7] is a specific type of DTP characterized by each constraint containing exactly one disjunct. Furthermore, a Temporal Constraint Satisfaction Problem (TCSP) [7] represents a DTP in which all disjuncts within a constraint apply to the same pair of variables.

Given a set of temporal variables V , the set of all possible assignments of integer numbers to the variables in V will be denoted by $\llbracket V \rrbracket$. Moreover, the subset of $\llbracket V \rrbracket$ corresponding to the set of solutions of a DTP $D = (V, C)$ will be denoted by $\text{sols}(D)$.

Note that here we consider DTPs with constraints on closed integer intervals and interpret their variables with integer values. Even if it may seem restrictive, most of the concepts and results introduced later can be extended to more general DTPs.

Consider a temporal constraint $c = x_1 - y_1 \in I_1 \vee \dots \vee x_k - y_k \in I_k$. The notation $\text{vars}(c)$ refers to the set of temporal variables involved in c , *i.e.*, the set of variables $\{x_1, \dots, x_k, y_1, \dots, y_k\}$. Furthermore, $\text{Lit}(c)$ denotes the disjuncts (temporal literals) of the constraint of c . Moreover, given another temporal constraint c' , c subsumes c' if and only if (i) $\text{vars}(c) \subseteq \text{vars}(c')$, and (ii) for all $(x - y \in I) \in \text{Lit}(c)$, there exists $(x - y \in I')$ such that $I \subseteq I'$. A sub-DTP of a DTP $D = (V, C)$ is a DTP (V, C') such that $C' \subseteq C$. Given two DTPs $D = (V, C)$ and $D' = (V', C')$, the union of D and D' , denoted by $D \cup D'$, is the DTP defined by $(V \cup V', C \cup C')$.

► **Example 2.** As illustration, consider the set of temporal variables $V = \{x_1, x_2, x_3, x_4, x_5\}$ and the following set of temporal constraints C :

$$\begin{aligned} c_1 &= x_1 - x_2 \in [4, 7] \vee x_2 - x_3 \in [-2, 2] \vee x_2 - x_4 \in [0, 8], & c_5 &= x_1 - x_3 \in [10, 12] \vee x_1 - x_4 \in [6, 7], \\ c_2 &= x_3 - x_4 \in [-20, 20], & c_6 &= x_1 - x_2 \in [-11, -6] \vee x_1 - x_3 \in [6, 9], \\ c_3 &= x_1 - x_2 \in [-15, -10] \vee x_2 - x_3 \in [8, 12], & c_7 &= x_2 - x_5 \in [5, 10] \vee x_3 - x_5 \in [0, 5], \\ c_4 &= x_1 - x_3 \in [-11, -8] \vee x_2 - x_4 \in [-6, -3], & c_8 &= x_1 - x_3 \in [0, 3] \vee x_1 - x_4 \in [12, 14]. \end{aligned}$$

Let the DTPs $D = (V, C)$ and $D' = (V, C \setminus \{c_8\})$. The DTP D' is consistent, whereas the DTP D is inconsistent. A solution σ of D' is given by the following assignment: $\sigma(x_1) = 10$, $\sigma(x_2) = \sigma(x_3) = 21$, $\sigma(x_4) = 4$ and $\sigma(x_5) = 16$. This assignment satisfies, for example, the temporal literal $x_1 - x_3 \in [-11, -8]$ of the constraint c_4 since $\sigma(x_1) - \sigma(x_3) = 10 - 21 = -11$.

2.2 Relaxations

An *c-rewriting rule* for a temporal clause c is a function μ mapping from $\text{Lit}(c)$ (the set of temporal literals in c) to $\mathcal{I}^{\mathbb{Z}}$. This function μ is applied to a temporal constraint c such that $\mu(c)$ denotes the temporal clause resulting from replacing each literal $l = x - y \in I$ within c with $x - y \in \mu(l)$.

A *local c-transformation* for a DTP instance $D = (V, C)$, where $C = \{c_1, \dots, c_n\}$, is a function λ that assigns an *c-rewriting rule* μ_i to each constraint c_i in C . This transformation is applied to D such that $\lambda(D)$ results in a new DTP instance (V, C') , where $C' = \bigcup_{i=1}^n \{\mu_i(c_i)\}$. In the sequel, by abuse of notation, we will sometimes denote $(\lambda(c_i))(c_i)$ by $\lambda(c_i)$ for notational convenience.

A *local c-relaxation* for a DTP instance $D = (V, C)$ is defined as a local *c-transformation* λ , where for each temporal constraint $c = x_1 - y_1 \in I_1 \vee \dots \vee x_k - y_k \in I_k$ in C , the transformation ensures that $I_i \subseteq (\lambda(c))(x_i - y_i \in I_i)$ for every $i \in \{1, \dots, k\}$. This transformation effectively relaxes the constraints, making them less restrictive.

15:4 A Framework for Assessing Inconsistency in Disjunctive Temporal Problems

Given two intervals $I = [l, u]$ and $I' = [l', u']$ with $I \subseteq I'$, we use $\delta(I, I')$ to denote the value $(u' - l') - (u - l)$. We extend δ to the cases where I or I' are half-unbounded or unbounded intervals in the following way: $\delta(I, I') = +\infty$ for the cases where I is left-bounded (resp. right-bounded) and I' is left-unbounded (resp. left-unbounded). Intuitively, this means that the value of infinity dominates in this scenario. In the case where $I =] - \infty, u]$ and $I' =] - \infty, u']$ (resp. $I = [l, +\infty[$ and $I' = [l', +\infty[$), $\delta(I, I')$ is defined by $u' - u$ (resp. $l - l'$). For the last case which corresponds to $I = I' =] - \infty, +\infty[$, $\delta(I, I')$ is defined by 0.

Let λ be a local c-relaxation of $D = (V, C)$, we use $\omega(\lambda)$ to denote the following value:

$$\sum_{c \in C} \sum_{l=(x-y \in I) \in \text{Lit}(c)} \delta(I, (\lambda(c))(l))$$

Furthermore, we use $\theta(\lambda)$ to denote the following value:

$$\max_{c \in C, l=(x-y \in I) \in \text{Lit}(c)} \delta(I, (\lambda(c))(l))$$

► **Example 3.** Consider again the DTP $D = (V, C)$ defined in Example 2. Let λ be a local c-transformation for D that assigns the c-rewriting rule μ_i to the constraint c_i in C , with the rules μ_i defined as follows:

$$\mu_1(x_1 - x_2 \in [4, 7]) = [4, 10], \mu_1(x_2 - x_3 \in [-2, 2]) = [-4, 3], \mu_1(x_2 - x_4 \in [0, 8]) = [0, 8];$$

$$\mu_2(x_3 - x_4 \in [-20, 20]) = [-30, 40];$$

$$\mu_5(x_1 - x_3 \in [10, 12]) = [0, 20], \mu_5(x_1 - x_4 \in [6, 7]) = [2, 15];$$

and $\mu_i(x - y \in I) = I$ for each $c_i \in \{c_3, c_4, c_6, c_7, c_8\}$ and each temporal $x - y \in I$ belonging to $\text{Lit}(c_i)$.

We have $\lambda(c_1) = x_1 - x_2 \in [4, 10] \vee x_2 - x_3 \in [-4, 3] \vee x_2 - x_4 \in [0, 8]$, $\lambda(c_2) = x_3 - x_4 \in [-30, 40]$, $\lambda(c_5) = x_1 - x_3 \in [0, 20] \vee x_1 - x_4 \in [2, 15]$ and $\lambda(c_i) = c_i$ for all $c_i \in \{c_3, c_4, c_6, c_7, c_8\}$.

Clearly, the local c-transformation λ is a local c-relaxation of D . It results in the less constraining DTP $\lambda(D) = (V, \{\lambda(c_1), \lambda(c_2), \lambda(c_5)\} \cup \{c_3, c_4, c_6, c_7, c_8\})$. We can notice that $\lambda(D)$ is consistent, whereas D is an inconsistent DTP. A solution σ of $\lambda(D)$ is given by the following assignment: $\sigma(x_1) = 20$, $\sigma(x_2) = \sigma(x_3) = 31$, $\sigma(x_4) = 8$ and $\sigma(x_5) = 26$. Moreover, we have $\omega(\lambda) = 3 + 3 + 30 + 18 + 12 = 66$ and $\theta(\lambda) = \max\{3, 3, 30, 18, 12\} = 30$.

In the following, we show that to achieve a consistent DTP from an inconsistent DTP, we can restrict ourselves to local c-relaxations that extend the interval of at most one temporal literal per temporal clause by modifying only one of its bounds. We also show that such a restriction preserves optimal relaxations with respect to minimizing the values generated by the functions $\omega(\cdot)$ et $\theta(\cdot)$.

► **Proposition 4.** Let $D = (V, C)$ be a DTP and a local c-relaxation λ of D . If $\lambda(D)$ is a consistent DTP then there exists a local c-relaxation λ' of D such that:

- (1) $\lambda'(D)$ is a consistent DTP,
- (2) for each $c \in C$, we have $|\{l = (x - y \in I) \in \text{Lit}(c) : I \neq (\lambda'(c))(l)\}| \leq 1$,
- (3) for each $c \in C$ and $l = (x - y \in I) \in \text{Lit}(c)$, we have $I = (\lambda'(c))(l)$ or $(\lambda'(c))(l) \setminus I$ is a bounded interval of $\mathbb{I}^{\mathbb{Z}}$,
- (4) we have $\theta(\lambda) \geq \theta(\lambda')$ and $\omega(\lambda) \geq \omega(\lambda')$.

Proof. Suppose that $\lambda(D)$ is a consistent DTP. Let σ a solution of $\lambda(D)$. From σ we will define a local c-relaxation λ' with the desired properties. As σ is a solution of $\lambda(D)$, we know that for each $c \in C$ there exists at least one temporal literal $l_c = (x - y \in I) \in \text{Lit}(c)$ such that $\sigma(x) - \sigma(y) \in (\lambda(c))(l_c)$. Select such a temporal literal $l_c = (x - y \in I)$ and define $\lambda'(c)$ by $(\lambda'(c))(l_c) = I$ if $(\lambda(c))(l_c) = I$ or $\sigma(x) - \sigma(y) \in I$, by the smallest interval

of $\mathcal{I}^{\mathbb{Z}}$ containing the values of I and $\sigma(x) - \sigma(y)$ in the contrary case. Moreover, we define $(\lambda'(c))(u - v \in I')$ by I' for all $(u - v \in I') \in \text{Lit}(c) \setminus \{l_c\}$. Remark that for each $c \in C$, we have $\sigma(x) - \sigma(y) \in (\lambda'(c))(l_c)$, by consequence σ is a solution of $\lambda'(D)$. Hence Property (1) is satisfied. Moreover, by construction of λ' , we can assert that the properties (2) and (3) are also satisfied. Always by construction, we can observe that $I \subseteq (\lambda'(c))(l) \subseteq (\lambda(c))(l)$ for all $c \in C$ and $l = (x - y \in I) \in \text{Lit}(c)$. It follows that $\delta(I, (\lambda'(c))(l)) \leq \delta(I, (\lambda(c))(l))$ for all $c \in C$ and $l = (x - y \in I) \in \text{Lit}(c)$. As a result, we have $\theta(\lambda') \leq \theta(\lambda)$ and $\omega(\lambda') \leq \omega(\lambda)$. ◀

In the sequel, given a DTP D , the set of the local c-relaxations λ of D such that $\lambda(D)$ is consistent will be denoted by $\text{LCR}(D)$.

► **Example 5.** Consider the DTP $D = (V, C)$ defined in Example 2 and its local c-relaxation λ given in Example 3 with the solution σ of $\lambda(D)$. By following the approach described in the proof of Proposition 4, we can construct a new local c-relaxation λ' from λ and σ that satisfies the properties specified in Proposition 4. The resulting local c-relaxation λ' assigns the c-rewriting rule μ'_i to the constraint c_i in C in the following way:

$$\mu'_2(x_3 - x_4 \in [-20, 20]) = [-20, 23];$$

$$\mu'_5(x_1 - x_3 \in [10, 12]) = [10, 12], \mu'_5(x_1 - x_4 \in [6, 7]) = [6, 12];$$

and $\mu'_i(x - y \in I) = I$ for each $c_i \in \{c_1, c_3, c_4, c_6, c_7, c_8\}$ and each temporal $x - y \in I$ belonging to $\text{Lit}(c_i)$.

We have $\lambda'(c_2) = x_3 - x_4 \in [-20, 23]$, $\lambda'(c_5) = x_1 - x_3 \in [10, 12] \vee x_1 - x_4 \in [6, 12]$ and $\lambda'(c_i) = c_i$ for all $c_i \in \{c_1, c_3, c_4, c_6, c_7, c_8\}$. It is clear that λ' is a local c-relaxation of D ensuring that $\lambda'(D)$ is a consistent DTP with σ serving as a solution. Hence, we have $\lambda' \in \text{LCR}(D)$. Moreover, we have $\omega(\lambda') = 3 + 5 = 8$ and $\theta(\lambda') = \max\{3, 5\} = 5$, whereas $\omega(\lambda) = 66$ and $\theta(\lambda) = 30$.

3 Rationality Postulates for Inconsistency Measurement

In this section, we describe various rationality postulates that can be used for defining inconsistency measures in the context of DTPs. Many of these rationality postulates are adaptations of those introduced in the propositional case. Additionally, we highlight several interesting relationships between the considered postulates.

Before presenting our rationality postulates, we first outline the concepts used to express them.

► **Definition 6** (Minimal Inconsistent Sub-DTP (MIS)). *Let $D = (V, C)$ be a DTP. A Minimal Inconsistent Sub-DTP (MIS) of D is an inconsistent sub-DTP D' of D such that each sub-DTP D'' of D' , with $D' \neq D''$, is consistent.*

A constraint c is said to be *free* in a DTP D if there exists no MIS $D' = (V, C)$ of D such that $c \in C$.

Consider now the dual concept of MIS.

► **Definition 7** (Maximal Consistent Sub-DTP (MCS)). *Let $D = (V, C)$ be a DTP. A Maximal Consistent Sub-DTP (MCS) of D is a consistent sub-DTP D' of D such that each sub-DTP D'' of D , with $D' \subsetneq D''$, is inconsistent.*

Clearly, a constraint is free if it belongs to all MCSes.

A constraint c is said to be *safe* in a DTP D if there is a variable $x \in \text{vars}(c)$ such that $v \notin \text{vars}(c')$ for every $c' \in C \setminus \{c\}$.

15:6 A Framework for Assessing Inconsistency in Disjunctive Temporal Problems

Let us note that our definition of safe constraints diverges from the concept of a safe formula as defined in [10]. Indeed, a safe formula is defined as one that shares no propositional variables with the remaining elements of the knowledge base.

We adopt the following notational conventions:

- $\text{MIS}(D)$: the set of MISes of D ,
- $\text{MCS}(D)$: the set of MCSes of D ,
- $\text{Free}(D)$: the set of free constraints of D ,
- $\text{Safe}(D)$: the set of safe constraints of D .

► **Example 8.** Revisiting the DTP $D = (V, C)$ defined in Example 2, we observe the following:

- $\text{MCS}(D) = \{(V, C \setminus \{c_5\}), (V, C \setminus \{c_8\})\}$,
- $\text{MIS}(D) = \{(V, \{c_1, c_4, c_5, c_8\}), (V, \{c_1, c_5, c_6, c_8\}), (V, \{c_3, c_4, c_5, c_8\}), (V, \{c_4, c_5, c_6, c_8\})\}$,
- $\text{Free}(D) = \{c_2, c_7\}$,
- $\text{Safe}(D) = \{c_7\}$.

► **Proposition 9.** *Let D be a DTP. If c is a safe constraint in D , then c is free in D .*

Proof. Let $D = (V, C)$ and consider a constraint $c \in C$ which is identified as a safe but not free constraint in C . Given that c is not free, there exists a MIS $D' = (V, C')$ of D such that $c \in C'$. Since D' is an MIS, we can define $D'' = (V, C' \setminus \{c\})$, which admits a solution σ . Given the safety of c in D , it follows that there exists a variable $x \in \text{vars}(c)$ that does not appear in any constraints of D'' . Assuming without loss of generality that $x - y \in I$ is a temporal literal in c , we can identify a specific value v such that $v - \sigma(y) \in I$. We then define a new assignment σ' for D' as follows: $\sigma'(x) = v$, and for all $y \in V \setminus \{x\}$, $\sigma'(y) = \sigma(y)$. Since σ' satisfies c and σ satisfies D'' , we conclude that σ' is a solution for D' . This leads to a contradiction, as D' being a MIS. ◀

Given a DTP D and an integer $k \in \mathbb{Z}$, we use $D \oplus k$ to denote the DTP obtained from D by replacing each interval $[l, u]$ with $[l + k, u + k]$.

► **Proposition 10.** *Let D be a DTP and $k \in \mathbb{Z}$. Then D is consistent iff $D \oplus k$ is consistent.*

Proof. This is mainly a consequence of the fact that for every solution σ , the assignment σ' is a solution of $D \oplus k$, where $\sigma'(x) = \sigma(x) + k$ for every variable x . ◀

An inconsistency measure \mathcal{I} is a function that maps a DTP to a non-negative real value. By denoting $\mathbb{R}_{\geq 0}^{\infty}$ the set of non-negative real value, an inconsistency measure is a function $\mathcal{I} : \mathcal{DTP} \rightarrow \mathbb{R}_{\geq 0}^{\infty}$ that satisfies the following property:

- $\mathcal{I}(D) = 0$ iff D is a consistent DTP (Consistency - Cons).

The property Cons stipulates that an inconsistency measure must distinguish between consistent and inconsistent DTPs.

In this work, many rationality postulates for defining inconsistency measures are analogous to those introduced in the propositional case (e.g., see [10, 19]). The considered postulates are as follows: for all DTPs $D = (V, C)$ and $D' = (V', C')$,

- $\mathcal{I}(D \cup D') \geq \mathcal{I}(D)$ (Monotonicity - Mono).
- If $c \in C$ is a safe temporal constraint of D then $\mathcal{I}(D) = \mathcal{I}((V, C \setminus \{c\}))$ (Safe Constraint Independence - SCI).
- If $c \in C$ is a free temporal constraint of D then $\mathcal{I}(D) = \mathcal{I}((V, C \setminus \{c\}))$ (Free Constraint Independence - FCI).
- If $c \in C$ is not free in D then $\mathcal{I}(D) > \mathcal{I}((V, C \setminus \{c\}))$ (Problematic Constraint Dependence - PCD).

- If D' is consistent and $V \cap V' = \emptyset$, then $\mathcal{I}(D \cup D') = \mathcal{I}(D)$ (Sub-DTP Independence - SDI).
- If c subsumes c' then $\mathcal{I}((V \cup \text{vars}(c), C \cup \{c\})) \geq \mathcal{I}((V \cup \text{vars}(c'), C \cup \{c'\}))$ (Subsumption - Sub).
- If c subsumes c' and $c \notin C$, then $\mathcal{I}((V \cup \text{vars}(c), C \cup \{c\})) \geq \mathcal{I}((V \cup \text{vars}(c'), C \cup \{c'\}))$ (Weak Subsumption - WSub).
- If $V \cap V' = \emptyset$, then $\mathcal{I}(D \cup D') = \mathcal{I}(D) + \mathcal{I}(D')$ (Variable Independence-Additivity - VIA).
- If $C \cap C' = \emptyset$, then $\mathcal{I}(D \cup D') \geq \mathcal{I}(D) + \mathcal{I}(D')$ (Super-Additivity - SA).
- For any $k \in \mathbb{Z}$, $\mathcal{I}(D) = \mathcal{I}(D \oplus k)$ (Shift Independence - SI).

The postulate **Mon** asserts that adding a new constraint cannot decrease the existing level of contradiction within the DTP. **SCI** and **FCI** maintain that safe constraints and free constraints, respectively, do not influence the level of conflict. **PCD** says that introducing non-free constraints must increase the level of contradiction. **SDI** states that an independent, consistent sub-DTP does not affect the overall amount of contradiction. **Sub** postulates that stricter constraints results in more conflicts. **WSub** is a weaker variant of **Sub**; the condition $c \notin C$ allows us to indicate that c is replaced with c' . **VIA** asserts that the total contradiction in the union of two DTPs, which do not share any variables, equals the sum of their individual contradictions. **SA** says that the total amount of contradiction in two disjoint DTPs cannot be less than the sum of their individual contradictions. Finally, **SI** posits that applying a shift to a DTP does not alter the level of contradiction.

It is important to note that in a consistent DTP, introducing a constraint that is less restrictive than an existing one does not modify the solution set. This observation also explains why, in **Sub** as opposed to **WSub**, we do not require the condition $c \notin C$: adding a weaker constraint does not affect the amount of contradiction.

The properties previously described are not entirely independent and display various interrelationships. For example, it is evident that **Sub** implies **WSub**. The proposition below outlines additional relationships among these properties.

► **Proposition 11.** *The following properties hold:*

1. *FCI implies SCI.*
2. *SA implies Mono.*
3. *Cons and VIA implies SDI.*

Proof.

Property 1. It is a direct consequent of Proposition 9: every safe constraint is a free constraint.

Property 2. Let \mathcal{I} be an inconsistency measure that satisfies **SA**. Let $D = (V, C)$ and $D' = (V', C')$ be two DTPs. We define a new DTP D'' as $D'' = (V', C' \setminus C)$. It follows that $D \cup D' = D \cup D''$. Furthermore, using property **SA** and noting that $D \cap D'' = \emptyset$, we deduce $\mathcal{I}(D \cup D') = \mathcal{I}(D \cup D'') \geq \mathcal{I}(D) + \mathcal{I}(D'')$. Consequently, we obtain $\mathcal{I}(D \cup D') \geq \mathcal{I}(D)$.

Property 3. Let \mathcal{I} be an inconsistency measure that satisfies both **Cons** and **VIA**. Let D and D' be two DTPs such that D' is consistent and $\text{vars}(D) \cap \text{vars}(D') = \emptyset$. Applying **VIA** under the condition that $\text{vars}(D) \cap \text{vars}(D') = \emptyset$, we deduce that $\mathcal{I}(D \cup D') = \mathcal{I}(D) + \mathcal{I}(D')$. Moreover, by invoking **Cons**, $\mathcal{I}(D') = 0$ holds. Therefore, this leads to $\mathcal{I}(D \cup D') = \mathcal{I}(D)$. ◀

As demonstrated by the following proposition, some of our properties are incompatible.

► **Proposition 12.** *There is no inconsistency measure that satisfies both **PCD** and **Sub**.*

Proof. Assume, for the sake of contradiction, that there exists a measure \mathcal{I} that satisfies both PCD and Sub. Consider the DTP $D = (\{x, y\}, C)$ where $C = \{x - y \in [0, 1], x - y \in [2, 3]\}$. Clearly, the constraint $x - y \in [2, 3]$ subsumes $x - y \in [2, 4]$. Employing Sub, it follows that $\mathcal{I}(D') \geq \mathcal{I}(D'')$, with $D' = (\{x, y\}, C \cup \{x - y \in [2, 3]\})$ and $D'' = (\{x, y\}, C \cup \{x - y \in [2, 4]\})$. Since $C \cup \{x - y \in [2, 3]\} = C$, we have $\mathcal{I}(D') = \mathcal{I}(D)$. Furthermore, $x - y \in [2, 4]$ is clearly problematic in D'' ; hence, by applying PCD, we deduce that $\mathcal{I}(D) < \mathcal{I}(D'')$. This leads to a contradiction. \blacktriangleleft

Over-constraining inconsistency measures can lead to uninteresting results, as demonstrated by the following proposition.

► **Proposition 13.** *An inconsistency measure \mathcal{I} satisfies Cons, Sub and SA iff \mathcal{I} is defined as follows:*

$$\mathcal{I}(D) = \begin{cases} \infty & \text{if } D \text{ is inconsistent} \\ 0 & \text{otherwise} \end{cases}$$

Proof.

The If Part. First, we establish that $\mathcal{I}(D) = \infty$ if and only if D is inconsistent, which implies that \mathcal{I} satisfies Cons.

The satisfaction of Sub follows from the observation: if c subsumes c' and $D = (V, C \cup \{c\})$ is a consistent DTP, then $\mathcal{I}(D) = \mathcal{I}((V \cup \text{vars}(c'), C \cup \{c'\})) = 0$; if D is inconsistent, then $\mathcal{I}(D) = \infty \geq \mathcal{I}((V \cup \text{vars}(c'), C \cup \{c'\}))$.

For SA, we consider: (i) if $D \cup D'$ is consistent, then both D and D' are consistent, leading to $\mathcal{I}(D \cup D') = 0$ and $\mathcal{I}(D) + \mathcal{I}(D') = 0$; (ii) if $D \cup D'$ is inconsistent, then $\mathcal{I}(D \cup D') = \infty \geq \mathcal{I}(D) + \mathcal{I}(D')$.

The Only-If Part. Let $D = (V, C)$ be a DTP. If D is consistent, then by Cons, $\mathcal{I}(D) = 0$. Now consider that D is inconsistent. Define a mapping f that associates each constraint c in C with any two distinct constraints not in C by adding two incompatible literals: $c' = c \vee (x - y \in [l, l])$ and $c'' = c \vee (x - y \in [l + 1, l + 1])$, where x and y are arbitrary variables, and l is an integer chosen such that c' and c'' are not in C . Using Sub, it follows that $\mathcal{I}(D) = \mathcal{I}((V, C \cup \bigcup_{c \in C} f(c)))$. Additionally, by SA, $\mathcal{I}((V, C \cup \bigcup_{c \in C} f(c))) \geq \mathcal{I}(D) + \mathcal{I}(D')$ where $D' = (V, \bigcup_{c \in C} f(c))$. Given Cons and the inconsistency of D , $\mathcal{I}(D') > 0$. Therefore, if $\mathcal{I}(D) \neq \infty$, this leads to $\mathcal{I}(D) > \mathcal{I}(D)$, a contradiction. Consequently, we deduce $\mathcal{I}(D) = \infty$. \blacktriangleleft

The following proposition demonstrates the need for caution when allowing infinity as an inconsistency value.

► **Proposition 14.** *If \mathcal{I} is an inconsistency measure that satisfies Cons, and there exists a DTP D such that $\mathcal{I}(D) = \infty$, then \mathcal{I} does not satisfy PCD.*

Proof. Let $D = (V, C)$ be a DTP such that $\mathcal{I}(D) = \infty$. Consider c to be a non-free constraint within D . We define c' as a constraint not present in C but logically equivalent to c . This equivalence can be achieved by utilizing redundancy in temporal literals; for example, $c \vee l \equiv c \vee l \vee l$. Given that c is non-free in D , we obtain that c' is non-free in $D = (V, C \cup \{c'\})$. Applying PCD, we deduce that $\mathcal{I}(D) < \mathcal{I}(D')$. However, this leads to a contradiction since $\mathcal{I}(D) = \infty$. \blacktriangleleft

4 Inconsistency Measures

In this section, we present several inconsistency measures, each based on a different approach. Some measures are adaptations of those previously established in the propositional case, while others are developed by leveraging the concept of local c-relaxation.

The considered inconsistency measures are defined as follows:

- $\mathcal{I}_{\text{mcs}}(D) = \min\{|C'| : C' \subseteq C, (V, C \setminus C') \in \text{MCS}(D)\}$
- $\mathcal{I}_{\text{mis}}(D) = |\text{MIS}(D)|$
- $\mathcal{I}_{\text{p}}(D) = |C \setminus \text{Free}(D)|$
- $\mathcal{I}_{\omega}(D) = \min\{\omega(\lambda) : \lambda \in \text{LCR}(D)\}$
- $\mathcal{I}_{\theta}(D) = \min\{\theta(\lambda) : \lambda \in \text{LCR}(D)\}$

The measure $\mathcal{I}_{\text{mcs}}(D)$ quantifies the minimum number of constraints that must be removed to restore consistency. $\mathcal{I}_{\text{mis}}(D)$ counts the total number of minimal inconsistent subsets within the DTP. $\mathcal{I}_{\text{p}}(D)$ calculates the number of constraints in C that do not participate in any minimal inconsistent subset. $\mathcal{I}_{\omega}(D)$ measures the minimum weight of a local c-relaxation required to achieve consistency. $\mathcal{I}_{\theta}(D)$ determines the minimum width of a local c-relaxation necessary for restoring consistency.

■ **Table 1** Properties of inconsistency measures. ✓ means “satisfies” and ✗ means “does not satisfy”.

Measure	Cons	Mono	SCI	FCI	PCD	SDI	Sub	WSub	VIA	SA	SI
\mathcal{I}_{mcs}	✓	✓	✓	✓	✗	✓	✗	✗	✓	✓	✓
\mathcal{I}_{mis}	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓
\mathcal{I}_{p}	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓
\mathcal{I}_{ω}	✓	✓	✓	✗	✗	✓	✗	✓	✓	✓	✓
\mathcal{I}_{θ}	✓	✓	✓	✗	✗	✓	✓	✓	✗	✗	✓

In Table 1, we present the properties satisfied by each considered measure.

The initial observation is that all our measures uphold the properties Cons, Mono, SCI, SDI, and SI. Cons is fulfilled for \mathcal{I}_{mcs} because D is the unique MCS of D if and only if D is consistent; For \mathcal{I}_{mis} and \mathcal{I}_{p} , it is satisfied as a consistent DTP has no Minimal MIS; in the cases of \mathcal{I}_{ω} and \mathcal{I}_{θ} , no local c-relaxation is required to achieve consistency in consistent DTPs. Mono is observed in \mathcal{I}_{mcs} since adding a constraint cannot reduce the number of constraints needed to be ignored for consistency; it holds for \mathcal{I}_{mis} and \mathcal{I}_{p} as adding a constraint does not eliminate any existing MIS; for \mathcal{I}_{ω} and \mathcal{I}_{θ} , any sub-DTP of a consistent DTP remains consistent, which means a local c-relaxation that leads to consistency after adding a constraint will also lead to consistency when applied to the DTP prior to the addition. SCI is met as the safe constraints are not involved in any conflicts, notably, they do not require relaxation to achieve consistency. SI is applicable for \mathcal{I}_{mcs} , \mathcal{I}_{mis} , and \mathcal{I}_{p} since applying a shift does not alter the MCSes and MISes of a DTP; for \mathcal{I}_{ω} and \mathcal{I}_{θ} , it is mainly because the value $\delta(l, l')$ remains unchanged when the same shift is applied to the literals l and l' .

The second observation is that the property Sub is satisfied exclusively by the measure \mathcal{I}_{θ} . A main reason why this property is not met by the other measures stems from its implication that adding an subsumed constraint should not alter the inconsistency value. However, in the case of the first four measures, adding such a constraint can impact the situation by introducing new MISes and necessitating the relaxation of the newly added subsumed constraint. In contrast, \mathcal{I}_{θ} handles the addition of a subsumed constraint without issue, as a local c-relaxation leading to consistency does not require widening intervals after incorporating a subsumed constraint.

It should be noted that the inconsistency measure \mathcal{I}_θ satisfies the weaker variant WSub. This is because any local c-relaxation in a DTP that achieves consistency will also maintain consistency when any constraint is replaced by one of its subsumed constraints.

A key distinction between the measures \mathcal{I}_{mcs} , \mathcal{I}_{mis} , and \mathcal{I}_p on one hand, and \mathcal{I}_ω and \mathcal{I}_θ on the other, is that the latter two incorporate internal information from the constraints. This specifically accounts for why both \mathcal{I}_ω and \mathcal{I}_θ do not satisfy FCI.

► **Proposition 15.** \mathcal{I}_ω and \mathcal{I}_θ do not satisfy FCI.

Proof. Let be the DTP $D = (V = \{x_1, x_2, x_3\}, C = \{c_1, c_2, c_3, c_4\})$ defined by:

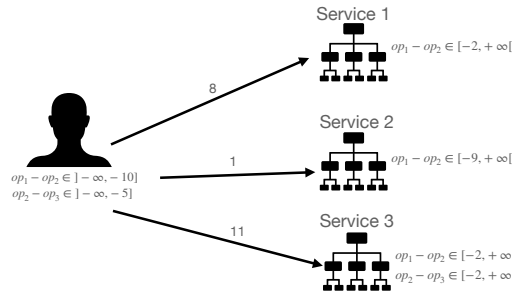
- $c_1 = x_1 - x_2 \in [5, 5] \vee x_1 - x_2 \in [20, 20]$,
- $c_2 = x_1 - x_2 \in [10, 10] \vee x_1 - x_2 \in [21, 21]$,
- $c_3 = x_2 - x_3 \in [0, 0]$,
- $c_4 = x_1 - x_3 \in [5, 10]$.

D is inconsistent and admits as unique MIS $(V, \{c_1, c_2\})$. Hence, $\text{Free}(D) = \{c_3, c_4\}$. We have $\mathcal{I}_\omega(D) = 5$ and $\mathcal{I}_\theta(D) = 3$. Moreover, we can easily see that have $\mathcal{I}_\omega((V, C \setminus \{c_4\})) = \mathcal{I}_\theta((V, C \setminus \{c_4\})) = 1$. Consequently, \mathcal{I}_ω and \mathcal{I}_θ do not satisfy the postulate FCI. ◀

► **Theorem 16.** The functions listed in Table 1 are inconsistency measures that satisfy the properties outlined in the same table.

5 Applications

In this section, we explore two applications of inconsistency measures. The core concept involves using these measures to select optimal solutions. Constraints in a DTP may represent the requirements of an individual agent or the integrity constraints of a computational service (e.g., $op_1 - op_2 \in]-\infty, 10]$ can be used to represent the constraint that operation op_2 must start no less than 10 time units after operation op_1). When conflicts arise either between the constraints of different agents or between the constraints of an agent and a service, inconsistency measures are employed to identify the most suitable resolution.



■ **Figure 1** Scenario depicting a service selection problem.

5.1 Service Selection

In the first application, we address the scenario where an agent with specific temporal constraints needs to select computational services to perform a set of operations. These services come with their own integrity constraints, which are also temporal in nature. We formally represent this situation with the tuple $\Omega = \langle V, C, S, f \rangle$, where V is a set of temporal

variables, C is a finite set of temporal constraints over V (the constraints of the considered agent), S is a set of computation services, and f is a function that assigns each service a finite set of temporal constraints (reflecting its integrity constraints).

■ **Algorithm 1** Consensus Achievement via Constraint Modification.

```

1: procedure ACHIEVECONSENSUS( $V, A, f, \mathcal{I}$ ) ▷ with  $A = \{a_1, \dots, a_k\}$ 
2:    $D_\xi \leftarrow (V, \bigcup_{a \in A} f(a))$  ▷ Define the initial DTP
3:    $i \leftarrow 1$ 
4:    $d \leftarrow \mathcal{I}(D_\xi)$ 
5:    $d_0 \leftarrow d$ 
6:   while  $D_\xi$  does not admit a solution do
7:      $D_t \leftarrow D_\xi$ 
8:     for each constraint  $c$  in  $f(a_i)$  do
9:        $D' \leftarrow D_\xi \setminus \{c\}$ 
10:      if  $I(D') < d$  then
11:         $D_t \leftarrow D'$ 
12:         $d \leftarrow I(D')$ 
13:      end if
14:    end for
15:     $D_\xi \leftarrow D_t$  ▷ Update the DTP
16:    if  $i = k$  then
17:      if  $d = d_0$  then ▷ no constraint reduces the amount of contradiction
18:         $D_\xi \leftarrow D_\xi \setminus \{c\}$  ▷  $c$  is an arbitrary constraint in  $D_\xi$ 
19:         $d \leftarrow I(D_\xi)$ 
20:      end if
21:       $d_0 \leftarrow d$ 
22:       $i \leftarrow 1$ 
23:    else
24:       $i \leftarrow i + 1$ 
25:    end if
26:  end while
27:  return  $D_\xi$ 
28: end procedure

```

In scenarios where the constraints of the considered agent clash with the integrity constraints of the computational services, inconsistency measures can be used to identify the most suitable service. This is achieved by computing $\mathcal{I}((V, C \cup f(s)))$ for each service $s \in S$. The inconsistency measures offer a quantitative assessment of the conflict severity between an agent's requirements and a service's constraints, facilitating an informed decision-making process.

For example, take the service selection problem illustrated in Figure 1. By employing the inconsistency measure \mathcal{I}_ω , the second service is identified as the most appropriate choice. This service exhibits fewer contradictions (1) with the constraints of the agent compared to others (8 and 11).

5.2 Multi-Agent Consensus

In this section, we explore an application of inconsistency measures to DTPs, aimed at facilitating consensus within a multi-agent system. This involves using these measures to strategically guide the modification of constraints, thus driving the system towards consensus.

We define a *consensus problem* as a tuple $\xi = \langle V, A, f \rangle$, where V is a set of temporal variables, A is a set of agents, and f is a function that assigns each agent in A a set of temporal constraints. We consider that there is a *consensus* if the DTP $D_\xi = (V, \bigcup_{a \in A} f(a))$ admits a solution.

Our approach to achieving consensus involves proposing that each agent, sequentially, weaken or remove one of its constraints. The critical aspect of this strategy is to provide agents with guidance on which modifications will bring them closest to consensus. This is where the role of inconsistency measures becomes crucial. More precisely, consider \mathcal{I} as the inconsistency measure in use, with $\xi = \langle V, A, f \rangle$ representing a consensus problem, and a an agent in A . An ordering \prec on the constraints in $f(a)$ can be defined as follows: $c \prec c'$ if and only if $\mathcal{I}((V, (f(a) \setminus \{c\}) \cup \bigcup_{a' \in A \setminus \{a\}} f(a')) < \mathcal{I}((V, (f(a) \setminus \{c'\}) \cup \bigcup_{a' \in A \setminus \{a\}} f(a'))$.

In Algorithm 1, we outline a variant of our approach designed to systematically achieve consensus. This algorithm iteratively removes the most problematic constraint, as determined by the inconsistency measure. This systematic elimination is designed to gradually resolve conflicts and align the system towards a solution.

6 Conclusion and perspectives

In this paper, we introduced a framework for defining inconsistency measures in Disjunctive Temporal Problems (DTPs), marking three main contributions. First, we established rationality postulates that lay foundational criteria for these measures. Second, we developed various inconsistency measures using diverse approaches. Finally, we demonstrated the applicability of these measures through two real-world applications, which underscores their potential to improve reasoning in temporal tasks.

For future work, we plan to explore additional rationality postulates to further enhance our framework. Additionally, we aim to define and investigate more inconsistency measures, expanding our current set. We also intend to assess the computational complexity of these measures and implement them.

References

- 1 James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983. doi:10.1145/182.358434.
- 2 Meriem Ammoura, Yakoub Salhi, Brahim Oukacha, and Badran Raddaoui. On an MCS-based inconsistency measure. *Int. J. Approx. Reasoning*, 80:443–459, 2017. doi:10.1016/J.IJAR.2016.06.004.
- 3 Philippe Besnard. Revisiting postulates for inconsistency measures. In *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, pages 383–396. Springer, 2014. doi:10.1007/978-3-319-11558-0_27.
- 4 Glauber De Bona, John Grant, Anthony Hunter, and Sébastien Konieczny. Classifying inconsistency measures using graphs. *J. Artif. Intell. Res.*, 66:937–987, 2019. doi:10.1613/JAIR.1.11852.
- 5 Jean-François Condotta. The augmented interval and rectangle networks. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000*, pages 571–579. Morgan Kaufmann, 2000.

- 6 Carl Corea, John Grant, and Matthias Thimm. Measuring Inconsistency in Declarative Process Specifications. In *Business Process Management - 20th International Conference, BPM 2022, Münster, Germany, September 11-16, 2022, Proceedings*, volume 13420 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2022. doi:10.1007/978-3-031-16103-2_20.
- 7 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, 1991. doi:10.1016/0004-3702(91)90006-6.
- 8 John Grant and Anthony Hunter. Measuring the good and the bad in inconsistent information. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2632–2637. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-438.
- 9 John Grant and Francesco Parisi. On measuring inconsistency in graph databases with regular path constraints. *Artif. Intell.*, 335:104197, 2024. doi:10.1016/J.ARTINT.2024.104197.
- 10 Anthony Hunter and Sébastien Konieczny. On the measure of conflicts: Shapley Inconsistency Values. *Artif. Intell.*, 174(14):1007–1026, 2010. doi:10.1016/J.ARTINT.2010.06.001.
- 11 Itay Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artif. Intell.*, 87(1-2):343–385, 1996. doi:10.1016/0004-3702(95)00109-3.
- 12 Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 494–502. Morgan Kaufmann, 2001.
- 13 Francesco Parisi and John Grant. On measuring inconsistency in definite and indefinite databases with denial constraints. *Artificial Intelligence*, 318:103884, 2023. doi:10.1016/J.ARTINT.2023.103884.
- 14 Yakoub Salhi. Inconsistency Measurement for Improving Logical Formula Clustering. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1891–1897. ijcai.org, 2020. doi:10.24963/IJCAI.2020/262.
- 15 Yakoub Salhi. Inconsistency measurement for paraconsistent inference. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2033–2039. ijcai.org, 2021. doi:10.24963/IJCAI.2021/280.
- 16 Yakoub Salhi and Michael Sioutis. A Decomposition Framework for Inconsistency Handling in Qualitative Spatial and Temporal Reasoning. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, pages 604–613, 2023. doi:10.24963/KR.2023/59.
- 17 Yakoub Salhi and Michael Sioutis. A Paraconsistency Framework for Inconsistency Handling in Qualitative Spatial and Temporal Reasoning. In *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 2049–2056. IOS Press, 2023. doi:10.3233/FAIA230498.
- 18 Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artif. Intell.*, 120(1):81–117, 2000. doi:10.1016/S0004-3702(00)00019-9.
- 19 Matthias Thimm. On the evaluation of inconsistency measures. In John Grant and Maria Vanina Martinez, editors, *Measuring Inconsistency in Information*, volume 73 of *Studies in Logic*. College Publications, February 2018.
- 20 Ioannis Tsamardinos, Thierry Vidal, and Martha E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints An Int. J.*, 8(4):365–388, 2003. doi:10.1023/A:1025894003623.

7 Appendix

The complete proofs of the satisfaction or the non satisfaction of each postulate for the inconsistency measures \mathcal{I}_ω and \mathcal{I}_θ are given in this appendix.

	\mathcal{I}_ω	\mathcal{I}_θ
Consistency (Cons)	✓ (Proposition 17)	✓ (Proposition 17)
Monotonicity (Mono)	✓ (Proposition 18)	✓ (Proposition 18)
Safe Constraint Independence (SCI)	✓ (Proposition 20)	✓ (Proposition 20)
Free Constraint Independence (FCI)	✗ (Proposition 15)	✗ (Proposition 15)
Problematic Constraint Dependence (PCD)	✗ (Proposition 21)	✗ (Proposition 21)
Sub-DTP Independence (SDI)	✓ (Proposition 22)	✓ (Proposition 22)
Subsumption (Sub)	✗ (Proposition 24)	✓ (Proposition 26)
Weak Subsumption (WSub)	✓ (Proposition 23)	✓ (Proposition 23)
VI-Additivity (VIA)	✓ (Proposition 29)	✗ (Proposition 27)
Super-Additivity (SA)	✓ (Proposition 28)	✗ (Proposition 27)
Shift Independence (SI)	✓ (Proposition 30)	✓ (Proposition 30)

► **Proposition 17.** \mathcal{I}_ω and \mathcal{I}_θ satisfy Cons.

Proof. Let $D = (V, C)$ be a DTP. Consider the particular local c-relaxation λ_{id}^D of D defined by $(\lambda_{id}^D(c))(x - y \in I) = I$ for all $c \in C$ and for all $(x - y \in I) \in \text{Lit}(c)$. Clearly, we have $\lambda_{id}^D(D) = D$. Hence, $\lambda_{id}^D \in \text{LCR}(D)$. Moreover, as $\omega(\lambda_{id}^D) = \theta(\lambda_{id}^D) = 0$ we can assert that $\mathcal{I}_\omega(D) = \mathcal{I}_\theta(D) = 0$.

Now, suppose that $\mathcal{I}_\omega(D) = 0$ or $\mathcal{I}_\theta(D) = 0$. We can assert that there exists a local c-relaxation $\lambda \in \text{LCR}(D)$ such that $\omega(\lambda) = 0$ or $\theta(\lambda) = 0$. By definition of ω and θ it follows that for all $c \in C$ and for all $l = (x - y \in I) \in \text{Lit}(c)$ we have $\delta(I, (\lambda(c))(l)) = 0$ and consequently, $I = (\lambda(c))(l)$. It results that $\lambda = \lambda_{id}^D$ and $\lambda(D) = D$. Furthermore, we know that $\lambda(D)$ is consistent since λ belongs to $\text{LCR}(D)$. It results that D is also consistent. ◀

► **Proposition 18.** \mathcal{I}_ω and \mathcal{I}_θ satisfy Mono.

Proof. Let $D = (V, C)$ and $D' = (V', C')$ be two DTPs and $\lambda \in \text{LCR}(D \cup D')$. Let λ' be the local c-transformation of D defined by $\lambda'(c) = \lambda(c)$ for all $c \in C$. Clearly, $\lambda'(c)$ is a local c-relaxation of D . Moreover, we have $\lambda'(D)$ which is consistent since $\lambda(D \cup D')$ is consistent and $\lambda'(D) \subseteq \lambda(D \cup D')$. Hence, λ' belongs to $\text{LCR}(D)$. Consequently, we have $\omega(\lambda') \geq \mathcal{I}_\omega(D)$ and $\theta(\lambda') \geq \mathcal{I}_\theta(D)$. On the other hand, by construction of λ' we can notice that $\omega(\lambda) \geq \omega(\lambda')$ and $\theta(\lambda) \geq \theta(\lambda')$. Now, suppose that λ is such that $\omega(\lambda) = \min\{\omega(\lambda'') : \lambda'' \in \text{LCR}(D \cup D')\} = \mathcal{I}_\omega(D \cup D')$ (resp. such that $\theta(\lambda) = \min\{\theta(\lambda'') : \lambda'' \in \text{LCR}(D \cup D')\} = \mathcal{I}_\theta(D \cup D')$). As $\omega(\lambda) \geq \omega(\lambda')$ (resp. $\theta(\lambda) \geq \theta(\lambda')$) and $\omega(\lambda') \geq \mathcal{I}_\omega(D)$ (resp. $\theta(\lambda') \geq \mathcal{I}_\theta(D)$), we can conclude that $\mathcal{I}_\omega(D \cup D') \geq \mathcal{I}_\omega(D)$ (resp. $\mathcal{I}_\theta(D \cup D') \geq \mathcal{I}_\theta(D)$). ◀

► **Proposition 19.** Let $D = (V, C)$ be a DTP and $c \in \text{Safe}(D)$. D is a consistent DTP iff $(V, C \setminus \{c\})$ is a consistent DTP.

Proof. Obviously, $(V, C \setminus \{c\})$ is a consistent DTP in the case where D is consistent. Now, suppose that $(V, C \setminus \{c\})$ is a consistent DTP and let us show that D is also consistent. Let σ a solution of $(V, C \setminus \{c\})$ and a variable $x \in \text{vars}(c)$ which does not belong to $\text{vars}(c')$ for all $c' \in C \setminus \{c\}$. We know that there exists in c a temporal literal of the form $x - y \in I$ (Case 1) or the form $y - x \in I$ (Case 2) with $y \in V$ and $I \in \mathcal{I}^{\mathbb{Z}}$. Let a value $a \in I$ and consider the assignment σ' of V defined by $\sigma'(u) = \sigma(u)$ for each $u \in V \setminus \{x\}$ and $\sigma'(x) = a + \sigma(y)$ if Case 1 occurs, $\sigma'(x) = \sigma(y) - a$ in the contrary case. Clearly, σ' satisfies the temporal constraints of C and is a solution of D . It results that D is consistent. ◀

► **Proposition 20.** \mathcal{I}_ω and \mathcal{I}_θ satisfy SCI.

Proof. Let $D = (V, C)$ be a DTP and $c \in C$ a safe temporal constraint of D . Let us prove that $\mathcal{I}_\omega(D) = \mathcal{I}_\omega((V, C \setminus \{c\}))$ and $\mathcal{I}_\theta(D) = \mathcal{I}_\theta((V, C \setminus \{c\}))$. As $D = (V, C \setminus \{c\}) \cup (V, \{c\})$, from Mono we have $\mathcal{I}_\omega(D) \geq \mathcal{I}_\omega((V, C \setminus \{c\}))$ and $\mathcal{I}_\theta(D) \geq \mathcal{I}_\theta((V, C \setminus \{c\}))$. Now, let us show that $\mathcal{I}_\omega(D) \leq \mathcal{I}_\omega((V, C \setminus \{c\}))$ and $\mathcal{I}_\theta(D) \leq \mathcal{I}_\theta((V, C \setminus \{c\}))$. Let $\lambda \in \text{LCR}((V, C \setminus \{c\}))$ such that $\omega(\lambda) = \min\{\omega(\lambda') : \lambda' \in \text{LCR}((V, C \setminus \{c\}))\}$ (resp. such that $\theta(\lambda) = \min\{\theta(\lambda') : \lambda' \in \text{LCR}((V, C \setminus \{c\}))\}$). Consider the local c-relaxation λ' of D defined by $(\lambda'(c))(l) = I$ for all $l = (x - y \in I) \in \text{Lit}(c)$ and $\lambda'(c') = \lambda(c')$ for all $c' \in C \setminus \{c\}$. Clearly, $\omega(\lambda') = \omega(\lambda)$ and $\theta(\lambda') = \theta(\lambda)$. Moreover, we can notice that c is a safe temporal constraint of $\lambda'(D) = (V, C')$ and $\lambda((V, C \setminus \{c\})) = (V, C' \setminus \{c\})$. From Proposition 19, it follows that $\lambda'(D)$ is consistent. Hence, $\lambda' \in \text{LCR}(D)$. Consequently, we have $\mathcal{I}_\omega(D) \leq \omega(\lambda')$ and $\mathcal{I}_\theta(D) \leq \theta(\lambda')$. From this and the fact that $\omega(\lambda') = \omega(\lambda) = \mathcal{I}_\omega((V, C \setminus \{c\}))$ and $\theta(\lambda') = \theta(\lambda) = \mathcal{I}_\theta((V, C \setminus \{c\}))$, we can assert that $\mathcal{I}_\omega(D) \leq \mathcal{I}_\omega((V, C \setminus \{c\}))$ and $\mathcal{I}_\theta(D) \leq \mathcal{I}_\theta((V, C \setminus \{c\}))$. We can conclude that $\mathcal{I}_\omega(D) = \mathcal{I}_\omega((V, C \setminus \{c\}))$ and $\mathcal{I}_\theta(D) = \mathcal{I}_\theta((V, C \setminus \{c\}))$. ◀

► **Proposition 21.** \mathcal{I}_ω and \mathcal{I}_θ do not satisfy PCD.

Proof. Let $D = (V = \{x_1, x_2, x_3\}, C = \{c_1, c_2, c_3\})$ be the DTP defined by:

- $c_1 = x_1 - x_2 \in [5, 5]$,
- $c_2 = x_1 - x_2 \in [15, 15]$,
- $c_3 = x_1 - x_2 \in [10, 10]$.

D is an inconsistent DTP. Moreover, we have $\text{MIS}(D) = \{\{c_1, c_2\}, \{c_1, c_3\}, \{c_2, c_3\}\}$ and $\text{Free}(D) = \emptyset$. On the other hand, we have $\mathcal{I}_\omega(D) = 10$ and $\mathcal{I}_\theta(D) = 5$. Now, by considering the DTP $(V, C \setminus \{c_3\})$ we have $\mathcal{I}_\omega((V, C \setminus \{c_3\})) = 10 = \mathcal{I}_\omega(D)$ and $\mathcal{I}_\theta((V, C \setminus \{c_3\})) = 5 = \mathcal{I}_\theta(D)$. From this example we can assert that \mathcal{I}_ω and \mathcal{I}_θ do not satisfy the postulate PCD. ◀

► **Proposition 22.** \mathcal{I}_ω and \mathcal{I}_θ satisfy SDI.

Proof. Let $D = (V, C)$ and $D' = (V', C')$ be two DTPs such that $V \cap V' = \emptyset$ and D' is consistent. Since \mathcal{I}_ω and \mathcal{I}_θ satisfy Mono we know that $\mathcal{I}_\omega(D \cup D') \geq \mathcal{I}_\omega(D)$ and $\mathcal{I}_\theta(D \cup D') \geq \mathcal{I}_\theta(D)$. Now, let us prove that $\mathcal{I}_\omega(D \cup D') \leq \mathcal{I}_\omega(D)$ and $\mathcal{I}_\theta(D \cup D') \leq \mathcal{I}_\theta(D)$. Let λ be a local c-relaxation of D belonging to $\text{LCR}(D)$ and let λ' be the local c-transformation of $D \cup D'$ defined by $\lambda'(c) = \lambda(c)$ for all $c \in C$ and $\lambda'(c) = c$ for all $c \in C'$. Clearly, $\lambda'(c)$ is a local c-relaxation of $D \cup D'$. Also, we have $\omega(\lambda) = \omega(\lambda')$ and $\theta(\lambda) = \theta(\lambda')$. Moreover, we can show that $\lambda'(D \cup D') = \lambda(D) \cup D'$. Since $\text{vars}(\lambda(D)) = \text{vars}(D) = V$, we have $\text{vars}(\lambda(D)) \cap \text{vars}(D') = \emptyset$. From all this and the fact that $\lambda(D)$ and D' are two consistent DTPs we can assert that $\lambda'(D \cup D') = \lambda(D) \cup D'$ is a consistent DTP. It follows that λ' belongs to $\text{LCR}(D \cup D')$. Hence, $\mathcal{I}_\omega(D \cup D') \leq \omega(\lambda') = \omega(\lambda)$ and $\mathcal{I}_\theta(D \cup D') \leq \theta(\lambda') = \theta(\lambda)$. Now, suppose that λ is such that $\omega(\lambda) = \min\{\omega(\lambda'') : \lambda'' \in \text{LCR}(D)\} = \mathcal{I}_\omega(D)$ (resp. such that $\theta(\lambda) = \min\{\theta(\lambda'') : \lambda'' \in \text{LCR}(D)\} = \mathcal{I}_\theta(D)$). With this additional property about λ we can deduce that $\mathcal{I}_\omega(D \cup D') \leq \mathcal{I}_\omega(D)$ (resp. $\mathcal{I}_\theta(D \cup D') \leq \mathcal{I}_\theta(D)$). From all this, we can conclude that $\mathcal{I}_\omega(D \cup D') = \mathcal{I}_\omega(D)$ and $\mathcal{I}_\theta(D \cup D') = \mathcal{I}_\theta(D)$. ◀

► **Proposition 23.** \mathcal{I}_ω and \mathcal{I}_θ satisfy WSub.

Proof. Let $D = (V, C)$ be a DTP and two temporal constraints c, c' such that $c \notin C$ and c subsumes c' . Let us denote by D' (resp. by D'') the DTP $(V \cup \text{vars}(c), C \cup \{c\})$ (resp. the DTP $(V \cup \text{vars}(c'), C \cup \{c'\})$). Firstly, note that in the case where $c' \in C$, we have by mono that $\mathcal{I}_\omega(D') \geq \mathcal{I}_\omega(D'')$ and $\mathcal{I}_\theta(D') \geq \mathcal{I}_\theta(D'')$ since $D' = D'' \cup (\text{vars}(c), \{c\})$. In the sequel, we will suppose that $c' \notin C$. Let λ' be a local c-relaxation belonging to $\text{LCR}(D')$ such that for

each $c'' \in C \cup \{c\}$, we have $|\{l = (x - y \in I) \in \text{Lit}(c'') : I \neq (\lambda'(c''))(l)\}| \leq 1$. Note that from Proposition 4, this last assumption is not restrictive. In the case where $\lambda'(c) = c$ we define the local c-transformation λ'' of D'' by $\lambda''(c'') = \lambda'(c'')$ for all $c'' \in C$ and for all $l \in \text{Lit}(c')$, $(\lambda''(c'))(l) = l$. In the case where $\lambda'(c) \neq c$, let $l' = (x - y \in I')$ be the temporal literal of c such that $(\lambda'(c))(l') \neq l'$ and let $l'' = (x - y \in I'')$ one temporal literal of c' such that $I' \subseteq I''$. For this case, we define the local c-transformation λ'' of D'' by $\lambda''(c'') = \lambda'(c'')$ for all $c'' \in C$, $(\lambda''(c'))(l')$ is defined by the smallest interval of $\mathcal{I}^{\mathbb{Z}}$ including $(\lambda'(c))(l') \cup I''$ and for all $l \in c'$ such that $l \neq l'$, $(\lambda''(c'))(l) = l$. Whatever the considered case and the definition of λ'' we can show that λ'' is a local c-relaxation of D'' such that $\omega(\lambda') \geq \omega(\lambda'')$, $\theta(\lambda') \geq \theta(\lambda'')$ such that $\lambda''(D'')$ is consistent (since any solution of $\lambda'(D')$ can be extended to a solution $\lambda''(D'')$). It follows that $\lambda'' \in \text{LCR}(D'')$, $\omega(\lambda') \geq \omega(\lambda'') \geq \mathcal{I}_{\omega}(D'')$ and $\omega(\lambda') \geq \omega(\lambda'') \geq \mathcal{I}_{\theta}(D'')$. Now suppose that λ' is such that $\omega(\lambda') = \mathcal{I}_{\omega}(D')$ (resp. $\theta(\lambda') = \mathcal{I}_{\theta}(D')$). With this additional assumption we can deduce that $\mathcal{I}_{\omega}(D') \geq \mathcal{I}_{\omega}(D'')$ and $\mathcal{I}_{\theta}(D') \geq \mathcal{I}_{\theta}(D'')$. ◀

► **Proposition 24.** \mathcal{I}_{ω} does not satisfy Sub.

Proof. Let $D = (V = \{x_1, x_2, x_3\}, C = \{c_1, c_2\})$ be the DTP defined by:

- $c_1 = x_1 - x_2 \in [0, 0]$,
- $c_2 = x_1 - x_2 \in [1, 1]$,
- $c_3 = x_1 - x_2 \in [4, 4]$.

D is an inconsistent DTP. Moreover, we have $\mathcal{I}_{\omega}(V, C) = 4$. Now consider the constraint $c_4 = x_1 - x_2 \in [4, 5]$. Clearly, c_3 subsumes c_4 . On the other hand, we have $\mathcal{I}_{\omega}(V, C \cup \{c_4\}) = \mathcal{I}_{\omega}(V, C) = 4$ and $\mathcal{I}_{\omega}(V, C \cup \{c_4\}) = \mathcal{I}_{\omega}(V, \{c_1, c_2, c_3, c_4\}) = 7$. From this example we can assert that \mathcal{I}_{ω} does not satisfy the postulate Sub. ◀

► **Proposition 25.** Let $D = (V, C)$ be a DTP and two temporal constraints c, c' such that $c \in C$ and c subsumes c' . We have $\mathcal{I}_{\theta}(D) = \mathcal{I}_{\theta}((V \cup \text{vars}(c'), C \cup \{c'\}))$.

Proof. In the case where $c' \in C$, the property is obvious, in the sequel we will suppose that $c' \notin C$. By Mono (Proposition 18) we know that $\mathcal{I}_{\theta}(D) \leq \mathcal{I}_{\theta}((V \cup \text{vars}(c'), C \cup \{c'\}))$. We will show that $\mathcal{I}_{\theta}(D) \geq \mathcal{I}_{\theta}((V \cup \text{vars}(c'), C \cup \{c'\}))$. Let λ be a local c-relaxation of $\text{LCR}(D)$ such that $\theta(\lambda) = \min\{\theta(\lambda'') : \lambda'' \in \text{LCR}(D)\} = \mathcal{I}_{\theta}(D)$. Moreover we suppose that for each $c'' \in C \cup \{c\}$, we have $|\{l = (x - y \in I) \in \text{Lit}(c'') : I \neq (\lambda(c''))(l)\}| \leq 1$. Note that, from Proposition 4, this last assumption is not restrictive. We define the local c-relaxation λ' of $(V \cup \text{vars}(c'), C \cup \{c'\})$ in the following way. In the case where $|\{l = (x - y \in I) \in \text{Lit}(c) : I \neq (\lambda(c))(l)\}| = 0$ (Case 1), $(\lambda'(c''))(l) = (\lambda(c''))(l)$ for all $c'' \in C$ and $l \in \text{Lit}(c')$. Moreover, $(\lambda'(c'))(l) = I$ for all $l = (x - y \in I) \in \text{Lit}(c')$. In the case where $|\{l = (x - y \in I) \in \text{Lit}(c) : I \neq (\lambda(c))(l)\}| = 1$ (Case 2), let $l' = (x' - y' \in I')$ be the literal of c such that $(\lambda(c))(l') = (x' - y' \in I') \neq I'$ and $l'' = (x' - y' \in I'')$ be a literal of c' such that $I' \subseteq I''$. For this case, λ' is defined by $(\lambda'(c''))(l) = (\lambda(c''))(l)$ for all $c'' \in C$ and $l \in \text{Lit}(c')$, $(\lambda'(c'))(l) = I$ for all $l = (x - y \in I) \in \text{Lit}(c') \setminus \{l'\}$ and $(\lambda'(c'))(l'')$ is defined by the smallest interval of $\mathcal{I}^{\mathbb{Z}}$ including $\lambda(c)(l') \cup I''$. Whatever the definition of λ' we have $\theta(\lambda') = \theta(\lambda)$ and $\lambda' \in \text{LCR}((V \cup \text{vars}(c'), C \cup \{c'\}))$. It follows that $\theta(\lambda) \geq \mathcal{I}_{\theta}((V \cup \text{vars}(c'), C \cup \{c'\}))$. Consequently, $\mathcal{I}_{\theta}(D) \geq \mathcal{I}_{\theta}((V \cup \text{vars}(c'), C \cup \{c'\}))$. ◀

► **Proposition 26.** \mathcal{I}_{θ} satisfies Sub.

Proof. Let $D = (V, C)$ be a DTP and two temporal constraints c, c' such that c subsumes c' . We have two cases that arise: $c \notin C$ or $c \in C$. By considering the case $c \notin C$, from WSub (Proposition 23), we know that $\mathcal{I}_{\theta}((V \cup \text{vars}(c), C \cup \{c\})) \geq \mathcal{I}_{\theta}((V \cup \text{vars}(c'), C \cup \{c'\}))$. Now, consider the case $c \in C$. For this case $\mathcal{I}_{\theta}((V \cup \text{vars}(c), C \cup \{c\})) = \mathcal{I}_{\theta}(V, C)$. Moreover,

from Proposition 25, we can assert that $\mathcal{I}_\theta((V, C)) = \mathcal{I}_\theta((V \cup \text{vars}(c'), C \cup \{c'\}))$. From all this, we can conclude that $\mathcal{I}_\theta((V \cup \text{vars}(c), C \cup \{c\})) \geq \mathcal{I}_\theta((V \cup \text{vars}(c'), C \cup \{c'\}))$ and that \mathcal{I}_θ satisfies Sub. \blacktriangleleft

► **Proposition 27.** \mathcal{I}_θ does not satisfy VIA and SA.

Proof. Let $D = (V = \{x_1, x_2, x_3, x_4\}, C = \{c_1, c_2, c_3, c_4\})$ be the DTP defined by:

- $c_1 = x_1 - x_2 \in [0, 0]$,
- $c_2 = x_1 - x_2 \in [1, 1]$,
- $c_3 = x_3 - x_4 \in [0, 0]$,
- $c_4 = x_3 - x_4 \in [1, 1]$.

Consider the two DTPs $D = (V = \{x_1, x_2\}, C = \{c_1, c_2\})$ and $D' = (V' = \{x_1, x_2\}, C' = \{c_3, c_4\})$. We have $V \cap V' = \emptyset$ and $C \cap C' = \emptyset$. Moreover, $\mathcal{I}_\theta(D \cup D') = 1$, $\mathcal{I}_\theta(D) = 1$ and $\mathcal{I}_\theta(D') = 1$. From this example we can assert that \mathcal{I}_θ do not satisfy the postulates VIA and SA. \blacktriangleleft

► **Proposition 28.** \mathcal{I}_ω satisfies SA.

Proof. Let $D = (V, C)$ and $D' = (V', C')$ be two DTPs such that $C \cap C' = \emptyset$. Let λ'' be a local c-relaxation of $\text{LCR}(D \cup D')$ such that $\omega(\lambda'') = \min\{\omega(\lambda''') : \lambda''' \in \text{LCR}(D \cup D')\} = \mathcal{I}_\omega(D \cup D')$. We define the local c-relaxation λ (resp. λ') of D (resp. of D') by $\lambda(c) = \lambda''(c)$ (resp. $\lambda'(c) = \lambda''(c)$) for all $c \in C$ (resp. for all $c' \in C'$). Clearly, since $C \cap C' = \emptyset$ we have $\omega(\lambda'') = \omega(\lambda) + \omega(\lambda')$. Moreover we have can show that $\lambda \in \text{LCR}(D)$ and $\lambda' \in \text{LCR}(D')$. Hence, $\omega(\lambda) \geq \mathcal{I}_\omega(D)$ and $\omega(\lambda') \geq \mathcal{I}_\omega(D')$. It results that $\mathcal{I}_\omega(D \cup D') = \omega(\lambda'') = \omega(\lambda) + \omega(\lambda') \geq \mathcal{I}_\omega(D) + \mathcal{I}_\omega(D')$. \blacktriangleleft

► **Proposition 29.** \mathcal{I}_ω satisfies VIA.

Proof. Let $D = (V, C)$ and $D' = (V', C')$ be two DTPs such that $V \cap V' = \emptyset$. As $V \cap V' = \emptyset$ we have $C \cap C' = \emptyset$. From SA (Proposition 28) we know that that $\mathcal{I}_\omega(D \cup D') \geq \mathcal{I}_\omega(D) + \mathcal{I}_\omega(D')$. Let us prove that $\mathcal{I}_\omega(D \cup D') \leq \mathcal{I}_\omega(D) + \mathcal{I}_\omega(D')$. Let λ be a local c-relaxation of $\text{LCR}(D)$ such that $\omega(\lambda) = \min\{\omega(\lambda''') : \lambda''' \in \text{LCR}(D)\} = \mathcal{I}_\omega(D)$ and let λ' be a local c-relaxation of $\text{LCR}(D')$ such that $\omega(\lambda') = \min\{\omega(\lambda''') : \lambda''' \in \text{LCR}(D')\} = \mathcal{I}_\omega(D')$. Let the local c-transformation λ'' defined by $\lambda''(c) = \lambda(c)$ for all $c \in C$ and $\lambda''(c) = \lambda'(c)$ for all $c \in C'$. We can show that $\omega(\lambda'') = \omega(\lambda) + \omega(\lambda')$ (since $C \cap C' = \emptyset$) and $\lambda'' \in \text{LCR}(D \cup D')$. This last belonging comes from the fact that $\lambda''(D \cup D') = \lambda(D) \cup \lambda'(D')$, $V \cap V' = \emptyset$, $\lambda(D)$ is a consistent DTP and $\lambda'(D')$ is a consistent DTP. It results that $\mathcal{I}_\omega(D \cup D') \leq \omega(\lambda'') = \omega(\lambda) + \omega(\lambda') = \mathcal{I}_\omega(D) + \mathcal{I}_\omega(D')$. From all this, we can conclude that $\mathcal{I}_\omega(D \cup D') = \mathcal{I}_\omega(D) + \mathcal{I}_\omega(D')$. \blacktriangleleft

► **Proposition 30.** \mathcal{I}_ω and \mathcal{I}_θ satisfies SI.

Proof. Let $D = (V, C)$ be a DTP and $k \in \mathbb{Z}$. Let λ be a local c-relaxation of $\text{LCR}(D)$ such that $\omega(\lambda) = \min\{\omega(\lambda'') : \lambda'' \in \text{LCR}(D)\} = \mathcal{I}_\omega(D)$ and let λ' be a local c-relaxation of $\text{LCR}(D)$ such that $\theta(\lambda') = \min\{\theta(\lambda'') : \lambda'' \in \text{LCR}(D)\} = \mathcal{I}_\theta(D)$. From λ and λ' we define the two local c-relaxations of $D \oplus k$ λ'' and λ''' in the following way: $(\lambda''(c))(l) = (\lambda(c))(l) \oplus k$ and $(\lambda'''(c))(l) = (\lambda'(c))(l) \oplus k$ for all $c \in C$ and $l \in \text{Lit}(c)$. We can show that $\omega(\lambda) = \omega(\lambda'')$ and $\theta(\lambda') = \theta(\lambda''')$. We can also show that $\lambda''(D \oplus k)$ and $\lambda'''(D \oplus k)$ are consistent. It follows that λ'' and λ''' belong to $\text{LCR}(D \oplus k)$. It results that $\mathcal{I}_\omega(D) = \omega(\lambda) = \omega(\lambda'') \geq \mathcal{I}_\omega(D \oplus k)$ and $\mathcal{I}_\theta(D) = \theta(\lambda') = \theta(\lambda''') \geq \mathcal{I}_\theta(D \oplus k)$.

Now, let us prove that $\mathcal{I}_\omega(D) \leq \mathcal{I}_\omega(D \oplus k)$ and $\mathcal{I}_\theta(D) \leq \mathcal{I}_\theta(D \oplus k)$. Let λ be a local c-relaxation of $\text{LCR}(D \oplus k)$ such that $\omega(\lambda) = \min\{\omega(\lambda'') : \lambda'' \in \text{LCR}(D \oplus k)\} = \mathcal{I}_\omega(D \oplus k)$ and let λ' be a local c-relaxation of $D \oplus k$ such that $\theta(\lambda') = \min\{\theta(\lambda'') : \lambda'' \in \text{LCR}(D \oplus k)\} =$

15:18 A Framework for Assessing Inconsistency in Disjunctive Temporal Problems

$\mathcal{I}_\theta(D \oplus k)$. From λ and λ' we define the two c-relaxation of D λ'' and λ''' in the following way: $(\lambda''(c))(l) = (\lambda(c))(l) \oplus (-k)$ and $(\lambda'''(c))(l) = (\lambda'(c))(l) \oplus (-k)$ for all $c \in C$ and $l \in \text{Lit}(c)$. We can show that $\omega(\lambda) = \omega(\lambda'')$ and $\theta(\lambda') = \theta(\lambda''')$. We can also show that $\lambda''(D)$ and $\lambda'''(D)$ are consistent. It follows that λ'' and λ''' belong to $\text{LCR}(D)$. It results that $\mathcal{I}_\omega(D \oplus k) = \omega(\lambda) = \omega(\lambda'') \geq \mathcal{I}_\omega(D)$ and $\mathcal{I}_\theta(D \oplus k) = \theta(\lambda') = \theta(\lambda''') \geq \mathcal{I}_\theta(D)$. From all this, we can conclude that $\mathcal{I}_\omega(D \oplus k) = \mathcal{I}_\omega(D)$ and $\mathcal{I}_\theta(D \oplus k) = \mathcal{I}_\theta(D)$. ◀

Real-Time Higher-Order Recursion Schemes

Eric Alsmann  

University of Kassel, Germany

Florian Bruse  

University of Kassel, Germany

Abstract

Higher-Order Recursion Schemes (HORS) have long been studied as a tool to model functional programs. Model-checking the tree generated by a HORS of order k against a parity automaton is known to be k -EXPTIME-complete. This paper introduces timed HORS, a real-time version of HORS in the sense of Alur/Dill’90, to be model-checked against a pair of a parity automaton and a timed automaton. We show that adding dense linear time to the notion of recursion schemes adds one exponential to the cost of model-checking, i.e., model-checking a timed HORS of order k can be done in $(k + 1)$ -EXPTIME. This is shown by an adaption of the region-graph construction known from the model-checking of timed CTL. We also obtain a hardness result for $k = 1$, but we strongly conjecture that it holds for all k . This result is obtained by encoding runs of 2-EXPTIME Turing machines into the trees generated by timed HORS.

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

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

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.16

1 Introduction

Higher-Order Recursion Schemes (HORS) are a well-studied framework in the context of infinite-state verification [13, 16]. A HORS is a higher-order grammar that generates a tree, e.g., the syntax tree of a functional program. The question of HORS model-checking is to decide whether a given alternating parity tree-automaton (APT) accepts the tree generated by a given HORS. This is known to be decidable in k -EXPTIME for recursion schemes of order k [20, 19]. In fact, the problem is k -EXPTIME-complete. Competitive model-checkers for this problem exist [17, 6, 5], which makes this problem practically feasible even though the theoretical complexity is high.

Another, seemingly unrelated, branch of verification of complex systems concerns real-time systems. These are systems that model time not in the discrete, transition-based fashion known from e.g., the modal μ -calculus, but rather via dense linear time. Typical verification questions then concern not the possibility of a given action, but, for example, whether the action can happen in a given timeframe as in “any request is granted in at most 5 milliseconds.” A standard model for real-time verification are timed automata (TA) [2], which serve as a finite representation of an infinite system that employs dense real time. Model-checking the systems generated by timed automata is well-understood for specification logics such as timed CTL, a real-time version of the well-known temporal logic CTL [3, 4].

A recent introduction to the world of real-time verification is Timed Recursive CTL [10], which extends the specification power of timed CTL to higher-order properties, including non-regular ones. The consequence of adding such an amount of expressive power is the increase of the complexity of the model-checking problem to 2-EXPTIME.

In this paper, we follow a different approach at real-time higher-order verification, namely by making the *system* to be verified both real-time and higher-order. This is done by extending the notion of HORS to that of timed HORS, i.e., HORS annotated by real-time



© Eric Alsmann and Florian Bruse;

licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 16; pp. 16:1–16:20

Leibniz International Proceedings in Informatics



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

information. The trees generated by timed HORS are to be model-checked simultaneously against an APT and a TA, both of which are loosely synchronized to the other. To our knowledge, this is the first introduction of real-time verification to the world of HORS.

The paper both introduces the concept of timed HORS, the respective model-checking problem, and studies the complexity of said problem. We show that adding real-time expressions to HORS increases the complexity of the model-checking problem by one exponential. The upper bound is shown for HORS of all orders, while the lower bound is shown only for order-1 HORS, due to space considerations. We are confident that the hardness result can be established for all orders due to parallels between the model-checking problems for HORS and HFL, the underlying theory of timed recursive CTL [18].

The structure of the paper is as follows: In Sect. 2, we recall the notions of HORS, their model-checking problem, as well as timed automata. In Sect. 3, we introduce timed HORS and the associated model-checking problem, supported by an example. In Sect. 4, we show that the model-checking problem for order- k timed HORS can be solved in $(k + 1)$ -fold exponential time, using an exponential reduction to the model-checking problem for untimed HORS. In Sect. 5, we establish a matching lower bound for $k = 1$, and we conclude with some remarks on further research in Sect. 6.

2 Preliminaries

We write $[n]$ for the set $\{1, \dots, n\}$. Let 2_0^n denote n if $k = 0$ and let 2_{k+1}^n denote 2^{2^k} . We use notation of the form $(t_1, _, t_3)$ with $_$ indicating that the value in question is not important, but exists.

2.1 Trees, Games, and Automata

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

A *parity game* $\mathcal{G} = (V, V_\exists, E, v_0, \Omega)$ is a game between \exists and \forall . Here, (V, E) is a directed graph, $V_\exists \subseteq V$ is the set of nodes owned by \exists , $v_0 \in V$ is the starting position, and $\Omega: V \rightarrow \mathbb{N}$ is the priority function with finite codomain. We write V_\forall for $V \setminus V_\exists$.

A *play* of \mathcal{G} is a sequence of nodes in V , connected by E and starting in v_0 . A finite play is extended by the player who owns the last node in the play by picking a successor of this node. If this is not possible, the player loses the game. A *maximal* play is either infinite or cannot be extended further. \exists wins an infinite play v_0, \dots if the maximal number that occurs infinitely often in the sequence $\Omega(v_0), \dots$ is even, \forall wins if it is odd. Positional strategies are defined as usual. A player wins a parity game iff they have a positional strategy for it. Computing who wins a parity game can be done in time $\mathcal{O}(p \cdot |E| \cdot |V|^{\lfloor \frac{p}{2} \rfloor})$ for a game with at most p priorities [15].

Let S be a set. By $\mathcal{B}^+(S)$ we denote the set of *positive Boolean expressions* over S , derived from the grammar $\varphi := s \mid \perp \mid \top \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$ with $s \in S$.

An *alternating parity tree-automaton* (APT) is a $\mathcal{P} = (Q, \Sigma, \delta, q_I, \Lambda)$ where Q is a finite, nonempty set of *states*, Σ is a tree alphabet containing a special nullary symbol ω , $\delta = \bigcup_{i \leq n} \delta^i$ is the *transition function* with $\delta^i: Q \times \Sigma_i \rightarrow \mathcal{B}^+(Q^i)$ and n being the maximal arity that occurs in Σ . We write $\delta(q, l(v))$ for $\delta^i(q, l(v))$ if $ar(l(v)) = i$. Finally, $q_I \in Q$ is the *starting state* and $\Lambda: Q \rightarrow \mathbb{N} \setminus \{0\}$ is the *priority function*, which does not assign value 0 to any state unless explicitly stated otherwise. The *size* of an APT is the number of its states.

A *run* of an APT \mathcal{P} on some Σ -tree T (for matching Σ) is a parity game $G(\mathcal{P}, T)$ called the *acceptance game*. It has positions from $T \times (Q \cup \bigcup_{i \leq n} \mathcal{B}^+(Q^i))$ where n is the maximal arity in Σ . Its starting position is (ϵ, q_I) . In a position of the form (v, q) , the unique successor is $(v, \delta(q, l(v)))$. Positions of the forms $(v, \varphi_1 \vee \varphi_2)$ and $(v, \varphi_1 \wedge \varphi_2)$ have two successors, namely (q, φ_1) and (q, φ_2) . A position of the form $(v, (q_0, \dots, q_{i-1}))$ has i successors $(v_0, q_0), \dots, (v_{i-1}, q_{i-1})$, a position of the form (v, \top) or (v, \perp) has no successors. Positions of the forms (v, \top) and $(v, \varphi_1 \wedge \varphi_2)$ and $(v, (q_0, \dots, q_{i-1}))$ belong to \forall , all other positions belong to \exists . Finally, $\Omega(v, q) = \Lambda(q)$; for all other positions we have $\Omega(v, \varphi) = 1$. \mathcal{P} *accepts* a tree T if \exists wins $G(\mathcal{P}, T)$.

2.2 Higher-Order Recursion Schemes

The following is quite terse, see e.g., [20] for more details.

The set of *types* is defined inductively via $\tau := \bullet \mid \tau \rightarrow \tau$ where \bullet is the type of trees, $\bullet \rightarrow \bullet$ is the type of functions that map trees to trees, etc. The *order* of a type is defined as $\text{ord}(\bullet) = 0$ and $\text{ord}(\tau_1 \rightarrow \tau_2) = \max\{1 + \text{ord}(\tau_1), \text{ord}(\tau_2)\}$. We write $\tau^i \rightarrow \tau'$ to denote the type $\tau \rightarrow \tau \rightarrow \dots \rightarrow \tau \rightarrow \tau'$ with i repetitions of τ .

Fix a tree alphabet Σ . Let $\mathcal{V} = \{x, y, \dots\}$ be a set of typed *variables*. We write $x: \tau$ to denote that x has type τ . The set of \mathcal{V} -terms over Σ is defined inductively: for each i -ary symbol $a \in \Sigma$, the *tree constructor* a is a term of type $\bullet^i \rightarrow \bullet$, and a variable $x \in \mathcal{V}$ of type τ is a term of type τ . Given terms t_1, t_2 of types $\tau_1 \rightarrow \tau_2$ and τ_1 , $(t_1 t_2)$ is a term of type τ_2 . We write $t: \tau$ to denote the type of a term. All terms are *subterms* of themselves; moreover t_1 and t_2 are subterms of $(t_1 t_2)$. We use standard conventions to reduce the number of parentheses, i.e., application associates to the left. The *order* of a subterm is the order of its type; the order of a term is the maximal order of any of its subterms. By $t[t_1/x_1, \dots, t_n/x_n]$ we denote the simultaneous capture-avoiding substitution of the t_i for all occurrences of the x_i in t . Here, we assume that the types of the term and the variable it is substituted for do match.

A higher-order recursion scheme (HORS) is a $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ such that

- Σ is a tree alphabet,
- \mathcal{N} is a set of typed nonterminals; we write $N: \tau$ to denote that N has type τ ,
- \mathcal{R} is a map from \mathcal{N} to the set of terms such that if N has type $\tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \bullet$, then $\mathcal{R}(N)$ is an $\{x_1, \dots, x_n\} \cup \mathcal{N}$ -term of type \bullet where the variables x_1, \dots, x_n are unique to N , and
- $S: \bullet \in \mathcal{N}$ is the starting symbol.

The order of a recursion scheme is the maximal order of the type of any of its nonterminals. The size $|\mathcal{G}|$ of a HORS $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ is defined as the number of distinct subterms on the right-hand-sides in \mathcal{R} .

Define a rewriting relation $\rightarrow_{\mathcal{G}}$ via

- $N t_1 \dots t_n \rightarrow_{\mathcal{G}} \mathcal{R}(N)[t_1/x_1, \dots, t_n/x_n]$ if $t_i: \tau_i$ for all i and $N: \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \bullet$,
- $a t_1 \dots t_n \rightarrow_{\mathcal{G}} a t'_1 \dots t'_n$ if $a: \bullet^n \rightarrow \bullet$ and $t_i \rightarrow_{\mathcal{G}} t'_i$ for all i .

Note that this defines \mathcal{N} -terms. For a given \mathcal{N} -term t , we define the tree t^ω generated by it as follows: $(a t_1 \dots t_n)^\omega$ is $a t_1^\omega \dots t_n^\omega$, and $(N t_1 \dots t_n)^\omega$ is ω where $\omega \notin \Sigma$ is a nullary constructor. Let $(T_i, l_i) = t_i^\omega$ for $i \in \{1, 2\}$. We write $t_1^\omega \sqsubseteq t_2^\omega$ if $T_1 \subseteq T_2$ and, for all $v \in T_1$, either $l_1(v) = \omega$ or $l_1(v) = l_2(v)$. Write $T(\mathcal{G})$ for the tree generated by a recursion scheme \mathcal{G} , defined via $\bigsqcup \{t^\omega \mid S \rightarrow_{\mathcal{G}} t\}$. This tree is well-defined due to confluence of the lambda-calculus [14]. By abuse of notation, we also write $T(t)$ for the tree generated by a closed term $t: \bullet$. We assume for the rest of the paper that all HORS in question generate trees that do not contain ω in order to ease notation.

The problem of model-checking higher-order recursion schemes is now the following: given a HORS \mathcal{G} over Σ and an APT \mathcal{P} over $\Sigma \cup \{\omega\}$, does \mathcal{P} accept $T_{\mathcal{G}}$? This problem is known to be complete for k -fold exponential time for schemes of order k [20].

2.3 Timed Automata

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

A clock evaluation is a mapping $\eta: \mathcal{X} \rightarrow \mathbb{R}^{\geq 0}$; it satisfies a clock constraint if

- $\eta \models \top$ always,
- $\eta \models x \oplus c$ iff $\eta(x) \oplus c$,
- $\eta \models \varphi_1 \wedge \varphi_2$ iff $\eta \models \varphi_1$ and $\eta \models \varphi_2$.

For a clock evaluation η and $d \in \mathbb{R}^{\geq 0}$, we write $\eta+d$ for the clock evaluation defined via $(\eta+d)(x) = \eta(x) + d$ for all $x \in \mathcal{X}$. For $R \subseteq \mathcal{X}$, $\eta|_R$ is the clock evaluation defined via $\eta|_R(x) = \eta(x)$ if $x \notin R$ and $\eta|_R(x) = 0$ if $x \in R$. For singleton sets $\{x\}$, we write $\eta|_x$ for $\eta|_{\{x\}}$.

Let *Prop* be a finite set of *propositions*. A *timed automaton* over clocks in \mathcal{X} and with propositions in *Prop* is an $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \Delta, \lambda)$ where

- L is the set of so-called *locations* of the timed automaton, including the *initial location* ℓ_0 ,
- \mathcal{X} is a finite set of clocks,
- $\iota: L \rightarrow CC(\mathcal{X})$ assigns a clock constraint called an *invariant* to each location,
- $\Delta \subseteq L \times CC(\mathcal{X}) \times 2^{\mathcal{X}} \times L$ is a finite set of transitions; we write $\ell \xrightarrow{g,R} \ell'$ for $(\ell, g, R, \ell') \in \Delta$.

In such a transition, g is the *guard* and R are the *resets* of the transition,

- $\lambda: L \rightarrow 2^{Prop}$ labels each location with the propositions valid there.

The *index* of a timed automaton \mathcal{A} is the largest constant that occurs in its guards or invariants. Its size is defined as

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

due to the coding of the constants in clock constraints in binary.

A TA \mathcal{A} defines a so-called *timed transition system* (tTS) $(\mathcal{S}, \rightarrow, s_0, \lambda)$ over pairs of locations and clock evaluations as follows:

- The state set of the system is $\mathcal{S} = \{(\ell, \eta) \in L \times (\mathcal{X} \rightarrow \mathbb{R}^{\geq 0}) \mid \eta \models \iota(\ell)\}$.
- The initial state s_0 is (ℓ_0, η_0) where $\eta_0(x) = 0$ for all $x \in \mathcal{X}$.
- *Delay transitions* keep the location but let time flow: for any $(\ell, \eta) \in \mathcal{S}$ and $d \in \mathbb{R}^{\geq 0}$ we have a transition $(\ell, \eta) \xrightarrow{d} (\ell, \eta + d)$ if $\eta + d' \models \iota(\ell)$ for all $0 \leq d' \leq d$.
- *Discrete transitions* are derived from Δ : For all $(\ell, \eta) \in \mathcal{S}$, $\ell' \in L$ and $R \subseteq \mathcal{X}$, we have $(\ell, \eta) \rightarrow (\ell', \eta|_R)$ if there is $g \in CC(\mathcal{X})$ with $(\ell, g, R, \ell') \in \Delta$, $\eta \models g$ and, finally, $\eta|_R \models \iota(\ell')$.
- The propositional labeling λ feeds through the labeling of a location, i.e., $\lambda(\ell, \eta) = \lambda(\ell)$.
- Moreover, we consider all states labeled by the clock constraints that hold there, i.e., $(\ell, \eta) \models \chi$ iff $\eta \models \chi$.

Note that the system defined by a TA is generally neither finite nor countable, both due to the uncountable state set and the infinitary labeling of states by arbitrary clock constraints. Also note that it contains an uncountable number of transition relations, namely the anonymous transition relation that stems from discrete transitions, as well as a transition relation d for each $d \in \mathbb{R}^{\geq 0}$, stemming from delay transitions.

We say that *time can pass* from some state (ℓ_0, η_0) by an amount of $d \in \mathbb{R}^{\geq 0}$ if there is a *Trace* in the system of the form $(\ell_0, \eta_0), (\ell_1, \eta_1), \dots, (\ell_n, \eta_n)$ such that, (I) for each $0 \leq i < n$, either $(\ell_i, \eta_i) \rightarrow (\ell_{i+1}, \eta_{i+1})$ or $(\ell_i, \eta_i) \xrightarrow{d'} (\ell_{i+1}, \eta_{i+1})$ for some $d' \in \mathbb{R}^{\geq 0}$, and (II) the sum of the delay transitions on this path is d . A state is a *timelock* if time cannot pass from this state, neither directly nor after some discrete transitions.

3 Real-Time Recursion Schemes

Let Σ be a tree alphabet. Its *timed version* is $\Sigma^t = \Sigma \times \mathcal{J}$ for \mathcal{J} a finite set of intervals with natural bounds plus infinity, where an element (a, J) is written a_J . Hence, symbols in Σ^t are of the form $a_{[2, \infty)}$ or $b_{(3, 5]}$ or $c_{[2, 2]}$.

A *timed HORS* is a recursion scheme over a timed alphabet Σ^t . Hence, a timed HORS is a recursion scheme over a tree alphabet where the terminals, or tree constructors, are each labeled with an interval. The size of a timed HORS \mathcal{G} is the size of its untimed version (obtained via the mapping $a_J \mapsto a$) times the logarithm of the largest finite interval bound mentioned by \mathcal{G} .

Since a timed HORS generates a tree just as an untimed HORS, this tree can be model-checked against an APT, either by simply ignoring the interval annotations (again, via the mapping $a_J \mapsto a$), or by treating Σ^t as an ordinary tree alphabet. However, the purpose of a timed HORS is to be validated against a pair of a (timed) APT and a TA, which are loosely synchronized. Before we give an intuition, we define timed APT.

Let \mathcal{X} be a set of clocks and let \mathcal{A} be a TA over \mathcal{X} with propositions in Q . Let Ξ be a finite set of clock constraints over \mathcal{X} . A *timed Ξ -APT* over Σ^ω is a $(Q, \Sigma, \delta, q_I, \Lambda)$ where Q, q_I, Λ are as for ordinary APT and δ is the union of the $\delta^i: Q \times 2^\Xi \times \Sigma_i \rightarrow \mathcal{B}^+(Q^i)$, i.e., the transition function consumes a state, an untimed symbol from Σ , and a subset of Ξ . Note that the state set of the timed APT is exactly the set of propositions used in the TA. The size of a timed APT is $|Q| \cdot 2^{|\Xi|} \cdot \log k$, where k is the biggest clock constraint in Ξ .

Let \mathcal{G} be a timed HORS over Σ^t , let \mathcal{A} be a timed TA over clocks in \mathcal{X} , and let \mathcal{P} be a timed Ξ -APT with states in Q with Ξ a set of clock constraints over \mathcal{X} . The semantics of an APT-TA pair over the tree generated by a timed HORS is explained in terms of an acceptance game. This works similarly to the acceptance game for ordinary, untimed APT, but with an extra component for the TA. Hence, a position of the game is defined by a node in the tree generated by the timed HORS, a state of the APT and a state of the tTS defined by the TA, i.e., a location of the TA and a suitable clock evaluation.

A play of the game proceeds like this: In a position as per above, first \exists must let time flow in the tTS by an amount dictated by the labeling a_J of the current tree node. Moreover, while letting time flow, she may visit only locations of the TA whose propositional labeling includes the current state of the APT. If she cannot let time flow like this, she loses the acceptance game. Once \exists has let time flow, the transition function of the timed APT is consulted. This function consumes the current state of the APT, the labeling of the current tree node, and, additionally, the clock constraints from Ξ valid in the state of the tTS defined by the TA. The transition function is then played out as for ordinary APT. Hence, the APT and the TA are synchronized in the sense that time flow in the tTS defined by the TA is restricted by the current state of the APT, while the behavior of the APT, in turn, depends on the clock constraints valid in the current state of the tTS.

Formally, the semantics of a Ξ -APT \mathcal{P} and its *companion TA* \mathcal{A} (over a suitable set of clocks) over a timed HORS \mathcal{G} is defined as a parity game $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$:

16:6 Real-Time Higher-Order Recursion Schemes

- The positions of the game are of the form $(v, \varphi, (\ell, \eta), b)$ where v is a node in $T_{\mathcal{G}}$, $\varphi \in Q \cup \bigcup_i \mathcal{B}(Q^i)$ is a subformula of the transition function of the timed APT, (ℓ, η) is a state in the timed transition system defined by \mathcal{A} , and b is a bit, indicating whether time has flown already in this position.
 - The initial position is $(\epsilon, q_I, (\ell_0, \eta_0), 0)$ where $\eta_0(\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathcal{X}$.
 - A position of the form $(v, q, (\ell, \eta), 0)$ is owned by \exists . If $l(v) = a_J$, any position of the form $(v, q, (\ell', \eta'), 1)$ is a successor position, provided (ℓ', η') can be reached in the tTS from (ℓ, η) by letting time flow for an amount within J , but by visiting *only locations where the propositional labeling includes q* .
 - A position of the form $(v, q, (\ell, \eta), 1)$ is owned by \exists . The unique successor is the position $(v, \varphi, (\ell, \eta), 1)$ where $\varphi = \delta(q, S, a)$ with $a_J = l(v)$ and $S = \{\chi \in \Xi \mid \eta \models \chi\}$.
 - A position of the form $(v, \varphi_1 \vee \varphi_2, (\ell, \eta), 1)$ is owned by \exists , and one of the form $(v, \varphi_1 \vee \varphi_2, (\ell, \eta), 1)$ is owned by \forall ; its successors are $(v, \varphi_i, (\ell, \eta), 1)$ for $i \in \{1, 2\}$.
 - A position of the form $(v, (q_0, \dots, q_{k-1}), (\ell, \eta), 1)$ is owned by \forall . Its successors are $(v_i, q_i, (\ell, \eta), 0)$ for $0 \leq i < k$.
 - The priority of a position of the form $(v, q, (\ell, \eta), 0)$ is $\Lambda(q)$, and 1 for the other positions.
- We say that $(\mathcal{A}, \mathcal{P})$ accepts $T(\mathcal{G})$ if \exists wins the above parity game.

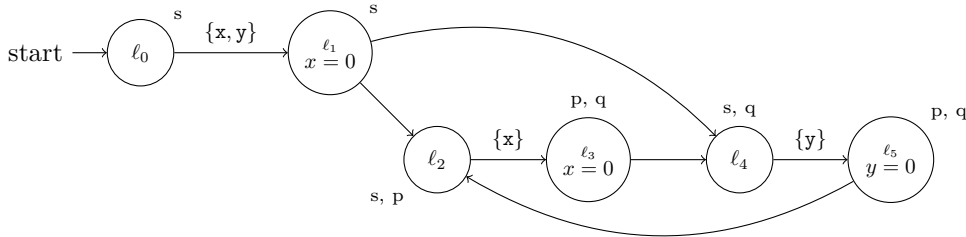
The model-checking problem for timed HORS is: Given a timed HORS \mathcal{G} and an APT-TA pair \mathcal{P}, \mathcal{A} as per above, decide whether \exists wins $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$.

► **Example 1.** Consider the problem of a scheduler that schedules two processes, each of them for a variable amount of time, depending on the internal state of the scheduler. We want to verify that the scheduler is fair, i.e., that no process gets scheduled for more than k seconds in a row, for some given constant k .

Let $\{J_1, \dots, J_n\}$ be a set of intervals denoting time slices scheduled to a process. W.l.o.g. assume that these are all unit intervals, i.e., of the form $[m, m]$ for some natural number m . Let Σ be a tree alphabet with a binary symbol a , unary symbols b, b' representing the two processes, and a nullary symbol c . We model the problem by a timed HORS with rules $S \mapsto S_1 c$ and

$$S_1(x: \bullet) \mapsto a_{[0,0]} x (S_2(b_{J_1}^1 x)) \quad \dots \quad S_n(x: \bullet) \mapsto a_{[0,0]} x (S_1(b_{J_n}^n x))$$

where the b^i are each either b or b' . Let \mathcal{A} be the TA defined by the following picture:



The sets on the edges denote resets, while the equalities in locations are invariants.

Finally, let \mathcal{P} be the $\{\mathbf{x} \leq k, \mathbf{y} \leq k\}$ -APT defined by $(\{s, p, q\}, \Sigma, \delta, s, \lambda)$ with λ being constant 2 and δ being defined by

$$\begin{aligned} \delta(s, _, a) &= (s, s) & \delta(_, S, b) &= (q) \\ \delta(_, \bar{S}, _) &= \perp & \delta(_, S, b') &= (p) & \delta(_, S, c) &= \top \end{aligned}$$

where $S = \{\mathbf{x} \leq k, \mathbf{y} \leq k\}$ and \bar{S} stands for any set of constraints that is not S .

The timed HORS in the above example generates a tree with the root and the rightmost branch labeled by $a_{[0,0]}$, while the left branches of these nodes are increasingly longer sequences of b and b' , ending in c . These encode increasingly longer scheduling decisions, encoded in reverse order. The APT traverses the rightmost branch in state s , while \exists necessarily keeps the TA in location ℓ_0 . Upon reading the first b or b' , the APT moves to state q or p , respectively, and, hence, the TA follows by switching to locations ℓ_4 , resp. ℓ_2 . Switching between the two resets exactly one of the clocks. If the APT encounters the TA in a clock evaluation where one of the clocks exceeds k , the automaton rejects, and it accepts any finite branch if it reaches the final c with none of the clocks having excessive value. The infinite branch of the tree containing the as accepts since $\lambda(s) = 2$.

4 Upper Bounds for Model-Checking

We show the model-checking problem for timed HORS to be in $(k+1)$ -EXPTIME for order- k timed HORS. This is done by an exponential reduction to the model-checking problem for untimed HORS. Hence, we reduce the input of a timed HORS, an APT, and a TA to a polynomially-sized untimed HORS and an exponential-sized APT such that the APT accepts the tree generated by the untimed HORS iff the APT-TA pair accepts the tree generated by the timed HORS.

As a preparation, we slightly alter the timed-HORS acceptance game. Let \mathcal{A} be a TA, let \mathcal{P} be an APT and let \mathcal{G} be a timed HORS, all over matching alphabets and propositions. Let \mathbf{z} be a clock not in the clock set \mathcal{X} of \mathcal{A} . The acceptance game $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$ is in *extra-clock semantics*, if transitions from positions of the form $(v, q, (\ell, \eta), 0)$ are as follows:

- First, the game transitions to the position $(v, q, (\ell, \eta|_{\mathbf{z}}), 0)$. Let $l(v) = a_J$. Any position of the form $(v, q, (\ell', \eta'), 1)$ with $\eta'(\mathbf{z}) \in J$ is a successor position, provided it can be obtained by letting time flow by visiting *only locations where the propositional labeling includes q* .

► **Lemma 2.** *In $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$, \exists wins from a position $(v, q, (\ell, \eta), 0)$ under extra-clock semantics iff she wins from that position under standard semantics. In particular, she wins the game from the initial position under extra-clock semantics iff she wins under standard semantics.*

We omit a formal proof since this is straightforward. The main point is that \mathbf{z} is not reset in \mathcal{A} , and, hence, can serve as a yardstick for elapsed time.

4.1 The Region Abstraction

The region abstraction is a classical result (see e.g., [1] or [4], Def. 9.42), that maps the infinite transition system defined by a TA onto a finite transition system. It uses an equivalence relation \simeq_m , for $m \in \mathbb{N}$, on clock evaluations, defined as follows: $\eta \simeq_m \eta'$ iff

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

Here, $\text{frac}(r)$ denotes the fractional part of a real number. Clock evaluations are considered equivalent if, for each clock, (I) either both clocks have a value greater than m , or (II) they compare in the same way with respect to all integers less than m . Moreover, the passage of time will make equivalent evaluations reach the next integral value first for the same clock.

It is straightforward to verify that \simeq_m is an equivalence relation for any m . An equivalence class in this relation is called a *region*. We denote the equivalence class of η under \simeq_m as $[\eta]_m$. When m is clear from the context, we may omit it and simply write $[\eta]$

An important observation is that the region abstraction is also a congruence w.r.t. the winner of the timed-HORS acceptance game:

► **Lemma 3.** *Let \mathcal{A} be a TA, \mathcal{P} a Ξ -APT, \mathcal{G} a timed HORS, all over matching alphabets and clocks. Let m be greater than or equal to the index of \mathcal{A} , any interval on a tree constructor in \mathcal{G} , and any clock constraint in Ξ . Let v be a node in $T(\mathcal{G})$, q a state of \mathcal{P} , ℓ a location in \mathcal{A} and $\eta \simeq_m \eta'$ two clock evaluations. Then \exists wins the acceptance game from position $(v, q, (\ell, \eta), 0)$ iff she wins the game from position $(v, q, (\ell, \eta'), 0)$.*

The proof is in App. A.1. By the lemma, the outcome of the acceptance game does not depend on the exact clock evaluation, but only on the region in question. This motivates notation of the form “ \exists wins from a position $(v, q, (\ell, [\eta]), 0)$ ”, which means that \exists wins for all positions $(v, q, (\ell, \eta'), 0)$ with $\eta' \in [\eta]$.

As mentioned above, given a TA \mathcal{A} and m greater than or equal to the index of \mathcal{A} , the region abstraction induces a finite transition system that faithfully represents the transition system defined by the TA. To make this precise, we define the notion of a *successor region*:

► **Definition 4.** *Let \mathcal{A} be a TA, and let m be greater than or equal than the index of \mathcal{A} . Let \simeq_m denote the region equivalence w.r.t. m . For each region $[\eta]_m$, the unique successor region is*

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

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

Let \mathcal{A} be a TA and let m be greater than or equal to the index of \mathcal{A} . Let Ξ be a set of clock constraints over the clocks of \mathcal{A} . The *region graph* $\mathcal{R}_{\Xi}^m(\mathcal{A})$ of \mathcal{A} (w.r.t. m and Ξ) is the transition system defined as follows:

- The state space is $\{(\ell, [\eta]_m) \in L \times (\mathcal{X} \rightarrow \mathbb{R}^{\geq 0}) / \simeq_m \mid \eta \models \iota(\ell)\}$ with initial state $(\ell_0, [\eta_0]_m)$.
- Discrete transitions from one state to another state are carried through, while delay transitions always lead to the successor region. Hence, we have $(\ell, [\eta]_m) \rightarrow (\ell', [\eta']_m)$ iff either $(\ell, \eta) \rightarrow (\ell', \eta')$ (discrete transition) or $\ell = \ell'$ and $[\eta']_m = \text{suc}([\eta]_m)$ (delay transition).
- The propositional labeling not only assigns atomic propositions to states via $p \in \lambda(\ell, [\eta])$ iff $p \in \lambda A(\ell)$ for any $p \in \text{Prop}$, but also interprets any clock constraint $\chi \in \Xi$ as an atomic proposition in the region graph via $\chi \in \lambda(\ell, [\eta])$ iff $(\ell, \eta) \models \chi$.

The following is a classical result.

► **Proposition 5 ([1]).** *Let \mathcal{A} be a TA over n clocks with l locations and let Ξ be a set of clock constraints over the clocks in \mathcal{A} . Let m be greater than or equal to the index of \mathcal{A} . Then $\mathcal{R}_{\Xi}^m(\mathcal{A})$ is an (untimed) transition system of size $l \cdot 2^{\mathcal{O}(n(\log n + \log m))} \cdot |\Xi|$, i.e., exponential in $|\mathcal{A}|$, and there is a trace $(\ell_0, \eta_0), \dots, (\ell_n, \eta_n)$ in the system defined by \mathcal{A} iff there is a path $(\ell_0, [\eta_0]) \rightarrow [s_1] \rightarrow \dots [s_{n'-1}] \rightarrow (\ell_n, [\eta_n])$ in $\mathcal{R}_{\Xi}^m(\mathcal{A})$.*

The last statement means that letting time flow in the tTS defined by \mathcal{A} corresponds to following a path in $\mathcal{R}_{\Xi}^m(\mathcal{A})$, and vice versa.

4.2 The Reduction

We now begin the reduction. The overall idea is to replace the companion TA, and the infinite transition system defined by it, by its corresponding region graph. This untimed transition system is then incorporated into the APT via a product construction. Hence, the new APT will have states of the form $(q, \ell, [\eta])$, where q is a state of the original APT, ℓ is a location in the companion TA, and $[\eta]$ is a region in its region graph. The transition function is then modified to simulate the passage of time from positions of the form $(v, q, (\ell, \eta), 0)$ using extra-clock semantics. However, direct simulation of the passage of time is tricky. In particular, it cannot be directly incorporated into the transition function without incurring blowup by another exponential in addition to the exponential blowup caused by passage to the region graph.

The above construction is supported by changing the timed HORS into an untimed HORS. This untimed HORS defines a tree that is similar in structure to that defined by the timed one, and it also makes the interval annotations of the tree defined by the timed HORS explicit via special gadgets. These gadgets help the new APT to properly advance time in its region graph components.

Let $\mathcal{G} = (\Sigma^t, \mathcal{N}, \mathcal{R}, S)$ be a timed HORS over a timed alphabet Σ^t , together with its untimed version Σ . Let \mathcal{J} be the intervals occurring on the right-hand sides of \mathcal{R} . Define the untimed alphabet $\Sigma^u = \Sigma \cup \{\text{reset}, \text{flow}\} \cup \bigcup_{J \in \mathcal{J}} \text{check}^J$ where $\text{reset}: \bullet \rightarrow \bullet$, $\text{flow}: \bullet^2 \rightarrow \bullet$ and $\text{check}^J: \bullet \rightarrow \bullet$ for all $J \in \mathcal{J}$. Then let $\mathcal{G}^u = (\Sigma^u, \mathcal{N} \cup \bigcup_{J \in \mathcal{J}} \{\text{start}^J, \text{choose}^J\}, \mathcal{R}', S)$ be an untimed HORS where \mathcal{R}' is defined as via:

$$\text{start}^J(x: \bullet) \mapsto \text{reset}(\text{choose}^J x) \quad \text{choose}^J(x: \bullet) \mapsto \text{flow}(\text{choose}^J x)(\text{check}^J x)$$

and the right-hand side for a symbol in N is defined as $\mathcal{R}'(N) = \widehat{\mathcal{R}(N)}$ where $(a_J \widehat{t_1 \cdots t_n}) = \text{start}^J(a \widehat{t_1 \cdots t_n}, \widehat{t_1 t_2} = \widehat{t_1} \widehat{t_2}$ if $t_1 \neq a_J$, $\widehat{x} = x$ for a variable x , and $\widehat{N'} = N'$ for $N' \in \mathcal{N}$.

Hence, a tree of the form $a_J t_1 \cdots t_n$ is replaced by a tree whose root is labeled by reset . The unique subtree below this root is an infinite tree where all nodes on the leftmost branch are labeled by flow , and the right son of each of the nodes labeled by flow is a tree of the form $\text{check}^J(a \widehat{t_1 \cdots t_n})$. Hence, there are infinitely many copies of the latter subtree, each reached by following the left son of the nodes labeled by flow for a finite number of times, and then branching to the right once.

The intuition is that the sequence of nodes labeled by flow will help the yet-to-be-defined untimed APT to simulate the flow of time in the tTTS defined by the TA by following a suitable path in its region graph component, i.e., one that advances the region component from one region to its successor region. The infinite sequence of nodes labeled by flow allows the APT to follow paths of arbitrary length in this fashion. Conversely, the passage of time is an atomic action in the semantics of the (timed) APT-TA pair, which must be broken up into its individual steps to avoid exponential blowup.

Let \mathcal{G} be as before and let $\mathcal{P} = (Q, \Sigma, \delta, q_I, \Lambda)$ be a timed Ξ -APT over Σ^t , let $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \Delta, \lambda)$ be a companion TA for it. Let $Q' = Q \cup \{q' \mid q \in Q\}$, let m be the smallest integer at least as big as the index of \mathcal{A} , each finite interval bound in \mathcal{J} and each clock constraint in Ξ . Let Ξ' be the union of Ξ and $\{z \in J \mid J \in \mathcal{J}\}$, where $z \notin \mathcal{X}$.

Define an untimed APT $\mathcal{P}_{\mathcal{A}} = (Q'', \Sigma^u, \delta', q'_I, \Lambda')$ via

- $Q'' = \{(q'', \ell, [\eta]) \mid q'' \in Q', (\ell, [\eta]) \text{ is a state in } \mathcal{R}_{\Xi'}^m(\mathcal{A})\}$
- the initial state is $q'_I = (q_I, \ell_0, [\eta_0])$,
- $\Lambda'(q'', _, _) = \Lambda(q'')$ if $q'' \in Q$, otherwise $\Lambda'(q'', _, _) = 1$

16:10 Real-Time Higher-Order Recursion Schemes

and δ' is defined as

$$\begin{aligned} \delta'((q, \ell, [\eta]), \text{reset}) &= (q', \ell, [\eta|_z]) \text{ if } q \in Q \\ \delta'((q', \ell, [\eta]), \text{flow}) &= ((q', \ell', [\eta']), (q', \ell, [\eta])) \text{ if } (\ell, [\eta]) \rightarrow (\ell', [\eta']) \text{ in } \mathcal{R}_{\Xi}^m(\mathcal{A}) \text{ and } q \in Q \\ \delta'((q', \ell, [\eta]), \text{check}^J) &= \perp \text{ if } \eta(\mathbf{z}) \notin J \\ \delta'((q', \ell, [\eta]), \text{check}^J) &= (q, \ell, [\eta]) \text{ if } \eta(\mathbf{z}) \in J \\ \delta'((q, \ell, [\eta]), a) &= \varphi' \text{ if } \delta(q, S, a) = \varphi \text{ and } S = \{\chi \in \Xi \mid \eta \models \chi\} \end{aligned}$$

where φ' is obtained from φ by replacing all occurrences of $q \in Q$ by $(q, \ell, [\eta])$. All unlisted transitions can be assumed to be \perp . The intuition is that the copies q' of states in Q serve for the sequence of flow nodes and have priority 1 to avoid staying on the sequence indefinitely. Branching leftward at a flow node advances time. The following is immediate from Prop. 5.

► **Observation 6.** *The size of $\mathcal{P}_{\mathcal{A}}$ is exponential in that of \mathcal{P} and \mathcal{A} .*

We now show that $\mathcal{P}_{\mathcal{A}}$ simulates the APT-TA pair of \mathcal{P} and \mathcal{A} faithfully.

► **Lemma 7.** *Let \mathcal{G}, \mathcal{A} and \mathcal{P} be a timed HORS, a TA and an APT over matching alphabets. $\mathcal{P}_{\mathcal{A}}$ accepts the tree generated by \mathcal{G}^u iff the APT-TA pair accepts $T_{\mathcal{G}}$.*

The proof is in App. A.1.

► **Theorem 8.** *The model-checking problem for order- k timed HORS is in $(k+1)$ -EXPTIME.*

Proof. By Lemma 7, the model-checking problem for a timed APT \mathcal{P} and a TA \mathcal{A} over a timed HORS \mathcal{G} can be reduced to the untimed model-checking problem of $\mathcal{P}_{\mathcal{A}}$ over \mathcal{G}^u . By Obs. 6, the size of $\mathcal{P}_{\mathcal{A}}$ is exponential in that of \mathcal{P} and \mathcal{A} . Since the model-checking problem for order- k HORS can be solved in k -fold exponential time [20], the result follows. ◀

5 Matching Lower Bounds

We now show that the model-checking problem for order-1 timed HORS is 2-EXPTIME hard. The proof reuses a construction from [10], adapted to the situation with timed HORS. We restrict ourselves to the order-1 case due to space considerations, but we conjecture that the proof can be extended to arbitrary order in the same fashion as in [10].

5.1 Encoding 2-EXPTIME Turing Machines

We use an idea from [12] to encode the word problem for deterministic 2-EXPTIME Turing machines, namely by encoding an accepting run of the machine into a square table of 2-fold exponential size. The rows of the table are the configurations of the run, in order. A row lists the tape contents of the machine and the position and state of the head.

The following is obtained by standard reductions: the machine never moves its head to the left of the starting position and it accepts by moving to a unique accepting state at the starting position and on empty tape. Moreover, instead of halting, it repeats this configuration ad infinitum, and it enters this configuration in time at most $2^{2^n} - 2$ on input of size n instead of merely time less than $2^{2^{p(n)}}$ for some polynomial p . Moreover, it suffices to consider the empty-word problem, i.e., the question whether the machine halts on empty input. The last assumption is easily shown to be valid, as for every machine \mathcal{M} and input w satisfying all the other assumptions, there is \mathcal{M}_w that first writes w on empty input and then behaves like \mathcal{M} .

Let $\mathcal{M} = (Q, \Sigma, \Gamma, q_I, q_{acc})$ be a DTM with state set Q , unique initial and accepting states q_I and q_{acc} , input and tape alphabets $\Sigma \subsetneq \Gamma$, with $\square \in \Gamma \setminus \Sigma$ the blank symbol and, $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$ the transition function. Let $\hat{\Gamma} = \Gamma \cup (Q \times \Gamma) \cup \{\#\}$ where $\#$ is a new boundary symbol that does not occur in Γ . Define the extended transition function $\hat{\delta}: \hat{\Gamma}^3 \rightarrow \hat{\Gamma} \setminus \{\#\}$ with $\hat{\delta}(b_1, b_2, b_3)$ being

- (b_2, q) , if $b_1 \in \Gamma \cup \{\#\}$, $b_2 \in \Gamma$ and $b_3 = (q', a)$ with $\delta(q', a) = (q, _ , L)$
- (b_2, q) , if $b_3 \in \Gamma \cup \{\#\}$, $b_2 \in \Gamma$ and $b_1 = (q', a)$ with $\delta(q', a) = (q, _ , R)$
- (a, q) , if $b_1, b_3 \in \Gamma \cup \{\#\}$, $b_2 = (q', a)$ with $\delta(q', a) = (q, b, N)$
- a , if $b_1, b_3 \in \Gamma \cup \{\#\}$, $b_2 = (q', b)$ with $\delta(q', a) = (q, a, R)$ or (q, a, L)
- a , if $b_2 = a$, $b_1 \neq (q', a')$ with $\delta(q', a') = (q, _ , R)$, $b_3 \neq (q', a')$ with $\delta(q', a') = (q, _ , L)$

This encodes the contents of a cell in the table of rows in dependence on the three cells of the previous row that are directly below it, resp. directly adjacent to the cell directly below it. It is easy to see that the preimage of any $b \in \hat{\Gamma} \setminus \{\#\}$ under $\hat{\delta}$ is easy to compute, since there are only $|\hat{\Gamma}^3|$ many possible candidates.

► **Definition 9.** A 2^{2^n} -certificate for the acceptance of \mathcal{M} is a $C: [2_2^n] \times [2_2^n] \rightarrow \hat{\Gamma}$ where

1. $C(r, c) = \#$ iff $c = 0$ or $c = 2^{2^n} - 1$,
2. $C(0, 1) = (q_I, \square)$ and $C(0, c) = \square$ if $2 \leq c < 2^{2^n} - 1$,
3. if $r > 0$, $0 \neq c \neq 2^{2^n} - 1$ and $C(r, c) = b \in \hat{\Gamma}$, then there are b_1, b_2, b_3 with $\hat{\delta}(b_1, b_2, b_3) = b$ and $C(r-1, c-1) = b_1$, $C(r-1, c) = b_2$ and $C(r-1, c+1) = b_3$.

Such a certificate is successful if also $C(2^{2^n} - 1, 0) = (q_{acc}, \square)$ holds.

Clearly, for every \mathcal{M} and n , there is a unique such certificate.

► **Proposition 10.** It is a 2-EXPTIME-hard problem to decide whether, given some n encoded in unary, and a DTM \mathcal{M} , there is a successful 2^{2^n} -certificate for \mathcal{M} and n in the sense above.

For a more detailed exposition of this, see [10].

5.2 Modeling of Large Numbers

In order to encode a certificate as in the previous section into the model-checking problem for timed HORS, we need to encode and manipulate numbers up to 2^{2^n} into a polynomially-sized presentation of a timed HORS, a TA and an APT. The heavy lifting for the encoding is done by the TA, via the use of clock values, while the manipulating is done using the HORS.

The encoding is done using a single clock. Let \mathcal{A} be the TA with one location ℓ , one clock \mathbf{x} , no invariants and no transitions and, hence, no guards. Hence, a state in the transition system is defined completely by its value of \mathbf{x} . Let $\Sigma_{\text{red}} = \{\text{and}, \text{and3}, \text{or}, \text{or3}, \text{neg}, \text{ff}, \text{mx}\}$ be a tree alphabet where ff and mx are of type \bullet , neg is of type $\bullet \rightarrow \bullet$, and and or are of type $\bullet^2 \rightarrow \bullet$ and and3 and or3 are of type $\bullet^3 \rightarrow \bullet$. Let $\Xi = \{\mathbf{x} \leq 2^n - 1\}$. We abbreviate this single constraint by max . Let $\mathcal{P}_{\text{cnt}} = (\{q_1^{\text{yes}}, q_0^{\text{yes}}, q_1^{\text{no}}, q_0^{\text{no}}, q^\top\}, \Sigma_{\text{red}}, \delta, q_1^{\text{yes}}, \Lambda)$ be a timed APT with $\Lambda(q_1^{\text{yes}}) = \Lambda(q_1^{\text{no}}) = \Lambda(q^\top) = 1$ and $\Lambda(q_0^{\text{yes}}) = \Lambda(q_0^{\text{no}}) = 0$, with δ to be defined soon. The intuition is that $q_1^{\text{yes}}, q_0^{\text{yes}}$ verify that a given bit is set in the representation of a large number, and that $q_1^{\text{no}}, q_0^{\text{no}}$ falsify this, with the difference between the two copies of each state to become important later. The state q^\top accepts any tree. We set the location ℓ in \mathcal{A} to satisfy all propositions associated to states in Q .

16:12 Real-Time Higher-Order Recursion Schemes

The transition function of the APT is defined via (where $i \in \{1, 2\}$)

$$\begin{aligned}
\delta(q_i^{\text{yes}}, _, \text{ff}) &= \perp & \delta(q_i^{\text{yes}}, _, \text{and}) &= (q_0^{\text{yes}}, q_1^{\text{yes}}) \\
\delta(q_i^{\text{no}}, _, \text{ff}) &= \top & \delta(q_i^{\text{yes}}, _, \text{or}) &= (q_1^{\text{yes}}, q^\top) \vee (q^\top, q_1^{\text{yes}}) \\
\delta(q_i^{\text{yes}}, \{\text{max}\}, \text{mx}) &= \top & \delta(q_i^{\text{no}}, _, \text{and}) &= (q_1^{\text{no}}, q^\top) \vee (q^\top, q_1^{\text{no}}) \\
\delta(q_i^{\text{no}}, \{\text{max}\}, \text{mx}) &= \perp & \delta(q_i^{\text{no}}, _, \text{or}) &= (q_0^{\text{no}}, q_1^{\text{no}}) \\
\delta(q_i^{\text{yes}}, \emptyset, \text{mx}) &= \perp & \delta(q_i^{\text{yes}}, _, \text{and3}) &= (q_1^{\text{yes}}, q_1^{\text{yes}}, q_1^{\text{yes}}) \\
\delta(q_i^{\text{no}}, \emptyset, \text{mx}) &= \top & \delta(q_i^{\text{yes}}, _, \text{or3}) &= (q_1^{\text{yes}}, q^\top, q^\top) \vee (q^\top, q_1^{\text{yes}}, q^\top) \vee (q^\top, q^\top, q_1^{\text{yes}}) \\
\delta(q^\top, _, _) &= \top & \delta(q_i^{\text{no}}, _, \text{and3}) &= (q_1^{\text{no}}, q^\top, q^\top) \vee (q^\top, q_1^{\text{no}}, q^\top) \vee (q^\top, q^\top, q_1^{\text{no}}) \\
&& \delta(q_i^{\text{no}}, _, \text{or3}) &= (q_1^{\text{no}}, q_1^{\text{no}}, q_1^{\text{no}}).
\end{aligned}$$

The intuition for the automaton is that the states q_i^{yes} reject at ff (for “false”) and that they accept at mx iff, in the companion TA, **max** is satisfied. The states q_i^{no} work in the opposite fashion. At tree constructors **and**, **or**, both state pairs behave as expected: for **and**, both subtrees have to be verified, while for **or**, only one of them needs to be verified, and opposite for q_i^{no} . Again, the pair q_i^{no} behaves in the opposite fashion, and **and3** and **or3** are the ternary variants of **and** and **or**. For reasons that we will explain later, at the binary tree constructors, the priority of the state going in leftward direction sometimes deviates from the default 1.

► **Definition 11.** *A tree over Σ_{red} encodes a number $k \in [2^{2^n}]$ if its representation in binary is $\sum_{i=0}^{2^n-1} b_i \cdot 2^i$ where $b_i = 0$ iff \mathcal{P}_{cnt} accepts the root of the tree with the clock of its companion TA set to $2^n - 1 - i$.*

Note that this definition talks about runs of the APT-TA pair with the clock of the TA initialized to a specific value, not necessarily 0. The intuition is that acceptance of \mathcal{P}_{cnt} for different initial values of the clock of the companion TA forms a set of bits: for each clock value in $[2^n]$, \mathcal{P}_{cnt} either accepts or it does not. These bits can be thought of as the binary representation of a number in $[2^{2^n} - 1]$. Note that the order of the bits is reversed from what one would intuitively expect: the least significant bit is the one where \mathbf{x} has value $2^{2^n} - 1$, while the most significant bit is the one where \mathbf{x} has value 0.

It is also fairly easy to see that there are trees that encode numbers. For example, the number 0 is encoded by the tree with just one node, labeled by ff. It now is time to add incrementation and decrementation of numbers.

In the binary representation of the increment of a number, a bit is set if either it is

- set in the representation of the number, and there is an unset bit of lesser significance, or
- not set in the representation of the number, but all bits of lesser significance are set.

Since bits are set in the encoding of our numbers if the APT \mathcal{P}_{cnt} accepts if the single clock of its companion TA is set to the right value, checking whether a bit of lesser significance is set in the encoding of a number, i.e., a tree, can be done by letting time flow for the right amount, and then simply checking if the APT accepts the tree. This works because bits of lesser significance are represented by clock values closer to $2^{2^n} - 1$.

In the following, for $a_{[1,1]}$ we write a_1 , and an undecorated constructor a means $a_{[0,0]}$. Consider the following definitions, where $T(\text{zero})$ clearly encodes 0:

$$\begin{aligned}
\text{zero} &\mapsto \text{ff} \\
\text{exists } (x: \bullet) &\mapsto \text{or}_1 (\text{exists } x) (\text{and max } x) \\
\text{all } (x: \bullet) &\mapsto \text{and}_1 (\text{all } x) (\text{or } (\text{neg max } x)) \\
\text{inc } (x: \bullet) &\mapsto \text{or}(\text{and } x (\text{exists } (\text{neg } x))) (\text{and } (\text{neg } x) (\text{all } x)) \\
\text{dec } (x: \bullet) &\mapsto \text{or}(\text{and } x (\text{exists } x)) (\text{and } (\text{neg } x) (\text{all } (\text{neg } x))) \\
\text{isZero? } (x: \bullet) &\mapsto \text{and}(\text{neg } x) (\text{all neg } x)
\end{aligned}$$

We have already seen that the tree generated by `zero` encodes the number 0. The following lemma proves that the above macros behave as their names suggest.

► **Lemma 12.** *Let e be a closed term of type \bullet such that $T(e)$ encodes some number $k \in [2^{2^n}]$. Then*

1. \mathcal{P}_{cnt} accepts $T(\text{exists } e)$ from states q_i^{yes} with its clock set to some value $k \in [2^n]$ iff it accepts $T(e)$ from q_i^{yes} with the clock set to some integral k' with $k < k' \leq 2^n - 1$, and it accepts from states q_i^{no} iff it accepts $T(e)$ from q_i^{no} for no such integral clock value.
2. \mathcal{P}_{cnt} accepts $T(\text{all } e)$ from states q_i^{yes} with its clock set to some value $k \in [2^n]$ iff it accepts $T(e)$ from q_i^{no} and for all integral clock values k' with $k < k' \leq 2^n - 1$, and it accepts from states q_i^{no} iff it rejects $T(e)$ from q_i^{no} for some such integral clock value.
3. $T(\text{inc } e)$ encodes the number $k + 1 \pmod{2^{2^n}}$ and $T(\text{dec } e)$ encodes the number $k - 1 \pmod{2^{2^n}}$.
4. \mathcal{P}_{cnt} accepts $T(\text{isZero? } e)$ with the clock set to 0 from states q_i^{yes} iff $T(e)$ encodes 0, and from states q_i^{no} iff the tree generated by $T(e)$ encodes a number different from 0.

The proof is in App. A.2.

5.3 The Hardness Proof

We now show that it is 2-EXPTIME-hard to decide the model-checking problem of timed HORS, i.e., to decide whether a given pair of a TA and a timed APT accepts the tree generated by an order-1 timed HORS. We reduce the following problem to the model-checking problem: given n encoded in unary and \mathcal{M} a DTM, is there a successful 2_2^n -certificate for \mathcal{M} ? By Prop. 10, this problem is 2-EXPTIME-hard.

For the remainder of this section, fix some n encoded in unary and a 2_2^n -bounded DTM. Let $\hat{\Gamma}$ be as in Def. 9. Note that the question for a successful 2_2^n -certificate asks for a table of width 2_2^n and height 2_2^n , filled with entries from $\hat{\Gamma}$. It is sufficient to check that the top left entry of the certificate is correct, and then to recursively check that the three entries directly below a given entry are as per $\hat{\delta}$.¹

We implement this recursive verification procedure by defining an APT-TA pair and an order-1 timed HORS such that the pair accepts the HORS iff the certificate is successful. The state of the APT represents an element of $\hat{\Gamma}$. The timed HORS generates a tree in which certain nodes represent an entry of the table, and these nodes have two (indirect) subtrees that encode the row and column number of the entry, to be recognized by the TA part of the APT-TA pair as in Def. 11. In addition, such a node representing a column entry has another three direct subtrees, which encode the three table entries in the previous row with the three adjacent column numbers as per $\hat{\delta}$. The tree contains further gadgets to verify that a given symbol can occur in the entry if this is easily verified, e.g., for the leftmost and rightmost columns and the boundary symbol $\#$. The scheme is defined as follows:

¹ This recursive procedure does not check the top right half of the table, but it is easy to see that it must be filled by blanks due to the assumptions about the DTM in question. See also [10] for more details.

16:14 Real-Time Higher-Order Recursion Schemes

$$\begin{aligned}
S &\mapsto F(\text{dec zero}) \text{ zero} \\
F(r: \bullet)(c: \bullet) &\mapsto \text{or}(\text{check } r \ c) (\text{next } r \ c) \\
\text{gtOne?}(x: \bullet) &\mapsto \text{and}(\text{neg}(\text{isZero? } x)) (\text{neg}(\text{isZero?}(\text{dec } x))) \\
\text{ltMax?}(x: \bullet) &\mapsto \text{and}(\text{neg}(\text{isZero?}(\text{inc } x))) (\text{isZero?}(\text{inc}(\text{inc } x))) \\
\text{check}(r: \bullet)(c: \bullet) &\mapsto \text{or3}(\text{or}(\text{isZero?}(\text{inc } r)) (\text{isZero? } r)) \\
&\quad (\text{and}(\text{isZero? } r) (\text{isZero?}(\text{dec } c))) \\
&\quad (\text{and3}(\text{isZero? } r) (\text{gtOne? } x) (\text{ltMax? } x)) \\
\text{next}(r: \bullet)(c: \bullet) &\mapsto \text{and}\left(\text{and3}(\text{neg}(\text{isZero? } r)) (\text{neg}(\text{isZero? } c)) (\text{ltMax? } c)\right) \\
&\quad \left(\text{and3}(F(\text{dec } r) (\text{dec } c)) (F(\text{dec } r) \ c) (F(\text{dec } r) (\text{inc } c))\right)
\end{aligned}$$

Intuitively, the only properly recursive nonterminal here is F . Any subtree generated by F with arguments r and c encodes a cell of the certificate, namely that whose row number is encoded by $T(r)$ and its column number is encoded by $T(c)$. Such a tree node poses a choice to the yet-to-be-defined APT whether it accepts this tree via (the tree generated by) **next** or via **check**. The latter variant allows the APT to verify that $T(r)$ and $T(c)$ satisfy the boundary conditions of the certificate, while the former variant verifies that $T(r)$ and $T(c)$ encode the indices of an interior cell of the certificate and make the APT verify recursively the properties of the three cells below that encoded by $T(r)$ and $T(c)$.

Define $\mathcal{P}_{\text{red}} = (Q, \Sigma_{\text{red}}, \delta', \{q^{(q_{\text{acc}}, \square)}\}, \Lambda)$ as an extension of the APT \mathcal{P}_{cnt} from the previous section, with δ' to be defined soon, with

$$Q = \{q_1^{\text{yes}}, q_0^{\text{yes}}, q_1^{\text{no}}, q_0^{\text{no}}, q^\top\} \cup \{q_l^b \mid b \in \{\#, \square, (q_I, \square)\}\} \cup \{q^{(b_1, b_2, b_3)} \mid b_1, b_2, b_3 \in \hat{\Gamma}\},$$

and $\Lambda(q) = 1$ for all $q_l^b, q^{(b_1, b_2, b_3)}$ and Λ as per \mathcal{P}_{cnt} for $q \in \{q_1^{\text{yes}}, q_0^{\text{yes}}, q_1^{\text{no}}, q_0^{\text{no}}, q^\top\}$. The companion TA remains the one-state, one-clock featureless TA from \mathcal{P}_{cnt} .

Consider the timed HORS $\mathcal{G}_{\text{red}} = (\Sigma_{\text{red}}, \mathcal{N}, \mathcal{R}, S)$ with \mathcal{N} and \mathcal{R} as given by the definitions above. The idea is that \mathcal{P}_{red} accepts from state q^b and with the clock of its companion TA set to 0 the tree $T(F \ r \ c)$ iff the numbers encoded by the trees $T(r)$ and $T(c)$ are m and m' such that $C(m, m') = b$ in the unique certificate for \mathcal{M} and n . In particular, \mathcal{P}_{red} accepts $T(S)$ from its starting state $q^{(q_{\text{acc}}, \square)}$ iff $C(2_2^n - 1, 0) = (q_{\text{acc}}, \square)$.

Towards this, consider the following definition of the transition function of \mathcal{P}_{red} :

$$\begin{aligned}
\delta'(q^b, _, \text{or}) &= (q_l^b, q^\top) \text{ if } b \in \{\#, (q_I, \square)\} \\
\delta'(q^\square, _, \text{or}) &= (q_l^\square, q^\top) \vee \bigvee_{\hat{\delta}(b_1, b_2, b_3) = \square} (q^\top, q^{(b_1, b_2, b_3)}) \\
\delta'(q^b, _, \text{or}) &= \bigvee_{\hat{\delta}(b_1, b_2, b_3) = b} (q^\top, q^{(b_1, b_2, b_3)}) \text{ if } b \notin \{\#, \square, (q_I, \square)\} \\
\delta'(q_l^\#, _, \text{or3}) &= (q_1^{\text{yes}}, q^\top, q^\top) & \delta'(q_l^{(q_i, \square)}, _, \text{or3}) &= (q^\top, q_1^{\text{yes}}, q^\top) \\
\delta'(q_l^\square, _, \text{or3}) &= (q^\top, q^\top, q_1^{\text{yes}}) & \delta'(q^{(b_1, b_2, b_3)}, _, \text{and}) &= (q_1^{\text{yes}}, q^{(b_1, b_2, b_3)}) \\
\delta'(q^{(b_1, b_2, b_3)}, _, \text{and3}) &= (q^{b_1}, q^{b_2}, q^{b_3})
\end{aligned}$$

The remaining transitions can all be assumed to lead to \perp , as they will not occur anyway.

We now formalize the above intuition about the semantics of the APT.

► **Lemma 13.** *Let $C: 2_2^n \times 2_2^n \rightarrow \hat{\Gamma}$ be the unique certificate for n and \mathcal{M} . Let $r, c: \bullet$ be expressions such that $T(r), T(c)$ encode $m, m' \in [2_2^n]$. Let $b, b_1, b_2, b_3 \in \hat{\Gamma}$.*

1. \mathcal{P}_{red} accepts $T(\text{check } r \ c)$ from state $q_1^\#$ iff either $m' = 0$ or $m' = 2_2^n - 1$.
 2. \mathcal{P}_{red} accepts $T(\text{check } r \ c)$ from state q_1^\square iff both $2 \leq m' < 2_2^n - 1$ and $m = 0$.
 3. \mathcal{P}_{red} accepts $T(\text{check } r \ c)$ from state $q_1^{(q_1, \square)}$ iff both $m = 0$ and $m' = 1$.
 4. \mathcal{P}_{red} accepts $T(\text{next } r \ c)$ from state $q^{(b_1, b_2, b_3)}$ iff (I) both $m > 0$ and $0 < m' < 2_2^n - 1$ and (II) \mathcal{P}_{red} accepts $T(F(\text{dec } r) \ (\text{dec } c))$ from state q^{b_1} and $T(F(\text{dec } r) \ c)$ from state q^{b_2} and $T(F(\text{dec } r) \ (\text{inc } c))$ from state q^{b_3} .
 5. \mathcal{P}_{red} accepts $T(F \ r \ c)$ from state q^b iff $C(m, m') = b$.
- Finally, \mathcal{P}_{red} accepts $T(\mathcal{G}_{\text{red}})$ iff $C(2_2^n - 1, 0) = (q_{\text{acc}}, \square)$.

The proof is in App. A.2.

This lemma then immediately yields the hardness result:

► **Theorem 14.** *It is 2-EXPTIME-hard to decide whether a given APT-TA pair accepts a given timed HORS.*

Proof. Take any DTM \mathcal{M} . By Prop. 10, it is 2-EXPTIME-hard to decide, given n , whether there is a successful 2_2^n -certificate for \mathcal{M} . By Lemma 13, this is equivalent to deciding whether \mathcal{P}_{red} , together with its trivial companion TA, accepts $T(\mathcal{G}_{\text{red}})$. The size of \mathcal{P}_{red} is polynomial in $|\mathcal{M}|$, the size of the TA is polynomial in n , and the size of \mathcal{G}_{red} is linear in n , since the bound max can be encoded in binary. Hence, this is a polynomial-time reduction to the timed-HORS model-checking problem which, in turn, must be 2-EXPTIME-hard. ◀

6 Conclusion

We have introduced the concept of timed APT and timed HORS, to be model-checked in conjunction with a TA. To our knowledge, this constitutes the first formalization of dense real time for the verification of trees generated by HORS. We have shown that real time adds at most one exponent to the model-checking problem, and that this is strict at order 1. This is in line with findings for settings where not the structure to be verified is higher-order, but the property a structure is verified against. For example, the model-checking problem for Recursive CTL is complete for exponential time [8], and adding real time to obtain Timed Recursive CTL makes the problem complete for two-fold exponential time [10].

While this paper introduces the concept of adding real time to recursion schemes, the expressiveness of the framework needs to be explored. For example, we could allow the APT to reset some clocks of its companion TA, which would make it easier to model certain problems. We would also like to invert the semantics of the TA in the sense that \forall controls the flow of time. This would allow us to model safety properties of the form “no matter how long it takes to perform an action, correctness is guaranteed”. However, the concept of timed automata is somewhat alien to adverse time flow, hence we expect such a change to be non-trivial.

We conjecture that the hardness result can be obtained for all k , not just for $k = 1$. This would be done by replacing the trees that encode large numbers by functions that consume such a tree and return such a tree, and functions that consume such functions and return trees encoding a number, etc. Such an approach has been used in similar settings [18, 11]. We also conjecture the existence of a restricted version of timed HORS and the APT-TA pairs used in the model-checking problem, such that the problem becomes characteristic not of time, but of space. This is suggested by similar fragments both for the timed setting [9] and for the model-checking problem for untimed HORS [7].

References

- 1 Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993. doi:10.1006/inco.1993.1024.
- 2 Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990. doi:10.1007/BFB0032042.
- 3 Rajeev Alur and P. Madhusudan. Decision problems for timed automata: A survey. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2004. doi:10.1007/978-3-540-30080-9_1.
- 4 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 5 Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. C-shore: a collapsible approach to higher-order verification. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 13–24. ACM, 2013. doi:10.1145/2500365.2500589.
- 6 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPICs*, pages 129–148. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICs.CSL.2013.129.
- 7 Florian Bruse. Space-efficient model-checking of higher-order recursion schemes. manuscript submitted.
- 8 Florian Bruse and Martin Lange. Temporal logic with recursion. *Inf. Comput.*, 281:104804, 2021. doi:10.1016/J.IC.2021.104804.
- 9 Florian Bruse and Martin Lange. The tail-recursive fragment of timed recursive CTL. *Inf. Comput.*, 294:105084, 2023. doi:10.1016/J.IC.2023.105084.
- 10 Florian Bruse and Martin Lange. Model checking timed recursive CTL. *Inf. Comput.*, 298:105168, 2024. doi:10.1016/J.IC.2024.105168.
- 11 Florian Bruse, Martin Lange, and Étienne Lozes. Space-efficient fragments of higher-order fixpoint logic. In Matthew Hague and Igor Potapov, editors, *Reachability Problems - 11th International Workshop, RP 2017, London, UK, September 7-9, 2017, Proceedings*, volume 10506 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2017. doi:10.1007/978-3-319-67089-8_3.
- 12 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.
- 13 Werner Damm. The IO- and oi-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi:10.1016/0304-3975(82)90009-3.
- 14 Gérard Huet. *Résolution d'Équations dans des Langages d'Order 1, 2, ω* . PhD thesis, Université de Paris VII, 1976.
- 15 Marcin Jurdzinski. Small progress measures for solving parity games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000. doi:10.1007/3-540-46541-3_24.
- 16 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

- 17 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.
- 18 Naoki Kobayashi, Étienne Lozes, and Florian Bruse. On the relationship between higher-order recursion schemes and higher-order fixpoint logic. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 246–259. ACM, 2017. doi:10.1145/3009837.3009854.
- 19 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.
- 20 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

A Omitted Proofs

A.1 Proofs for Section 4

► **Lemma 3.** *Let \mathcal{A} be a TA, \mathcal{P} a Ξ -APT, \mathcal{G} a timed HORS, all over matching alphabets and clocks. Let m be greater than or equal to the index of \mathcal{A} , any interval on a tree constructor in \mathcal{G} , and any clock constraint in Ξ . Let v be a node in $T(\mathcal{G})$, q a state of \mathcal{P} , ℓ a location in \mathcal{A} and $\eta \simeq_m \eta'$ two clock evaluations. Then \exists wins the acceptance game from position $(v, q, (\ell, \eta), 0)$ iff she wins the game from position $(v, q, (\ell, \eta'), 0)$.*

Proof. We show the result using extra-clock semantics. Let \mathcal{A} , \mathcal{P} , \mathcal{G} and m, η, η' be as in the lemma. We show the following, stronger result: \exists wins the acceptance game from any position of the forms $(v, q, (\ell, \eta), 0)$ or $(v, \varphi, (\ell, \eta), 1)$ iff she wins from positions $(v, q, (\ell, \eta'), 0)$, resp. $(v, \varphi, (\ell, \eta'), 1)$. The proof is by induction over plays of the acceptance game. Since the claim of the lemma is fully symmetric, it suffices to show that a winning strategy for η yields one for η' . The invariant will be that \exists keeps the game for η' in positions of the form $(v, q, (\ell, \eta'), 0)$ or $(v, q, (\ell, \eta'), 1)$ such that she wins the respective game from the position with η' replaced by some η with $\eta \simeq_m \eta'$. Clearly, the initial position for the claim of the lemma has this form.

The interesting case is that of a position of the form $(v, q, (\ell, \eta'), 0)$ with $l(v) = a.J$. By assumption, \exists has a winning strategy from $((v, q, (\ell, \eta), 0)$. This means there is a trace $(\ell, \eta|_z) = (\ell_1, \eta_1), \dots, (\ell_n, \eta_n)$ where all the ℓ_i satisfy q and (ℓ_i, η_i) reaches (ℓ_{i+1}, η_{i+1}) either via a delay transition or a discrete transition, and $\eta_n(z) \in J$. We produce a sequence $(\ell, \eta'|_z) = (\ell_1, \eta'_1), \dots, (\ell_n, \eta'_n)$ that satisfies the same conditions and, moreover, satisfies $\eta'_i \simeq_m \eta_i$ for all i .

It is easy to see that $\eta|_z \simeq_m \eta'|_z$. The rest of the sequence is generated by induction. If the next pair in the sequence is reached by a discrete transition for the η_i , then the same transition is available for η'_i since clock evaluations in the same region satisfy the same clock constraints (which appear as guards and location invariants here). If the next pair is reached via a delay transition using delay d , there is some d' such that delaying form η'_i reaches an equivalent region. This is due to the definition of \simeq_m : letting time flow in equivalent regions will pass through the same sequences or regions, since clocks reach integral values in the same order. Hence, such a sequence $(\ell_1, \eta'_1), \dots, (\ell_n, \eta'_n)$ exists. Since $\eta'_n \simeq \eta_n$, also $\eta'_n(z) \in J$. This finishes the case of positions of the form $(v, q, (\ell, \eta'), 0)$.

For the other cases, the invariant is easy to maintain since the TA part of the position does not change. This finishes the proof. ◀

► **Lemma 7.** *Let \mathcal{G}, \mathcal{A} and \mathcal{P} be a timed HORS, a TA and an APT over matching alphabets. $\mathcal{P}_{\mathcal{A}}$ accepts the tree generated by \mathcal{G}^u iff the APT-TA pair accepts $T_{\mathcal{G}}$.*

Proof. The invariant for \exists to maintain for either direction of the proof is to stay in a position $(v, (q, \ell, [\eta]))$ such that she wins from position $(v, \varphi, (\ell, \eta), 0)$ or $(v, q, (\ell, \eta), 1)$ in $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$, or to stay in a position of the latter form such that she wins from position $(v, (q', \ell, [\eta]))$ or $(v, (q, \ell, [\eta]))$ in the acceptance game $G(\mathcal{P}_{\mathcal{A}}, \mathcal{G}^u)$, where $q \in Q$, the state set of \mathcal{P} .

The interesting part is that of a position of the form $(v, (q, \ell, [\eta]))$ in $G(\mathcal{P}_{\mathcal{A}})$. Assume that there is a corresponding position $(v, q, (\ell, \eta), 0)$ such that \exists wins from there in $G(\mathcal{A}, \mathcal{P}, \mathcal{G})$. Using extra-clock semantics, the game first transitions to $(v, q, (\ell, \eta|z), 0)$. Then, \exists lets time flow, visiting only locations where q holds, up to some $(v, q, (\ell', \eta'), 0)$ with $\eta'(z) \in J$ where $a_J = l(v)$. By Prop. 5, there is a path corresponding to the trace taken by \exists in $G(\mathcal{P}_{\mathcal{A}}, \mathcal{G})$ passing through the respective regions, ending up in $(\ell', [\eta'])$. It is not hard to see that all regions on this path also must satisfy the proposition q . Hence, \exists can move through the gadget defined by \mathbf{start}^J by letting the initial transition for \mathbf{reset} reset the clock z , passing from $[\eta]$ to $[\eta|z]$, then choosing the leftmost branch at \mathbf{flow} choosing successor pairs or regions and locations following the trace from $G(\mathcal{P}_{\mathcal{A}}, \mathcal{G}^u)$, and exiting via the right branch once she has replicated the trace. This yields a position $(v, (q, \ell', [\eta']))$ and by the definition of \simeq_m , also $\eta'(z) \in J$, whence the check at \mathbf{check}^J is passed.

The other transitions concern only the transition function of the original APT and do not manipulate time, whence it is not hard to verify the invariant. The converse direction follows a similar pattern, also invoking Prop. 5. This finishes the proof. ◀

A.2 Proofs for Section 5

► **Lemma 12.** *Let e be a closed term of type \bullet such that $T(e)$ encodes some number $k \in [2^{2^n}]$. Then*

1. \mathcal{P}_{cnt} accepts $T(\mathbf{exists} e)$ from states q_i^{yes} with its clock set to some value $k \in [2^n]$ iff it accepts $T(e)$ from q_i^{yes} with the clock set to some integral k' with $k < k' \leq 2^n - 1$, and it accepts from states q_i^{no} iff it accepts $T(e)$ from q_i^{no} for no such integral clock value.
2. \mathcal{P}_{cnt} accepts $T(\mathbf{all} e)$ from states q_i^{yes} with its clock set to some value $k \in [2^n]$ iff it accepts $T(e)$ from q_i^{no} and for all integral clock values k' with $k < k' \leq 2^n - 1$, and it accepts from states q_i^{no} iff it rejects $T(e)$ from q_i^{no} for some such integral clock value.
3. $T(\mathbf{inc} e)$ encodes the number $k + 1 \pmod{2^{2^n}}$ and $T(\mathbf{dec} e)$ encodes the number $k - 1 \pmod{2^{2^n}}$.
4. \mathcal{P}_{cnt} accepts $T(\mathbf{isZero?} e)$ with the clock set to 0 from states q_i^{yes} iff $T(e)$ encodes 0, and from states q_i^{no} iff the tree generated by $T(e)$ encodes a number different from 0.

Proof. The proof is by verification. We begin with the first item. The second item is shown similarly to the first item.

Let k be as in the lemma and assume that there is k' with $k < k' \leq 2^n - 1$ such that the automaton accepts $T(e)$ from q_i^{yes} . Let $n = k' - k$. We can construct a winning play for \exists by choosing the pair $(q_1^{\text{yes}}, q^\top)$ for $n - 1$ times when reading the \mathbf{and}_1 . For each such transition, time passes for one unit, so x has value $k' - 1$ after doing this. Then let \exists choose $(q^\top, q_1^{\text{yes}})$ at the next occurrence of a_1^\vee . Since $k' \leq 2^n - 1$, a winning play for the tree $\mathbf{and} \max x$ is easily constructed from the winning play for $T(e)$ from clock value k' . Conversely, assume that \mathcal{P}_{cnt} accepts $T(\mathbf{exists} e)$ from q_i^{yes} from some clock value $k \in [2^n]$. There are two cases: either \exists chooses $(q^\top, q_1^{\text{yes}})$ eventually, or she always chooses $(q_1^{\text{yes}}, q^\top)$. Note that, since the priority of q_1^{yes} is 1, this is a losing play. Hence, \exists chooses the first option after n many choices of the second option, whence time has passed by $n + 1$ when reaching the tree

$T(\text{and } \max x)$. Clearly, $k + n + 1 \leq 2^n - 1$ for otherwise the play is obviously not winning. Hence $k' = k + n + 1$ is as desired. The statement for the state q_i^{no} is proved in a similar fashion.

The third item follows from the first and second items and the pattern of binary incrementation outlined above, the last item follows from the second item and Def. 11. \blacktriangleleft

► **Lemma 13.** *Let $C: 2_2^n \times 2_2^n \rightarrow \hat{\Gamma}$ be the unique certificate for n and \mathcal{M} . Let $r, c: \bullet$ be expressions such that $T(r), T(c)$ encode $m, m' \in [2_2^n]$. Let $b, b_1, b_2, b_3 \in \hat{\Gamma}$.*

1. \mathcal{P}_{red} accepts $T(\text{check } r c)$ from state $q_1^\#$ iff either $m' = 0$ or $m' = 2_2^n - 1$.
 2. \mathcal{P}_{red} accepts $T(\text{check } r c)$ from state q_1^\square iff both $2 \leq m' < 2_2^n - 1$ and $m = 0$.
 3. \mathcal{P}_{red} accepts $T(\text{check } r c)$ from state $q_1^{(q_1, \square)}$ iff both $m = 0$ and $m' = 1$.
 4. \mathcal{P}_{red} accepts $T(\text{next } r c)$ from state $q^{(b_1, b_2, b_3)}$ iff (I) both $m > 0$ and $0 < m' < 2_2^n - 1$ and (II) \mathcal{P}_{red} accepts $T(F(\text{dec } r) (\text{dec } c))$ from state q^{b_1} and $T(F(\text{dec } r) c)$ from state q^{b_2} and $T(F(\text{dec } r) (\text{inc } c))$ from state q^{b_3} .
 5. \mathcal{P}_{red} accepts $T(F r c)$ from state q^b iff $C(m, m') = b$.
- Finally, \mathcal{P}_{red} accepts $T(\mathcal{G}_{\text{red}})$ iff $C(2_2^n - 1, 0) = (q_{\text{acc}}, \square)$.

Proof. The first three items are shown directly. We show the first item, the other two are shown similarly. Note that the root $T(\text{check } r c)$ is labeled by **or3**, and that $\delta(q_1^\#, \text{or3}) = (q_1^{\text{yes}}, q^\top, q^\top)$. Since every tree is accepted by q^\top , it remains to argue that the automaton accepts $T(\text{and}(\text{zero } (\text{inc } r) \text{zero } r))$ from q_1^{yes} iff $m' = 0$ or $m' = 2_2^n - 1$. The former is equivalent to the automaton accepting $T(\text{zero } (\text{inc } r))$ and $T(\text{zero } r)$ from q_1^{yes} . By Lemma 12, this is the case iff m' , i.e., the number encoded by r , is either $2_2^n - 1$ or 0, which is what we wanted to show.

The proof for the last two items is by simultaneous induction over m' . Before we begin the induction consider the structures of the respective trees.

The tree root of $T(\text{next } r c)$ is labeled by **and**, and both its left and right subtrees are labeled by **and3**. We have $\delta(q^{(b_1, b_2, b_3)}, _, \text{and}) = (q_1^{\text{yes}}, q^{(b_1, b_2, b_3)})$ and $\delta(q^{(b_1, b_2, b_3)}, _, \text{and3}) = (b_1, b_2, b_3)$, we obtain that \mathcal{P}_{red} accepts $T(\text{next } r c)$ iff it accepts the following:

- $T(\text{neg } (\text{isZero? } r))$ from q_1^{yes} ,
- $T(\text{neg } (\text{isZero? } c))$ from q_1^{yes} ,
- $T(\text{ltMax? } c)$ from q_1^{yes} ,
- $T(F(\text{dec } r) (\text{dec } c))$ from q^{b_1} ,
- $T(F(\text{dec } r) c)$ from q^{b_2} , and
- $T(F(\text{dec } r) (\text{inc } c))$ from q^{b_3} .

By Lemma 12, the first three are equivalent to $m > 0$ and $0 < m' < 2_2^n - 1$, as in the claim.

Similar considerations yield that \mathcal{P}_{red} accepts the tree generated by $T(F r c)$ iff it accepts one of $T(\text{next } r c)$ and $T(\text{check } r c)$. Closer inspection of the requirements w.r.t. m and m' yield that at most one of the latter can be the case (since acceptance of $T(\text{next } r c)$ requires $m > 0$ and $0 < m' < 2_2^n - 1$, while acceptance of $T(\text{check } r c)$ requires $m = 0$ or $m' = 0$ or $m' = 2_2^n - 1$).

Now let $m = 0$. The fourth item follows readily from the previous considerations, since both acceptance of $T(\text{next } r c)$ and the premise of the item require $m > 0$. For the fifth item, the claim follows since only acceptance of $T(\text{check } r c)$ is in question (since $m > 0$), and the first three items cover exactly the possibilities for $C(0, m')$, namely $\#$ (for $m' = 0$ and $m' = 2_2^n - 1$), (q_1, \square) (for $m' = 1$) and \square (for $1 < m' < 2_2^n - 1$).

16:20 Real-Time Higher-Order Recursion Schemes

Now let $m > 0$ and assume that the fourth and fifth item have been shown for $m - 1$. For the fourth item, the claim now follows easily from the itemized list above and the induction hypothesis applied to the last three items of the list. For the fifth item of the lemma, if $m' = 0$ or $m' = 2_2^n - 1$, apply the first item of the lemma, and if $0 < m' < 2_2^n - 1$, apply the fourth item of the lemma.

The last, unitemized claim then follows straightforwardly. ◀

Time Series Anomaly Detection Leveraging MSE Feedback with AutoEncoder and RNN

Ibrahim Delibasoglu ✉ 

Department of Computer and Information Science (IDA), Linköping University, Sweden
Software Engineering, Sakarya University, Turkey

Fredrik Heintz ✉ 

Department of Computer and Information Science (IDA), Linköping University, Sweden

Abstract

Anomaly detection in time series data is a critical task in various domains, including finance, healthcare, cybersecurity and industry. Traditional methods, such as time series decomposition, clustering, and density estimation, have provided robust solutions for identifying anomalies that exhibit distinct patterns or significant deviations from normal data distributions. Recent advancements in machine learning and deep learning have further enhanced these capabilities. This paper introduces a novel method for anomaly detection that combines the strengths of autoencoders and recurrent neural networks (RNNs) with an reconstruction error feedback mechanism based on Mean Squared Error. We compare our method against classical techniques and recent approaches like OmniAnomaly, which leverages stochastic recurrent neural networks, and the Anomaly Transformer, which introduces association discrepancy to capture long-range dependencies and DCDetector using contrastive representation learning with multi-scale dual attention. Experimental results demonstrate that our method achieves superior overall performance in terms of precision, recall, and F1 score. The source code is available at <http://github.com/mribrahim/AE-FAR>

2012 ACM Subject Classification Computing methodologies → Machine learning algorithms

Keywords and phrases Time series, Anomaly, Neural networks

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.17

Supplementary Material *Software:* <https://github.com/mribrahim/AE-FAR/>

1 Introduction

Time series data, characterized by its sequential and temporal nature, is extensively utilized across numerous applications, including finance, healthcare, manufacturing, and environmental monitoring. Detecting anomalies within time series data is a critical task to implement an early warning mechanism for unusual patterns that may indicate events such as system failures and frauds [8, 1]. Anomalies, often referred to as outliers or deviants, are data points that deviate markedly from the expected values. In industry, anomalies are often so rare and it is too hard to label them for supervised learning. Hence, most studies in the literature focus on unsupervised methods such as clustering [15] and density estimation [2], or learning representations for only the normal data (supervising only for normal data). Because deep neural networks have the capacity to learn the representation of the normal data, reconstruction from the embedded of that data can be used to determine the anomalies. It means that reconstruction-based models [20] learn how to reconstruct the normal data, and high error in the reconstructed data indicates the anomalies. Similarly, forecasting-based methods [6] are also used to detect anomalies.

Contrastive representation got attention in computer vision tasks [3, 5], and applied for time series problem in a recent study [21]. Contrastive representation learning aims an embedding space emphasizing the distinction between similar and dissimilar data points. Combination of forecasting and reconstruction-based networks are also implemented in the



© Ibrahim Delibasoglu and Fredrik Heintz;

licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 17; pp. 17:1–17:12

Leibniz International Proceedings in Informatics



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

literature [22, 19]. Although prior studies have achieved significant success, they may still suffer from performance degradation, particularly when anomalous points are not uniformly distributed, and anomaly scores or reconstruction errors may vary in the different regions of the data.

In this paper, we propose reconstruction networks based on Mean Squared Error (MSE) Feedback, augmented with Attention and Recurrent Neural Network (RNN) modules. We implement two variations of this architecture: AE-FAR, which employs an Autoencoder with Feedback Attention Reconstruction, and VAE-FAR, which utilizes a Variational Autoencoder (VAE) with Feedback Attention Reconstruction. In VAE-FAR, we integrate an Long Short-Term Memory (LSTM)-VAE with dual attention modules. Specifically, we implement two parallel graph attention mechanisms proposed in MTAD-GAT [22]. These modules are designed to capture temporal dependencies within time series data and relationships between features, enhancing the model’s ability to detect anomalies effectively. Previous approaches mostly labels the anomalies if the reconstruction/forecasting error is larger than a prior threshold or dynamically determined threshold. According to the results we observed during our experiments, the reconstruction error does not progress similarly on the entire data set, there are fluctuations. Therefore, it is important to apply different thresholds at different time intervals, i.e. to identify the peaks in the errors. Since the previous and next values must be considered in the peak detection problem, a similar structure is placed in the proposed architecture by the reconstruction error feedback. We integrate a lightweight autoencoder model, a RNN module, and an attention mechanism in our model. Proposed architecture aims to improve reconstruction accuracy by dynamically adjusting the reconstruction of input sequences based on the MSE feedback with an RNN module. Our main motivation is that MSE feedback mechanism further enhances the model’s ability to identify anomalies supported by the anomaly criterion. Our anomaly detection method is based on moving average and standard deviation within a sliding window for the reconstruction error.

- Our network is lightweight compared to the state-of-the-art methods and performs well with the proposed MSE feedback module and thresholding for anomaly detection.
- We train a simple autoencoder model and LSTM-VAE improved with graph attention mechanisms. Then we incorporate attention mechanism for the reconstructed values and obtained mean squared error. These pre-trained models are integrated with RNN to further enhance the reconstruction process by remembering the reconstruction MSE errors.
- We use a sliding window-based anomaly detection mechanism that focuses on sudden error changes rather than relying on a predefined general threshold. We show that our model’s architecture inherently supports the sliding window-based thresholding approach.
- Through extensive experiments, we show that AE-FAR achieves an overall F1 score of 93.38 on the MSL, SWAT and SMD datasets and 58.48 on the pulp-and-paper industry dataset. VAE-FAR has an overall F1 score of 93.65 on the MSL, SWAT and SMD datasets, and 50.78 on the pulp-and-paper industry dataset.

2 Related works

Classical methods for time series anomaly detection have evolved significantly, adapting to various data characteristics. Techniques such as time series decomposition, clustering, and density estimation have offered robust solutions for identifying anomalies in time series data characterized by distinct patterns or substantial deviations from normal data distributions. Examples of classical anomaly detection methods include the Local Outlier Factor (LOF) [2]

and Deep Autoencoding Gaussian Mixture Model (DAGMM) [24], both of which are grounded in density estimation principles. Distance to the cluster center is used as an anomaly score in clustering based methods. ITAD [16] applies a tensor-based decomposition to model normal behavior patterns and uses clustering techniques to group similar patterns. Deep-SVDD [15] trains a neural network to learn a representation of normal data and the objective is to map normal instances close to a central point in the latent space. IForest [11] detects anomalies by isolating observations through a recursive partitioning process. It randomly selects a feature and then chooses a split value between the minimum and maximum values of that feature. Autoregressive models predicting the future values based on past observations are another type of anomaly detection method. Recently, with the rise of deep learning, RNNs and their variants such as LSTM networks have been extensively applied, capable of capturing long-term dependencies and temporal patterns for detecting anomalies in diverse domains. CL-MPPCA [18] an extension of ARIMA, is one such method that compares predicted values with actual values and detects deviations that exceed a certain threshold. It combines the capabilities of LSTM-based neural networks and mixture of several probabilistic PCA models.

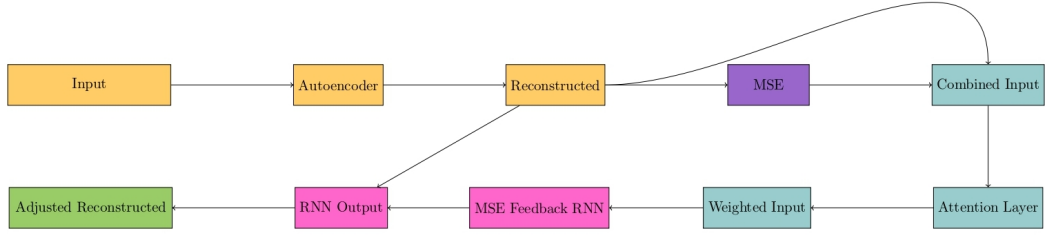
Autoencoders are a type of neural networks designed to learn embedded representations of data, typically for the purposes of dimensionality reduction or feature learning. They consist of two main components: an encoder that maps the input data to a lower-dimensional latent space, and a decoder that reconstructs the input data from this latent representation. VAEs extend the autoencoder by encoding inputs into distributions, typically Gaussian. The decoder reconstructs the data from these distributions instead of fixed points in the latent space. Employing VAE, LSTM-VAE [13] and some improved versions [17, 9] of LSTM-VAE are applied for the anomaly problem. OmniAnomaly [17] employs a stochastic RNN framework with GRU, integrating VAEs to model the temporal dependencies. InterFusion [10] proposes a hierarchical VAE to model inter-metric and temporal relationships. MAD-GAN [9] is Generative Adversarial Network(GAN) based method in which LSTM is used in generator and discriminator networks. DGHL [4] proposes a hierarchical latent space representation with convolution networks. BEATGAN [23] is also a reconstruction-based method using generative adversarial networks. MTAD-GAT [22] combines forecasting-based and reconstruction-based networks and anomalies are detected by using both the these outputs. AnomalyTransformer [20] introduces a new Anomaly-Attention mechanism to compute the association discrepancy focusing on the difference between normal and anomalous patterns. DCDetector [21] is a contrastive learning based multi-scale dual attention model.

3 Method

Let X be a multivariate time-series sequence of length N :

$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$$

The unsupervised time series anomaly detection problem aims to ascertain the anomalous nature of X without the availability of labeled data. This work proposes a composite model that combines an AE or VAE, an attention mechanism, and RNN to address this problem. The general overall of proposed architecture is shown in Figure 1. AE-FAR integrates multiple neural network components to enhance time series prediction accuracy through error correction. It consists of three main parts: an autoencoder, an attention layer, and RNN.



■ **Figure 1** General overview of the proposed AE-FAR architecture.

3.1 Overall architecture

We train a simple AE to reconstruct the input \mathbf{X} . \hat{X} represents the reconstructed input values with AE network. In the proposed composite model we use the MSE error of that AE.

$$\begin{aligned} \hat{X} &= \text{Autoencoder}(X) \\ \text{MSE} &= \frac{1}{n} \sum_{i=1}^n (x_i - \hat{x}_i)^2 \end{aligned} \quad (1)$$

The combination of \hat{X} and MSE is passed to the attention mechanism. The attention layer is responsible for assigning different importance levels to various parts of the input sequence. That helps the model focus on the most relevant features during the prediction process. The attention layer uses a sequential model consisting of linear transformations, a Tanh activation function, and a Softmax function to compute attention weights. In the following equation, α represents the attention weights. These weights are then applied to the input features to produce a weighted input.

$$\begin{aligned} z &= [\hat{X}, \text{MSE}] \\ \alpha &= \text{Softmax}(\text{Linear}(\tanh(\text{Linear}(z)))) \\ z' &= \alpha \odot z. \end{aligned} \quad (2)$$

The MSE Feedback RNN module is a recurrent neural network that processes sequences of inputs and provides feedback based on the mean squared error between the predicted and actual values. This module includes an RNN layer followed by a fully connected layer. The RNN processes the input sequence and outputs hidden states, which are then transformed by the fully connected layer to produce the final output. Finally, the adjusted reconstruction, incorporating RNN predictions, forms the model's output, aiming to refine and improve the accuracy of predictions over time.

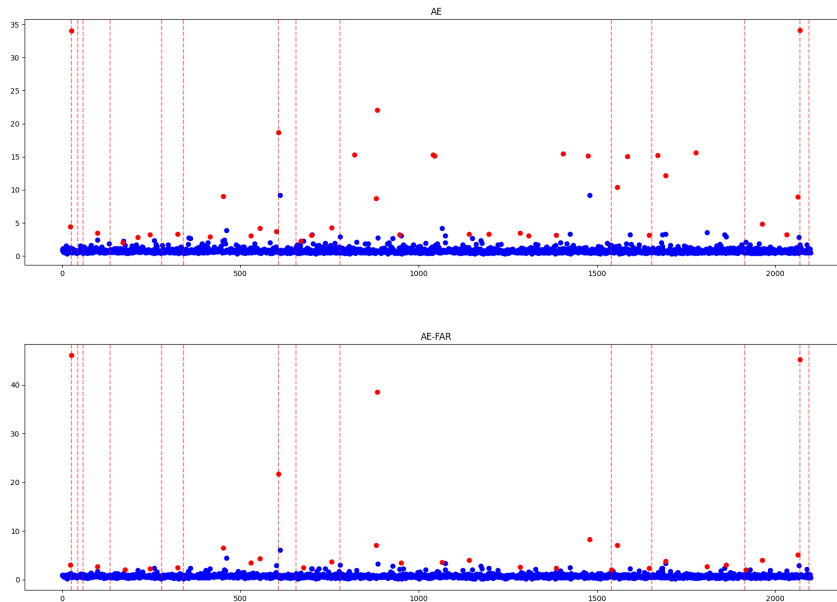
$$\begin{aligned} h &= \text{RNN}(z') \\ \hat{y} &= \hat{X} + h \end{aligned} \quad (3)$$

AE-FAR integrates the autoencoder, attention mechanism, and MSE Feedback RNN to enhance the forecasting capabilities. Combining all parts, the model output is:

$$\hat{y} = \hat{X} + \text{RNN}(\alpha \odot [\hat{X}, \text{MSE}]) \quad (4)$$

This approach aims to enhance the reconstructed output \hat{y} by incorporating error-driven adjustments from the RNN, thereby improving predictive accuracy in time-series analysis. Sample reconstruction errors and anomaly points are shown in Figure 2 comparing the

results of the AE with the proposed AE-FAR model, highlighting the influence of the MSE feedback module. The figure illustrates a region containing 14 actual anomalies, represented by vertical red dotted lines. The AE method detects 39 anomaly points, whereas the AE-FAR method identifies 30 anomaly points. It is observed that the AE-FAR method has a lower mean error value compared to the AE method, and its anomaly detections are closer to the actual anomaly points. The mean distances between the predicted anomaly indexes and the ground truth anomaly indexes are 87.5 and 72.8 for the AE method and the AE-FAR method, respectively.



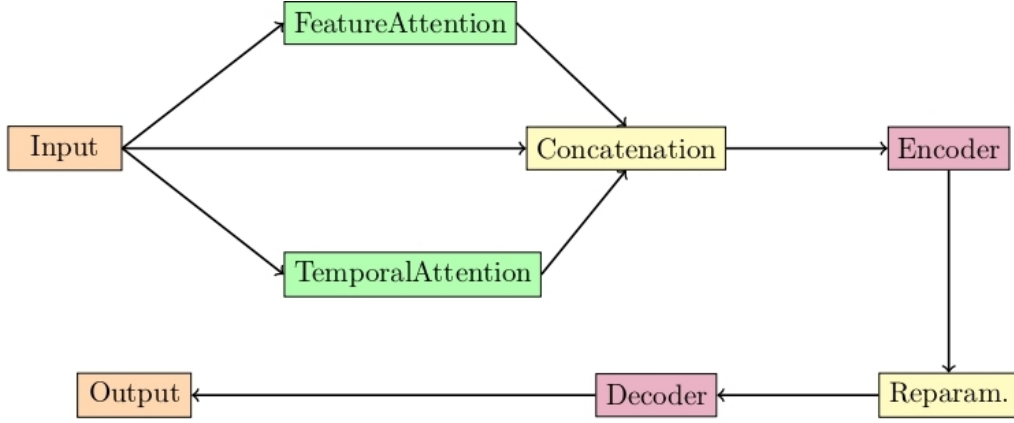
■ **Figure 2** Reconstruction error and detected anomaly points. Vertical red dotted lines indicate the real anomalies.

3.2 LSTM-VAE with attention layers

We employ a different variant of that proposed architecture by using VAE instead of AE. In the VAE network, we use graph-based feature and attention layers proposed in MTAD-GAT [22]. The general overview of the VAE improved with the attention layers is shown in the Figure 3. Feature attention layer aims to emphasize the most relevant features for each time step while temporal attention layer applies attention across the temporal dimension, focusing on the most significant time steps for each feature. The encoded representation is formed by concatenating the original input X with the outputs of the feature and temporal attention layers. Reparameterize function implements the reparameterization trick to sample latent variables from a Gaussian distribution inferred by the encoder's output. Decoder module reconstructs the input sequence from the sampled latent variables.

3.3 Anomaly criterion

We use a sliding window-based approach to dynamically compute thresholds based on statistical properties of the reconstruction error. The proposed anomaly detection strategy is based on the MSE values using a moving average and standard deviation within a sliding



■ **Figure 3** General overview of the VAE with graph attention.

window. This approach is particularly useful in anomaly detection where the MSE distribution may change over time. Our model’s architecture inherently supports the sliding window-based thresholding mechanism, making it more effective in dynamically adjusting to varying anomaly patterns within the time series data. Window size is a significant hyper-parameter in time series analysis to split time series into instances instead of using only a single point as input. We use the same window size for the the sliding window used to compute the moving average and moving standard deviation. The adaptive nature of the sliding window approach ensures that the detection mechanism remains sensitive to new patterns, providing a robust tool for maintaining robust performance in anomaly detection.

Given a sequence of MSE values $mse_list = [mse_1, mse_2, \dots, mse_t]$, dynamic threshold is calculated based on average and standard deviation for a window size w as follows:

$$\begin{aligned}
 avg[t] &= \frac{1}{w} \sum_{i=0}^w mse_{t-i} \\
 std[t] &= \sqrt{\frac{1}{w} \sum_{i=0}^w (mse_{t-i} - avg[t])^2}
 \end{aligned} \tag{5}$$

The dynamic threshold $T[t]$ for each input X in a time step is then calculated using a user-defined threshold factor β , and anomalies(A) are identified at each time step t if the MSE mse_t exceeds the computed dynamic threshold $T[t]$:

$$\begin{aligned}
 T[t] &= avg[t] + \beta \times std[t] \\
 A_t &= \begin{cases} 1 & \text{if } mse_t > T[t] \\ 0 & \text{otherwise} \end{cases}
 \end{aligned} \tag{6}$$

4 Experiments

4.1 Benchmark datasets

We perform experiments using four datasets: the Mars Science Laboratory (MSL) [7] rover dataset, the Server Machine Dataset (SMD) [17], Secure Water Treatment (SWAT) [12], and pulp-and-paper manufacturing industry dataset [14]. MSL is collected by NASA and it

reflects the rover’s operational status and environment. SMD contains data collected from various server machines in a data center. It includes metrics such as CPU usage, memory usage, and network traffic, aimed at detecting anomalies that may indicate hardware failures or network issues. SWAT includes sensor data from critical infrastructure systems. The pulp-and-paper industry dataset used in our experiments has a total of 59 input features, after removing the categorical features x_{28} and x_{61} . This dataset consists of 18,398 rows, of which only 124 rows are labeled as anomalies. A significant characteristic that distinguishes this dataset from others is the presence of non-consecutive anomalies, which poses a unique challenge for anomaly detection methods. We create two subset datasets (called Paper-1 and Paper-2) for testing purposes, containing 29 and 58 anomalies respectively.

We adopt a widely recognized adjustment technique to ensure a fair comparison with existing methods in the literature. That adjustment approach refines predicted anomaly labels in time series data by ensuring that if any single point in an anomalous segment is detected, the entire segment is marked as anomalous. This strategy is justified by the observation that detecting a single anomalous point triggers an alert for the entire segment in real-world applications. While this adjustment technique significantly impacts datasets like MSL, SMD, and SWAT, it has no effect on Paper-1 and Paper-2 datasets. We employ a neighborhood-based matching strategy to assess the performance on the paper industry dataset. This approach was chosen due to the difficulty of precisely determining the exact timing of anomalies in this particular industry dataset. We apply a slight temporal misalignments between true and predicted anomalies by defining a window size k . True positives are correctly predicted anomalies within k indices of actual anomalies, while false positives are predictions without a corresponding true anomaly in the window. This approach enhances robustness in detecting anomalies in time series data by accounting for small deviations, thus providing a more accurate assessment of model performance in practical scenarios. It offers a comprehensive evaluation of precision, recall, and F1 score, rounded for clarity. Table 4 represents the performance comparison with this approach.

4.2 Implementation details

We use a fixed window size 50 for all datasets to split time series into instances. The same value of 50 is also used for k to apply a slight temporal misalignments for pulp-and-paper industry dataset. The AE model consists of four linear layers: the first layer compresses the input from a flattened dimension of $window * inputsize$ to 32 neurons, the second layer further reduces the size to 16 neurons, and then gradually expanded to the original input size. For the VAE network, the dimension of *hidden size* and *latent size* are 64 and 32, respectively. RNN component is an 8-layer RNN that processes the combined input with dimensions $inputsize + 1$. It uses a hidden size of $inputdim/2$ for the hidden state and maps the output to the original input dimension using a fully connected layer. The attention mechanism takes an input dimension of $inputsize + 1$ and maps it to an attention dimension 64 . The experiments with these hyperparameter selection are implemented in PyTorch with NVIDIA GeForce RTX 3060 graphic card. We use Adam optimizer with an initial learning rate of 10^{-4} and set the batch size to 128. We used early stopping during training by monitoring the validation error, with an early stop value set to 10, to prevent overfitting and ensure optimal model performance. We choose a fixed threshold value as the default comparison, and for the proposed anomaly criterion, we define different threshold factors β 4, 5 and 6.5 for SMD, SWAT and MSL, respectively.

■ **Table 1** Ablation study on MSL, SMD and SWAT. * indicates the proposed anomaly criterion instead of fixed threshold.

Methods	MSL			SMD			SWAT		
	<i>P</i>	<i>R</i>	<i>F1</i>	<i>P</i>	<i>R</i>	<i>F1</i>	<i>P</i>	<i>R</i>	<i>F1</i>
AE	95.70	20.91	34.32	38,10	68,09	48,86	91.62	76.41	83.33
AE-FAR	83.57	90.45	86.87	62.72	52.01	56.86	92.12	76.41	83.53
VAE-FAR	52.87	96.87	68.41	81.28	76.81	78.98	97.43	77.09	86.07
<i>AE*</i>	85.23	99.54	91.83	91.89	87.78	89.79	94.87	96.97	95.91
<i>AE – FAR*</i>	92.59	95.13	93.85	92.01	88.20	90.06	97.40	96.36	96.87
<i>VAE – FAR*</i>	90.23	94.89	92.50	89.95	94.59	92.21	94.49	98.07	96.24

4.3 Results and analysis

We use commonly-used evaluation measures: precision, recall, F1 score for performance comparison. The ablation study represented in Table 1 evaluates the performance on MSL, SMD and SWAT benchmark datasets. The first three rows represent the performance with fixed general threshold, while the other rows represent the performance with the proposed anomaly criterion. The best results obtained with both approaches are shown separately in **bold**. The results clearly demonstrate the effectiveness of the anomaly criterion method improving the AE, AE-FAR and VAE-FAR models. The AE with anomaly criterion shows substantial gains in recall and F1 scores, especially for the MSL and SWAT datasets. VAE-FAR has better recall compared to the AE-FAR with the general fixed threshold. Both of the AE-FAR and VAE-FAR models, when combined with the proposed anomaly criterion, consistently outperforms other configurations, achieving the highest F1 scores across all datasets. This highlights the robustness and accuracy of the proposed methods in multivariate time-series anomaly detection.

Table 2 represents the ablation study for subsets of pulp-and-paper industry datasets. The AE-FAR model alone shows high precision for the Paper-1 dataset at 66.67%, but recall is very low at 14.81%, resulting in an F1 score of 24.24%. This suggests that the model is very conservative in identifying anomalies, leading to fewer false positives but many missed anomalies with the fixed threshold. On the other hand, VAE-FAR has more stabil and better results compared to the AE-FAR with the fixed threshold. Using the proposed anomaly criterion significantly improves performance as shown in last three rows. For the Paper-1 dataset, it achieves a balanced precision of 56.41% and a high recall of 81.48%, resulting in the highest F1 score of 66.67% among all methods. VAE-FAR* has the second best value with an F1 score of 58.46 with the anomaly criterion. For the Paper-2 dataset, AE-FAR* achieves a precision of 37.29% and a recall of 77.19%, resulting in the highest F1 score of 50.29%. This demonstrates the effectiveness of the anomaly criterion in improving the model’s ability to detect anomalies accurately. The AE model benefits significantly from the anomaly criterion, especially in terms of recall, but at the cost of precision. The AE-FAR model alone shows high precision but struggles with recall, indicating a conservative anomaly detection approach. The combination of AE-FAR with the anomaly criterion achieves the best overall performance, striking a balance between precision and recall and resulting in the highest F1 scores for both datasets.

Table 3 shows the performance comparison of different anomaly detection methods in the literature. For three real world datasets MSL, SMD and SWAT: AE-FAR and VAE-FAR achieve an overall F1-Score of 93.59 and 93.65, while DCdetector and AnomalyTransformer have 93.37 and 93.32, respectively. Table 4 presents the performance comparison with the

■ **Table 2** Ablation study on pulp-and-paper industry dataset. * indicates the proposed anomaly criterion instead of fixed threshold.

Methods	Paper-1			Paper-2		
	<i>P</i>	<i>R</i>	<i>F1</i>	<i>P</i>	<i>R</i>	<i>F1</i>
AE	44.74	62.96	52.31	23.58	43.86	30.67
AE-FAR	66.67	14.81	24.24	40.0	14.04	20.78
VAE-FAR	45.83	40.74	43.14	31.34	36.84	33.87
<i>AE*</i>	38.46	92.59	54.35	29.49	80.70	43.19
<i>AE – FAR*</i>	56.41	81.48	66.67	37.29	77.19	50.29
<i>VAE – FAR*</i>	50.0	70.37	58.46	32.73	63.16	43.11

■ **Table 3** Results on multivariate benchmark datasets. All results are presented as percentages; the best values are in **bold**, and the second-best are underlined.

<i>Dataset</i>	<i>MSL</i>			<i>SWAT</i>			<i>SMD</i>		
	<i>P</i>	<i>R</i>	<i>F1</i>	<i>P</i>	<i>R</i>	<i>F1</i>	<i>P</i>	<i>R</i>	<i>F1</i>
LOF	47.72	85.25	61.18	72.15	65.43	68.62	56.34	39.86	46.68
IForest	53.94	86.54	66.45	49.29	44.95	47.02	42.31	73.29	53.64
DAGMM	89.60	63.93	74.62	89.92	57.84	70.40	67.30	49.89	57.30
ITAD	69.44	84.09	76.07	63.13	52.08	57.08	86.22	73.71	79.48
CL-MPPCA	73.71	88.54	80.44	76.78	81.50	79.07	82.36	76.07	79.09
Deep-SVDD	91.92	76.63	83.58	80.42	84.45	82.39	78.54	79.67	79.10
BeatGAN	89.75	85.42	87.53	64.01	87.46	73.92	72.90	84.09	78.10
OmniAnomaly	89.02	86.37	87.67	81.42	84.30	82.83	83.68	86.82	85.22
InterFusion	81.28	92.70	86.62	80.59	85.58	83.01	87.02	85.43	86.22
AnomalyTransformer	92.09	<u>95.15</u>	93.59	91.55	96.73	94.07	89.40	95.45	92.33
DCdetector	93.69	99.69	96.60	93.11	99.77	<u>96.33</u>	83.59	91.10	87.18
AE-FAR	<u>92.59</u>	95.13	<u>93.85</u>	97.40	96.36	96.87	92.01	88.20	90.06
VAE-FAR	90.23	94.89	92.50	<u>94.49</u>	<u>98.07</u>	96.24	89.95	<u>94.59</u>	<u>92.21</u>

DCDetector on two subsets of a pulp-and-paper manufacturing industry anomaly detection dataset: Paper-1 and Paper-2. For Paper-1, Introducing the proposed anomaly criterion to DCDetector slightly changes the results, with a precision of 31.43%, recall of 40.74%, and F1 score of 35.48%. There is a small reduction in precision and recall, which suggests that the proposed thresholding might be filtering out some true positives, leading to a lower overall performance. For Paper-2, proposed anomaly criteria improves the performance slightly, achieving a precision of 28.21%, recall of 38.60%, and F1 score of 32.59%. On the other hand, AE-FAR outperforms the other methods significantly for both Paper-1 and Paper-2 datasets. Although VAE-FAR shows lower performance compared to AE-VAE, it is seen that it gives much better performance than DCDetector. The proposed anomaly criterion for DCDetector shows minor improvements but does not suffice to compete with the performance of AE-FAR. These findings highlight the effectiveness of the proposed approach in handling the rare anomalies in the pulp-and-paper manufacturing industry datasets.

5 Conclusion

The proposed AE-FAR/VAE-FAR models effectively combine AE/VAE, an attention mechanism, and RNN to improve anomaly detection in time series data. The AE/VAE is employed to reconstruct the input data, aiming to capture the underlying normal patterns. They

■ **Table 4** Results on pulp-and-paper manufacturing industry dataset. All results are presented as percentages; the best values are in **bold**, and the second-best are underlined.

<i>Dataset</i>	<i>Paper-1</i>			<i>Paper-2</i>		
	P	R	F1	P	R	F1
DCdetector	35.29	44.44	39.34	23.17	33.33	27.34
DCdetector*	31.43	40.74	35.48	28.21	38.60	32.59
AE-FAR	56.41	81.48	66.67	37.29	77.19	50.29
VAE-FAR	<u>50.0</u>	<u>70.37</u>	<u>58.46</u>	<u>32.73</u>	<u>63.16</u>	<u>43.11</u>

consist of an encoder that maps the input data to a lower-dimensional latent space and a decoder that reconstructs the input from this latent representation. The attention mechanism is integrated to enhance the model’s focus on significant parts of the input data. The RNN component processes the reconstructed data produced by the autoencoder and reconstruction errors. The RNN output is used to adjust the reconstructed input, providing the final output of the model. This integrated approach leads to more accurate anomaly detection results. The proposed models outperform state-of-the-art approaches overall on four datasets.

Future work. We need to focus on anomaly criterion selecting threshold factor β automatically, and better detection and localization for discontinuous anomalies such as pulp-and-paper dataset.

References

- 1 Archana Anandkrishnan, Senthil Kumar, Alexander Statnikov, Tanveer Faruquie, and Di Xu. Anomaly detection in finance: editors’ introduction. In *KDD 2017 Workshop on Anomaly Detection in Finance*, pages 1–7. PMLR, 2018.
- 2 Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000. doi:10.1145/342009.335388.
- 3 Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in neural information processing systems*, 33:9912–9924, 2020.
- 4 Cristian I Challu, Peihong Jiang, Ying Nian Wu, and Laurent Callot. Deep generative model with hierarchical latent factors for time series anomaly detection. In *International Conference on Artificial Intelligence and Statistics*, pages 1643–1654. PMLR, 2022. URL: <https://proceedings.mlr.press/v151/challu22a.html>.
- 5 Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020. URL: <http://proceedings.mlr.press/v119/chen20j.html>.
- 6 Ailin Deng and Bryan Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4027–4035, 2021. doi:10.1609/AAAI.V35I5.16523.
- 7 Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 387–395, 2018.
- 8 Bedeuro Kim, Mohsen Ali Alawami, Eunsoo Kim, Sanghak Oh, Jeongyong Park, and Hyounghick Kim. A comparative study of time series anomaly detection models for industrial control systems. *Sensors*, 23(3):1310, 2023. doi:10.3390/S23031310.

- 9 Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In *International conference on artificial neural networks*, pages 703–716. Springer, 2019. doi:10.1007/978-3-030-30490-4_56.
- 10 Zhihan Li, Youjian Zhao, Jiaqi Han, Ya Su, Rui Jiao, Xidao Wen, and Dan Pei. Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 3220–3230, 2021. doi:10.1145/3447548.3467075.
- 11 Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth IEEE international conference on data mining*, pages 413–422. IEEE, 2008. doi:10.1109/ICDM.2008.17.
- 12 Aditya P Mathur and Nils Ole Tippenhauer. Swat: A water treatment testbed for research and training on ics security. In *2016 international workshop on cyber-physical systems for smart water networks (CySWater)*, pages 31–36. IEEE, 2016. doi:10.1109/CYSWATER.2016.7469060.
- 13 Daehyung Park, Yuuna Hoshi, and Charles C Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018. doi:10.1109/LRA.2018.2801475.
- 14 Chitta Ranjan, Mahendranath Reddy, Markku Mustonen, Kamran Paynabar, and Karim Pourak. Dataset: rare event classification in multivariate time series. *arXiv preprint arXiv:1809.10717*, 2018.
- 15 Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402. PMLR, 2018.
- 16 Youjin Shin, Sangyup Lee, Shahroz Tariq, Myeong Shin Lee, Okchul Jung, Daewon Chung, and Simon S Woo. Itad: integrative tensor-based anomaly detection system for reducing false positives of satellite systems. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 2733–2740, 2020. doi:10.1145/3340531.3412716.
- 17 Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2828–2837, 2019. doi:10.1145/3292500.3330672.
- 18 Shahroz Tariq, Sangyup Lee, Youjin Shin, Myeong Shin Lee, Okchul Jung, Daewon Chung, and Simon S Woo. Detecting anomalies in space using multivariate convolutional lstm with mixtures of probabilistic pca. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2123–2133, 2019. doi:10.1145/3292500.3330776.
- 19 Lawrence C Wong. *Time Series Anomaly Detection using Prediction-Reconstruction Mixture Errors*. PhD thesis, Massachusetts Institute of Technology, 2022.
- 20 Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly transformer: Time series anomaly detection with association discrepancy. *arXiv preprint arXiv:2110.02642*, 2021. arXiv:2110.02642.
- 21 Yiyuan Yang, Chaoli Zhang, Tian Zhou, Qingsong Wen, and Liang Sun. Dcdetector: Dual attention contrastive representation learning for time series anomaly detection. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3033–3045, 2023. doi:10.1145/3580305.3599295.
- 22 Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 841–850. IEEE, 2020. doi:10.1109/ICDM50108.2020.00093.
- 23 Bin Zhou, Shenghua Liu, Bryan Hooi, Xueqi Cheng, and Jing Ye. Beatgan: Anomalous rhythm detection using adversarially generated time series. In *IJCAI*, volume 2019, pages 4433–4439, 2019. doi:10.24963/IJCAI.2019/616.

17:12 AE-FAR: Autoencoder with Feedback Attention Reconstruction

- 24 Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*, 2018.

Full Characterisation of Extended CTL*

Massimo Benerecetti  

Università di Napoli Federico II, Italy

Laura Bozzelli  

Università di Napoli Federico II, Italy

Fabio Mogavero  

Università di Napoli Federico II, Italy

Adriano Peron  

Università di Trieste, Italy

Abstract

The precise identification of the expressive power of logic languages used in formal methods for specifying and verifying run-time properties of critical systems is a fundamental task and *characterisation theorems* play a crucial role as model-theoretic tools in this regard. While a clear picture of the expressive power of linear-time temporal logics in terms of word automata and predicate logics has long been established, a complete mapping of the corresponding relationships for branching-time temporal logics has proven to be a more elusive task over the past four decades with few scattered results. Only recently, an *automata-theoretic characterisation* of both CTL* and its full- ω -regular extension ECTL* has been provided in terms of *Symmetric Hesitant Tree Automata* (HTA), with and without a suitable *counter-freeness restriction* on their linear behaviours. These two temporal logics also correspond to the *bisimulation-invariant* semantic fragments of *Monadic Path Logic* (MPL) and *Monadic Chain Logic* (MCL), respectively. Additionally, it has been proven that the counting extensions of CTL* and ECTL*, namely CCTL* and CECTL*, enjoy equivalent graded versions of the HTAs for the corresponding non-counting logics. However, while Moller and Rabinovich have proved CCTL* to be equivalent to full MPL, thus filling the gap for the standard branching-time logic, no similar result has been given for CECTL*. This work completes the picture, by proving the expressive equivalence of CECTL* and full MCL, by means of a *composition theorem* for the latter logic. This also indirectly establishes the equivalence between HTAs and their first-order extensions HFTAs, as originally introduced by Walukiewicz.

2012 ACM Subject Classification Theory of computation \rightarrow Automata over infinite objects; Theory of computation \rightarrow Modal and temporal logics; Theory of computation \rightarrow Tree languages

Keywords and phrases Branching-Time Temporal Logics, Monadic Chain Logic, Tree Automata

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.18

Funding Indam GNCS 2024 project “Certificazione, Monitoraggio, ed Interpretabilità in Sistemi di Intelligenza Artificiale”. M. Benerecetti, F. Mogavero, and A. Peron are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

1 Introduction

In the domain of *formal verification* of complex systems, temporal logics [33] play the crucial role of *specification languages* [34] for the correct behaviour of system components over time. These languages are generally divided into two main categories: *linear-time logics* and *branching-time logics*. Logics in the first category focus on properties that span the entirety of each possible behaviour in isolation, while those in the second one are designed to address the interactions among those behaviours. Prominent examples of linear-time logics are *Linear-Time Temporal Logic* (LTL) [40, 41] and its full ω -regular extension ELTL [52]. On the other hand, typical representatives of branching-time logics fall within the families



© Massimo Benerecetti, Laura Bozzelli, Fabio Mogavero, and Adriano Peron; licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 18; pp. 18:1–18:18

Leibniz International Proceedings in Informatics



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

of *Dynamic Logics* [17] and *Computation Tree Logics* [12, 11, 13, 14, 50]. Notable examples of such logics include CTL, CTL*, and ECTL*, together with the corresponding counting versions CCTL, CCTL*, and CECTL*.

The semantics of these temporal logics is often defined using suitable variants of *predicate logic*, usually *First-Order Logic* (FO) or *Second-Order Logic* (SO), interpreted either over *linearly-ordered structures*, such as infinite words, or over *partially-ordered structures*, such as infinite trees. At the same time, the extensive literature on automata-theoretic techniques [48] has been instrumental in providing effective technical tools for solving related decision problems. Predicate logics and automata theory also offer a comprehensive and coherent framework to evaluate and compare the expressive power and the computational properties of temporal languages, as evidenced by numerous *characterisation theorems*.

The foundational result in this context is Kamp's theorem [28], which establishes the equivalence between LTL and FO over infinite words. This result links FO-*definability* with *recognisability* by *counter-free finite-state automata* [30, 44, 45, 39], via the notions of *star-free language*, *aperiodic language*, and *aperiodic syntactic monoid*. Altogether these results fully characterise the expressive power of LTL in terms of predicate logics and automata. A similar correspondence also exists between ELTL, *Monadic Second-Order Logic* (MSO), and regular automata on infinite words [8, 9, 35, 10].

The landscape for branching-time temporal logics is much more complex due to the non-linear structure of the models and additional factors like *bisimulation invariance* [49] and *counting quantifiers* [16], and until recently it was far from being as complete as the linear-time counterpart. The more complete results were, indeed, the full correspondences among (1) the μ -CALCULUS [29], the *bisimulation-invariant* fragment of MSO interpreted over trees, and *Symmetric Alternating Parity Tree Automata* [27] and (2) the *alternation-free* fragment of μ -CALCULUS (AF μ -CALCULUS), the bisimulation-invariant fragments of WMSO over bounded-branching trees, and *Symmetric Alternating Weak Tree Automata* [1, 26]. These equivalences extend to the general case when counting quantifiers are incorporated into the modal logics [26, 25]. For four decades, the scenarios for CTL* and ECTL* remained significantly more fragmented. In the eighties, it was proved that, on binary trees, CTL* is equivalent to *Monadic Path Logic* (MPL) [24] and ECTL* to *Monadic Chain Logic* (MCL) [47]. The single result concerning CTL* was later extended to arbitrary-branching trees, at the turn of the century, addressing both bisimulation-invariance [37] and counting quantifiers [38]. Only very recently have corresponding classes of automata been proposed for these logics. Specifically, in [3], it was shown that, on arbitrary-branching trees, CTL* and ECTL* are equivalent to two versions of *Symmetric Hesitant Tree Automata*, namely HTA_{cf} and HTA, with and without a suitable *counter-freeness restriction* on their linear behaviours. Additionally, it was proved that HTA are equivalent to the bisimulation-invariant fragment of MCL. Thus, we finally have complete correspondences among (a) CTL*, the *bisimulation-invariant* fragment of MPL, and HTA_{cf}, and (b) ECTL*, the *bisimulation-invariant* fragment of MCL, and HTA. The first result was further extended to show, on arbitrary-branching trees, the equivalence of (c) CCTL*, MPL, and a graded version of HTA_{cf}, called HGTA_{cf}. However, the same result has not been obtained for CECTL*. This logic has been proven equivalent to a graded version of HTA, called HGTA, while MCL has only been shown equivalent to a potentially more-general first-order extension of HTA, called HFTA, inspired to a class of automata proposed by Walukiewicz [51].

The objective of this work is to complete the picture regarding the standard branching-time temporal logics by showing the equivalence of CECTL* with MCL. The key idea behind this result is to establish a composition theorem for MCL. *Composition Theorems*

serve as model-theoretic tools that simplify reasoning about complex structures by breaking down a statement about the whole into several statements about its individual components. A first example of this approach is the renowned Feferman-Vaught Theorem [15], which reduces the first-order theory of any product of structures to the first-order theory of its factors. An initial application to linear orders was proposed by Läuchli [31], as an alternative to the automata-theoretic technique on words by Büchi [7, 8, 9], and subsequently advanced in a series of works by Shelah and Gurevich [43, 19, 21, 22, 20]. Thomas then applied the approach to binary-branching tree structures [46, 47], culminating in the composition theorem for MPL in collaboration with Hafer [24]. This result was later extended to arbitrary-branching trees by Moller and Rabinovich [37, 38]. In the present work, we merge and generalise the techniques considered in [47, 38] to obtain the corresponding result for MCL, by relying on Ehrenfeucht-Fraïssé games tailored to this logic. Specifically, we show that verifying an MCL formula with quantifier rank m and a unique free chain variable over a tree boils down to verifying an MSO sentence over a word that is the encoding of a suitable vector of m chains induced by the interpretation of that variable. This allows us to translate, via structural induction, every MCL formula with a single first-order variable into an equivalent CECTL* state formula. Given that the translation from CECTL* to MCL is relatively straightforward, we obtain the stated result, settling one of the problems left open in [3]. It is important to note that the automata-theoretic technique developed in [3] cannot be directly applied here. In principle, given the equivalences of MCL with HFTA and CECTL* with HGTA, one might be tempted to show the equivalence of the two logics by proving the equivalence of the two automaton classes. However, the natural compositional transformation of the first-order formulae encoded in the transition function of an HFTA to the corresponding graded modal formulae does not satisfy the hesitant constraint required by a HGTA. The approach proposed in this work circumvents that difficulty and allows us to prove, though indirectly, that HFTA and HGTA are two equivalent types of automata.

2 Preliminaries

Let \mathbb{N} be the set of natural numbers. For $i, j \in \mathbb{N}$ with $i \leq j$, $[i, j]$ denotes the set of natural numbers k such that $i \leq k \leq j$. For a finite or infinite word ρ over some alphabet, $|\rho|$ is the length of ρ ($|\rho| = \omega$ if ρ is infinite) and for all $0 \leq i < |\rho|$, $\rho(i)$ is the $(i + 1)$ -th letter of ρ .

Kripke Trees. A tree T is a non-empty subset of \mathbb{N}^* which is prefix closed (i.e., for each $w \cdot n \in T$ with $n \in \mathbb{N}$, $w \in T$). Elements of T are called nodes and the empty word ε is the root of T . For $w \in T$, a *child* of w in T is a node in T of the form $w \cdot n$ for some $n \in \mathbb{N}$, and a *descendant* of w in T is a node of T of the form $w \cdot w'$ for some $w' \in \mathbb{N}^*$. For $w \in T$, the *subtree of T rooted at node w* is the tree consisting of the nodes of the form w' such that $w \cdot w' \in T$. A *subtree of T* is a tree T' such that for some $w \in T$, T' is a subset of the subtree of T rooted at w . A *path* of T is a subtree π of T which is totally ordered by the child-relation (i.e., each node of π has at most one child in π). In the following, a path π of T is also seen as a word over \mathbb{N} in accordance to the total ordering in π induced by the child relation. A *chain* of T is a subset of a path of T , while a *branch* of T is a path of T starting at the root. A tree is *non-blocking* if each node has some child. A non-blocking tree T is infinite, and maximal paths in T are infinite as well.

For an alphabet Σ , a Σ -labelled tree is a pair $\mathcal{S} = (T, Lab)$ consisting of a tree and a labelling $Lab : T \mapsto \Sigma$ assigning to each node in T a symbol in Σ . For a subtree T' of T , we denote by $\mathcal{S}_{T'}$ the Σ -labelled subtree $(T', Lab|_{T'})$ of \mathcal{S} . In this paper, we consider formalisms

whose specifications are interpreted over labeled trees. For the easy of presentation, we focus on labeled trees which are non-blocking. All the results of this paper can be easily adapted to the general case, where the non-blocking assumption is relaxed. For a finite set AP of atomic propositions, a *Kripke tree* over AP is a non-blocking 2^{AP} -labelled tree.

Automata over Infinite and Finite Words. We first recall the class of parity nondeterministic automata on infinite words (parity NWA for short) which are tuples $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$, where Σ is a finite input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \mapsto 2^Q$ is the transition function, $q_I \in Q$ is an initial state, and $\Omega : Q \mapsto \mathbb{N}$ is a parity acceptance condition over Q assigning to each state a natural number (color). The NWA \mathcal{A} is *deterministic* if for all states q and input symbols a , $\delta(q, a)$ is a singleton $\{q'\}$ (in this case, we write $\delta(q, a) = q'$). We use the acronym DWA for the subclass of deterministic NWA.

Given a word ρ over Σ , a *path* of \mathcal{A} over ρ is a word π over Q of length $|\rho| + 1$ ($|\rho| + 1$ is ω if ρ is infinite) such that $\pi(i+1) \in \delta(\pi(i), \rho(i))$ for all $0 \leq i < |\rho|$. A *run* over ρ is a path over ρ starting at the initial state. The NWA \mathcal{A} is *counter-free* if for all $n > 0$, states $q \in Q$ and finite words ρ over Σ , the following holds: if there is a path from q to q over ρ^n , then there is also a path from q to q over ρ .

A run π of \mathcal{A} over an infinite word ρ is *accepting* if the highest color of the states appearing infinitely often along π is even. The ω -language $L(\mathcal{A})$ accepted by \mathcal{A} is the set of infinite words ρ over Σ such that there is an accepting run π of \mathcal{A} over ρ .

A parity acceptance condition $\Omega : Q \mapsto \mathbb{N}$ is a *Büchi* condition if $\Omega(Q) \subseteq \{1, 2\}$. A *Büchi* NWA is a parity NWA whose acceptance condition is Büchi.

We also consider NWA over finite words (NWA_f for short) which are defined as parity NWA but the parity condition Ω is replaced with a set $F \subseteq Q$ of accepting states. A run π over a finite word is *accepting* if its last state is accepting.

Monadic Chain Logic. We recall now Monadic Chain Logic (MCL for short) [47] interpreted over arbitrary Kripke trees. MCL is the well-known fragment of MSO where second-order quantification is restricted to chains of the given Kripke tree. For technical convenience, we consider a one-sorted variant of MCL where first-order variables are encoded as second-order variables which are singletons. It is straightforward to show that this variant is equivalent to standard MCL.

Formally, given a finite set AP of atomic propositions and a finite set Vr_2 of second-order variables (or *chain* variables), the syntax of the considered variant of MCL is the set of formulae built according to the following grammar:

$$\varphi := \mathbf{sing}(X) \mid X \subseteq p \mid X \subseteq Y \mid X \leq Y \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists^c X. \varphi$$

where $p \in \text{AP}$ and $X, Y \in \text{Vr}_2$. Intuitively, $\mathbf{sing}(X)$ asserts that X is a singleton, $X \subseteq p$ means that p holds at each node of X , and $X \leq Y$ means that each node of Y is a descendant of each node of X . As usual, a *free variable* of a formula φ is a variable occurring in φ that is not bound by a quantifier. A *sentence* is a formula with no free variables. The language of MCL consists of its sentences.

Semantics of MCL. Formulae of MCL are interpreted over Kripke trees on AP. Given a Kripke tree $\mathcal{S} = (T, \text{Lab})$ over AP, a *second-order valuation for \mathcal{S}* is a mapping $\mathbf{V}_2 : \text{Vr}_2 \mapsto 2^T$ assigning to each second-order variable a chain of T. For an MCL formula φ , the satisfaction relation $(\mathcal{S}, \mathbf{V}_2) \models \varphi$, meaning that \mathcal{S} satisfies the formula φ under the valuation \mathbf{V}_2 , is defined as follows (the treatment of Boolean connectives is standard):

$$\begin{aligned}
(\mathcal{S}, V_2) \models \text{sing}(X) &\Leftrightarrow V_2(X) \text{ is a singleton;} \\
(\mathcal{S}, V_2) \models X \subseteq p &\Leftrightarrow p \in \text{Lab}(w) \text{ for each } w \in V_2(X); \\
(\mathcal{S}, V_2) \models X \subseteq Y &\Leftrightarrow V_2(X) \subseteq V_2(Y); \\
(\mathcal{S}, V_2) \models X \leq Y &\Leftrightarrow \text{for all } w \in V_2(X) \text{ and } w' \in V_2(Y), w' \text{ is a descendant of } w \text{ in } T; \\
(\mathcal{S}, V_2) \models \exists^c X. \varphi &\Leftrightarrow (\mathcal{S}, V_2[X \mapsto C]) \models \varphi \text{ for some chain } C \text{ of } T.
\end{aligned}$$

where $V_2[X \mapsto C]$ denotes the second-order valuation for \mathcal{S} defined as: $V_2[X \mapsto C](X) = C$ and $V_2[X \mapsto C](Y) = V_2(Y)$ if $Y \neq X$. Note that the satisfaction relation $(\mathcal{S}, V_2) \models \varphi$, for fixed \mathcal{S} and φ , depends only on the values assigned by V_2 to the variables occurring free in φ . In particular, if φ is a sentence, we say that \mathcal{S} *satisfies* φ , written $\mathcal{S} \models \varphi$, if $(\mathcal{S}, V_2) \models \varphi$ for some valuation V_2 . In this case, we also say that \mathcal{S} is a model of φ .

3 Branching-Time Temporal Logics

In this section, we recall syntax and semantics of *Counting-CTL** (CCTL* for short) [38], which extends standard CTL* [14] with counting operators, as well as the counting extension CECTL* [3] of ECTL* [50], a branching-time temporal logic more expressive than CCTL*. For technical convenience, we shall consider an equivalent syntactic variant of ECTL*, which employs NWA_f over finite words, instead of right-linear grammars, as the building blocks of formulae.¹ We also consider a fragment of CECTL*, that we call *counter-free* CECTL*, where all the NWA_f over finite words are required to be counter-free. We prove that counter-free CECTL* and CCTL* have the same expressive power.

3.1 The Logic CCTL*

The syntax of CCTL* is given by specifying inductively the set of *state formulae* φ and the set of *path formulae* ψ over a given finite set AP of atomic propositions:

$$\begin{aligned}
\varphi &::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid E\psi \mid D^n\varphi \\
\psi &::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid X\psi \mid \psi U\psi
\end{aligned}$$

where $p \in \text{AP}$, X and U are the standard “next” and “until” temporal modalities, E is the existential path quantifier, and D^n , with $n \in \mathbb{N} \setminus \{0\}$, is the counting operator. The language of CCTL* consists of the state formulae of CCTL*. Standard CTL* is the fragment of CCTL* where counting operators D^n with $n > 1$ are not allowed.

Given a Kripke tree $\mathcal{S} = (T, \text{Lab})$ (over AP), a node w of T , an infinite path π of T , and $0 \leq i < |\pi|$, the satisfaction relations $(\mathcal{S}, w) \models \varphi$ for a state formula φ (meaning that φ holds at node w of \mathcal{S}), and $(\mathcal{S}, \pi, i) \models \psi$ for a path formula ψ (meaning that ψ holds at position i of the path π in \mathcal{S}) are defined as follows (Boolean connectives are treated as usual):

$$\begin{aligned}
(\mathcal{S}, w) \models p &\Leftrightarrow p \in \text{Lab}(w); \\
(\mathcal{S}, w) \models E\psi &\Leftrightarrow (\mathcal{S}, \pi, 0) \models \psi \text{ for some infinite path } \pi \text{ of } T \text{ starting at node } w; \\
(\mathcal{S}, w) \models D^n\varphi &\Leftrightarrow \text{there are at least } n \text{ distinct children } w' \text{ of } w \text{ in } T \text{ s.t. } (\mathcal{S}, w') \models \varphi; \\
(\mathcal{S}, \pi, i) \models \varphi &\Leftrightarrow (\mathcal{S}, \pi(i)) \models \varphi; \\
(\mathcal{S}, \pi, i) \models X\psi &\Leftrightarrow (\mathcal{S}, \pi, i+1) \models \psi; \\
(\mathcal{S}, \pi, i) \models \psi_1 U\psi_2 &\Leftrightarrow \text{for some } j \geq i: (\mathcal{S}, \pi, j) \models \psi_2 \text{ and } (\mathcal{S}, \pi, k) \models \psi_1 \text{ for all } i \leq k < j.
\end{aligned}$$

Note that $D^1\varphi$ corresponds to $EX\varphi$. A Kripke tree \mathcal{S} satisfies (or is a model of) a state formula φ , written $\mathcal{S} \models \varphi$, if $\mathcal{S}, \varepsilon \models \varphi$.

¹ In [3], the considered syntactic variant of CECTL* is called Counting Computation Dynamic logic (CCDL) since it essentially corresponds to a branching-time extension of *Linear Dynamic Logic* [18].

3.2 The Logic CECTL*

Like CCTL*, the syntax of CECTL* is composed of *state formulae* φ and *path formulae* ψ over a given finite set AP of atomic propositions, defined as follows:

$$\begin{aligned}\varphi &::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid E\psi \mid D^n\varphi \\ \psi &::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \langle \mathcal{A} \rangle \psi\end{aligned}$$

where $p \in \text{AP}$ and $\langle \mathcal{A} \rangle$ is the *existential sequencing* modality applied to a *testing* $\text{NWA}_f \mathcal{A}$. We define a *testing* $\text{NWA}_f \mathcal{A} = \langle 2^{\text{AP}}, \text{Q}, \delta, q_I, \text{F}, \tau \rangle$ as consisting of an $\text{NWA}_f \langle 2^{\text{AP}}, \text{Q}, \delta, q_I, \text{F} \rangle$ over finite words over 2^{AP} and a test function τ mapping states in Q to CECTL* path formulae. Intuitively, along an infinite path π of a Kripke tree, the testing automaton accepts the labeling of a (possibly empty) infix $\pi(i) \dots \pi(j-1)$ of π if the embedded NWA_f has an accepting run $q_i \dots q_j$ over the labeling of such an infix so that, for each position $k \in [i, j]$, the formula holds at position k along π . A test function τ is *trivial* if it maps each state to \top . We also use the shorthand $[\mathcal{A}]\psi \triangleq \neg \langle \mathcal{A} \rangle \neg\psi$ (*universal sequencing* modality). The language of CECTL* consists of the state formulae of CECTL*. A CECTL* formula φ is *counter-free* if (i) the testing automata \mathcal{A} occurring in φ are counter-free and (ii) *either* \mathcal{A} is deterministic *or* the test function of \mathcal{A} is trivial.

Given a Kripke tree $\mathcal{S} = (\text{T}, \text{Lab})$, an infinite path π of T, and $0 \leq i < |\pi|$, the semantics of modality $\langle \mathcal{A} \rangle$ is defined as follows, where $\mathcal{A} = \langle 2^{\text{AP}}, \text{Q}, \delta, q_I, \text{F}, \tau \rangle$:

$$(\mathcal{S}, \pi, i) \models \langle \mathcal{A} \rangle \psi \iff \text{for some } j \geq i, (i, j) \in \text{R}_{\mathcal{A}}(\mathcal{S}, \pi) \text{ and } (\mathcal{S}, \pi, j) \models \psi$$

where $\text{R}_{\mathcal{A}}(\mathcal{S}, \pi)$ is the set of pairs (i, j) with $j \geq i$ s.t. there is an accepting run $q_i \dots q_j$ of the NWA_f embedded in \mathcal{A} over $\text{Lab}(\pi(i)) \dots \text{Lab}(\pi(j-1))$ and, for all $k \in [i, j]$, it holds that $(\mathcal{S}, \pi, k) \models \tau(q_k)$. The notion of a model of a CECTL* formula is defined as for CCTL*.

3.3 Expressiveness equivalence of CCTL* and counter-free CECTL*

We first show that CCTL* can be embedded into counter-free CECTL*. Let \mathcal{A} be the testing counter-free NWA_f having trivial tests and accepting all and only the words of length 1. Moreover, for a counter-free CECTL* path formula ψ_1 , let \mathcal{A}_{ψ_1} be the testing counter-free $\text{DWA}_f \mathcal{A} = \langle 2^{\text{AP}}, \{q_1\}, \delta, q_1, \{q_1\}, \tau \rangle$ defined as follows: $\delta(q_1, a) = q_1$ for each input symbol a , and $\tau(q_1) = \psi_1$. Then, the next and until formulae $X\psi_1$ and $\psi_1 \text{U} \psi_2$ can be expressed as: $X\psi_1 \equiv \langle \mathcal{A} \rangle \psi_1$ and $\psi_1 \text{U} \psi_2 \equiv \psi_2 \vee \langle \mathcal{A}_{\psi_1} \rangle \langle \mathcal{A} \rangle \psi_2$. Hence, we obtain the following result.

► **Proposition 3.1.** *Given a CCTL* formula, one can build an equivalent counter-free CECTL* formula.*

For the converse translation from counter-free CECTL* to CCTL*, by the known equivalence between *Monadic Path Logic* (MPL) and CCTL* [38], it suffices to show that each counter-free CECTL* formula can be translated into an equivalent MPL sentence. We first recall the logic MPL [23], the well-known fragment of MSO where second-order quantification is restricted to paths of the given Kripke tree.

Monadic Path Logic (MPL) [23]. Given a finite set AP of atomic propositions, a finite set Vr_1 of first-order variables, and a finite set Vr_2 of second-order variables, the syntax of standard MPL is the set of formulae built according to the following grammar:

$$\varphi := p(x) \mid x \leq y \mid x \in X \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x. \varphi \mid \exists^p X. \varphi$$

where $p \in \text{AP}$, $x, y \in \text{Vr}_1$, $X \in \text{Vr}_2$, and $\exists^P X$ is the path quantifier which ranges over paths of the given Kripke tree. We also exploit the standard logical connectives \vee and \rightarrow as abbreviations, the universal first-order quantifier $\forall x$, defined as $\forall x.\varphi \triangleq \neg\exists x.\neg\varphi$, and the universal path quantifier $\forall^P X$, defined as $\forall^P X.\varphi \triangleq \neg\exists^P X.\neg\varphi$. We also make use of the shorthands (i) $x = y$ for $x \leq y \wedge y \leq x$, (ii) $x < y$ for $x \leq y \wedge \neg(y \leq x)$; (iii) $\exists x \in X.\varphi$ for $\exists x.(x \in X \wedge \varphi)$, and (iv) $\forall x \in X.\varphi$ for $\forall x.(x \in X \rightarrow \varphi)$. Moreover, the child relation is definable in MPL by the binary predicate $\text{child}(x, y) \triangleq x < y \wedge \neg\exists z.(x < z \wedge z < y)$ which exploits only first-order quantification.

Given a Kripke tree $\mathcal{S} = (\text{T}, \text{Lab})$, a *first-order valuation for \mathcal{S}* is a mapping $\text{V}_1 : \text{Vr}_1 \mapsto \text{T}$ assigning to each first-order variable a node of T . A *path valuation for \mathcal{S}* is a second-order valuation $\text{V}_2 : \text{Vr}_2 \mapsto 2^{\text{T}}$ assigning to each second-order variable a path of T . Given an MPL formula φ , a first-order valuation V_1 for \mathcal{S} , and a path valuation V_2 for \mathcal{S} , the satisfaction relation $(\mathcal{S}, \text{V}_1, \text{V}_2) \models \varphi$ is defined as follows (the treatment of Boolean connectives is standard):

$$\begin{aligned} (\mathcal{S}, \text{V}_1, \text{V}_2) \models p(x) &\Leftrightarrow p \in \text{Lab}(\text{V}_1(x)); \\ (\mathcal{S}, \text{V}_1, \text{V}_2) \models x \leq y &\Leftrightarrow \text{V}_1(y) \text{ is a descendant of } \text{V}_1(x) \text{ in } \text{T}; \\ (\mathcal{S}, \text{V}_1, \text{V}_2) \models x \in X &\Leftrightarrow \text{V}_1(x) \in \text{V}_2(X); \\ (\mathcal{S}, \text{V}_1, \text{V}_2) \models \exists x.\varphi &\Leftrightarrow (\mathcal{S}, \text{V}_1[x \mapsto w], \text{V}_2) \models \varphi \text{ for some } w \in \text{T}; \\ (\mathcal{S}, \text{V}_1, \text{V}_2) \models \exists^P X.\varphi &\Leftrightarrow (\mathcal{S}, \text{V}_1, \text{V}_2[X \mapsto \pi]) \models \varphi \text{ for some path } \pi \text{ of } \text{T}. \end{aligned}$$

where $\text{V}_1[x \mapsto w]$ denotes the first-order valuation for \mathcal{T} defined as: $\text{V}_1[x \mapsto w](x) = w$ and $\text{V}_1[x \mapsto w](y) = \text{V}_1(y)$ if $y \neq x$.

From counter-free CECTL* to MPL. The translation presented in the following is based on known results about counter-free DWA [32, 6].

► **Proposition 3.2.** *Given a counter-free CECTL* formula, one can build an equivalent MPL sentence.*

Proof. Let ψ be a counter-free CECTL* path formula. We construct an MPL formula $\widehat{\psi}(x, X)$ with exactly one *free* first-order variable x and at most one *free* second-order variable X such that for each Kripke tree \mathcal{S} , infinite path π of \mathcal{S} , and position $i \geq 0$, it holds that

$$(\mathcal{S}, \pi, i) \models \psi \text{ if and only if } (\mathcal{S}, x \rightarrow \pi(i), X \rightarrow \pi) \models \widehat{\psi}(x, X)$$

Moreover, $\widehat{\psi}(x, X)$ does not depend on X if ψ is a state formula. Thus, given a state CECTL* formula φ , the MPL sentence equivalent to φ is given by $\exists x.(\text{root}(x) \wedge \widehat{\varphi}(x))$ with $\text{root}(x) \triangleq \neg\exists y.y < x$.

The MPL formula $\widehat{\psi}(x, X)$ is defined by structural induction on ψ as follows, where we exploit the predicate $\text{Inf}(Y)$ expressing that the path Y is infinite ($\text{Inf}(Y)$ can be easily specified in MPL by using only first-order quantification).

- $\psi = p$ with $p \in \text{AP}$: $\widehat{\psi}(x, X) \triangleq p(x)$.
- $\psi = \neg\psi_1$: $\widehat{\psi}(x, X) \triangleq \neg\widehat{\psi}_1(x, X)$.
- $\psi = \psi_1 \wedge \psi_2$: $\widehat{\psi}(x, X) \triangleq \widehat{\psi}_1(x, X) \wedge \widehat{\psi}_2(x, X)$.
- $\psi = \mathbf{E}\psi_1$: $\widehat{\psi}(x, X) \triangleq \exists^P Y. (\text{Inf}(Y) \wedge x \in Y \wedge \widehat{\psi}_1(x, Y) \wedge \forall y \in Y. x \leq y)$.
- $\psi = \mathbf{D}^n\psi_1$: $\widehat{\psi}(x, X) \triangleq \exists x_1 \dots \exists x_n. (\bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_{i=1}^n (\text{child}(x, x_i) \wedge \widehat{\psi}_1(x_i, X)))$.
- $\psi = \langle \mathcal{A} \rangle \psi_1$, where \mathcal{A} is a counter-free testing NWA_f such that either \mathcal{A} is deterministic or the test function of \mathcal{A} is trivial. Since a counter-free NWA_f on finite words can be converted into an equivalent counter-free DWA_f [36], we can assume that the NWA_f with

tests \mathcal{A} is deterministic. Let $\mathcal{A} = \langle 2^{\text{AP}}, \mathcal{Q}, \delta, q_I, F, \tau \rangle$. By [32, 6], for each state $q \in \mathcal{Q}$, we can construct an FO formula $\xi_q(x, y)$ with two free variables x and y such that for each infinite word ρ over 2^{AP} and positions $i, j \geq 0$, it holds that $(\rho, x \rightarrow i, y \rightarrow j) \models \xi_q(x, y)$ iff $i \leq j$ and the unique run of \mathcal{A} over $\rho[i, j - 1]$ leads to state q . Let $\widehat{\xi}_q(x, y, X)$ be the MPL formula obtained by “relativizing” the FO formula $\xi_q(x, y)$ with respect to path X , i.e., by replacing each subformula $\exists x. \theta$ of $\xi_q(x, y)$ with $\exists x. (x \in X \wedge \theta)$. Then, $\widehat{\psi}(x, X)$ is defined as follows:

$$\bigvee_{q \in F} \exists y \in X. \left(x \leq y \wedge \widehat{\xi}_q(x, y, X) \wedge \widehat{\tau}(q)(y, X) \wedge \forall z \in X. [x \leq z < y \rightarrow \bigvee_{q \in \mathcal{Q}} (\widehat{\xi}_q(x, z, X) \wedge \widehat{\tau}(q)(z, X))] \wedge \widehat{\psi}_1(y, X) \right) \quad \blacktriangleleft$$

It remains an open question whether counter-free NWA_f with non-trivial tests can be captured in MPL. Thus, by Propositions 3.1–3.2 and the known equivalence of MPL and CCTL^* [38], we obtain the following result.

► **Theorem 3.3.** *CCTL^* and counter-free CECTL^* are equivalent formalisms, i.e., they specify the same class of tree languages.*

4 Expressiveness equivalence of MCL and CECTL^*

It is known [3] that each CECTL^* state formula has an equivalent MCL sentence. In this section, we show that the two logics CCDL and MCL are in fact expressively equivalent. We provide a proof of this result which relies on an adaptation of the compositional argument given in [38] for showing that each MPL sentence has an equivalent CCTL^* state formula.

4.1 Model-theoretic fundamentals

We first introduce some notations and definitions. The *quantifier rank* $\text{qr}(\varphi)$ of an MCL formula φ is the maximum number of nested quantifiers occurring in it. In the following, a Kripke tree (over AP) is called *structure* (over AP).

Fix a finite set AP of atomic propositions. Given $h \in \mathbb{N}$, an *h-structure* \mathcal{S}_h is a tuple of the form $\mathcal{S}_h = (\mathcal{S}, C_1, \dots, C_h)$ such that \mathcal{S} is a structure and C_1, \dots, C_h are chains of \mathcal{S} . An *h-word structure* is defined similarly but we require that the structure \mathcal{S} is an infinite word over 2^{AP} (recall that an infinite word over 2^{AP} corresponds to a structure where each node has exactly one child). Note that a structure can be seen as a 0-structure.

An *h-MCL formula* is an MCL formula having at most h free variables (recall that in the one-sorted formalization of MCL, all the variables range over chains). Note that a 0-MCL formula is a sentence. An *h-structure* $\mathcal{S}_h = (\mathcal{S}, C_1, \dots, C_h)$ satisfies an *h-MCL formula* $\varphi(X_1, \dots, X_h)$ if $\mathcal{S} \models \varphi(C_1, \dots, C_h)$ (which means that $(\mathcal{S}, \mathbb{V}_2) \models \varphi(X_1, \dots, X_h)$ for any valuation \mathbb{V}_2 such that $\mathbb{V}_2(X_i) = C_i$ for each $i \in [1, h]$). Two *h-MCL formulas* are *equivalent* if they are satisfied by the same *h-structures*. Two *h-MCL formulas* are *word-equivalent* if they are satisfied by the same *h-word structures*.

Equivalence relations between *h-structures*. Let $m \in \mathbb{N}$. Given two *h-structures*, $\mathcal{S}_h = (\mathcal{S}, C_1, \dots, C_h)$ and $\mathcal{S}'_h = (\mathcal{S}', C'_1, \dots, C'_h)$, we say that \mathcal{S}_h and \mathcal{S}'_h are *m-rank equivalent*, written $\mathcal{S}_h \equiv_m \mathcal{S}'_h$, if no *h-MCL formula* $\varphi(X_1, \dots, X_k)$ of quantifier rank at most m can distinguish them, i.e., $\mathcal{S} \models \varphi(C_1, \dots, C_k)$ iff $\mathcal{S}' \models \varphi(C'_1, \dots, C'_k)$. If $\mathcal{S}_h \equiv_m \mathcal{S}'_h$ and \mathcal{S}_h and \mathcal{S}'_h are *h-word structures*, we write $\mathcal{S}_h \equiv_m^\omega \mathcal{S}'_h$. The equivalence relation \equiv_m over the class of

h -structures has finite index and each equivalence class can be characterized by an h -MCL formula of quantifier rank at most m , called m -type for h -MCL formulas. In particular, the following result follows from standard arguments.

► **Proposition 4.1.** *Let $h \in \mathbb{N}$. Then, the following properties hold for each $m \geq 0$:*

1. *the equivalence \equiv_m over the set of h -structures defines finitely-many equivalence classes;*
2. *for each equivalence class Λ_m of \equiv_m over h -structures, there is an h -MCL formula β (called m -type for h -MCL formulas) with $\text{qr}(\beta) \leq m$ which characterizes it: that is, $\mathcal{S}_h \models \beta$ iff $\mathcal{S}_h \in \Lambda_m$, for all h -structures \mathcal{S}_h ;*
3. *each h -MCL formula φ with $\text{qr}(\varphi) \leq m$ is equivalent to a disjunction of m -types;*
4. *the variants of Properties 1–3 for the class of h -word structures.*

Proof. We focus on Properties 1–3. We observe that by variable renaming, we can assume that h -MCL formulas φ with $\text{qr}(\varphi) \leq m$ only use variables from a *finite* set. Hence, by a straightforward induction on $\text{qr}(\varphi)$, the following holds.

▷ **Claim.** There is a *finite* set Υ of h -MCL formulas with quantifier rank at most m such that each h -MCL formula φ with $\text{qr}(\varphi) \leq m$ is equivalent to some formula in Υ .

Let $\Upsilon = \{\psi_1, \dots, \psi_N\}$ be the finite set of h -MCL formulas with quantifier rank at most m satisfying the previous claim. We consider the h -MCL formulas of the form

$$\bar{\psi}_1 \wedge \dots \wedge \bar{\psi}_N$$

where $\bar{\psi}_i$ is either ψ_i or $\neg\psi_i$ for all $i \in [1, N]$. Let us denote by $\beta_1, \dots, \beta_\ell$ these formulas (note that $\ell = 2^N$). By construction, for each h -structure \mathcal{S}_h , there is exactly one $i \in [1, \ell]$ such that $\mathcal{S}_h \models \beta_i$. Moreover, each formula $\psi_i \in \Upsilon$ can be expressed as the disjunction of all and only the formulas in $\{\beta_1, \dots, \beta_\ell\}$ whose associated conjunct $\bar{\psi}_i$ is ψ_i . Thus, by the previous claim, Properties 1–3 easily follow. ◀

Local isomorphism on h -structures. The equivalence relation \equiv_0 over h -structures can be characterized as follows. Given two h -structures, $\mathcal{S}_h = (\mathcal{S}, C_1, \dots, C_h)$ and $\mathcal{S}'_h = (\mathcal{S}', C'_1, \dots, C'_h)$, we say that \mathcal{S}_h and \mathcal{S}'_h are *locally-isomorphic* (for MCL) if the following conditions hold, where $\mathcal{S} = \langle \mathbb{T}, \text{Lab} \rangle$ and $\mathcal{S}' = \langle \mathbb{T}', \text{Lab}' \rangle$:

- for all $i \in [1, h]$, C_i is a singleton iff C'_i is a singleton;
- for all $i \in [1, h]$ and $p \in \text{AP}$, $C_i \subseteq T_p$ iff $C'_i \subseteq T'_p$, where $T_p = \{w \in \mathbb{T} \mid p \in \text{Lab}(w)\}$ and $T'_p = \{w \in \mathbb{T}' \mid p \in \text{Lab}'(w)\}$;
- for all $i, j \in [1, h]$, $C_i \subseteq C_j$ iff $C'_i \subseteq C'_j$;
- for all $i, j \in [1, h]$, $C_i \leq C_j$ iff $C'_i \leq C'_j$.

Note that two structures are always locally-isomorphic and two h -structures are 0-rank equivalent iff they are locally-isomorphic.

Ehrenfeucht-Fraïssé Games for MCL. The rank-equivalence relation \equiv_m over the class of h -structures has an elegant characterization in terms of *Ehrenfeucht-Fraïssé games* (EF-games) over h -structures. The EF-game $\mathcal{G}_m(\mathcal{S}_h, \mathcal{S}'_h)$ over two h -structures $\mathcal{S}_h = (\mathcal{S}, C_1, \dots, C_h)$ and $\mathcal{S}'_h = (\mathcal{S}', C'_1, \dots, C'_h)$ is played by two players called the *spoiler* and the *duplicator*. Each play consists of m -rounds. At i -th round, with $i \in [1, m]$, the spoiler chooses a chain in one of the two structures \mathcal{S} and \mathcal{S}' , after which the duplicator responds by choosing a chain in the other structure which she believes *matches* the chain chosen by the spoiler. After m -rounds, there will be m chains $\bar{C}_1, \dots, \bar{C}_m$ selected in the structure \mathcal{S} , and corresponding m chains

$\overline{C}'_1, \dots, \overline{C}'_i$ selected in the structure \mathcal{S}' . The duplicator wins if the two $(h+m)$ -structures $(\mathcal{S}, C_1, \dots, C_h, \overline{C}_1, \dots, \overline{C}_m)$ and $(\mathcal{S}', C'_1, \dots, C'_h, \overline{C}'_1, \dots, \overline{C}'_m)$ are locally-isomorphic (note that this entails that the original h -structures \mathcal{S}_h and \mathcal{S}'_h need to be locally-isomorphic). Otherwise, the spoiler wins. We say that the duplicator has a *winning strategy* in the game $\mathcal{G}_m(\mathcal{S}_h, \mathcal{S}'_h)$ if it is possible for him to win each play whatever choices are made by the opponent. The h -structures \mathcal{S}_h and \mathcal{S}'_h are *m -game equivalent*, written $\mathcal{S}_h \sim_m \mathcal{S}'_h$ if the duplicator has a winning strategy in the game $\mathcal{G}_m(\mathcal{S}_h, \mathcal{S}'_h)$. If $\mathcal{S}_h \sim_m \mathcal{S}'_h$ and \mathcal{S}_h and \mathcal{S}'_h are h -word structures, we write $\mathcal{S}_h \sim_m^\omega \mathcal{S}'_h$. By classical arguments, one can show that the m -game equivalence relation corresponds to the rank equivalence \equiv_m .

► **Proposition 4.2.** *For all $h, m \in \mathbb{N}$ and h -structures \mathcal{S}_h and \mathcal{S}'_h , $\mathcal{S}_h \equiv_m \mathcal{S}'_h$ iff $\mathcal{S}_h \sim_m \mathcal{S}'_h$.*

Proof. First, we observe that for each $m \geq 0$, \sim_m is the unique *equivalence* relation satisfying the following properties for all h -structures $\mathcal{S}_h = (\mathcal{S}, \dots)$ and $\mathcal{S}'_h = (\mathcal{S}', \dots)$:

1. if $m = 0$, then $\mathcal{S}_h \sim_0 \mathcal{S}'_h$ iff \mathcal{S}_h and \mathcal{S}'_h are locally isomorphic;
2. if $m > 0$, then:

(forth) for each chain C of \mathcal{S} , there is a chain C' of \mathcal{S}' such that $(\mathcal{S}_h, C) \sim_{m-1} (\mathcal{S}'_h, C')$, where (\mathcal{S}_h, C) and (\mathcal{S}'_h, C') denote the $(h+1)$ -structures defined in the obvious way;

(back) for each chain C' of \mathcal{S}' , there is a chain C of \mathcal{S} such that $(\mathcal{S}_h, C) \sim_{m-1} (\mathcal{S}'_h, C')$.

Thus, it suffices to show that the equivalence relation \equiv_m satisfies the previous conditions with \sim_m replaced with \equiv_m . If $m = 0$, the result trivially follows. Now, assume that $m > 0$. We focus on the forth condition. Let C be a chain of \mathcal{S} . According to Proposition 4.1, let β be the unique $(m-1)$ -type for $(h+1)$ -structures such that $(\mathcal{S}_h, C) \models \beta$. Hence, $\mathcal{S}_h \models \exists^c X_{h+1}. \beta$. By Proposition 4.1, β is an $(h+1)$ -MCL formula with $\text{qr}(\beta) \leq m-1$. Since $\mathcal{S}_h \equiv_m \mathcal{S}'_h$, it follows that $\mathcal{S}'_h \models \exists^c X_{h+1}. \beta$. Hence, there exists a chain $C' \in \mathcal{S}'$ such that $(\mathcal{S}'_h, C') \models \beta$. Thus, being β the $(m-1)$ -type for $(h+1)$ -structures, we obtain that $(\mathcal{S}_h, C) \equiv_{m-1} (\mathcal{S}'_h, C')$, and we are done. ◀

4.2 A Composition Theorem for MCL

We provide now a characterization of the game-equivalence relation \sim_m over 1-structures on AP, for a given $m \geq 1$, in terms of the game-equivalence relation \sim_{2m}^ω over word-structures defined over a suitable set of atomic propositions.

Fix $m \geq 1$. Referring to Proposition 4.1, let $\beta_1, \dots, \beta_\ell$ be the m -types of the equivalence relation \equiv_m over the class of structures. Recall that for each structure \mathcal{S} , there is exactly one m -type β_i for some $i \in [1, \ell]$ such that $\mathcal{S} \models \beta_i$ (we say that β_i is the m -type of \mathcal{S}). Given a structure \mathcal{S} , a node w of \mathcal{S} , a child w_C of w in \mathcal{S} , and $i \in [1, \ell]$, let $N_{\mathcal{S}}(w, w_C, i)$ be the (possibly infinite) cardinality of the set of children w' of w in \mathcal{S} such that $w' \neq w_C$ and the substructure (i.e., the labeled subtree) of \mathcal{S} rooted at w' has m -type β_i . We denote by $f_{\mathcal{S}}(w, w_C)$ the mapping in $[1, \ell] \rightarrow [0, m]$, where for each $i \in [1, \ell]$, $f_{\mathcal{S}}(w, w_C)(i)$ is defined as:

$$f_{\mathcal{S}}(w, w_C)(i) \triangleq \begin{cases} N_{\mathcal{S}}(w, w_C, i) & \text{if } N_{\mathcal{S}}(w, w_C, i) < m \\ m & \text{otherwise.} \end{cases}$$

Thus, $f_{\mathcal{S}}(w, w_C)(i)$ approximates $N_{\mathcal{S}}(w, w_C, i)$ with the greatest number in $[0, m]$ which is smaller or equal to $N_{\mathcal{S}}(w, w_C, i)$. We consider the finite set AP_m of propositions defined as:

$$\text{AP}_m \triangleq 2^{\text{AP} \cup \{c\}} \times ([1, \ell] \mapsto [0, m]) \times [1, \dots, \ell].$$

Let (\mathcal{S}, C) be a 1-structure with labeling Lab . For each infinite branch π of \mathcal{S} such that π contains the chain C (note that if C is infinite, then π is uniquely determined), we denote by $\omega(\mathcal{S}, \pi, C)$ the infinite word over 2^{AP_m} defined as follows for all positions $j \geq 0$:

$$\omega(\mathcal{S}, \pi, C)(j) = \{(\text{Lab}(\pi(j)) \cup b, f_{\mathcal{S}}(\pi(j), \pi(j+1)), k_j)\}$$

where (i) $\flat = \{c\}$ if $\pi(j) \in C$, and $\flat = \emptyset$ otherwise, and (ii) k_j is such that β_{k_j} is the m -type of the substructure of \mathcal{S} rooted at $\pi(j+1)$. Thus the label of the j th position of $\omega(\mathcal{S}, \pi, C)(j)$ is a singleton and corresponds to the label of the j th node of the infinite path π of \mathcal{S} extended with additional information concerning the m -type of the subtree rooted at node $\pi(j+1)$, the m -types of the subtrees rooted at the children of $\pi(j)$ which are not in π , and the indication whether node $\pi(j)$ belong to the chain C or not. Note that the infinite word $\omega(\mathcal{S}, \pi, C)$ can be seen as the word-structure (π, Lab') , where $Lab'(\pi(j)) = \omega(\mathcal{S}, \pi, C)(j)$ for each $j \geq 0$.

The importance of the word-structure $\omega(\mathcal{S}, \pi, C)$ is that it captures the whole of the 1-structure (\mathcal{S}, C) with respect to the distinguishing power of 1-MCL formulas with quantifier rank at most m . In particular, we establish the following crucial result.

► **Lemma 4.3** (From MCL-games on 1-structures to MCL-games on word-structures). *Let $m \geq 1$, (\mathcal{S}, C) and (\mathcal{S}', C') be two 1-structures, and π and π' be two infinite branches of \mathcal{S} and \mathcal{S}' , respectively, such that $C \subseteq \pi$ and $C' \subseteq \pi'$. Then:*

$$\omega(\mathcal{S}, \pi, C) \sim_{2m}^{\omega} \omega(\mathcal{S}', \pi', C') \Rightarrow (\mathcal{S}, C) \sim_m (\mathcal{S}', C').$$

Proof. We need some additional definitions and preliminary observations. Let $\mathcal{S}_1 = (\mathcal{S}, C)$ be a 1-structure, and π be an infinite branch of \mathcal{S} such that $C \subseteq \pi$. We observe that each chain of \mathcal{S} which is not contained in the branch π can be partitioned into two chains C_1 and C_2 of \mathcal{S} such that C_1 is a subset of the path π and there exists a child w' of some node w of π such that w' is not a π -node and C_2 is a non-empty chain of the subtree of \mathcal{S} rooted at node w' . This justifies the following definition. A π -term of \mathcal{S}_1 is either a chain of π , or a tuple of the form (C_1, w, T, C_2) such that the following holds:

- C_1 is a finite chain of π and w is a node of π which is a descendant of all nodes in C_1 ;
- there is a child w' of w in \mathcal{S} such that $w' \notin \pi$, T is the labeled subtree of \mathcal{S} rooted at node w' , and C_2 is a non-empty chain of T .

For a compound π -term $t = (C_1, w, T, C_2)$ of \mathcal{S}_1 , we write $T(t)$ for T , $C_1(t)$ for C_1 , $C_2(t)$ for C_2 , and $C_w(t)$ for w . For a simple π -term t consisting of a chain of π , $C_1(t)$ is for t , and $T(t)$, $C_2(t)$, and $C_w(t)$ denote the empty set. For each $h \geq 0$, a (π, h) -term of \mathcal{S}_1 is a tuple of the form (t_1, \dots, t_h) where t_1, \dots, t_h are π -terms of \mathcal{S}_1 . We make the following observation:

Disjointness property. For π -terms (C_1, w, T, C_2) and (C'_1, w', T', C'_2) of \mathcal{S}_1 , either $T = T'$ or $T \cap T' = \emptyset$. Moreover, if $T \cap T' = \emptyset$, then $C_2 \cap C'_2 = \emptyset$ (in particular, C_2 and C'_2 are not related by the descendant relation).

Fix two 1-structures $\mathcal{S}_1 = (\mathcal{S}, C)$ and $\mathcal{S}'_1 = (\mathcal{S}', C')$, an infinite branch π of \mathcal{S}_1 with $C \subseteq \pi$, and an infinite branch π' of \mathcal{S}'_1 with $C' \subseteq \pi'$. Let $\mathcal{S}_\omega = \omega(\mathcal{S}, \pi, C)$ and $\mathcal{S}'_\omega = \omega(\mathcal{S}', \pi', C')$. Given $m \geq 1$, $h \in [0, m]$, a (π, h) -term $\text{tr} = (t_1, \dots, t_h)$ of \mathcal{S}_1 , and a (π', h) -term $\text{tr}' = (t'_1, \dots, t'_h)$ of \mathcal{S}'_1 , we say that tr and tr' are m -consistent if the following holds:

1. for all $i \in [1, h]$, $T(t_i) \neq \emptyset$ iff $T(t'_i) \neq \emptyset$. Moreover, if $T(t_i) \neq \emptyset$, then:
 - let t_{i_1}, \dots, t_{i_p} and $t'_{i'_1}, \dots, t'_{i'_p}$ be the ordered sequences of compound terms in tr and tr' , respectively, having tree-component $T(t_i)$ and $T(t'_i)$, respectively. Then, $p = p'$, and $i_j = i'_j$ for all $j \in [1, p]$. Moreover, $(\mathcal{S}_{T(t_i)}, C_2(t_{i_1}), \dots, C_2(t_{i_p})) \sim_{m-p} (\mathcal{S}'_{T(t'_i)}, C_2(t'_{i_1}), \dots, C_2(t'_{i_p}))$.
2. $(\mathcal{S}_\omega, C_1(t_1), C_w(t_1), \dots, C_1(t_h), C_w(t_h)) \sim_{2m-2h}^{\omega} (\mathcal{S}'_\omega, C_1(t'_1), C_w(t'_1), \dots, C_1(t'_h), C_w(t'_h))$.

By the disjointness property and since $m \geq 1$ (recall that the special proposition c of AP_m marks the nodes of the chains C and C' of $\omega(\mathcal{S}, \pi, C)$ and $\omega(\mathcal{S}', \pi', C')$, respectively), the following result easily follows.

▷ **Claim 4.4.** Let $\text{tr} = (t_1, \dots, t_h)$ be a (π, h) -term of \mathcal{S}_1 and $\text{tr}' = (t'_1, \dots, t'_h)$ be a (π', h) -term of \mathcal{S}'_1 . If tr and tr' are m -consistent, then the $(h+1)$ -structures $(\mathcal{S}, C, C_1(t_1) \cup C_2(t_1), \dots, C_1(t_h) \cup C_2(t_h))$ and $(\mathcal{S}', C', C_1(t'_1) \cup C_2(t'_1), \dots, C_1(t'_h) \cup C_2(t'_h))$ are locally isomorphic.

Assume that $\omega(\mathcal{S}, \pi, C) \sim_{2m}^{\omega} \omega(\mathcal{S}', \pi', C')$. We need to prove that $\mathcal{S}_1 \sim_m \mathcal{S}'_1$. By Claim 4.4 (for the case where $h = 0$), \mathcal{S}_1 and \mathcal{S}'_1 are locally isomorphic. Now, given $0 \leq h < m$, assume that after h -rounds in the EF-game $\mathcal{G}_m(\mathcal{S}_1, \mathcal{S}'_1)$, there are h π -terms t_1, \dots, t_h selected in the structure \mathcal{S} and corresponding h π' -terms t'_1, \dots, t'_h selected in the structure \mathcal{S}' such that $\text{tr} = (t_1, \dots, t_h)$ and $\text{tr}' = (t'_1, \dots, t'_h)$ are m -consistent with respect to \mathcal{S}_1 and \mathcal{S}'_1 . Moreover, assume that at the $(h+1)$ -round the spoiler chooses a π -term t_{h+1} in \mathcal{S} (the case where the choice is made on the structure \mathcal{S}' is similar). We show that the duplicator can respond by choosing a π' -term t'_{h+1} in \mathcal{S}' such that the tuples $(t_1, \dots, t_h, t_{h+1})$ and $(t'_1, \dots, t'_h, t'_{h+1})$ are still m -consistent with respect to \mathcal{S}_1 and \mathcal{S}'_1 . Hence, by Claim 4.4, the result follows. We focus on the case where t_{h+1} is a compound term π -term of the form (C_1, w, T, C_2) (the case where t_{h+1} is a chain of π is simpler). We distinguish two cases:

- *there is some term t_i in tr such that $T(t_i) = T$:* let t_{i_1}, \dots, t_{i_N} be the ordered sequence of compound terms in tr having T as tree-component. Note that $C_w(t_{i_j}) = w$ for all $j \in [1, N]$. Since tr and tr' are m -consistent with respect to \mathcal{S}_1 and \mathcal{S}'_1 , it holds that $T(t'_i) = T'_i \neq \emptyset$, $t'_{i_1}, \dots, t'_{i_N}$ is the ordered sequence of compound terms in tr' having T' as tree-component, and there is a node w' of π' (the parent of the T' -root) such that $C_w(t'_{i_j}) = w'$ for all $j \in [1, N]$. Moreover, $(\mathcal{S}_T, C_2(t_{i_1}), \dots, C_2(t_{i_N})) \sim_{m-N} (\mathcal{S}'_{T'}, C_2(t'_{i_1}), \dots, C_2(t'_{i_N}))$ and $(\mathcal{S}_\omega, C_1(t_1), C_w(t_1), \dots, C_1(t_h), C_w(t_h)) \sim_{2m-2h}^{\omega} (\mathcal{S}'_\omega, C_1(t'_1), C_w(t'_1), \dots, C_1(t'_h), C_w(t'_h))$. Thus, since $N < m$, in the EF-game over the substructures \mathcal{S}_T and $\mathcal{S}'_{T'}$, the duplicator can pick a chain C'_2 of T' such that $(\mathcal{S}_T, C_2(t_{i_1}), \dots, C_2(t_{i_N}), C_2) \sim_{m-(N+1)} (\mathcal{S}'_{T'}, C_2(t'_{i_1}), \dots, C_2(t'_{i_N}), C'_2)$. Moreover, since $h < m$, in the EF-game over the word structures \mathcal{S}_ω and \mathcal{S}'_ω , the duplicator can pick a chain C'_1 of π' such that $(\mathcal{S}_\omega, C_1(t_1), C_w(t_1), \dots, C_1(t_h), C_w(t_h), C_1, \{w\}) \sim_{2m-2(h+1)}^{\omega} (\mathcal{S}'_\omega, C_1(t'_1), C_w(t'_1), \dots, C_1(t'_h), C_w(t'_h), C'_1, \{w'\})$. Note that w' must be a descendant of all nodes in C'_1 . Hence, by setting t'_{h+1} to (C'_1, w', T', C'_2) , the result follows.
- *there is no term in tr having tree-component T :* let t_{i_1}, \dots, t_{i_N} be the (possibly empty) ordered sequence of compound terms in tr whose second component is w . Since $h < m$ and tr and tr' are m -consistent, in the EF-game over the word structures \mathcal{S}_ω and \mathcal{S}'_ω , the duplicator can pick a chain C'_1 of π' and a node $w' \in \pi'$ which is a descendant of all nodes in C'_1 such that $(\mathcal{S}_\omega, C_1(t_1), C_w(t_1), \dots, C_1(t_h), C_w(t_h), C_1, \{w\}) \sim_{2m-2(h+1)}^{\omega} (\mathcal{S}'_\omega, C_1(t'_1), C_w(t'_1), \dots, C_1(t'_h), C_w(t'_h), C'_1, \{w'\})$. Hence, $t'_{i_1}, \dots, t'_{i_N}$ is the (possibly empty) ordered sequence of compound terms in tr' whose second component is w' . Since the label of w in \mathcal{S}_ω and the label of w' in \mathcal{S}'_ω coincide and $N < m$, by construction of the word-structures \mathcal{S}_ω and \mathcal{S}'_ω , there must be a child w'' of w' in \mathcal{S}' such that $w'' \notin \pi'$ and for the subtree T' of \mathcal{S}' rooted at w'' , it holds that $\mathcal{S}_T \equiv_m \mathcal{S}'_{T'}$ and $T' \neq T(t'_{i_j})$ for all $j \in [1, N]$. Being $C_2 \subseteq T$ and $m \geq 1$, in the EF-game over the substructures \mathcal{S}_T and $\mathcal{S}'_{T'}$, the duplicator can pick a chain C'_2 of T' such that $(\mathcal{S}_T, C_2) \sim_{m-1} (\mathcal{S}'_{T'}, C_2)$. We set $t'_{h+1} = (C'_1, w', T', C'_2)$, and the result follows. ◀

We can now state a composition theorem for 1-MCL formulas over 1-structures, which allows to express such formulas in terms of MCL sentences over word-structures on 2^{AP_m} (or, equivalently, MSO sentences over infinite words on 2^{AP_m}).

► **Theorem 4.5 (Composition Theorem for MCL).** *For all $m \geq 1$ and 1-MCL formulas $\varphi(X)$ over AP with $\text{qr}(\varphi) \leq m$, there is an MCL sentence ψ over AP_m such that for each 1-structure (\mathcal{S}, C) and infinite branch π of \mathcal{S} with $C \subseteq \pi$, we have $(\mathcal{S}, C) \models \varphi(X) \Leftrightarrow \omega(\mathcal{S}, \pi, C) \models \psi$.*

Proof. According to Proposition 4.1, we consider the following formulas:

- the m -types $\alpha_1(X), \dots, \alpha_\ell(X)$ for 1-structures. By Proposition 4.1, there is $I \subseteq \{1, \dots, \ell\}$ such that $\varphi(X)$ is equivalent to $\bigvee_{p \in I} \alpha_p(X)$.
- The $2m$ -types $\gamma_1, \dots, \gamma_h$ for the MCL-sentences over word-structures on AP_m .

We denote by Γ the finite set of $2m$ -types γ_i with $i \in [1, h]$ such that there exist a 1-structure (\mathcal{S}, C) and an infinite branch π of \mathcal{S} with $C \subseteq \pi$ so that $(\mathcal{S}, C) \models \varphi(X)$ and $\omega(\mathcal{S}, \pi, C) \models \gamma_i$. The desired MCL sentence ψ is then given by $\bigvee_{\gamma_i \in \Gamma} \gamma_i$. We prove the following, hence the result directly follows.

▷ **Claim.** For each 1-structure (\mathcal{S}, C) and infinite branch π of \mathcal{S} so that $C \subseteq \pi$, it holds that $(\mathcal{S}, C) \models \varphi(X)$ if and only if $\omega(\mathcal{S}, \pi, C) \models \gamma_i$, for some $\gamma_i \in \Gamma$.

To prove the claim, let (\mathcal{S}, C) and π be as in the claim. By Proposition 4.1, there is a unique $i \in [1, h]$ such that $\omega(\mathcal{S}, \pi, C) \models \gamma_i$.

If $(\mathcal{S}, C) \models \varphi(X)$, then by construction, $\gamma_i \in \Gamma$. Assume now that $\gamma_i \in \Gamma$. It remains to show that $(\mathcal{S}, C) \models \varphi(X)$. We assume the contrary and derive a contradiction. Hence, there exists $p' \in [1, \ell] \setminus I$ such that $(\mathcal{S}, C) \models \alpha_{p'}(X)$. Since $\gamma_i \in \Gamma$, there exist $p \in I$, a 1-structure (\mathcal{S}', C') , and an infinite branch π' of \mathcal{S}' with $C' \subseteq \pi'$ so that $(\mathcal{S}', C') \models \alpha_p(X)$ and $\omega(\mathcal{S}', \pi', C') \models \gamma_i$. Since $\omega(\mathcal{S}, \pi, C) \models \gamma_i$, it follows that $\omega(\mathcal{S}, \pi, C) \equiv_{2m}^{\omega} \omega(\mathcal{S}', \pi', C')$. Thus, by Proposition 4.2 and Lemma 4.3, we obtain that $(\mathcal{S}, C) \equiv_m (\mathcal{S}', C')$, which is a contradiction since (\mathcal{S}, C) and (\mathcal{S}', C') have distinct m -types. ◀

4.3 From MCL to CECTL*

By exploiting Theorem 4.5, we show that the logics MCL and CECTL* have the same expressiveness.

► **Theorem 4.6.** *MCL and CECTL* are equally expressive.*

Proof. By [3], each CECTL* state formula has an equivalent MCL sentence. Thus, it suffices to show that for each MCL sentence, there is an equivalent CECTL* state formula. Let φ be an MCL sentence. The result is shown by an induction argument on the quantifier rank $\text{qr}(\varphi)$.

Base case. Let φ be an MCL sentence such that $\text{qr}(\varphi) = 1$. Since the existential chain quantifier \exists^c distributes over disjunction, φ is equivalent to a Boolean combination of sentences of the form $\exists^c X. \psi$, where ψ is a conjunction of *atoms in X*, i.e., atoms of the form $X \subseteq p$ or $X \leq X$ or $\text{sing}(X)$, or negations of atoms in X .

Fix an MCL sentence of the form $\exists^c X. \psi$, where ψ is a conjunction of atoms in X or negations of atoms X . We show that there is an equivalent CECTL* formula. Hence, the result follows. We assume that $\exists^c X. \psi$ is satisfiable (otherwise, $\exists^c X. \psi$ is equivalent to $\neg \top$, and the result trivially follows). We distinguish two cases:

- ψ holds when X is bound to the empty chain. In this case, $\exists^c X. \psi$ is equivalent to \top , and the result follows.
- ψ does not hold when X is bound to the empty chain: in this case, the atomic formula $X \leq X$ corresponds to $\text{sing}(X)$ and there exist distinct atomic propositions $p_1, \dots, p_n, q_1, \dots, q_m$ such that ψ can be rewritten as

$$\xi \wedge \bigwedge_{i=1}^{i=n} X \subseteq p_i \wedge \bigwedge_{j=1}^{j=m} \neg(X \subseteq q_j)$$

where either $\xi = \top$, or $\xi = \text{sing}(X)$, or $\xi = \neg\text{sing}(X)$. We focus on the case where $\xi = \neg\text{sing}(X)$ (the other cases being similar). Note that for an atom $X \subseteq p_i$ and two chains C and C' of a structure (Kripke tree) $\mathcal{S} = \langle T, \text{Lab} \rangle$ such that $C \subseteq C'$, $(\mathcal{S}, \mathbb{V}_2[X \mapsto C]) \models X \subseteq p_i$ entails that $(\mathcal{S}, \mathbb{V}_2[X \mapsto C']) \models X \subseteq p_i$ (intuitively, the satisfaction relation is *downward-closed* for atoms $X \subseteq p_i$). Moreover, $(\mathcal{S}, \mathbb{V}_2[X \mapsto C]) \models \neg(X \subseteq q_j)$ iff there is a node $w \in C$ such that $q_j \notin \text{Lab}(w)$. It follows that a structure (Kripke tree) $\mathcal{S} = \langle T, \text{Lab} \rangle$ is a model of $\exists^c X. \psi$ iff there exist $\ell \in [2, m+2]$ and a finite chain C of \mathcal{S} having cardinality ℓ such that (i) $p_i \in \text{Lab}(w)$ for all $i \in [1, n]$ and $w \in C$, and (ii) for all $j \in [1, m]$, there exists $w \in C$ so that $q_j \notin \text{Lab}(w)$. These requirements can be easily captured by a CTL* formula (and thus by a CECTL* formula as well). Hence, the result follows.

Induction step. Let $m \geq 1$ and assume that for each MCL sentence with quantifier rank at most m , there is an equivalent CECTL* state formula. Fix an MCL sentence of the form $\exists^c X. \varphi(X)$ with $\text{qr}(\varphi) \leq m$. We show that $\exists^c X. \varphi(X)$ has an equivalent CECTL* state formula. Hence, the result follows. For the fixed $m \geq 1$, let $\beta_1, \dots, \beta_\ell$ be the m -types for MCL sentences over structures on AP. Since $\beta_1, \dots, \beta_\ell$ have quantifier rank at most m , by the induction hypothesis, there exist CECTL* state formulas $\hat{\beta}_1, \dots, \hat{\beta}_\ell$ such that β_i and $\hat{\beta}_i$ are equivalent for each $i \in [1, \ell]$. Recall that $\text{AP}_m = 2^{\text{AP} \cup \{c\}} \times ([1, \ell] \mapsto [0, m]) \times [1, \ell]$. Let AP'_m obtained from AP_m by removing the special proposition c from the first component $2^{\text{AP} \cup \{c\}}$ of AP_m . For a structure \mathcal{S} (over AP) and an infinite branch π of \mathcal{S} , we write $\omega(\mathcal{S}, \pi)$ to mean the word-structure $\omega(\mathcal{S}, \pi, \emptyset)$. Note that $\omega(\mathcal{S}, \pi)$ corresponds to an infinite word over AP'_m . Recall that for each MSO sentence ϕ over infinite words, one can construct a Büchi NWA accepting the models of ϕ . Moreover, Büchi NWA are closed under projection and a Büchi NWA can be converted into an equivalent parity DWA [42]. Thus, since MCL over word structures corresponds to MSO over infinite words, by applying Theorem 4.5 to the 1-MCL formula $\varphi(X)$, there exists a parity NWA \mathcal{D}_φ over AP'_m such that the following holds.

▷ **Claim 4.7.** For each structure \mathcal{S} , there exists a chain C of \mathcal{S} such that $(\mathcal{S}, C) \models \varphi(X)$ if and only if there exists an infinite branch π of \mathcal{S} so that $\omega(\mathcal{S}, \pi) \in \text{L}(\mathcal{D}_\varphi)$.

Let $\Upsilon \triangleq ([1, \ell] \mapsto [0, m]) \times [1, \ell]$. Now, for each $(f, k) \in \Upsilon$, we define a CECTL* path formula $\theta_{(f,k)}$ expressing, for a given structure \mathcal{S} , infinite branch π of \mathcal{S} , and node $w \in \pi$, that:

- for each $i \in [1, \ell]$, let N be the (possibly infinite) number of distinct children w' of w such that the substructure of \mathcal{S} rooted at w' has m -type β_i . Then:
 - case $i \neq k$: $N = f(i)$ if $f(i) < m$, and $N \geq f(i)$ otherwise;
 - case $i = k$: $N = f(i) + 1$ if $f(i) < m$, and $N \geq f(i) + 1$ otherwise.
- The substructure of \mathcal{S} rooted at the child w' of w along π has m -type β_k .

$$\theta_{(f,k)} \triangleq (\mathbf{X} \hat{\beta}_k) \wedge \bigwedge_{i \in [1, \ell]} \theta_{(f,k)}^i, \text{ where}$$

$$\theta_{(f,k)}^i \triangleq \begin{cases} \mathbf{D}^{f(i)} \hat{\beta}_i, & \text{if } f(i) = m \wedge i \neq k; \\ \mathbf{D}^{f(i)} \hat{\beta}_i \wedge \neg \mathbf{D}^{f(i)+1} \hat{\beta}_i, & \text{if } f(i) < m \wedge i \neq k; \\ \mathbf{D}^{f(i)+1} \hat{\beta}_i, & \text{if } f(i) = m \wedge i = k; \\ \mathbf{D}^{f(i)+1} \hat{\beta}_i \wedge \neg \mathbf{D}^{f(i)+2} \hat{\beta}_i, & \text{if } f(i) < m \wedge i = k. \end{cases}$$

Since $\beta_1, \dots, \beta_\ell$ are the m -types for MCL sentences, by construction, the following holds.

▷ **Claim 4.8.** Given a structure \mathcal{S} , an infinite branch π of \mathcal{S} , and $i \geq 0$, there is exactly one element $(f, k) \in \Upsilon$ such that $(\mathcal{S}, \pi, i) \models \theta_{(f,k)}$.

Let $\mathcal{D}_\varphi = \langle 2^{\text{AP}} \times \Upsilon, \mathbb{Q}_D, \delta_D, q_{D,I}, \Omega_D \rangle$ be the parity DWA of Claim 4.7. For each $(f, k) \in \Upsilon$, we consider the parity NWA $\mathcal{N}_{(f,k)} = \langle 2^{\text{AP}}, \mathbb{Q}_N, \delta_N, (q_{D,I}, f, k), \Omega_N \rangle$ over 2^{AP} with initial state $(q_{D,I}, f, k)$ which simulates \mathcal{D}_φ by keeping track in the current state of the guessed second component of the next input symbol. Formally $\mathbb{Q}_N = \mathbb{Q}_D \times \Upsilon$, $\delta_N((q', f', k'), a) = \bigvee_{(f'', k'') \in \Upsilon} (\delta_D(q', (a, f', k')), f'', k'')$ and $\Omega_N(q', (f', k')) = \Omega_D(q')$ for all $q' \in \mathbb{Q}_D$, $a \in 2^{\text{AP}}$, and $(f', k') \in \Upsilon$. Note that for $(f, k) \neq (f', k')$, the parity NWA $\mathcal{N}_{(f,k)}$ and $\mathcal{N}_{(f',k')}$ differ only for the initial state. Moreover, let τ be the testing function assigning to each state $(q', f', k') \in \mathbb{Q}_N$ the CECTL* path formula $\theta_{(f',k')}$. By construction and Claim 4.7, we obtain the following characterization of the structures satisfying the MCL sentence $\exists^c X. \varphi(X)$.

▷ **Claim 4.9.** For each structure $\mathcal{S} = (\mathbb{T}, \text{Lab})$, $\mathcal{S} \models \exists^c X. \varphi(X)$ iff for some infinite branch π of \mathcal{S} and some $(f, k) \in \Upsilon$, there is an accepting run ν of $\mathcal{N}_{(f,k)}$ over $\text{Lab}(\pi(0))\text{Lab}(\pi(1))\dots$ such that $(\mathcal{S}, \pi, i) \models \tau(\nu(i))$ for all $i \geq 0$.

Given a finite path π_f of a structure $\mathcal{S} = (\mathbb{T}, \text{Lab})$, a *good run* of $\mathcal{N}_{(f,k)}$ over π_f is a finite path ν_f of $\mathcal{N}_{(f,k)}$ over the *Lab*-labeling of π_f such that $(\mathcal{S}, \pi_f(i)) \models \tau(\nu(i))$ for all $0 \leq i < |\pi_f|$.

We now show that the characterization of the set of models of $\exists^c X. \varphi(X)$ in Claim 4.9 can be captured by a CECTL* formula. For all states $(q, f, k) \in \mathbb{Q}_N$ and set $P \subseteq \mathbb{Q}_N$, we denote by ${}_{(q,f,k)}\mathcal{N}_P$ the testing NWA_f with test function τ and whose embedded NWA_f is obtained from the automata $\mathcal{N}_{(f,k)}$ by setting a fresh copy of (q, f, k) as initial state, and P as set of accepting states. This fresh copy behaves as (q, f, k) and has the same test as (q, f, k) , and ensures that the automaton cannot accept the empty word. Finally, let $\mathbb{Q}_{N, \text{even}}$ be the set of states in \mathbb{Q}_N having even color, and for each $(q, f, k) \in \mathbb{Q}_N$, let $\mathbb{Q}_N > (q, f, k)$ be the set of states in \mathbb{Q}_N having color greatest than the color of (q, f, k) . We consider the CECTL* state formula $\text{E}\psi$ where the CECTL* path formula ψ is defined as follows:

$$\begin{aligned} \psi &\triangleq \bigvee_{(f,k) \in \Upsilon} \bigvee_{(q', f', k') \in \mathbb{Q}_{N, \text{even}}} (\psi_1(f, k, q', f', k') \wedge \psi_2(f, k, q', f', k')) \\ \psi_1(f, k, q', f', k') &\triangleq \langle {}_{(q_{D,I}, f, k)}\mathcal{N}_{\{(q', f', k')\}} \rangle [{}_{(q', f', k')}\mathcal{N}_{\mathbb{Q}_N > (q', f', k')}] \neg \top \\ \psi_2(f, k, q', f', k') &\triangleq [{}_{(q_{D,I}, f, k)}\mathcal{N}_{\{(q', f', k')\}}] \langle {}_{(q', f', k')}\mathcal{N}_{\{(q', f', k')\}} \rangle \top \end{aligned}$$

Thus, an infinite branch π of a structure \mathcal{S} satisfies the path formula ψ iff there exist $(f, k) \in \Upsilon$ and $(q', f', k') \in \mathbb{Q}_{N, \text{even}}$ such that the following conditions hold:

- there is a good run of $\mathcal{N}_{(f,k)}$ over some non-empty prefix $\pi(0) \dots \pi(i)$ of π from state $(q_{D,I}, f, k)$ to the state with even color (q', f', k') . Moreover, no good run of $\mathcal{N}_{(f,k)}$ over some non-empty infix of π from position i which starts and ends at state (q', f', k') visits a state with color greatest than color of (q', f', k') .
- for each good run of $\mathcal{N}_{(f,k)}$ over some non-empty prefix $\pi(0) \dots \pi(i)$ of π from state $(q_{D,I}, f, k)$ to state (q', f', k') , there is a good run of $\mathcal{N}_{(f,k)}$ over some non-empty infix of π from position i which starts and ends at state (q', f', k') .

The first condition is expressed by the conjunct $\psi_1(f, k, q', f', k')$ of ψ , while the second condition is expressed by the conjunct $\psi_2(f, k, q', f', k')$. By Claim 4.9, correctness of the construction directly follows from the following claim whose proof relies on the mutual-exclusivity condition expressed in Claim 4.8.

▷ **Claim 4.10.** For each structure $\mathcal{S} = (\mathbb{T}, \text{Lab})$ and infinite branch π of \mathcal{S} , $(\mathcal{S}, \pi, 0) \models \psi$ iff for some $(f, k) \in \Upsilon$, there exists an accepting run ν of $\mathcal{N}_{(f,k)}$ over $\text{Lab}(\pi(0))\text{Lab}(\pi(1))\dots$ such that $(\mathcal{S}, \pi, i) \models \tau(\nu(i))$ for all $i \geq 0$.

The left-right implication in Claim 4.10 easily follows from construction. For the right-left implication, assume that for some $(f, k) \in \Upsilon$, there is an accepting run $\nu = (q_0, f_0, k_0)(q_1, f_1, k_1)\dots$ of $\mathcal{N}_{(f,k)}$ over $\rho = \text{Lab}(\pi(0))\text{Lab}(\pi(1))\dots$ with $(q_0, f_0, k_0) = (q_{D,I}, f, k)$ such that $(\mathcal{S}, \pi, i) \models \theta_{(f_i, k_i)}$ for all $i \geq 0$. Since ν is accepting, there exists a state $(q', f', k') \in \mathbb{Q}_{N, \text{even}}$ having an even color n such that n is the maximum color associated to the states which occur infinitely many times along ν . We show that $(\mathcal{S}, \pi, 0) \models \psi_1(f, k, q', f', k') \wedge \psi_2(f, k, q', f', k')$. Hence, the result follows. We focus on the conjunct $\psi_2(f, k, q', f', k')$ (the proof for the conjunct $\psi_1(f, k, q', f', k')$ is similar). By construction of $\psi_2(f, k, q', f', k')$, it suffices to show that for all $j \geq 0$ and accepting runs ν_f of $(q_{D,I}, f, k)\mathcal{N}_{\{(q', f', k')\}}$ over $\rho[0, j]$ whose states satisfy the associated tests, then ν_f is a prefix of ν . Let $\nu_f = (q'_0, f'_0, k'_0)\dots(q'_{j+1}, f'_{j+1}, k'_{j+1})$ be such a finite run over $\rho[0, j]$ such that $(q'_0, f'_0, k'_0) = (q_{D,I}, f, k)$ and for all $i \in [0, j+1]$, $(\mathcal{S}, \pi, i) \models \theta_{(f'_i, k'_i)}$. Since $(\mathcal{S}, \pi, i) \models \theta_{(f_i, k_i)}$ for all $i \geq 0$, by Claim 4.8, it follows that $(f'_i, k'_i) = (f_i, k_i)$ for all $i \in [0, j+1]$. Thus, since \mathcal{D}_φ is deterministic, we deduce that $q'_i = q_i$ for all $i \in [0, j+1]$, and the result follows. This concludes the proof of Claim 4.10.

At this point, the equivalence between $\exists^c X. \varphi(X)$ and $\text{E}\psi$ directly follows from Claims 4.9 and 4.10. This concludes the proof of Theorem 4.6. ◀

5 Conclusion

In this work, we adopted a compositional approach to prove the expressive equivalence of *Monadic Chain Logic* (MCL) and the counting extension CECTL^* of ECTL^* . Recent work [3] has established that the graded version (HGTA) of *Hesitant Tree Automata* (HTA) and their first-order extension (HFTA) represent the automata counterparts of the logics CECTL^* and MCL, respectively. As a corollary of our main results, we obtain the following chain of equivalence:

► **Corollary 5.1.** *The logics CECTL^* and MCL and the classes of automata HGTA and HFTA are all equivalent formalisms.*

It would be interesting to explore the applicability of a compositional approach to *Monadic Tree Logic* (MTL) [2], a fragment of MSO where second-order quantifiers range over trees. The goal here is to gain insights into the expressiveness of various extensions of standard temporal logics for strategic reasoning, such as Substructure Temporal Logic (STL), a temporal logic that allows implicit predication over substructures/subtrees [4, 5].

References

- 1 A. Arnold and D. Niwiński. Fixed Point Characterization of Weak Monadic Logic Definable Sets of Trees. In *Tree Automata and Languages*, pages 159–188. North-Holland, 1992.
- 2 M. Benerecetti, L. Bozzelli, F. Mogavero, and A. Peron. Quantifying over Trees in Monadic Second-Order Logic. In *LICS'23*, pages 1–13. IEEECS, 2023.
- 3 M. Benerecetti, L. Bozzelli, F. Mogavero, and A. Peron. Automata-Theoretic Characterisations of Branching-Time Temporal Logics. In *ICALP'24*, LIPIcs 297, pages 128:1–20. Leibniz-Zentrum fuer Informatik, 2024.
- 4 M. Benerecetti, F. Mogavero, and A. Murano. Substructure Temporal Logic. In *LICS'13*, pages 368–377. IEEECS, 2013.

- 5 M. Benerecetti, F. Mogavero, and A. Murano. Reasoning About Substructures and Games. *TOCL*, 16(3):25:1–46, 2015.
- 6 U. Boker, K. Lehtinen, and S. Sickert. On the Translation of Automata to Linear Temporal Logic. In *FOSSACS'22*, LNCS 13242, pages 140–160. Springer, 2022. [path\(doi : 10.1007/978 - 3 - 030 - 99253 - 8₈\)](#).
- 7 J.R. Büchi. Weak Second-Order Arithmetic and Finite Automata. *MLQ*, 6(1-6):66–92, 1960.
- 8 J.R. Büchi. On a Decision Method in Restricted Second-Order Arithmetic. In *ICLMPs'62*, pages 1–11. Stanford University Press, 1962.
- 9 J.R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *Studies in Logic and the Foundations of Mathematics*, volume 44, pages 1–11. Elsevier, 1966.
- 10 Y. Choueka. Theories of Automata on ω -Tapes: A Simplified Approach. *JCSS*, 8(2):117–141, 1974.
- 11 E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach. In *POPL'83*, pages 117–126. ACM, 1983.
- 12 E.A. Emerson and E.M. Clarke. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *LP'81*, LNCS 131, pages 52–71. Springer, 1982.
- 13 E.A. Emerson and J.Y. Halpern. “Sometimes” and “Not Never” Revisited: On Branching Versus Linear Time. In *POPL'83*, pages 127–140. ACM, 1983.
- 14 E.A. Emerson and J.Y. Halpern. “Sometimes” and “Not Never” Revisited: On Branching Versus Linear Time. *JACM*, 33(1):151–178, 1986.
- 15 S. Feferman and R. Vaught. The First-Order Properties of Products of Algebraic Systems. *FM*, 47(1):57–103, 1959.
- 16 K. Fine. In So Many Possible Worlds. *NDJFL*, 13:516–520, 1972. [path\(doi : 10.1305/NDJFL/1093890715\)](#).
- 17 M.J. Fischer and R.E. Ladner. Propositional Dynamic Logic of Regular Programs. *JCSS*, 18(2):194–211, 1979. [path\(doi : 10.1016/0022 - 0000\(79\)90046 - 1\)](#).
- 18 G. De Giacomo and M.Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI'13*, pages 854–860. IJCAI' & AAAI Press, 2013.
- 19 Y. Gurevich. Modest Theory of Short Chains. I. *JSL*, 44(4):481–490, 1979. [path\(doi : 10.2307/2273287\)](#).
- 20 Y. Gurevich. Monadic Second-Order Theories. In *Model-Theoretical Logics*, pages 479–506. Springer, 1985.
- 21 Y. Gurevich and S. Shelah. Modest Theory of Short Chains. II. *JSL*, 44(4):491–502, 1979. [path\(doi : 10.2307/2273288\)](#).
- 22 Y. Gurevich and S. Shelah. Rabin’s Uniformization Problem. *JSL*, 48(4):1105–1119, 1979.
- 23 Y. Gurevich and S. Shelah. The Decision Problem for Branching Time Logic. *JSL*, 50(3):668–681, 1985. [path\(doi : 10.2307/2274321\)](#).
- 24 T. Hafer and W. Thomas. Computation Tree Logic CTL* and Path Quantifiers in the Monadic Theory of the Binary Tree. In *ICALP'87*, LNCS 267, pages 269–279. Springer, 1987. [path\(doi : 10.1007/3 - 540 - 18088 - 5₂2\)](#).
- 25 D. Janin. *A Contribution to Formal Methods: Games, Logic and Automata*. Habilitation thesis, Université Bordeaux I, Bordeaux, France, 2005.
- 26 D. Janin and G. Lenzi. On the Relationship Between Monadic and Weak Monadic Second Order Logic on Arbitrary Trees, with Applications to the μ -Calculus. *FI*, 61(3-4):247–265, 2004. URL: <http://content.iospress.com/articles/fundamenta-informaticae/fi61-3-4-04>.
- 27 D. Janin and I. Walukiewicz. On the Expressive Completeness of the Propositional μ -Calculus with Respect to Monadic Second Order Logic. In *CONCUR'96*, LNCS 1119, pages 263–277. Springer, 1996. [path\(doi : 10.1007/3 - 540 - 61604 - 7₆0\)](#).
- 28 H.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, CA, USA, 1968.

- 29 D. Kozen. Results on the Propositional μ Calculus. *TCS*, 27(3):333–354, 1983. path(*doi* : 10.1016/0304 – 3975(82)90125 – 6).
- 30 R.E. Ladner. Application of Model Theoretic Games to Discrete Linear Orders and Finite Automata. *IC*, 33(4):281–303, 1977. path(*doi* : 10.1016/S0019 – 9958(77)90443 – 0).
- 31 H. Läuchli. A Decision Procedure for the Weak Second-Order Theory of Linear Order. In *LC'66*, volume 50, pages 189–197. North-Holland, 1968.
- 32 O. Maler and A. Pnueli. On the Cascaded Decomposition of Automata, its Complexity, and its Application to Logic. Unpublished, 1995.
- 33 Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer, 1992.
- 34 Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems - Safety*. Springer, 1995.
- 35 R. McNaughton. Testing and Generating Infinite Sequences by a Finite Automaton. *IC*, 9(5):521–530, 1966. path(*doi* : 10.1016/S0019 – 9958(66)80013 – X).
- 36 R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.
- 37 F. Moller and A.M. Rabinovich. On the Expressive Power of CTL*. In *LICS'99*, pages 360–368. IEEECS, 1999.
- 38 F. Moller and A.M. Rabinovich. Counting on CTL*: On the Expressive Power of Monadic Path Logic. *IC*, 184(1):147–159, 2003. path(*doi* : 10.1016/S0890 – 5401(03)00104 – 4).
- 39 D. Perrin and J. Pin. First-Order Logic and Star-Free Sets. *JCSS*, 32(3):393–406, 1986. path(*doi* : 10.1016/0022 – 0000(86)90037 – 1).
- 40 A. Pnueli. The Temporal Logic of Programs. In *FOCS'77*, pages 46–57. IEEECS, 1977.
- 41 A. Pnueli. The Temporal Semantics of Concurrent Programs. *TCS*, 13:45–60, 1981. path(*doi* : 10.1016/0304 – 3975(81)90110 – 9).
- 42 S. Safra. On the Complexity of ω -Automata. In *FOCS'88*, pages 319–327. IEEECS, 1988.
- 43 S. Shelah. The Monadic Theory of Order. *AM*, 102(3):379–419, 1975.
- 44 W. Thomas. Star-Free Regular Sets of ω -Sequences. *IC*, 42(2):148–156, 1979.
- 45 W. Thomas. A Combinatorial Approach to the Theory of ω -Automata. *IC*, 48(3):261–283, 1981.
- 46 W. Thomas. Logical Aspects in the Study of Tree Languages. In *CAAP'84*, pages 31–50. CUP, 1984.
- 47 W. Thomas. On Chain Logic, Path Logic, and First-Order Logic over Infinite Trees. In *LICS'87*, pages 245–256. IEEECS, 1987.
- 48 W. Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science (vol. B)*, pages 133–191. MIT Press, 1990.
- 49 J. van Benthem. *Modal Correspondence Theory*. PhD thesis, University of Amsterdam, Amsterdam, Netherlands, 1977.
- 50 M.Y. Vardi and P. Wolper. Yet Another Process Logic. In *LP'83*, LNCS 164, pages 501–512. Springer, 1984.
- 51 I. Walukiewicz. Monadic Second Order Logic on Tree-Like Structures. *TCS*, 275(1-2):311–346, 2002. path(*doi* : 10.1016/S0304 – 3975(01)00185 – 2).
- 52 P. Wolper. Temporal Logic Can Be More Expressive. *IC*, 56(1-2):72–99, 1983. path(*doi* : 10.1016/S0019 – 9958(83)80051 – 5).

Model Checking Linear Temporal Properties on Polyhedral Systems

Massimo Benerecetti  

Università di Napoli Federico II, Italy

Marco Faella  

Università di Napoli Federico II, Italy

Fabio Mogavero  

Università di Napoli Federico II, Italy

Abstract

We study the problem of model checking linear temporal logic formulae on finite trajectories generated by polyhedral differential inclusions, thus enriching the landscape of models where such specifications can be effectively verified. Each model in the class comprises a static and a dynamic component. The static component features a finite set of observables represented by (non-necessarily convex) polyhedra. The dynamic one is given by a convex polyhedron constraining the dynamics of the system, by specifying the possible slopes of the trajectories in each time instant. We devise an exact algorithm that computes a symbolic representation of the region of points that existentially satisfy a given formula φ , i.e., the points from which there exists a trajectory satisfying φ .

2012 ACM Subject Classification Theory of computation \rightarrow Modal and temporal logics

Keywords and phrases Model Checking, Real-Time Systems, LTLf, RTLf

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.19

Funding PNRR MUR project PE0000013-FAIR and Indam GNCS 2024 project “Certificazione, Monitoraggio, ed Interpretabilità in Sistemi di Intelligenza Artificiale”. The authors are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

1 Introduction

Formal verification has been a central topic in computer science for decades, and model checking has emerged as a key technique for this purpose. In this paper, we focus on *continuous-time* and *infinite-state* systems, which are essential for cyber-physical applications [20]. We represent the state of our systems using a vector of real-valued variables, whose dynamics are governed by a constant polyhedral inclusion of the type $\dot{x} \in F$, where F is a convex polyhedron. Such dynamics correspond to the single-location dynamics of linear hybrid automata (LHAs) [13]. Whereas reachability in LHAs is undecidable [14], we show in this paper that model checking a linear temporal property on a single location is a decidable, albeit challenging, problem.

As specification language, we consider a real-time interpretation of linear temporal logic on finite traces (LTL_f), that we call RTL_f following Reynolds [22]. Compared to LTL_f (and LTL), RTL_f does not include an explicit next operator, which is commonly omitted when considering continuous time domains, but includes both a strict and non-strict version of the until operator. In our interpretation, time is real-valued and each atomic proposition denotes a polyhedral region of the state-space. Hence, users can exploit the familiar syntax of LTL to express complex properties involving continuous variables and their relationships.

The polyhedral inclusions that define our trajectories bestow a considerable degree of flexibility, affording room for behaviours, commonly referred to as *Zeno behaviours*, which may lack a plausible physical rationale or clash with the symbolic abstraction adopted in this



© Massimo Benerecetti, Marco Faella, and Fabio Mogavero;
licensed under Creative Commons License CC-BY 4.0

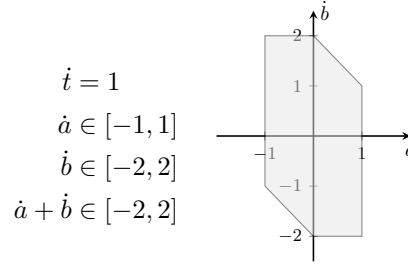
31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 19; pp. 19:1–19:23

Leibniz International Proceedings in Informatics



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



■ **Figure 1** The flow and its projection on the (\dot{a}, \dot{b}) plane.

paper. To avoid these issues, since our observables are polyhedral regions of the state-space, we restrict our attention to trajectories that transition between polyhedra finitely often within any bounded time interval. We call this notion *well behavedness* and compare it with similar notions in the existing literature.

Our main contribution is a symbolic algorithm to determine the set of initial states from which the system supports a well-behaved trajectory that satisfies a given property, a problem that we call the *existential denotation problem* for RTL_f . The algorithm is based on a translation from RTL_f to LTL_f , followed by the classical automata construction for LTL_f . Then, the finite-state automaton is used as a guide for a backward symbolic computation of the existential denotation of the input formula.

The results of the existential denotation problem can be used in two ways, depending on the interpretation given to the input model. Indeed, the non-determinism inherent in a polyhedral inclusion can be meant either in an angelic (i.e., controllable) or demonic (i.e., uncontrollable) sense. In the first case, a constraint of the type $\dot{x} \in [1, 2]$ is taken to mean that the variable x can be steered by the system to grow with any rate between 1 and 2. In the second case, the same constraint signals that the environment may choose any growth rate between 1 and 2. Given a model with angelic non-determinism, one may use the results in this paper to verify that the system can be controlled into satisfying a specified property. If instead the non-determinism is meant to be interpreted as demonic, one will specify an error condition and check from which states the environment can generate a trajectory that engenders the error. Our work has potential applications in a variety of domains, including robotics and control systems, and offers new insights into the analysis of polyhedral systems.

A Motivating Example. Consider a system of two tanks connected with a pump and holding a liquid. An inlet pours liquid into the first tank at an uncertain and time-varying rate, which however is known to be contained in $[1, 2]$. The pump shifts liquid from the first tank to the second tank at a varying rate contained in $[1, 2]$. Finally, an outlet extracts liquid from the second tank at a varying rate contained in $[0, 3]$. If we represent the level in the first (resp., second) tank with variable a (resp., b) and we add a clock t to measure the passage of time, the above constraints lead to the dynamic laws reported in Figure 1.

Notice that the above semantics allows levels to become negative: we guarantee that this does not happen using the formula $\varphi_{\text{inv}} = \mathbf{G}(a \geq 0 \wedge b \geq 0)$. Suppose that we want to find the initial states from which the system, within the first 10 time units, can first reach a configuration where $a \geq b + 1$ and later reach another configuration where $b \geq a + 1$. This property is captured by the following formula:

$$\varphi_1^{\text{gap}} = \varphi_{\text{inv}} \wedge (t = 0) \wedge \mathbf{G}(t \leq 10) \wedge \mathbf{F}(a \geq b + 1 \wedge \mathbf{F}(b \geq a + 1)).$$

This example also shows that, despite not directly supporting time bounds on the temporal operators, RTL_f allows to talk about *absolute* time, by introducing an extra variable t into the model to represent time. In Section 6, we show how our algorithm can readily compute the set of initial points supporting a trajectory that satisfies the above formula, as well as several variations thereof.

Related Work. Several temporal logics have been proposed in the literature to express properties of real-time systems. Some proposals enrich classical temporal logic with new operators specific for real time, like decorating the *until* operator with time bounds. That is the case of MTL [16], MITL [2], and STL [18]. Other approaches, including ours, reinterpret the original LTL on real time. In particular, Reynolds investigates the validity problem for LTL interpreted over real time [22].

The dynamics we support generalise the single-mode (i.e., single-location) dynamics of timed automata [1] and constant-rate multi-mode systems (MMS) [4], and correspond to the single-mode dynamics of linear hybrid automata (LHA) [13]. In the case of MMS's, reachability is a decidable problem, yet full LTL model checking is not. Notably, Blondin et al. have recently delineated a range of decidable syntactic fragments in this context [8]. When it comes to LHAs, even the reachability problem is undecidable [14]. This has not prevented the development of approximate or incomplete approaches, included in tools like SpaceEx [12] and NYCS [6].

If we go even higher in expressivity ladder of models for single-mode systems, the polyhedral inclusion characterising our model can be considered as a special case of an affine system with controllable inputs (i.e., a dynamics of the type $\dot{x} = Ax + b + Bu$ where $A = 0$ and the control input u plays the role of nondeterminism). In that model, a sound but incomplete synthesis approach for LTL specifications was proposed [15].

Structure of the Paper. The paper is organised as follows. Section 2 introduces polyhedral systems and their trajectories, and discusses the notion of well-behavedness and its relationship with other standard regularity conditions. Section 3 defines the classical (i.e., discrete) and continuous semantics of LTL_f and RTL_f , respectively. Section 4 provides the technical framework to discretise trajectories into traces and RTL_f formulae into LTL_f formulae. Section 5 presents our algorithm for the existential denotation problem, and Section 6 describes the experiments performed on our prototype implementation.

2 Polyhedral Systems, Trajectories, and Signals

We study continuous-time and continuous-state dynamic systems, whose state $x \in \mathbb{R}^n$ evolves non-deterministically under a differential inclusion of the type $\dot{x} \in \text{Flow}$, for a fixed convex polyhedron Flow . In the following, we shall use the symbol \mathbb{R}^+ to denote the set of non-negative reals and \bar{X} to denote the complement of a set $X \subseteq \mathbb{R}^n$.

Polyhedra. A *convex polyhedron* is the intersection of a finite number of strict or non-strict half-spaces. A *polyhedron* is a finite union of convex polyhedra and a *polytope* is a bounded convex polyhedron. We denote by $\text{Poly}(\mathbb{R}^n)$ (resp., $\text{CPoly}(\mathbb{R}^n)$) the set of polyhedra (resp., convex polyhedra) on \mathbb{R}^n . We shall use the letters P, Q to refer to convex polyhedra and letters A, B, G for general polyhedra, instead. For a polyhedron G , we denote by $\text{Patch}(G)$ its representation as a finite set of convex polyhedra, called the *patches* of G . Also, $\text{cl}(P)$ is *topological closure* of P , obtained by replacing all strict half-spaces with non-strict ones.

Atomic propositions. In the rest of the paper, we assume a finite set AP of atomic proposition symbols. Each atomic proposition $p \in AP$ is interpreted as a polyhedron $[p] \in \text{Poly}(\mathbb{R}^n)$, called its *interpretation*. That is, $[p]$ is the set of points where p holds. For a set of atomic propositions $\alpha \subseteq AP$, we denote with $\llbracket \alpha \rrbracket$ the interpretation of the set α , namely the set of points where all and only the propositions in α hold. That is,

$$\llbracket \alpha \rrbracket = \bigcap_{p \in \alpha} [p] \cap \bigcap_{p \in AP \setminus \alpha} \overline{[p]}.$$

By definition, $\llbracket \alpha \rrbracket$ is a polyhedron. Observe that $\llbracket \{p\} \rrbracket \subseteq [p]$ and the inclusion may be strict. For instance, if $[p] = \{x \geq 0\}$ and $[q] = \{x \geq 2\}$, then $\llbracket \{p\} \rrbracket = \{0 \leq x < 2\}$. Moreover, for any two sets of atomic propositions $\alpha_1, \alpha_2 \subseteq AP$, either $\llbracket \alpha_1 \rrbracket = \llbracket \alpha_2 \rrbracket$ or $\llbracket \alpha_1 \rrbracket \cap \llbracket \alpha_2 \rrbracket = \emptyset$. Hence, the image of 2^{AP} under $\llbracket \cdot \rrbracket$ is a partition of \mathbb{R}^n into polyhedra.

Trajectories under polyhedral differential inclusions. We are interested in dynamic systems that obey a given polyhedral differential inclusion. Therefore, we assume a fixed convex polyhedron $Flow \subseteq \mathbb{R}^n$ called the *flow constraint*, and we omit it from the notation whenever possible. We call the pair $\mathcal{P} = (Flow, [\cdot])$ a *polyhedral system*.

For a number $T \in \mathbb{R}^+$, we use $\langle 0, T \rangle$ as a shorthand for one of the two right-closed intervals, either $(0, T]$ or $[0, T]$, with left endpoint 0 and right endpoint T . Given a point $x \in \mathbb{R}^n$, a *finite-time trajectory* (*trajectory* from now on) starting from x is a function $f: \langle 0, T \rangle \rightarrow \mathbb{R}^n$, such that: (i) $\lim_{t \rightarrow 0} f(t) = x$, (ii) f is continuous, (iii) f is differentiable everywhere in its domain except for a finite number of points, (iv) whenever the derivative $\dot{f}(t)$ is defined, it holds that $\dot{f}(t) \in Flow$. When $\langle 0, T \rangle = [0, T]$ (*resp.*, $\langle 0, T \rangle = (0, T]$) we say that f is *left-closed* (*resp.*, *left-open*). We use $\text{Traj}(x)$ to denote the set of all trajectories starting from x .

The interpretation $[\cdot]$ of the atomic propositions induces a mapping from trajectories to functions of type $\langle 0, T \rangle \rightarrow 2^{AP}$, called *bounded signals* [17] (*signals* from now on), over which we shall base the semantics of the logics defined in Section 3. Namely, given a trajectory f , we denote with σ_f the signal assigning to each time instant t the set of atomic propositions that are true at $f(t)$. Formally,

$$\sigma_f(t) \triangleq \{p \in AP \mid f(t) \in [p]\}.$$

For a signal σ and a time $t \in \langle 0, T \rangle$, we denote by $\sigma_{\sim t}$, with $\sim \in \{>, \geq\}$, the left-open or left-closed *suffix of σ starting at t* defined as follows: $\sigma_{\sim t}(t') = \sigma(t + t')$, for all t' such that $t' \sim 0$ and $t + t' \in \langle 0, T \rangle$.

2.1 Well-Behavedness and Finite Variability

A *well-behaved* trajectory $f: \langle 0, T \rangle \rightarrow \mathbb{R}^n$ is a trajectory that crosses any hyperplane a finite number of times, *i.e.*, for all hyperplanes H there is a finite set of times $0 = t_0 < t_1 < \dots < t_k = T$ such that, during every open interval (t_i, t_{i+1}) , the trajectory f lies in the same closed half-space induced by H . We denote by $\text{Traj}_{\text{wb}}(x)$ the set of all well-behaved trajectories starting from a point $x \in \mathbb{R}^n$.

Considering the membership in a half-space as an observable, the condition above states that the truth value of the observable along the trajectory f changes only a finite number of times in every bounded time interval. This last property is equivalent to the notion of *discrete variation*, as observed in [9].

These notions can be compared to classical notion in analysis such as *analyticity* and *Lipschitz continuity*. Recall that a trajectory f is *analytic in a point* t in its domain if it is smooth at t and the Taylor's series of f at t converges to f in some open neighbourhood of t . Moreover, f is said to be *analytic* if it is analytic in every point of its domain and *piecewise analytic* if it is analytic in every point of its domain except for a finite number.¹ The following result follows from Theorem 16 of [9].

► **Proposition 1.** *On the set of trajectories, piecewise analyticity implies well-behavedness.*

A trajectory f , instead, is *Lipschitz continuous* on $X \subseteq \mathbb{R}^+$ if there exists $K \geq 0$ such that, for all $t_1, t_2 \in X$,

$$\|f(t_1) - f(t_2)\| \leq K \cdot |t_1 - t_2|,$$

where $\|\cdot\|$ denotes the Euclidean norm. Moreover, f is *locally Lipschitz continuous* if for all $t \in \mathbb{R}^+$ there exists a neighbourhood of t , where f is Lipschitz continuous.

► **Proposition 2.** *On the set of trajectories, Lipschitz continuity and well-behavedness are incomparable notions.*

Next, we provide an alternative characterisation of well-behavedness. A *polyhedral partition* of \mathbb{R}^n is a finite set of mutually disjoint convex polyhedra whose union is \mathbb{R}^n .

► **Proposition 3.** *A trajectory is well-behaved iff, for all polyhedral partitions of \mathbb{R}^n and all time instants $t \in \mathbb{R}^+$, the trajectory changes polyhedron a finite number of times during $[0, t]$.*

We say that a signal $\sigma: \langle 0, T \rangle \rightarrow 2^{AP}$ has *finite variability* if it changes its value only a finite number of times. Formally, this means that there exists a strictly-increasing finite sequence of time points $0 = t_0 < \dots < t_k = T$ and a finite sequence of observables $\{\alpha_i\}_{i=0}^{k-1} \subseteq 2^{AP}$ such that, for all indexes $0 \leq i < k$ and time instants $t \in (t_i, t_{i+1})$, it holds true that $\sigma(t) = \alpha_i$. We call any such sequence of time points $\tau = \{t_i\}_{i=0}^k \subseteq \mathbb{R}^+$ a *time-slicing* of σ and denote the set of these sequences $TS(\sigma)$. Note that this set does not depend on whether the signal is left-open or not, *i.e.*, $TS(\sigma) = TS(\sigma_{>0})$.

As we shall show in the Section 4, the notion of (finite) time-slicing is an essential component of the solution technique proposed in this paper, which reduces the problem of checking RTL_f formulae to that of checking LTL_f formulae interpreted on finite discrete abstractions of bounded signals. The existence of a time-slicing for a signal as defined above, however, relies on the finite variability property of that signal, as infinite-variability bounded signals do not admit finite time-slicing.

An immediate consequence of Proposition 3 is that for any polyhedral system \mathcal{P} , all well-behaved trajectories induce finite variability signals.

► **Proposition 4.** *If a trajectory f is well-behaved, then the corresponding signal σ_f has finite variability.*

In the rest of the paper we shall leave the polyhedral system implicit, consider only well-behaved trajectories and, therefore, only finite variability signals. The following table summarises the main semantic notions and their intuitive meaning.

¹ Note that this is a slight adaptation of the classical notion to the case of functions defined on a bounded domain.

■ **Table 1** Main notions used in the paper: three types of trace-like objects (from the most concrete to the most abstract), three types of runs of an automaton, and the time decomposition of a signal.

Type	Name	Role	Symbol
$\langle 0, T \rangle \rightarrow \mathbb{R}^n$	Trajectory	Behaviour of a polyhedral system	f
$\langle 0, T \rangle \rightarrow 2^{AP}$	Signal	Interpretation of RTL_f	σ
$\{0, 1, \dots, k\} \rightarrow 2^{AP}$	Trace	Interpretation of LTL_f	w
$\{0, 1, \dots, k\} \rightarrow S$	Discrete run	Behaviour of a finite automaton	r^d
$\langle 0, T \rangle \rightarrow S$	Continuous run	Continuous behaviour of a finite automaton	r^c
$\langle 0, T \rangle \rightarrow (\mathbb{R}^n \times S)$	Hybrid run	Pairing of a trajectory and a continuous run	ρ
$\{0, 1, \dots, k\} \rightarrow \mathbb{R}^+$	Time-slicing	Time decomposition of a signal to generate traces	τ

3 Linear Temporal Logics

Linear Temporal Logic (LTL) was introduced by Pnueli to specify and verify properties of reactive systems [19]. Given a set of atomic propositions AP , an LTL formula is composed of atomic propositions, the Boolean connectives *conjunction* (\wedge) and *negation* (\neg), and the temporal operators *next* (X) and two flavors of *until*: strict (\dot{U}) and non-strict (U).

LTL formulae are built up in the usual way from the above operators and connectives, according to the following grammar:

$$\varphi := p \mid \neg \varphi \mid \varphi \wedge \varphi \mid X\varphi \mid \varphi U \varphi \mid \varphi \dot{U} \varphi,$$

where p is an atomic proposition in AP . We denote by $|\varphi|$ the length of formula φ .

The semantics of LTL is typically given *w.r.t.* infinite sequences (*i.e.*, words) of sets of atomic propositions in AP , *a.k.a.* *discrete traces*, to capture properties of discrete infinite computations. Since we are interested in the verification of continuous systems, we shall also consider a semantics based on signals, in a similar vein to some previous works [22]. In Section 5, we describe how the discrete and the continuous semantics are related, a connection that we leverage to reduce verification of continuous properties to a combination of verification of discrete properties and geometric reasoning. Both for the discrete and the continuous version, we consider the bounded semantic fragments, where formulae are interpreted over finite words and bounded signals, respectively.

Discrete Semantics. In this paper, we consider the semantic fragment LTL_f [10], where formulae are interpreted over non-empty finite words $w = w_0w_1 \dots w_n$ of symbols in the alphabet $\Sigma = 2^{AP}$. For all $i = 0, \dots, n$, we denote by $w_{\geq i}$ the suffix of w starting from w_i . The satisfaction relation $w \models \varphi$ is defined as follows:

- $w \models \varphi$, for $\varphi \in AP$, if and only if $\varphi \in w_0$;
- $w \models \neg \varphi$ if and only if $w \not\models \varphi$ does not hold;
- $w \models \varphi_1 \wedge \varphi_2$ if and only if $w \models \varphi_1$ and $w \models \varphi_2$;
- $w \models X\varphi$ if and only if $|w| > 1$ and $w_{\geq 1} \models \varphi$;
- $w \models \varphi_1 U \varphi_2$ if and only if there exists $i \geq 0$ such that $w_{\geq i} \models \varphi_2$ and, for all j such that $0 \leq j < i$, it holds $w_{\geq j} \models \varphi_1$;
- $w \models \varphi_1 \dot{U} \varphi_2$ if and only if there exists $i > 0$ such that $w_{\geq i} \models \varphi_2$ and, for all j such that $0 < j < i$, it holds $w_{\geq j} \models \varphi_1$.

► **Theorem 5** ([10]). *For all LTL_f formulae φ there exists a finite automaton \mathcal{A}_φ that accepts all and only the models of φ .*

Continuous Semantics. As it is customary, RTL_f , the continuous version of LTL_f , is identified as the fragment without the next-time operator X . The semantics of RTL_f formulae is given with respect to signals $\sigma: \langle 0, T \rangle \rightarrow 2^{AP}$ in the following way:

- $\sigma \models \varphi$, for $\varphi \in AP$, if and only if:
 - $\varphi \in \sigma(0)$, if σ is left-closed, and
 - there exists $t' \in \langle 0, T \rangle$ such that $\varphi \in \sigma(t')$, for all $t'' \in (0, t')$, otherwise;
- $\sigma \models \varphi_1 \cup \varphi_2$ if and only if there exists $t \in \langle 0, T \rangle$ such that $\sigma_{\geq t} \models \varphi_2$ and $\sigma_{\geq t'} \models \varphi_1$, for all $t' \in \langle 0, T \rangle$ with $t' < t$;
- $\sigma \models \varphi_1 \dot{\cup} \varphi_2$ if and only if there exists $0 < t \leq T$ such that $\sigma_{\geq t} \models \varphi_2$ and $\sigma_{\geq t'} \models \varphi_1$, for all $0 < t' < t$.

While the base case for left-closed signals is standard, we stipulate that a left-open signal satisfies an atomic proposition $p \in AP$ if there exists an initial left-open interval contained in the domain of the signal, where p is observed.

Note that on a left-open signal the semantics of the operators \cup and $\dot{\cup}$ coincide. Moreover, unlike in LTL_f where the operators $\dot{\cup}$ and \cup are inter-derivable thanks to the presence of the operator X , in RTL_f this is not the case and $\dot{\cup}$ turns out to be strictly more expressive than \cup (a proof of this result can be found in [21]). Indeed, in both LTL_f and RTL_f , we have that $\varphi_1 \cup \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \varphi_1 \dot{\cup} \varphi_2)$ and in LTL_f only it holds, in addition, that $\varphi_1 \dot{\cup} \varphi_2 \equiv X(\varphi_1 \cup \varphi_2)$. The semantics of RTL_f essentially corresponds to a bounded version of the logic by the same name from [22], except that we consider both left-open and left-closed signals and we omit the past operator *Since*.

The Problem. In this work we are interested in solving the problem of computing the *existential denotation* of an RTL_f formula defined as follows.

► **Definition 6.** *Given an RTL_f formula φ and a polyhedral system \mathcal{P} on the same set of atomic propositions, the existential denotation of φ on \mathcal{P} is the set of points of \mathbb{R}^n from which there exists a well-behaved trajectory whose signal satisfies φ .*

Note that a solution to the existential denotation problem also allows us to solve the *model-checking problem*, where we ask whether a given point $x \in \mathbb{R}^n$ is the source of some trajectory in \mathcal{P} whose signal satisfies the formula.

4 Discretisation

To address the model-checking problem for RTL_f , we reduce it to a suitable decision problem for the discrete version of the logic. Specifically, we show that, for all RTL_f formulae φ on a set of atomic propositions AP , there exists an LTL_f formula $\text{dsc}(\varphi)$ on the extended set $AP \cup \{\text{sing}\}$ such that a signal σ satisfies φ iff the discrete traces induced by σ satisfy $\text{dsc}(\varphi)$. This result is proved at the end of this section as Theorem 11. First, we need to define and characterise the discrete versions of signals (Section 4.1) and formulae (Section 4.2).

4.1 Discretising Signals

Recall from Section 2.1 that a time-slicing $\tau = \{t_i\}_{i=0}^k \in TS(\sigma)$ of a signal σ decomposes σ into a finite sequence of slices corresponding to an alternation of singular and open time intervals. Introduce the function $\text{slice}_\tau^\sigma: [0, t_k] \rightarrow \{0, \dots, 2k\}$, associating each time instant $t \in [0, t_k]$ with the index of its slice $\text{slice}_\tau^\sigma(t)$. Formally:

$$\text{slice}_\sigma^\tau(t) = \begin{cases} 2i, & \text{if } t = t_i; \\ 2i + 1, & \text{if } t \in (t_i, t_{i+1}). \end{cases}$$

Given a time-slicing τ of a signal σ , we now define the discrete trace $\text{trace}(\sigma, \tau)$ by lumping together in a single object the time instants of each open interval (t_i, t_{i+1}) and inserting between any two such intervals the observables of the singular time point separating them. We maintain the distinction between open and singular intervals by means of an accessory atomic proposition *sing* that holds true in all and only the time points (*i.e.*, singular intervals) t_i of the time-slicing τ . Denote again by α_i the set of observables holding true in the open interval (t_i, t_{i+1}) . The discretisation $\text{trace}(\sigma, \tau)$ is the finite word defined below for both left-closed and left-open signals. We use $\text{trace}(\sigma, \tau)_i \subseteq AP \cup \{\text{sing}\}$ to denote the i -th symbol of the discrete trace. Formally, for a left-closed signal $\sigma: [0, T] \rightarrow 2^{AP}$ and an index $j \in \{0, \dots, 2k\}$, we set:

$$\text{trace}(\sigma, \tau)_j \triangleq \begin{cases} \sigma(t_i) \cup \{\text{sing}\}, & \text{if } j \text{ is even and } i = j/2; \\ \alpha_i, & \text{if } j \text{ is odd and } i = (j-1)/2. \end{cases}$$

For a left-open signal $\sigma: (0, T] \rightarrow 2^{AP}$ and an index $j \in \{0, \dots, 2k-1\}$, we set:

$$\text{trace}(\sigma, \tau)_j \triangleq \begin{cases} \alpha_i, & \text{if } j \text{ is even and } i = j/2; \\ \sigma(t_i) \cup \{\text{sing}\}, & \text{if } j \text{ is odd } i = (j+1)/2. \end{cases}$$

Before continuing with the discretisation of the specification, we state a commutativity property enjoyed by the composition of the discretisation function with the suffix operation on signals, time-slicings, and traces. In particular, for some $t \leq T$, we define $(\{t_i\}_{i=0}^k)_{\geq t} \triangleq \{t'_i\}_{i=0}^{k'}$, with $k' \triangleq k-l$, $t'_0 \triangleq 0$, and $t'_i \triangleq t_{i+l} - t$, for all $i \in \{1, \dots, k'\}$, where $l \in \{0, 1, \dots, k\}$ is the maximum index such that $t_l \leq t$, which also corresponds to $\lfloor \frac{\text{slice}_\sigma^\tau(t)}{2} \rfloor$. Note that, if $\tau \in TS(\sigma)$, then $\tau_{\geq t} \in TS(\sigma_{\geq t}) = TS(\sigma_{>t})$.

► **Lemma 7.** *Let $\sigma: \langle 0, T \rangle \rightarrow 2^{AP}$ be a signal, $\tau \in TS(\sigma)$ one of its time-slicings, $t \in \langle 0, T \rangle$ a time instant in the signal domain, and $h = \text{slice}_\sigma^\tau(t)$ the corresponding slice index. Then, it holds true that:*

- $\text{trace}(\sigma, \tau)_{\geq h} = \text{trace}(\sigma_{\geq t}, \tau_{\geq t})$, if $\text{sing} \in \text{trace}(\sigma, \tau)_h$;
- $\text{trace}(\sigma, \tau)_{\geq h} = \text{trace}(\sigma_{>t}, \tau_{\geq t})$, otherwise.

In addition, it is immediate to see that a trace of a signal satisfies the following property concerning the auxiliary *sing* atomic proposition. In words, (a) singular and open intervals alternate throughout the trace, (b) the trace must end in a singular interval, and (c) the trace starts in a singular interval *iff* the underlying signal is left-closed.

► **Proposition 8.** *For a signal $\sigma: \langle 0, T \rangle \rightarrow 2^{AP}$ and a time-slicing $\tau \in TS(\sigma)$, it holds that $\text{trace}(\sigma, \tau) \models \mathbf{G}((\text{sing} \leftrightarrow \mathbf{X}\neg\text{sing}) \vee \text{last}) \wedge \mathbf{F}(\text{last} \wedge \text{sing})$, where $\text{last} \triangleq \neg\mathbf{X}\top$. Moreover, σ is left-closed *iff* $\text{trace}(\sigma, \tau) \models \text{sing}$.*

4.2 Discretising Formulae

We can now introduce the required transformation from RTL_f to LTL_f . Intuitively, this translation exploits the segmentation induced by a time-slicing of a signal to verify whether the observable changes along the signal actually satisfy the property prescribed by the RTL_f formula. Formally, we set the following:

$$\begin{aligned}
\text{dsc}(p) &\triangleq p \\
\text{dsc}(\neg\varphi) &\triangleq \neg\text{dsc}(\varphi), \\
\text{dsc}(\varphi_1 \wedge \varphi_2) &\triangleq \text{dsc}(\varphi_1) \wedge \text{dsc}(\varphi_2), \\
\text{dsc}(\varphi_1 \cup \varphi_2) &\triangleq \text{dsc}(\varphi_1) \cup (\text{dsc}(\varphi_2) \wedge (\text{dsc}(\varphi_1) \vee \text{sing})), \\
\text{dsc}(\varphi_1 \dot{\cup} \varphi_2) &\triangleq (\text{sing} \wedge \text{X dsc}(\varphi_1 \cup \varphi_2)) \vee (\neg\text{sing} \wedge \text{dsc}(\varphi_1 \cup \varphi_2)).
\end{aligned}$$

To prove the correctness of the above transformation, we first need to state two properties enjoyed by the semantics of RTL_f . In the following, we say that a signal $\sigma: \langle 0, T \rangle \rightarrow 2^{AP}$ is B -uniform, for an interval $B \subseteq \langle 0, T \rangle$, if $\sigma(t) = \sigma(t')$, for all $t, t' \in B$.

► **Lemma 9.** *For all RTL_f formulae φ , signals $\sigma: \langle 0, T \rangle \rightarrow 2^{AP}$, and open intervals $B \subseteq \langle 0, T \rangle$ such that σ is B -uniform, the following holds true: $\sigma_{\sim_1 t_1} \models \varphi$ iff $\sigma_{\sim_2 t_2} \models \varphi$, for all $t_1, t_2 \in B$ and $\sim_1, \sim_2 \in \{\geq, >\}$.*

Proof. The proof proceeds by structural induction on the RTL_f formula φ .

- **[Base case $\varphi = p \in AP$].** *W.l.o.g.*, let us assume $\sigma_{\sim_1 t_1} \models \varphi$. It is easy to see that there necessarily exists $t \in B$ such that $p \in \sigma(t)$. Indeed, if $\sim_1 = \geq$, by the semantics of atomic propositions on left-closed signals, we can choose $t = t_1$, since $p \in \sigma_{\geq t_1}(0) = \sigma(t_1)$. If, $\sim_1 = >$, instead, again by the semantics of atomic propositions, this time for left-open signals, there exists a non-empty open interval $(t_1, t') \subseteq (t_1, T]$ such that $p \in \sigma_{> t_1}(t'' - t_1) = \sigma(t'')$, for all $t'' \in (t_1, t')$. Since, by hypothesis, B is a non-empty open interval with $t_1 \in B$, the intersection $B \cap (t_1, t')$ is non-empty as well. Therefore, we can arbitrarily choose t as an element of this intersection. At this point, consider the left-closed subinterval $C \triangleq [t_2, \text{sup}(B))$ of B . Due to the B -uniformity of the signal σ , it holds that $p \in \sigma(t') = \sigma(t)$, for all $t' \in C$. Hence, by using C as witness, it is immediate to show that $\sigma_{\sim_2 t_2} \models \varphi$, independently from the specific relation \sim_2 .
- **[Inductive case].** The Boolean operators \neg and \wedge are trivial to deal with, so we focus on the strict until operator only, *i.e.*, we consider the case $\varphi = \varphi_1 \dot{\cup} \varphi_2$. *W.l.o.g.*, let us assume $\sigma_{\sim_1 t_1} \models \varphi$. Independently from the relation \sim_1 , by definition of the semantics of the temporal operator $\dot{\cup}$, there exists $t \in (t_1, T]$ such that $\sigma_{\geq t} \models \varphi_2$ and $\sigma_{\geq t'} \models \varphi_1$, for all $t' \in (t_1, t)$. Since, by hypothesis, B is an open interval and $t_1 \in B$, the intersection $B \cap (t_1, t)$ is necessarily non-empty. Thus, there exists an instant $t' \in B$ such that $\sigma_{\geq t'} \models \varphi_1$. So, by the inductive hypothesis applied to the formula φ_1 , it holds that $\sigma_{\geq t'} \models \varphi_1$, for all $t' \in B$. Now, two cases may arise depending on whether t belongs to B as well.
 - **[$t < \text{sup}(B)$].** Since $\sigma_{\geq t} \models \varphi_2$, by the inductive hypothesis applied to the formula φ_2 , it holds that $\sigma_{\geq t'} \models \varphi_2$, for all $t' \in B$. Then, as an immediate consequence, any $t \in (t_2, \text{sup}(B))$ satisfies $\sigma_{\geq t} \models \varphi_2$ and $\sigma_{\geq t'} \models \varphi_1$, for all $t' \in (t_2, t)$. Hence, $\sigma_{\sim_2 t_2} \models \varphi$ holds, independently from the specific relation \sim_2 .
 - **[$t \geq \text{sup}(B)$].** Since $t_2 \in B$, it holds that $t_2 < t$. Hence, to prove that $\sigma_{\sim_2 t_2} \models \varphi$, it only remains to show that $\sigma_{\geq t'} \models \varphi_1$, for all $t' \in (t_2, t)$. Obviously, the open interval (t_2, t) can be decomposed into the disjoint union of the two adjacent intervals $(t_2, \text{sup}(B))$ and $[\text{sup}(B), t)$. At this point, the required property clearly follows from the fact that $(t_2, \text{sup}(B)) \subset B$ and $[\text{sup}(B), t) \subset (t_1, t)$, as φ_1 holds true on all points of these two intervals. ◀

► **Lemma 10.** *For all RTL_f formulae φ , signals $\sigma: \langle 0, T \rangle \rightarrow 2^{AP}$, and time instants $t \in \langle 0, T \rangle$, the following holds true: $\sigma_{> t} \models \varphi$ iff there exists a time instant $t' \in (t, T]$ such that $\sigma_{\geq t'} \models \varphi$, for all $t'' \in (t, t')$.*

The following theorem, which leverages the above two lemmas, establishes the correctness of the discretisation and allows us in the next section to reduce verification of RTL_f properties against signals to verification of LTL_f properties against discrete traces.

► **Theorem 11.** *For all RTL_f formulae φ , signals σ , and time-slicings $\tau \in \text{TS}(\sigma)$, it holds that $\sigma \models \varphi$ iff $\text{trace}(\sigma, \tau) \models \text{dsc}(\varphi)$.*

Proof. The proof proceeds by structural induction on the formula, where we consider an arbitrary time-slicing $\tau = \{t_i\}_{0 \leq i \leq k}$ of σ .

- **[Base case $\varphi = p \in \mathbf{AP}$].** For the base case, we distinguish the two cases of left-closed and left-open signals. If σ is left-closed, then, by definition of $\text{trace}(\sigma, \tau)$, it holds that $\text{trace}(\sigma, \tau)_0 = \sigma(0) \cup \{\text{sing}\}$. Hence, being $\text{dsc}(p) = p$, we have $\sigma \models p$ iff $\text{trace}(\sigma, \tau) \models \text{dsc}(p)$. If, on the other hand, σ is left-open, then $\text{trace}(\sigma, \tau)_0 = \sigma(t)$, for every $t \in (0, t_1)$, since, by definition of time-slicing of σ , the observables are constant in each open interval (t_i, t_{i+1}) . Now, $\sigma \models p$ iff $p \in \sigma(t)$, for all $t \in (0, t_1)$. It immediately follows, then, that $\sigma \models p$ iff $\text{trace}(\sigma, \tau) \models \text{dsc}(p)$.
- **[Inductive case].** We shall focus on the inductive case where $\varphi = \varphi_1 \dot{\cup} \varphi_2$, since the cases of the Boolean operators are trivial, while the case for \cup is essentially a simplified version of $\dot{\cup}$. In the following, let $\zeta \triangleq \text{dsc}(\varphi_1 \cup \varphi_2)$.

For the first direction of the equivalence, let us consider the case of a left-closed signal $\sigma: [0, T] \rightarrow 2^{AP}$ and assume $\sigma \models \varphi_1 \dot{\cup} \varphi_2$. Then, by the semantics, there is a $\bar{t} \in (0, T]$ with $\sigma_{\geq \bar{t}} \models \varphi_2$ and, for all $t' \in (0, \bar{t})$, it holds $\sigma_{\geq t'} \models \varphi_1$. Being σ left-closed, it holds that $\text{sing} \in \text{trace}(\sigma, \tau)_0$, hence, we only need to show that $\text{trace}(\sigma, \tau)_{\geq 0} \models \mathbf{X}\zeta$, i.e., $\text{trace}(\sigma, \tau)_{\geq 1} \models \text{dsc}(\varphi_1) \cup (\text{dsc}(\varphi_2) \wedge (\text{dsc}(\varphi_1) \vee \text{sing}))$, by definition of $\text{dsc}(\varphi_1 \cup \varphi_2)$. We have two more cases, depending on whether \bar{t} belongs to the time-slicing τ or is contained in one of its open intervals. If $\text{slice}_\sigma^\tau(\bar{t})$ belongs to $\{t_i\}_{0 \leq i \leq k}$, let \bar{j} be the position in the discrete trace corresponding to the instant \bar{t} , i.e., $\bar{j} \triangleq \text{slice}_\sigma^\tau(\bar{t}) > 0$. Then, $\text{sing} \in \text{trace}(\sigma, \tau)_{\bar{j}}$ and $\text{trace}(\sigma_{\geq \bar{t}}, \tau_{\geq \bar{t}}) = \text{trace}(\sigma, \tau)_{\geq \bar{j}}$, by Lemma 7. By the inductive hypothesis, $\text{trace}(\sigma_{\geq \bar{t}}, \tau_{\geq \bar{t}}) \models \text{dsc}(\varphi_2)$ and, hence, we obtain $\text{trace}(\sigma, \tau)_{\geq \bar{j}} \models \text{dsc}(\varphi_2) \wedge \text{sing}$. If, on the other hand, $\bar{t} \in (t_i, t_{i+1})$, for some index $0 \leq i < k$, then σ is clearly \mathbf{B} -uniform, if we take $\mathbf{B} = (t_i, \bar{t}] \subset (t_i, t_{i+1})$. Hence, by Lemma 9, we have $\sigma_{\geq t} \models \varphi_2$, for all $t \in \mathbf{B}$ and, by Lemma 10 and the fact that $\inf(\mathbf{B}) = t_i$, we conclude $\sigma_{> t_i} \models \varphi_2$. Taking $\bar{j} \triangleq \text{slice}_\sigma^\tau(\bar{t}) = \text{slice}_\sigma^\tau(t_i) + 1$, we have that $\text{sing} \notin \text{trace}(\sigma, \tau)_{\bar{j}}$ and $\text{trace}(\sigma_{> t_i}, \tau_{\geq t_i}) = \text{trace}(\sigma, \tau)_{\geq \bar{j}}$, by Lemma 7. By applying the inductive hypothesis, we obtain $\text{trace}(\sigma_{> t_i}, \tau_{\geq t_i}) \models \text{dsc}(\varphi_2)$. In this case, we know from the assumption that $\sigma_{\geq t'} \models \varphi_1$, for all $t_i < t' < \bar{t}$. Then, by applying again Lemma 9 and Lemma 10, we obtain that $\text{trace}(\sigma_{> t_i}, \tau_{\geq t_i}) \models \text{dsc}(\varphi_1)$. Thus, we can conclude $\text{trace}(\sigma, \tau)_{\geq \bar{j}} \models \text{dsc}(\varphi_2) \wedge \text{dsc}(\varphi_1)$. Regardless of the case, we have obtained that $\text{trace}(\sigma, \tau)_{\geq \bar{j}} \models \text{dsc}(\varphi_2) \wedge (\text{dsc}(\varphi_1) \vee \text{sing})$. Let us now consider any $t' \in \mathbf{J}$, where $\mathbf{J} = (0, t_i)$, if $\bar{t} = t_i$, and $\mathbf{J} = (0, t_i]$, if $\bar{t} \in (t_i, t_{i+1})$, for some index $0 \leq i < k$. We have two cases, depending on whether $t' = t_j$ or $t' \in (t_j, t_{j+1})$, for some $0 < j < i$. By applying to t' the same reasoning we applied to \bar{t} above, using the inductive hypothesis and Lemmas 7, 9, and 10, we obtain that $\text{trace}(\sigma, \tau)_{\geq j} \models \text{dsc}(\varphi_1)$, for $j = \text{slice}_\sigma^\tau(t')$. Since, in addition, $\{\text{slice}_\sigma^\tau(t') \mid t' \in \mathbf{J}\} = \{1, \dots, \bar{j} - 1\}$, we can conclude that $\text{trace}(\sigma, \tau)_{\geq j} \models \text{dsc}(\varphi_1)$, for all $1 \leq j < \bar{j}$. Together with $\text{trace}(\sigma, \tau)_{\geq \bar{j}} \models \text{dsc}(\varphi_2) \wedge (\text{dsc}(\varphi_1) \vee \text{sing})$, this gives us $\text{trace}(\sigma, \tau)_{\geq 1} \models \text{dsc}(\varphi_1) \dot{\cup} (\text{dsc}(\varphi_2) \wedge (\text{dsc}(\varphi_1) \vee \text{sing}))$, which, in turn, implies $\text{trace}(\sigma, \tau) \models \text{sing} \wedge \mathbf{X}(\text{dsc}(\varphi_1) \dot{\cup} \text{dsc}(\varphi_2) \wedge (\text{dsc}(\varphi_1) \vee \text{sing}))$, since $\text{sing} \in \text{trace}(\sigma, \tau)_0$ in this case. Hence, $\text{trace}(\sigma, \tau) \models \text{sing} \wedge \mathbf{X}\zeta$ and, finally, $\text{trace}(\sigma, \tau) \models \text{dsc}(\varphi_1 \dot{\cup} \varphi_2)$ as required.

For the other direction of the equivalence, assume $trace(\sigma, \tau) \models \mathbf{dsc}(\varphi_1 \dot{\cup} \varphi_2)$. Since we are considering a left-closed signal σ , we have $sing \in trace(\sigma, \tau)_0$ and, therefore, $trace(\sigma, \tau) \models (sing \wedge \mathbf{X}\zeta)$. As a consequence, $trace(\sigma, \tau)_{\geq 1} \models \mathbf{dsc}(\varphi_1) \cup (\mathbf{dsc}(\varphi_2) \wedge (\mathbf{dsc}(\varphi_1) \vee sing))$. By the semantics of \cup , there exists an index $j \geq 1$ such that $trace(\sigma, \tau)_{\geq j} \models (\mathbf{dsc}(\varphi_2) \wedge (\mathbf{dsc}(\varphi_1) \vee sing))$ and $trace(\sigma, \tau)_{\geq z} \models \mathbf{dsc}(\varphi_1)$, for all $1 \leq z < j$. We have two cases, depending on whether $trace(\sigma, \tau)_j$ contains the proposition $sing$ or not. If $sing \in trace(\sigma, \tau)_j$, then $trace(\sigma, \tau)_{\geq j} \models \mathbf{dsc}(\varphi_2) \wedge sing$ and $j = slice_\sigma^\tau(t_i)$, for some $0 < i \leq k$. By Lemma 7, $trace(\sigma, \tau)_{\geq j} = trace(\sigma_{\geq t_i}, \tau_{\geq t_i})$. Therefore, $trace(\sigma_{\geq t_i}, \tau_{\geq t_i}) \models \mathbf{dsc}(\varphi_2)$. By the inductive hypothesis, then, $\sigma_{\geq t_i} \models \varphi_2$. For the other case, $sing \notin trace(\sigma, \tau)_j$, hence, $trace(\sigma, \tau)_{\geq j} \models \mathbf{dsc}(\varphi_2) \wedge \mathbf{dsc}(\varphi_2)$ and $j = slice_\sigma^\tau(t)$, for all $t \in (t_i, t_{i+1})$ and some $0 \leq i < k$. For all such t , then, we obtain $trace(\sigma, \tau)_{\geq j} = trace(\sigma_{> t}, \tau_{\geq t})$, thanks to Lemma 7 and, then, also $trace(\sigma_{> t}, \tau_{\geq t}) \models \mathbf{dsc}(\varphi_2) \wedge \mathbf{dsc}(\varphi_2)$. By the inductive hypothesis, it holds $\sigma_{> t} \models \varphi_2 \wedge \varphi_2$, for each such t . Lemma 9, then, gives us $\sigma_{\geq t} \models \varphi_2 \wedge \varphi_2$, for all $t \in (t_i, t_{i+1})$. Now, take any $t' \in J$, where $J = (0, t_i)$, if $sing \in trace(\sigma, \tau)_j$, and $J = (0, t_i]$, otherwise. Clearly, $slice_\sigma^\tau(t') \in \{1, \dots, j-1\}$ and we have two cases, depending on whether t' is an element of τ or lies in one of its open intervals. In the first case, let $t_z \triangleq t'$ and $z \triangleq slice_\sigma^\tau(t_z) < j$. Since $trace(\sigma, \tau)_{\geq z} \models \mathbf{dsc}(\varphi_1)$ and, by Lemma 7, $trace(\sigma, \tau)_{\geq z} = trace(\sigma_{\geq t_z}, \tau_{\geq t_z})$, we conclude $trace(\sigma_{\geq t_z}, \tau_{\geq t_z}) \models \mathbf{dsc}(\varphi_1)$ and, by the inductive hypothesis, also $\sigma_{\geq t_z} \models \varphi_1$. If, on the other hand, $t' \in (t_l, t_{l+1})$, for some l , let us set $z \triangleq slice_\sigma^\tau(t') < j$. Lemma 7 in this case gives us $trace(\sigma, \tau)_{\geq z} = trace(\sigma_{> t'}, \tau_{\geq t'})$. We know that $trace(\sigma, \tau)_{\geq z} \models \mathbf{dsc}(\varphi_1)$, hence, $trace(\sigma_{> t'}, \tau_{\geq t'}) \models \mathbf{dsc}(\varphi_1)$. By the inductive hypothesis, $\sigma_{> t'} \models \varphi_1$ and, by Lemma 9, also $\sigma_{\geq t'} \models \varphi_1$. Putting everything together, we have shown that there is a time $t \in (0, T]$ such that $\sigma_{\geq t} \models \varphi_2$ and $\sigma_{\geq t'} \models \varphi_1$, for all $t' \in (0, t]$, which coincides with the semantic condition for $\sigma \models \varphi_1 \dot{\cup} \varphi_2$.

The proof of the inductive case when the signal σ is left-open is essentially the same, except that the first letter $trace(\sigma, \tau)_0$ of $trace(\sigma, \tau)$ does not contain $sing$, as it corresponds to the first open interval (t_0, t_1) of the time-slicing, and that $\mathbf{dsc}(\varphi_1 \dot{\cup} \varphi_2)$ reduces to $\neg sing \wedge \zeta$ in this case. \blacktriangleleft

In conclusion, as an immediate corollary of the above result, we have the following theorem, where with every RTL_f formula φ we associate the LTL_f formula

$$\widehat{\varphi} \triangleq \mathbf{dsc}(\varphi) \wedge sing \wedge \mathbf{G}((sing \leftrightarrow \mathbf{X}\neg sing) \vee last) \wedge \mathbf{F}(last \wedge sing). \quad (1)$$

► **Theorem 12.** *For all RTL_f formulae φ , left-closed signals σ , and time-slicings $\tau \in TS(\sigma)$, it holds that $\sigma \models \varphi$ iff $trace(\sigma, \tau) \models \widehat{\varphi}$.*

5 Model Checking RTL_f on Polyhedral Systems

In this section, we describe the algorithm that solves the existential denotation problem for RTL_f on polyhedral systems. Unless differently specified, we consider a fixed RTL_f formula φ over the set of atomic propositions AP , and a fixed polyhedral system \mathcal{P} on AP . Before describing the algorithm itself, we introduce two auxiliary operators on polyhedra.

5.1 The Basic Operators

The algorithm presented in Section 5.3 (Algorithm 1) requires a function $reach^b(A, B)$ that takes as arguments a possibly non-convex polyhedron A and a convex polyhedron B , and identifies the set of points of A that can reach B while staying in $A \cup B$. The b superscript

19:12 Model Checking Linear Temporal Properties on Polyhedral Systems

can be either 0 or +, corresponding to different timing constraints: a point from A belongs to $reach^0(A, B)$ if it can *immediately* enter into B , whereas it belongs to $reach^+(A, B)$ if it can enter into B after a positive delay. Formally, for all polyhedra A and convex polyhedra B :

$$\begin{aligned} reach^0(A, B) &\triangleq \{x \in A \mid \exists f \in Traj_{wb}(x), t > 0. \forall t' \in (0, t] : f(t') \in B\}; \\ reach^+(A, B) &\triangleq \{x \in A \mid \exists f \in Traj_{wb}(x), t > 0 : f(t) \in B \text{ and } \forall t' \in (0, t) : f(t') \in A\}. \end{aligned}$$

Moreover, we need to split the result of $reach^b(A, B)$, which is a general polyhedron, into convex polyhedra, each contained in one of the patches of A . To this aim, we introduce the following *split* function. For all polyhedra A and $A' \subseteq A$, the function $split(A', A)$ returns a set of pairs $\{(P_i, X_i)\}_{i=1, \dots, n}$ such that: (i) P_i and X_i are convex polyhedra such that $X_i \subseteq P_i$, (ii) each P_i is one of the patches of A , and (iii) A' is the union of the X_i 's. It is straightforward to implement the function *split* using Boolean operations on polyhedra.

Computing the reach operators. We now show how to compute the value of $reach^b$ with a finite number of geometric operations. First, define the *positive pre-flow* $P_{\swarrow_{>0}}$ of a convex polyhedron P as the set of points that can reach P after a positive delay. Formally:

$$P_{\swarrow_{>0}} \triangleq \{x \in \mathbb{R}^n \mid \exists d \in Flow, t > 0 : x + d \cdot t \in P\}.$$

Lemma 13 below deals with $reach^0$, whereas Lemma 14 provides an algorithm for $reach^+$. Their proofs can be found in Appendix A.

► **Lemma 13.** *For all polyhedra A and convex polyhedra B the following holds:*

$$reach^0(A, B) = A \cap cl(B) \cap B_{\swarrow_{>0}}.$$

When it comes to computing $reach^+$, we shall make use of the *May Reach While Avoiding* operator $RWA^m(Y, Z)$, that collects the points from which some admissible trajectory can reach a point in the set Y while avoiding all the points in the set Z . The operator is formally defined as follows:

$$RWA^m(Y, Z) \triangleq \{x \in \mathbb{R}^n \mid \exists f \in Traj_{wb}(x), t \geq 0 : f(t) \in Y \text{ and } \forall t' \in [0, t) : f(t') \in Y \cup \overline{Z}\}.$$

An algorithm for computing RWA^m using symbolic operations on polyhedra is presented in [7]. The following lemma formalises the connection between RWA^m and $reach^+$.

► **Lemma 14.** *For all polyhedra A and convex polyhedra B the following holds:*

$$reach^+(A, B) = \bigcup_{P \in Patch(A)} RWA^m(T_P, \overline{A}), \quad \text{where } T_P \triangleq P \cap (cl(P) \cap B)_{\swarrow_{>0}}.$$

As far as the *computational complexity* is concerned, first notice that the implementation of the algorithm is based on symbolic operations on polyhedra, whose complexity is already exponential in the worst case. A loose measure of complexity can be obtained by counting the number of symbolic operations involved.

The operator $reach^0$ involves a constant number of geometric operations, specifically intersections of polyhedra, closure operations and positive time-elapse [5]. The computation of $reach^+(A, B)$, instead, requires at most $|Patch(A)|$ calls to RWA^m . An analysis of the algorithm for RWA^m (see Theorem 3 in [7]) shows that computing $RWA^m(Y, \overline{Z})$ requires at most $k \cdot m^{O(m)}$ symbolic operations, where k and m are, respectively, the number of convex patches of Y and Z . The analysis also shows that the number of patches of the output cannot exceed $m^{O(m)}$. Summarising, $reach^+(A, B)$ requires at most $m^{O(m)}$ operations, with m the number of convex patches of A , since B is a single patch, and its output contains at most $m^{O(m)}$ patches.

5.2 The Finite Automaton

Our algorithm works on a finite automaton that checks the satisfaction of φ , while ensuring a number of extra properties. The automaton is obtained by applying the classic LTL_f-to-automata construction to the formula $\hat{\varphi}$ defined in (1).

Let $\mathcal{A}_{\hat{\varphi}} = (S, \delta, \lambda, S_0, S_F)$ be the finite automaton corresponding to $\hat{\varphi}$, according to Theorem 5. Recall that λ labels each state in S with a subset of $\widehat{AP} = AP \cup \{\text{sing}\}$. For convenience, we write $\llbracket s \rrbracket$ for $\llbracket \lambda(s) \rrbracket$ to denote the polyhedron interpreting the set of propositions labelling s .

We assume w.l.o.g. that $\mathcal{A}_{\hat{\varphi}}$ satisfies the following properties. Properties (a) and (c) are directly encoded in $\hat{\varphi}$, while property (b) is enforced via a simple modification of the automaton.

► **Proposition 15.** *The finite automaton $\mathcal{A}_{\hat{\varphi}}$ satisfies the following properties:*

- (a) *The initial states are labelled with sing .*
- (b) *The initial states have no predecessors.*
- (c) *The underlying graph is bipartite in $(S_{\text{sing}}, S_{\text{open}})$, where S_{sing} is the set of all states labelled with sing , while S_{open} is its complement.*

We denote by $\text{Run}^d(f)$ the set of all initial runs of $\mathcal{A}_{\hat{\varphi}}$ on the discrete traces of f . For a trajectory f and a time slicing $\tau = \{t_i\}_{i=0}^k \in \text{TS}(\sigma_f)$, let w be the corresponding discrete trace and r^d one of the runs of $\mathcal{A}_{\hat{\varphi}}$ on w . We define the *continuous run* r^c for r^d and τ as follows:

$$r^c(t) = \begin{cases} r^d(2 \cdot i) & \text{if } t = t_i, \text{ for some } i, \\ r^d(2 \cdot i + 1) & \text{if } t \in (t_i, t_{i+1}), \text{ for some } i. \end{cases}$$

We denote with $\text{Run}^c(f)$ the set of continuous runs induced by f as just described.

Moreover, we define the notion of *hybrid run* as the function $\rho = \lambda t . (f(t), r^c(t))$ pairing a trajectory with one of its continuous runs. Let $\text{HRun}(x)$ be the set of hybrid runs (f, r^c) , where $f \in \text{Traj}_{\text{wb}}(x)$ and $r^c \in \text{Run}^c(f)$.

5.3 The Algorithm

We now describe the main step in the procedure to solve the existential denotation problem, expressed in pseudo-code as the function $\exists\text{Denot}(\cdot)$ in Algorithm 1. Theorem 18 at the end of this section describes the top-level invocations that start the process, which begins from a final state of the automaton and then works recursively backward towards the initial states.

Roughly speaking, a call to $\exists\text{Denot}(s, P, X, V)$ computes the points from where there exists a hybrid run of the automaton ending in the state s and in a point in the convex polyhedron X . Moreover, X is assumed to be contained in P , and P must be a patch of $\llbracket s \rrbracket$. The role of the parameter V is explained below. In the following, for a state $s \in S$, let $\text{type}(s) = 0$, if $\text{sing} \in \lambda(s)$, and $\text{type}(s) = +$, otherwise.

To ensure termination, the algorithm keeps track of the patches associated with *open states* in S_{open} that have been visited in the current sequence of recursive calls. Those are the patches in which the induced trajectory must spend some positive amount of time. This information is kept in the map V , that associates with each state s the set of patches of $\llbracket s \rrbracket$ already encountered by the algorithm.

When s is an initial state, the result is clearly X itself (Line 1). Otherwise, an updated map V' is computed, where the patch P is added to $V(s)$ if s is an open state (Line 3). The for loop at Lines 4–9 iterates over the incoming edges of s . For each such edge (s', s) ,

19:14 Model Checking Linear Temporal Properties on Polyhedral Systems

Line 5 sets A to the region of $\llbracket s' \rrbracket$ that has *not* been already visited. Line 6 computes the set of points of A that can reach some point in X , either leaving A immediately, if s' is a singular state ($type(s) = 0$), or lingering in A for some time, if it is open ($type(s) = +$). Line 7 splits the resulting set A' into a set of distinct pairs (Q_i, Y_i) , where Y_i is the maximal convex polyhedron contained in A' and in the patch Q_i of A . Each such pair (Q_i, Y_i) , then, gives rise to a recursive call on the state s' with targets Y_i and Q_i at Line 9. The results of all such calls are gathered in **Result**, which is returned at Line 10.

■ **Algorithm 1** Function $\exists\text{Denot}(s, P, X, V)$. For simplicity, we omit from the notation two implicit arguments: the finite automaton $\mathcal{A}_{\hat{\varphi}} = (S, \delta, \lambda, S_0, S_F)$ and the polyhedral system \mathcal{P} .

```

input :  $s \in S$ ;
           $P$ : convex polyhedron in  $\text{Patch}(\llbracket s \rrbracket)$ ;
           $X$ : convex polyhedron included in  $P$ ;
           $V$ : map from states  $u \in S$  to a subset of the patches of  $\llbracket u \rrbracket$ ;
output: A polyhedron in  $\mathbb{R}^n$ 

1 if  $s \in S_0$  then return  $X$ 
2 Result  $\leftarrow \emptyset$ 
3  $V' \leftarrow$  if  $s \in S_{\text{sing}}$  then  $V$  else  $V[s \mapsto V(s) \cup \{P\}]$ 
4 foreach state  $s' \in S$  such that  $(s', s) \in \delta$  do
5    $A \leftarrow \llbracket s' \rrbracket \setminus V(s')$ 
6    $A' \leftarrow \text{reach}^{type(s')}(A, X)$ 
7    $\{(Q_1, Y_1), \dots, (Q_n, Y_n)\} \leftarrow \text{split}(A', A)$ 
8   for  $i = 1, \dots, n$  do
9     Result  $\leftarrow$  Result  $\cup \exists\text{Denot}(s', Q_i, Y_i, V')$ 
10 return Result

```

The following lemmas state the characteristic properties of the function $\exists\text{Denot}$, namely termination (Lemma 16), and soundness and completeness (Lemma 17).

► **Lemma 16.** *For all convex polyhedra P and X , such that $P \in \text{Patch}(\llbracket s \rrbracket)$ and $X \subseteq P$, and maps $V : S \rightarrow_s 2^{\text{Patch}(\llbracket s \rrbracket)}$, the call to $\exists\text{Denot}(s, P, X, V)$ terminates after at most $|S|^{O(m \cdot |S|)} \cdot m^{O(m^2 \cdot |S|)}$ symbolic operations on polyhedra, with m the maximum number of patches in the denotation of any state.*

Proof. First, we prove that the recursion depth is bounded by $1 + 2 \cdot \sum_{s \in S} |\text{Patch}(\llbracket s \rrbracket)|$. Let $\chi = (s_0, P_0), (s_1, P_1), \dots$ be the sequence of first and second arguments in a stack of recursive calls to $\exists\text{Denot}$, with (s_0, P_0) being the bottom of the stack. Recall that by design $s_i \in S$ and P_i is one of the patches of $\llbracket s_i \rrbracket$. Consider a pair (s_i, P_i) with $s_i \in S_{\text{open}}$. The recursive call issued from a state s_i at recursion level i adds the patch P_i to $V(s_i)$ (Line 3). From that point on, i.e., at recursion levels $j > i$, if state s' considered at Line 4 is s_i , the assignment at Line 5 ensures that the patch P_i is not passed to the next recursive call. Hence, either $s_j \neq s_i$ or $P_j \neq P_i$. Equivalently, the pair (s_i, P_i) , cannot occur again in the sequence χ . By the generality of (s_i, P_i) , we obtain that the sequence χ contains no duplicate pairs whose state is in S_{open} . Since states in χ strictly alternate between S_{open} and S_{sing} , this proves the bound on the recursion depth. Termination follows from the fact that the number of recursive calls at each level is plainly finite. As to the bound on the symbolic operations, observe that, since the output of reach^+ contains at most $m^{O(m)}$ patches and the loop at Line 4 iterates on the states of the automaton, the branching degree of the recursion tree of

the algorithm is bounded by $|S| \cdot m^{O(m)}$. Its depth, instead, is bounded by $1 + 2 \cdot m \cdot |S|$ as shown above. Hence, the overall number of symbolic operations required by algorithm is bounded by $|S|^{O(m \cdot |S|)} \cdot m^{O(m^2 \cdot |S|)}$. ◀

A hybrid run ρ , with time-slicing $\{t_i\}_{i=0}^k$, ends in the pair (X, s) , for a set of points $X \subseteq \mathbb{R}^n$ and a state $s \in S$, if either $s \in S_{sing}$ and ρ is in (X, s) at the last instant of time in its domain, or $s \in S_{open}$ and ρ resides in (X, s) for some final open time interval bounded by t_k . Formally:

- if $s \in S_{sing}$, then $\rho(t_k) \in X \times \{s\}$;
- otherwise, there exists $t^* \in (t_{k-1}, t_k)$ such that $\rho(t) \in X \times \{s\}$, for all $t \in [t^*, t_k)$.

Moreover, we denote by $Visited(\rho)$ the set of pairs (P, s) , composed of a patch $P \in Patch(\llbracket s \rrbracket)$ and a state s , traversed by ρ at any time. We say that a hybrid run ρ avoids a pair (P, s) if $(P, s) \notin Visited(\rho)$. This notion of avoidance generalises to pairs (A, s) , with A a set of patches, and to sets of such pairs, in the obvious way.

The following lemma shows that for every hybrid run ρ there exists a similar hybrid run ρ'' that crosses a given pair (P, s) , with $s \in S_{open}$, at most once.

► **Lemma 17.** *For all states $s \in S$, convex polyhedra $P \in Patch(\llbracket s \rrbracket)$ and $X \subseteq P$, and maps $V : S \rightarrow_s 2^{Patch(\llbracket s \rrbracket)}$ such that $P \notin V(s)$, we have that $\exists Denot(s, P, X, V)$ returns the set of all points x from which there is a hybrid run $\rho \in HRun(x)$ such that: (a) ρ ends in (X, s) ; (b) ρ avoids V ; (c) if $s \in S_{open}$, then ρ avoids (P, s) , except for the last slice.*

The following theorem describes the initial arguments required by Algorithm 1 to solve the existential denotation problem.

► **Theorem 18.** *For all RTL_f formulas φ and polyhedral systems \mathcal{P} on the same set of atomic propositions, let $\hat{\varphi}$ be the corresponding LTL_f formula, $\mathcal{A}_{\hat{\varphi}}$ be the finite automaton for $\hat{\varphi}$, and*

$$X = \bigcup_{s \in S_F} \bigcup_{P \in Patch(\llbracket s \rrbracket)} \exists Denot(s, P, P, \emptyset).$$

Then, X is the set of points from which there exists a trajectory that satisfies φ .

Proof. Assume there exists a trajectory f from point x that satisfies φ , i.e., $x = f(0)$ and $\sigma_f \models \varphi$. Let us pick an arbitrary $\tau = \{t_i\}_{0 \leq i \leq k}$ in $TS(\sigma_f)$ and let $y \triangleq f(t_k)$. Then, by Theorem 12, $trace(\sigma_f, \tau) \models \hat{\varphi}$. By definition of $\mathcal{A}_{\hat{\varphi}}$, there exists an accepting run $r \in Runs(\mathcal{A}_{\hat{\varphi}})$ for $trace(\sigma_f, \tau)$ that ends in some final state $s \in S_F$. Let α be the last symbol of $trace(\sigma_f, \tau)$, then y belongs to some patch P of $\llbracket s_F \rrbracket = \llbracket \alpha \rrbracket$. Let now $\rho = (f, r^c)$ be the hybrid run from x whose second component r^c is the continuous run of r and τ . Clearly, ρ ends in (P, s_F) , hence it satisfies condition (a) of the statement of Lemma 17 (conditions (b) and (c) hold trivially for ρ). Therefore, Lemma 17 ensures that $x \in \exists Denot(s, P, P, \emptyset)$ and the thesis follows.

For the other direction, let x be a point in $\exists Denot(s, P, P, \emptyset)$, for some $s \in S_F$ and $P \in Patch(\llbracket s \rrbracket)$. By Lemma 17, there exists a hybrid run $\rho = (f, r^c)$ from x that ends in (P, s) , where P is a patch of $\llbracket s \rrbracket$ and r^c is a continuous run of some discrete run $r \in Runs(\mathcal{A}_{\hat{\varphi}})$ and some time-splitting τ for f . The run r is accepting since it ends in the same final state $s \in S_F$ as r^c and it accepts the word $trace(\sigma_f, \tau)$. This means that $trace(\sigma_f, \tau) \models \hat{\varphi}$ and Theorem 12, then, ensures that $\sigma_f \models \varphi$. ◀

6 Experiments

In this section, we report on the experiments performed with our implementation, which is based on Parma Polyhedral Library [5] as the underlying engine for the symbolic manipulation of polyhedra. Our prototype implementation starts from an RTL_f formula φ and computes its discretisation $\widehat{\varphi}$ according to Equation (1). This formula is translated into standard LTL (see [10]) in order to obtain a non-deterministic Büchi automaton recognising its models using SPOT [11]. The NBA is then turned into an NFA \mathcal{A} recognising the finite traces satisfying $\widehat{\varphi}$. The obtained automaton, together with the polyhedral system providing the flow constraints and the polyhedral denotations of the atomic propositions of φ , are finally fed to $\exists\text{Denot}$ (Algorithm 1).

We ran some experiments based on the two-tank model described in the introduction. The experiments consist of two families of RTL_f properties, called φ_k^{gap} and φ_k^{nogap} , of the following form:

$$\varphi_k^\star \triangleq \mathbf{G} \text{inv} \wedge t_0 \wedge \mathbf{G} t_{\max} \wedge \underbrace{\mathbf{F}(p \wedge \mathbf{F}(q \wedge \cdots \wedge \mathbf{F}(p \wedge \mathbf{F}q)))}_{k \text{ times}}$$

where $k \geq 1$, $\star \in \{\text{gap}, \text{nogap}\}$, and the interpretations of the atomic propositions is reported in the following table:

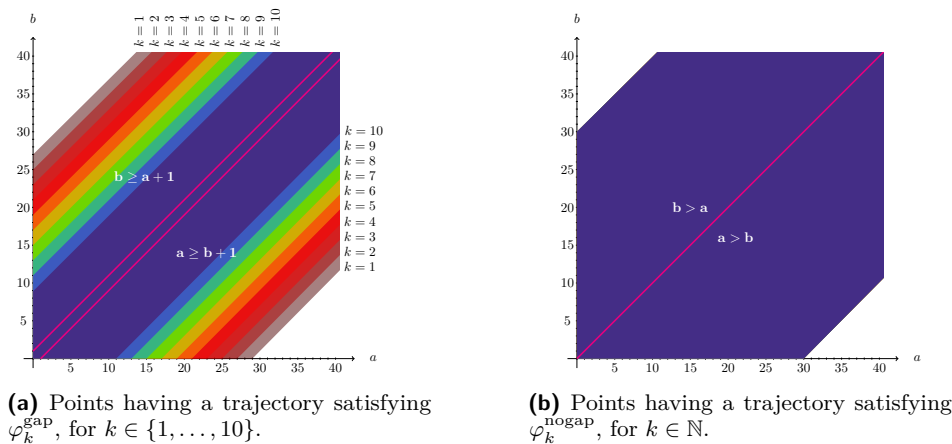
	$[p]$	$[q]$	$[\text{inv}]$	$[t_0]$	$[t_{\max}]$
φ_k^{gap}	$a \geq b + 1$	$b \geq a + 1$	$a \geq 0 \wedge b \geq 0$	$t = 0$	$t \leq 10$
φ_k^{nogap}	$a > b$	$b > a$	$a \geq 0 \wedge b \geq 0$	$t = 0$	$t \leq 10$

Both families require a trajectory that satisfies the invariant inv , starts at time $t = 0$ (represented by the proposition t_0) and ends at time 10 (enforced by the formula $\mathbf{G} t_{\max}$), and alternates k times between the propositions p and q . The only difference between the two families is in the polyhedral interpretations $[p]$ and $[q]$ of the atomic propositions p and q .

From a semantic standpoint, the first family φ_k^{gap} requires a trajectory to alternate k times between two disjoint and non-adjacent half-spaces. Since the flow constraint is a bounded (convex) polyhedron, the intensities of the derivatives are bounded, hence there is a minimum amount of time that any trajectory, reaching a point of the half-space $a \geq b + 1$, requires to reach the half-space $b \geq a + 1$. As a consequence, the number of alternations possible from different points may differ. The further away from the border of the half-spaces a point is, the fewer alternations are possible.

In the second family φ_k^{nogap} , instead, the two half-spaces between which to alternate are adjacent. Therefore, no minimum time is needed to move from one to the other. This means that, if a trajectory can reach $a > b$ and, from there, also reach $b > a$, then it may keep alternating between the two an arbitrary number of times.

Figure 2 shows the denotations of the two families of formulas, both limited to the cross section for $t = 0$. In particular, Figure 2a shows the different regions of points satisfying the RTL_f property indexed with the corresponding value of k . As explained above, the bigger the value of k , the smaller the region of points. For example, only the points in the dark blue region in the middle satisfy $\varphi_{10}^{\text{gap}}$, whereas the points satisfying φ_9^{gap} additionally include the two light blue strips. Observe also that the region of points in the half-space $a \geq b + 1$ satisfying the property φ_k^{gap} is bigger than the region of points in the half-space $b \geq a + 1$ that satisfies the same property. This is due to the fact that a trajectory from the points in latter region must spend additional time to first reach the half-space $a \geq b + 1$, leaving less time, with respect to the points in the former region, to perform the alternations. Figure 2b,



■ **Figure 2** The results of the experiments for the two families of RTL_f properties.

instead, is perfectly symmetric and shows that all the points from where one can reach the diagonal $a = b$ in the allotted time can alternate between the two half-spaces k times, regardless of the value of k .

References

- 1 R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994. doi:10.1016/0304-3975(94)90010-8.
- 2 R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1):116–146, 1996. doi:10.1145/227595.227602.
- 3 R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Softw. Eng.*, 22:181–201, March 1996. doi:10.1109/32.489079.
- 4 R. Alur, A. Trivedi, and D. Wojtczak. Optimal scheduling for constant-rate multi-mode systems. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, pages 75–84, 2012.
- 5 R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008. doi:10.1016/j.scico.2007.08.001.
- 6 M. Benerecetti and M. Faella. Automatic synthesis of switching controllers for linear hybrid systems: Reachability control. *ACM Trans. on Embedded Computing Systems*, 16(4), 2017. doi:10.1145/3047500.
- 7 M. Benerecetti, M. Faella, and S. Minopoli. Automatic synthesis of switching controllers for linear hybrid systems: Safety control. *Theoretical Computer Science*, 493:116–138, 2013. doi:10.1016/J.TCS.2012.10.042.
- 8 M. Blondin, P. Offtermatt, and A. Sansfaçon-Buchanan. Verifying linear temporal specifications of constant-rate multi-mode systems. In *LICS*, pages 1–13, 2023. doi:10.1109/LICS56636.2023.10175721.
- 9 E. Davis. Infinite loops in finite time: Some observations. In *Proc. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'92)*. Cambridge, MA, USA, October 25-29, 1992, pages 47–58. Morgan Kaufmann, 1992.
- 10 G. De Giacomo and M.Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 854–860. IJCAI/AAAI, 2013. URL: <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997>.

- 11 A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0—a framework for ltl and-automata manipulation. In *International Symposium on Automated Technology for Verification and Analysis*, pages 122–129. Springer, 2016.
- 12 G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *CAV 11: Proc. of 23rd Conf. on Computer Aided Verification*, pages 379–395, 2011. doi:10.1007/978-3-642-22110-1_30.
- 13 T.A. Henzinger. The theory of hybrid automata. In *11th IEEE Symp. Logic in Comp. Sci.*, pages 278–292, 1996. doi:0.1109/LICS.1996.561342.
- 14 T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What’s decidable about hybrid automata? *J. of Computer and System Sciences*, 57(1):94–124, 1998. doi:10.1006/jcss.1998.1581.
- 15 M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008. doi:10.1109/TAC.2007.914952.
- 16 R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990. doi:10.1007/BF01995674.
- 17 O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004. doi:10.1007/978-3-540-30206-3_12.
- 18 O. Maler, D. Nickovic, and A. Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. *Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, pages 475–505, 2008. doi:10.1007/978-3-540-78127-1_26.
- 19 A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.
- 20 R. Poovendran. Cyber–physical systems: Close encounters between two parallel worlds. *Proceedings of the IEEE*, 98(8):1363–1366, 2010.
- 21 M. Reynolds. The complexity of the temporal logic with “until” over general linear time. *Journal of Computer and System Sciences*, 66(2):393–426, 2003. doi:10.1016/S0022-0000(03)00005-9.
- 22 M. Reynolds. The complexity of temporal logic over the reals. *Annals of Pure and Applied Logic*, 161(8):1063–1096, 2010. doi:10.1016/J.APAL.2010.01.002.

A Additional Proofs

► **Proposition 2.** *On the set of trajectories, Lipschitz continuity and well-behavedness are incomparable notions.*

Proof. In \mathbb{R}^2 , the trajectory $f_1(t) = (t, t^2)$ is well-behaved but not Lipschitz continuous. Note that f_1 is locally Lipschitz continuous. For the other non-implication, let $f_2(0) = (0, 0)$, $f_2(t) = (t, t^2 \cdot \sin(t^{-1}))$, for all $t \in (0, \pi^{-1})$, and $f_2(t) = (t, t - \pi^{-1})$, for all $t \geq \pi^{-1}$. Then, f_2 is Lipschitz continuous because it is differentiable in $(0, +\infty)$ and its derivative is bounded. However, it is not well-behaved because it crosses the hyperplane $y = 0$ infinitely often in any time interval $(0, a)$, with $a > 0$. ◀

► **Proposition 4.** *If a trajectory f is well-behaved, then the corresponding signal σ_f has finite variability.*

Proof. Take any polyhedral partitioning $\{P_i\}_{i \in I}$ of \mathbb{R}^n that respects the propositions in AP , meaning that, for all $i \in I$ and $p \in AP$, either $P_i \cap [p] = \emptyset$ or $P_i \subseteq [p]$. Since f is well-behaved, it must change convex polyhedron in $\{P_i\}_{i \in I}$ a finite number of times in $\langle 0, T \rangle$. Let P_{i_1}, \dots, P_{i_z} be the sequence of convex polyhedra traversed by f in that interval and $\tau = \{t_i\}_{i=0}^k \subseteq \mathbb{R}^+$ the sequence of instants in which f changes polyhedron in the sequence, with the possible addition of instants 0 and T , if needed. Since the polyhedral partitioning respects AP , every P_{i_j} is contained in $\llbracket \alpha \rrbracket$, for some $\alpha \subseteq AP$. Hence, τ is a suitable time-slicing of f . The thesis follows then from the finite length of τ . \blacktriangleleft

► **Lemma 10.** For all RTL_f formulae φ , signals $\sigma: \langle 0, T \rangle \rightarrow 2^{AP}$, and time instants $t \in \langle 0, T \rangle$, the following holds true: $\sigma_{>t} \models \varphi$ iff there exists a time instant $t' \in (t, T]$ such that $\sigma_{\geq t'} \models \varphi$, for all $t'' \in (t, t')$.

Proof. The proof proceeds by induction on the Boolean structure of the RTL_f formula φ , where we consider as base cases the atomic propositions and the $\dot{\cup}$ temporal formulae. Since the inductive cases of Boolean operators \neg and \wedge are trivial to deal with, here we focus on the base cases for atomic propositions and $\dot{\cup}$ only. Recall that $\dot{\cup}$ is a derived operator.

- **[Base case $\varphi = p \in AP$].** Since $\sigma_{>t}$ is a left-open signal, by the semantic of atomic propositions, $\sigma_{>t} \models p$ holds iff there exists a non-empty open interval $(t, t') \subseteq (t, T]$ such that $p \in \sigma_{>t}(t'' - t) = \sigma(t'')$, for all $t'' \in (t, t')$, which also means $\sigma_{\geq t'} \models p$, again by the semantic of atomic propositions, this time on left-closed signals. Hence, the truth of the statement is immediately verified.
- **[Base case $\varphi = \varphi_1 \dot{\cup} \varphi_2$, only-if direction].** By the semantics of the temporal operator $\dot{\cup}$, if $\sigma_{>t} \models \varphi_1 \dot{\cup} \varphi_2$, then there exists $t_2 \in (t, T]$ such that $\sigma_{\geq t_2} \models \varphi_2$ and $\sigma_{\geq t_1} \models \varphi_1$, for all $t_1 \in (t, t_2)$. As an immediate consequence, by using precisely $t' \triangleq t_2$ as witness of the second property, we have $\sigma_{\geq t'} \models \varphi_1 \dot{\cup} \varphi_2$, for all $t'' \in (t, t')$.
- **[Base case $\varphi = \varphi_1 \dot{\cup} \varphi_2$, if direction].** Let $\tau = \{t_i\}_{i=0}^k \in TS(\sigma_{>t})$ be a time-slicing of the suffix $\sigma_{>t}$ of the signal σ and $j \in \{1, \dots, k\}$ the smallest index such that either (a) $\sigma_{\geq t+t_j} \models \varphi_2$ or (b) $\sigma_{\geq t+t''} \models \varphi_2$, for all $t'' \in (t_{j-1}, t_j)$. The existence of such an index is ensured by the fact that at every time instant t'' of the non-empty open interval (t, t') the until formula $\varphi_1 \dot{\cup} \varphi_2$ is satisfied. Now, suppose by contradiction that $\sigma_{>t} \not\models \varphi_1 \dot{\cup} \varphi_2$. Since φ_2 is satisfied either at time instant $t + t_j$ or at all time instants $t + t''$, with $t'' \in (t_{j-1}, t_j)$, the only possibility for the until formula to be falsified is the existence of at least one time instant t_1 , either in $(t, t + t_j)$ or in $(t, t_{j-1}]$, such that $\sigma_{\geq t_1} \not\models \varphi_1$. However, this would clearly lead to $\sigma_{\geq t'} \not\models \varphi_1 \dot{\cup} \varphi_2$, for all $t'' \in (t, t_1)$, which contradicts the hypothesis. \blacktriangleleft

The following lemma is a straightforward adaptation of Lemma 1 from [3], used in the proofs of Lemma 13 and Lemma 14.

► **Lemma 19 ([3]).** For any convex flow constraint $Flow$, convex polyhedron X and points $x_1, x_2 \in X$, the following two conditions are equivalent for all $t^* \geq 0$:

1. there exists a trajectory f such that $f(0) = x_1$ and $f(t^*) = x_2$;
2. there is a straight-line trajectory $f'(t) \triangleq x_1 + d \cdot t$, with $d \in Flow$, such that $f'(0) = x_1$, $f'(t^*) = x_2$ and $f'(t) \in X$, for all $t \in [0, t^*]$.

► **Lemma 13.** For all polyhedra A and convex polyhedra B the following holds:

$$\text{reach}^0(A, B) = A \cap \text{cl}(B) \cap B_{\setminus < 0}.$$

19:20 Model Checking Linear Temporal Properties on Polyhedral Systems

Proof. Let $\Delta \triangleq A \cap cl(B) \cap B_{\angle_{>0}}$. To show that $\Delta \subseteq X^0$, we just need to observe first that if $x \in \Delta$, then $x \in A$. Moreover, $x \in B_{\angle_{>0}}$, which means that there is a direction $d \in Flow$ such that $x + d \cdot t \in B$, for some $t > 0$. In addition, since $x \in cl(B)$ and B is convex, we have that $x + d \cdot t' \in B$, for all $t' \in (0, t]$. But $f(t) = x + d \cdot t$ is clearly an admissible trajectory that satisfies the required conditions for x to be in X^0 . Hence, the conclusion.

For the other direction, let $x \in X^0$, f be any admissible witness trajectory and $t > 0$ be such that $f(t') \in B$ for all $t' \in (0, t]$. Let, in addition, $y \triangleq f(t)$. By convexity of B , the segment connecting x and y lies entirely in $cl(B)$. Since y can be reached from x following an admissible trajectory, by convexity of $Flow$ and Lemma 19, there is a direction $d \in Flow$ such that $y = x + d \cdot t$. Clearly, the set

$$\{z \in \mathbb{R}^n \mid z = x + d \cdot t', \text{ for some } t' \in [0, t]\}$$

contains all and only the points of the segment from x to y and is, therefore, contained in $cl(B)$. Hence, we conclude that $x \in A \cap cl(B) \cap B_{\angle_{>0}}$, as required. \blacktriangleleft

► **Lemma 14.** *For all polyhedra A and convex polyhedra B the following holds:*

$$reach^+(A, B) = \bigcup_{P \in Patch(A)} RWA^m(T_P, \bar{A}), \quad \text{where } T_P \triangleq P \cap (cl(P) \cap B)_{\angle_{>0}}.$$

Proof. First, observe that, by definition, $T_P \subseteq P \subseteq A$. As a consequence, we obtain that:

$$RWA^m(T_P, \bar{A}) = \{x \in \mathbb{R}^n \mid \exists f \in Traj_{wb}(x), t \geq 0 : f(t) \in T_P \text{ and } \forall t' \in [0, t] : f(t') \in A\}.$$

Consider any point $x \in RWA^m(T_P, \bar{A})$, a trajectory f witnessing its membership to the set $RWA^m(T_P, \bar{A})$, a time instant $t \in \mathbb{R}^+$ such that $f(t) \in T_P$ and $f(t') \in A$, for all $t' \in [0, t]$, and let $y \triangleq f(t) \in T_P$. Clearly, $y \in P$ and also $y \in (cl(P) \cap B)_{\angle_{>0}}$. This means that there is an admissible straight trajectory f' that, in a strictly positive amount of time, leads from y to a point belonging both to the closure of P and to B . Let $t^* > 0$ be a time instant such that $f'(t^*) \in cl(P) \cap B$. Since f' is a straight trajectory and P is a convex polyhedron, $f'(t')$ is contained in P , hence also in A , for all $t' \in [0, t^*]$. By concatenating f with f' we obtain an admissible trajectory f'' defined as follows: $f''(t') = f(t')$, for all $t' \in [0, t]$, and $f''(t') = f'(t' - t)$, for all $t' \in (t, t + t^*]$. Clearly, f'' leads from $x \in A$ to a point $z \in B$, while never leaving A except, possibly, in the last instant. In addition, $x = f(0) = f''(0)$ and $f(0)$ is required to belong to A . By combining these observations, we obtain that for all $x \in RWA^m(T_P, \bar{A})$ it holds that there exists $f \in Traj_{wb}(x)$ and $t \in \mathbb{R}_{>0}$ with $f(t) \in B$ and for all $t' \in (0, t)$, $f(t') \in A$. Hence, for all $P \in Patch(A)$, we have that $RWA^m(T_P, \bar{A}) \subseteq reach^+(A, B)$.

For the other direction, assume $x \in reach^+(A, B)$. Then there exist $f \in Traj_{wb}(x)$ and $t \in \mathbb{R}_{>0}$, with $f(t) \in B$ and $f(t') \in A$, for all $t' \in (0, t)$. Since A is a polyhedron and f is well-behaved, the trajectory can only change convex polyhedron in $Patch(A)$ a finite number of times. This means that there is a last patch of A traversed by f before entering B in which f lingers for a positive amount of time. Let P be such a patch. Since as soon as f exits from P it enters B , it must do so by passing at time t through a point in B that lies on the border between P and B , that is $f(t) \in cl(P) \cap B$. Let $t^* \in [0, t)$ be a time interval such that $f(t') \in P$, for all $t' \in (t^*, t)$. By convexity of P and $Flow$ and thanks to Lemma 19, we obtain that $f(t') \in P \cap (cl(P) \cap B)_{\angle_{>0}} = T_P$, for all $t' \in (t^*, t)$. But then f is a witness of the membership of x to the set $RWA^m(T_P, \bar{A})$. As a consequence, we obtain that $reach^+(A, B) \subseteq \bigcup_{P \in Patch(A)} RWA^m(T_P, \bar{A})$. \blacktriangleleft

The next lemma is instrumental in proving the completeness of Algorithm 1 in Lemma 17.

► **Lemma 20.** *For all hybrid runs ρ , states $s \in S_{open}$, and patches $P \in Patch(\llbracket s \rrbracket)$, there exists a hybrid run ρ' such that:*

- ρ' starts and ends in the same pairs as ρ ;
- ρ' passes at most once through the pair (P, s) ;
- $Visited(\rho') \subseteq Visited(\rho)$;
- the length of the shortest discrete trace of ρ' is smaller than or equal to that of ρ .

Proof. Assume that ρ passes at least twice through the pair (P, s) . Let t_1, t_2 be two times in the domain of ρ belonging to the first and to the last visit to (P, s) . In particular, $\rho(t_i) \in (P, s)$, for $i = 1, 2$. Let $\rho(t_i) = (x_i, s)$, we define a new hybrid run ρ' by connecting with a straight trajectory point x_1 to x_2 . By convexity of the flow, such trajectory is feasible. By convexity of P , such trajectory is entirely contained in P . The rest of ρ' follows exactly ρ . It is easy to see that ρ' satisfies all properties required by the thesis. ◀

► **Lemma 17.** *For all states $s \in S$, convex polyhedra $P \in Patch(\llbracket s \rrbracket)$ and $X \subseteq P$, and maps $V : S \rightarrow_s 2^{Patch(\llbracket s \rrbracket)}$ such that $P \notin V(s)$, we have that $\exists Denot(s, P, X, V)$ returns the set of all points x from which there is a hybrid run $\rho \in HRun(x)$ such that: (a) ρ ends in (X, s) ; (b) ρ avoids V ; (c) if $s \in S_{open}$, then ρ avoids (P, s) , except for the last slice.*

Proof.

Soundness. First, we prove that the base case of the algorithm is sound, that is, that the points returned at Line 1 satisfy the lemma items. Any initial state of $\mathcal{A}_{\hat{\varphi}}$ is in itself a run of $\mathcal{A}_{\hat{\varphi}}$ of length 1 from an initial state. If s is an initial state, by Proposition 15(a) it includes the *sing* proposition. Then, for all $x \in X$ let f be the trajectory of duration 0 defined by $f(0) = x$. Its discrete trace w_f contains a single symbol and induces the run $r^d = s$ in $\mathcal{A}_{\hat{\varphi}}$. The hybrid run (f, r^c) ends in (X, s) , giving Item (a); and avoids V , because its only point is (x, s) and, by assumption, $x \in P \notin V(s)$. Thus, we have Item (b). Item (c) trivially holds since $s \notin S_{open}$.

Next, we consider the points added to the result at Line 9. We proceed by induction on the length k of the longest sequence of pairs $(s_i, P_i)_{i=0, \dots, k-1}$ such that: (i) the sequence $(s_i)_{i=0, \dots, k-1}$ is a (not necessarily initial) run of $\mathcal{A}_{\hat{\varphi}}$ ending in $s_{k-1} = s$, (ii) $P_{k-1} = P$, (iii) each P_i is a patch of $\llbracket s_i \rrbracket$, (iv) if $s_i \in S_{open}$ then $P_i \notin V(s_i)$, (v) all the pairs (s_i, P_i) such that $s_i \in S_{open}$ are distinct. Note that Items (i) and (v) imply that the length of these sequences is bounded by twice the number of distinct non-singular pairs. We call the length so defined $k(s, P, V)$.

In the base case, $k(s, P, V) = 1$. Then, the algorithm does not perform any recursive call, because for each predecessor s' of s , the set $A' \triangleq reach^{type(s')}(A, X)$ computed at Line 6 is empty, with $A \triangleq \llbracket s' \rrbracket \setminus V(s')$. Indeed, assume by contradiction, that there is a predecessor s' of s whose set A' is not empty. Therefore, there must be a pair $(Q, Y) \in split(A', A)$, where Q is a patch of A and the sequence $(s', Q)(s, P)$ has all the properties (i)–(v) needed to prove that $k(s, P, V) > 1$, contradicting the assumption of the base case. We conclude that either s has no predecessors, or A' is empty. Hence, no points are added to the result at Line 9.

For the inductive case, assume that the longest sequence described above has length greater than 1. Let s' be a predecessor of s and let $\{(Q_1, Y_1), \dots, (Q_n, Y_n)\}$ be $split(A', A)$. For all $i = 1, \dots, n$, we apply the inductive hypothesis to s' , Q_i , and V' , where $V' = V[s \mapsto V(s) \cup \{P\}]$, if $s \in S_{open}$, and $V' = V$, otherwise, as prescribed by Line 3. In order to apply the inductive hypothesis, we prove that $k(s', Q_i, V') < k(s, P, V)$ in both cases. Assume by contradiction that $h \triangleq k(s', Q_i, V') \geq k(s, P, V)$ and let $\xi \triangleq (s_i, P_i)_{i=0, \dots, h-1}$ be the sequence

of length h corresponding to (s', Q_i, V') . We extend ξ' with the pair (s, P) , thus obtaining the sequence $\xi' \triangleq \xi \cdot (s, P)$ of length $h + 1$. We can show that ξ' satisfies all five Items (i)-(v) w.r.t. (s, P, V) . Items (i)-(iii) are trivially true, so we can focus on the remaining two:

- If $s \in S_{open}$, then Item (iv) follows from the assumption that $P \notin V(s)$, while Item (v) is due to the fact that no pair in ξ can be equal to (s, P) , since $P \in V'(s)$.
- If $s \in S_{sing}$, then both Items (iv) and (v) hold trivially.

Hence, ξ' is a sequence satisfying (i)-(v) w.r.t. (s, P, V) , so $k(s, P, V) \geq h + 1$, which contradicts the hypothesis.

Now, consider a point x in $\exists \text{Denot}(s', Q_i, Y_i, V')$ and the witness hybrid run $\rho' = (f', r')$ provided by the inductive hypothesis, whose trajectory f' goes from x to $Y_i \subseteq A$, and let $\{t_j\}_{j=0}^k$ be its time-slicing. In the following we shall extend ρ' to reach (X, s) , using the definition of *reach*, while satisfying the Items (a), (b), and (c) of the lemma. We again distinguish two cases.

- [$s' \in S_{sing}$] By Proposition 15(c), $\text{type}(s') = 0$ and $s \in S_{open}$. In this case, f' must end in some point $z \in Y_i$. Since $Y_i \subseteq A' = \text{reach}^0(A, X)$, let f'' be the trajectory that starts in z , immediately enters X , and remains inside X in the interval $(0, \epsilon)$, for some $\epsilon > 0$. Let f be the concatenation of f' and f'' . It is immediate to observe that $\tau = \{t_j\}_{j=0}^{k+1}$, with $t_{k+1} = (t_k + \epsilon)$, is a time-slicing of f .

Let us set $\rho \triangleq (f, r)$, where r is the continuous run of f that has the following form:

$$r(t) = \begin{cases} r'(t) & \text{if } 0 \leq t \leq t_k \\ s & \text{if } t_k < t < t_k + \epsilon. \end{cases}$$

Clearly, ρ is a hybrid run in $\text{HRun}(x)$ with τ one of its time-slicings.

As, by construction, ρ ends in (X, s) , we obtain that Item (a) holds. Item (b) is satisfied, since ρ' avoids V' , by assumption $P \notin V(s)$, and V is pointwise included in V' . Item (c), instead, follows from the fact that $V' = V[s \mapsto V(s) \cup P]$.

- [$s' \in S_{open}$] Obviously, $\text{type}(s') = +$, and $s \in S_{sing}$. Recall that ρ' ends in (Y_i, s') and let $t' > t_{k-1}$ be any time instant such that $y \triangleq f'(t') \in Y_i$. Observe that $r(t') = s'$, since there cannot be a state change within the same time slice. Let f'' be the witness trajectory given by the property of $\text{reach}^+(A, X)$, which starts in y , reaches X , and in the intermediate times remains inside $A = \llbracket s' \rrbracket \setminus V(s')$. Now, let f be the trajectory obtained by concatenating the prefix of f' ending in y with f'' and $\tau = \{t_0, t_1, \dots, t_{k-1}, t^*\}$, with $t^* = t' + \epsilon$, where ϵ is the duration of f'' . Observe that $f(t) \in \llbracket s' \rrbracket$, for all $t \in (t_{k-1}, t^*)$. Indeed, $(t_{k-1}, t') \subseteq (t_{k-1}, t_k)$ and, by hypothesis, f' lies in $\llbracket s' \rrbracket$ in latter interval. Moreover, f'' lies in $A \subseteq \llbracket s' \rrbracket$ in the interval $(0, \epsilon)$, hence f lies in $A \subseteq \llbracket s' \rrbracket$ in the interval (t', t^*) . Clearly, the signal of f is constant, and equal to $\lambda(s') \cap AP$, in the entire interval (t_{k-1}, t^*) , therefore τ is a proper time slicing for f . Let us set $\rho \triangleq (f, r)$, where r is the continuous run of f that has the following form:

$$r(t) = \begin{cases} r'(t) & \text{if } 0 \leq t \leq t_{k-1} \\ s' & \text{if } t_{k-1} < t < t^* \\ s & \text{if } t = t^*. \end{cases}$$

Trivially, ρ satisfies Item (c). Moreover, ρ satisfies Item (b), since ρ' avoids $V' = V$ and at all times f'' is either contained in A , which is disjoint from $V(s')$, or in X , which is disjoint from $V(s)$ by assumption. Item (a) holds as well, since $w_f = w_{f'} \cdot \lambda(s)$. Indeed, the trajectory f'' entirely lies in $\llbracket s' \rrbracket$ except for its last point, which belongs to $\llbracket s \rrbracket$.

Completeness. Given $s \in S$, $P \in \text{Patch}(\llbracket s \rrbracket)$, $X \subseteq P$, and V , let y be a point from which there is a hybrid run satisfying Items (a)-(c). Among those hybrid runs, let $\rho = (f, r^c)$ be one that induces a *shortest* discrete trace, and let $\tau = \{t_i\}_{i=0}^k$ be the corresponding time-slicing. Formally, $(\rho, \tau) \in \arg \min_{(f, \tau) \in \text{HRun}(x), \tau \in \text{TS}(\sigma_f)} |\text{trace}(\sigma_f, \tau)|$.



Let $w \triangleq \text{trace}(\sigma_f, \tau)$ and $k(y, s, P, X, V)$ be the length of w . We prove that $y \in \exists \text{Denot}(s, P, X, V)$ by induction on $k(y, s, P, X, V)$.

- Base case [$k(y, s, P, X, V) = 1$]: By Item (a), s is an initial state. By Proposition 15(a), $s \in S_{\text{sing}}$. Since the only left-closed trajectories with a discrete trace of length one are those with zero duration, we have that f starts and ends in y , which implies $y \in X$, by Item (a). By Line 1 of Algorithm 1, we have that $y \in \exists \text{Denot}(s, P, X, V)$, thus, the thesis follows.
- Inductive case [$k(y, s, P, X, V) > 1$]: Let $w = w' \cdot \alpha$, with $\alpha \subseteq \widehat{AP}$ and $s' \in S$ the state of $\mathcal{A}_{\hat{\varphi}}$ preceding s in r^c . Note that $\alpha = \lambda(s)$ and $s \notin S_0$, by Proposition 15(b). We distinguish two cases.

[$s \in S_{\text{sing}}$]: By Proposition 15(c), $s' \in S_{\text{open}}$. Let $A' = \text{reach}^+(A, X)$, with $A = \llbracket s' \rrbracket \setminus V(s')$. Since f is well-behaved and lies in A' in the last open slice (t_{k-1}, t_k) of τ , there exists a pair $(Q, Y) \in \text{split}(A', A)$ and $\epsilon > 0$ such that f lies in $Y \subseteq Q$ at all times in $(t_{k-1}, t_{k-1} + \epsilon]$. Consider the prefix $\rho' = (f_{\leq t+\epsilon}, r^c_{\leq t+\epsilon})$, clearly ρ' ends in (Y, s') and its discrete trace is obtained from w by removing the last symbol α . By applying Lemma 20 to ρ' and (Q, s') , there exists a hybrid run ρ'' that starts and ends where ρ' does, passes only once through (Q, s') , satisfies $\text{Visited}(\rho'') \subseteq \text{Visited}(\rho')$, and the induced discrete trace is no longer than the one of ρ' . Therefore, $k(y, s', Q, Y, V) < k(y, s, P, X, V)$. Hence, y satisfies the inductive hypothesis w.r.t. s', Q, Y , and V , as witnessed by ρ'' , and so $y \in \exists \text{Denot}(s', Q, Y, V)$. Since $(s', s) \in \delta$ and $(Q, Y) \in \text{split}(A', A)$, the algorithm at Line 9 adds y to the set **Result**, which is then returned.

[$s \in S_{\text{open}}$]: By Proposition 15(c), $s' \in S_{\text{sing}}$. Let $A' = \text{reach}^0(A, X)$, with $A = \llbracket s' \rrbracket \setminus V(s')$. Since ρ ends in (X, s) , there exists $(Q, Y) \in \text{split}(A', A)$ such that $f(t_{k-1}) \in Y$. Next, consider the prefixes $f' = f_{\leq t_{k-1}}$, $\rho' = \rho_{\leq t_{k-1}}$, and $\tau' = \{t_i\}_{i=0}^{k-1}$. Clearly, ρ' ends in $(\{f(t_{k-1})\}, s')$. Let $V' = V[s \mapsto V(s) \cup \{P\}]$ as in Line 3 of the algorithm. By Items (b) and (c) on ρ w.r.t. y, s, P, X , and V , it holds that ρ' avoids V and (P, s) . Hence, ρ' avoids V' . It follows that ρ' satisfies Items (a)-(c) with respect to y, s', Q, Y , and V' . Moreover, the discrete trace $\text{trace}(\sigma_{f'}, \tau')$ is strictly shorter than w by construction. We then have that $k(y, s', Q, Y, V') < k(y, s, P, X, V)$ and, by inductive hypothesis, we obtain that $y \in \exists \text{Denot}(s', Q, Y, V')$. Since $(s', s) \in \delta$ and $(Q, Y) \in \text{split}(A', A)$, the algorithm at Line 9 adds y to **Result**, which is then returned. ◀

FastMinTC+: A Fast and Effective Heuristic for Minimum Timeline Cover on Temporal Networks

Giorgio Lazzarinetti  

Università degli Studi Milano-Bicocca, Milano, Italy

Sara Manzoni  

Università degli Studi Milano-Bicocca, Milano, Italy

Italo Zoppis  

Università degli Studi Milano-Bicocca, Milano, Italy

Riccardo Dondi  

Università degli Studi di Bergamo, Bergamo, Italy

Abstract

The analysis and summarization of temporal networks are crucial for understanding complex interactions over time, yet pose significant computational challenges. This paper introduces FASTMINTC+, an innovative heuristic approach designed to efficiently solve the Minimum Timeline Cover (MINTCOVER) problem in temporal networks. Our approach focuses on the optimization of activity timelines within temporal networks, aiming to provide both effective and computationally feasible solutions. By employing a low-complexity approach, FASTMINTC+ adeptly handles massive temporal graphs, improving upon existing methods. Indeed, comparative evaluations on both synthetic and real-world datasets demonstrate that our algorithm outperforms established benchmarks with remarkable efficiency and accuracy. The results highlight the potential of heuristic approaches in the domain of temporal network analysis and open up new avenues for further research incorporating other computational techniques, for example deep learning, to enhance the adaptability and precision of such heuristics.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Design and analysis of algorithms; Theory of computation → Mathematical optimization; Theory of computation → Discrete optimization

Keywords and phrases Temporal Networks, Activity Timeline, Timeline Cover, Vertex Cover, Optimization, Heuristic

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.20

1 Introduction

Modern applications are increasingly incorporating new data abstractions that necessitate redefined approaches to data summarization and synthesis. Notably, with the widespread availability of temporal information, many datasets, traditionally modeled as networks, are now being treated as temporal networks [10, 18], i.e., graphs $G = (V, E)$ that include temporal edges representing interactions among a set of entities V , where each edge $(u, v, t) \in E$ captures the interaction at time t between entities u and v .

This paper introduces a novel heuristic, FastMinTC+, aimed at summarizing temporal networks – a critical area for data compression, visualization, interactive analysis, and noise reduction. Temporal network summarization poses unique challenges stemming from their inherent complexity and the diverse objectives of summarization, for which different methods have been proposed [17]. These methods employ a variety of techniques, including temporal motifs [19], graphlets [11], vocabulary-based summaries [26], evolutionary patterns [27], and community evolution [21]. While effective, such techniques can be complex and difficult to interpret. To simplify, research has shifted towards using activity time intervals to represent



© Giorgio Lazzarinetti, Sara Manzoni, Italo Zoppis, and Riccardo Dondi;
licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 20; pp. 20:1–20:18

Leibniz International Proceedings in Informatics

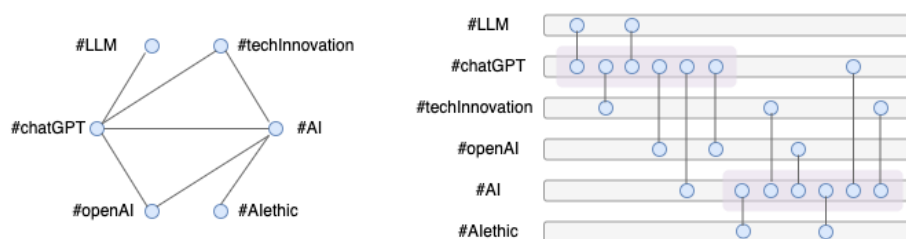


LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

20:2 FastMinTC+: An Heuristic for the MinTCover Problem

interactions between entities [24, 25, 8, 5, 4, 6, 7]. An interaction between two entities is accounted for if, at the time of their interaction, at least one of the entities is active. This summarization task involves identifying these latent activity intervals for all entities, producing an activity timeline that encompasses the entire network. This problem, known as *Minimum Timeline Cover* (MINTCOVER) problem, was introduced by Rozenshtein et al. [25], with the goal of identifying crucial time intervals that elucidate significant network events.

To illustrate the importance of activity timelines in understanding events, consider the launch of *ChatGPT* by *OpenAI* in November 2022. This event, which quickly captured public attention due to its advances in *AI* and large language model (*LLM*) capabilities, sparked widespread social media engagement and discussions on *AI ethics* and potential *technological innovations*. Figure 1 shows a co-occurrence graph on the left, where vertices represent hashtags and edges connect hashtags that appear together in posts. On the right side of Figure 1, a temporal network model visualizes these interactions, highlighting how data structured in timelines of (entity, time-interval) pairs can offer deep insights into significant events. These timelines, indicated in purple, outline key moments such as the initial launch and subsequent debates around AI, underlining the central entities' roles in shaping discussions. This method of mapping event timelines is central to solving the MINTCOVER problem.



■ **Figure 1** Example of a co-occurrence graph (left side) and temporal network framework (right side) with event timelines (in purple). The analysis of hashtags coming from social media discussions on two distinct events in time intuitively shows the relationship between activity timelines and events, which are the objective of the MINTCOVER problem.

Despite the interpretability and effectiveness of this problem, studies on the hardness and parametrized complexity highlight that MINTCOVER is NP-hard and, when considering more than one time interval, not even approximable within any constant factor (deciding whether there exists a solution of span 0 is indeed an NP-complete problem) [25, 8, 5, 4]. From hence, our research focuses on the development of approximation and heuristic algorithms capable of generating satisfactory timeline covers in feasible time frames. In this study, we introduce a novel local search heuristic for the MINTCOVER problem, employing a low-complexity approach in order to make it feasible even on large-scale graphs. The performance of our method are shown with an experimental evaluation on both synthetic and real-world datasets. The experiments show that our algorithm improves upon existing methods, both for efficiency and accuracy.

The rest of the paper is organized as follow. In Section 2 we formally define the MINTCOVER problem and we present some related works on approximate solutions. In Section 3, we describe our heuristic, FASTMINTC+, for solving the MINTCOVER problem. In Section 4 we provide the experimental results on the outlined comparison. Final considerations and future direction are described in Section 5.

2 Preliminaries

2.1 Problem Definition and Notions

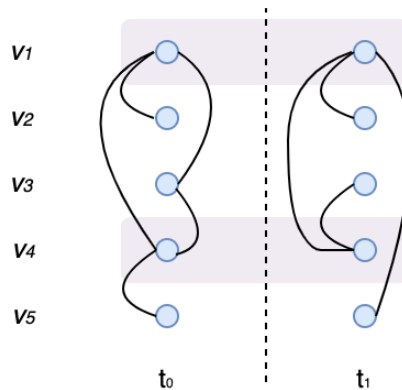
Let $G = (V, E)$ be a temporal graph, with V a set of vertices and E a set of temporal edges, where each edge is a triple $(u, v, t) \in E$, such that $(u, v) \in V$ and t is a timestamp indicating the time that an interaction between vertices u and v takes place. We consider unweighted undirected graphs. Given a vertex $u \in V$ we define, $E(u) = \{(u, v, t) \in E\}$ as the set of temporal edges incident in u , $N((u, t)) = \{v | (u, v, t) \in E\}$ as the set of vertices incident in u at timestamp t , $T(u) = \{t | (u, v, t) \in E\}$ as the set of timestamps of edges incident in u .

Following the definition provided in [7], given a temporal graph $G = (V, E)$ and a vertex $u \in V$, the *local degree* of u in a timestamp t , denoted as $deg_L((u, t)) = |N((u, t))|$, is the number of temporal edges incident in u at a timestamp t , while the *global degree* of a vertex u , denoted as $deg(u) = \sum_{t=1}^T deg_L((u, t))$ is the number of temporal edges incident in u in the overall time domain. Moreover, we define the overall *density* of an undirected graph as the ratio of the number of edges $|E|$ with respect to the maximum possible edges: $d = \frac{2|E|}{|V|(|V|-1)|T|}$. We consider graph to be *sparse* when $|E| = O(|V| + |T|)$.

Given two numbers s_u, e_u , with $s_u \leq e_u$ we define $I_u = [s_u, e_u]$ as the activity interval of vertex u and $\mathcal{T} = \{I_u\}_{u \in V}$ as an active timeline of G . Given an interval $I_u = [s_u, e_u]$, $\delta(I_u) = e_u - s_u$ is the *span* of interval I_u .

► **Definition 1** (Timeline Cover). *Given a temporal graph $G = (V, E)$ and an activity timeline $\mathcal{T} = \{I_u\}_{u \in V}$, we say that \mathcal{T} covers G if $\forall (u, v, t) \in E$, $t \in I_u$ or $t \in I_v$.*

Figure 2 shows an example of timeline covering over a 2-timestamps graph with 5 nodes.



■ **Figure 2** A temporal graph with 5 vertices and 9 edges, distributed over two timestamps. In purple we can see a possible timeline cover represented by intervals $I_{v1} = [0, 1]$ and $I_{v4} = [0, 1]$. This is a timeline cover since every edge $(u, v, t) \in E$ is such that t is in either I_u or in I_v .

The trivial timeline $I_u = [\min T(u), \max T(u)]$ provides a cover but may have unnecessarily long intervals. Indeed, the task is to find a timeline that has the most compact intervals possible according to some objective functions: the sum-span of a timeline \mathcal{T} , $S(\mathcal{T}) = \sum_{u \in V} \delta(I_u)$, or the max-span of a timeline, $\Delta\mathcal{T} = \max_{u \in V} \delta(I_u)$, are the objective functions proposed in the literature [25].

According to these quality measures, it is possible to define two problems:

► **Definition 2** (MINTCOVER₊). *Given a temporal network $G = (V, E)$, find a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ that covers G and minimizes the sum-span $S(\mathcal{T})$.*

► **Definition 3** (MINTCOVER_∞). *Given a temporal network $G = (V, E)$, find a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ that covers G and maximize the max-span $\Delta\mathcal{T}$.*

The selection of either the SUM or MAX formulation of the problem is contingent upon the specific application context. Generally, the MAX formulation facilitates the derivation of a worst-case bound on the duration of all activity intervals; however, this approach is susceptible to outliers, whereby a single long interval may precipitate solutions characterized by disproportionately high costs. Conversely, the SUM formulation is advisable in scenarios characterized by considerable variability in the duration of events anticipated within the activity timeline.

These problems can be further extended to allow k active intervals per vertex. It has been shown that also when there is only one activity interval per vertex, i.e., $k = 1$, while the MAX problem can be reduced to 2-SAT, and solved optimally in linear time, the SUM problem is NP-hard [8]. For this reason, in this research, we focus on designing and implementing an heuristic algorithm for the SUM problem, with $k = 1$, given the scarcity of heuristics even for this case, with the goal of proposing an efficient and effective approach also for massive graphs. For this problem we provide an exact Integer Linear Programming (ILP) formulation in Appendix A, considered mainly to evaluate the performance of our method on small datasets. We will refer to this problem (MINTCOVER_+) as MINTCOVER .

2.2 Related Works

The MINTCOVER problem, being a recent and NP-hard problem, has limited researches which focus primarily on the study of the parametric complexity and the development of approximate algorithms aimed at theoretical outcomes. Thus, in the following we describe these approximate solutions and, since in building our heuristic we mainly based on established Minimum Vertex Cover (MVC) heuristics, we also introduce the state-of-the-art solutions for MVC.

2.2.1 Approximate Solutions

For MINTCOVER , few approximate solutions have been proposed. In [25], Rozenstein et al. propose an inner point iterative method as a strategy to solve this problem by initially considering a subproblem named COALESCE. This subproblem involves finding optimal activity timelines that include predetermined time points for each vertex, called *inner points*. These points are essentially estimates on where a vertex is presumed to be active. The challenge is to construct intervals around these points to cover all interactions. Remarkably, the authors develop a method to find a 2-approximate solution to the COALESCE problem in linear time, by first providing an ILP formulation, then by relaxing the integrality constraint to write the dual, whose solution is then used iteratively to refine the MINTCOVER algorithm. Each iteration adjusts the inner points based on the activity intervals derived from the COALESCE solution, until the changes in the solution become negligible, indicating that the intervals are covering all interactions with minimal total duration. This method not only ensures comprehensive interaction coverage but also strives to minimize the overall activity time across the network. If from the one hand the authors were able to find a 2-approximate solution to the COALESCE problem, they cannot provide an approximation factor for MINTCOVER , even though they can compute a solution for this latter problem in linear time.

An $O(T \log n)$ factor approximation algorithm, that consists of two main phases, is proposed by Dondi et al. in [6, 7]. The algorithm works by considering the union graph $G_u = (V, E_u)$ of the temporal graph $G = (V, E)$, which is a static labeled graph, where labels for each edge are the union of all the timestamps t associated to edges (u, v, t) , for each pair (u, v) . The first phase consists in finding a minimum vertex cover with a 2-factor approximation algorithm [12] for the subgraph of the union graph that contains edges with at least three labels. For each vertex of this set is defined an activity interval that spans the entire temporal graph starting from 1. The vertices in this set are then removed from the temporal graph G , resulting in a temporal graph G' such that each pair of vertices is connected by at most two temporal edges. Then the second phase, inspired by an approximation algorithm for SETCOVER [9], uses a randomized rounding algorithm to find an approximation solution to a variant of the problem called Minimum Non-Consecutive Timeline Cover (MIN-NC-TCOVER) where each vertex can be active in non consecutive timestamps. Then they define a solution of MINTCOVER where each vertex is active in an interval that includes the minimum and maximum timestamp where the vertex is active in the computed solution of MIN-NC-TCOVER.

2.2.2 Heuristics for Minimum Vertex Cover

MINTCOVER is a variant in temporal graphs of MVC, a classical NP-hard optimization problem that consists of, given an undirected unweighted graph $G = (V, E)$, finding a minimum sized subset $S \subseteq V$ such that every edge in G has at least one endpoint in S .

The MVC problem is often addressed by iteratively solving its decision version, which involves identifying a vertex cover of a specified size k . The process begins with the construction of a vertex cover. If a vertex cover of k vertices is found, one vertex is removed, reducing the target size to $k - 1$, and a local search is initiated to find a smaller vertex cover. The current candidate solution, denoted as C , includes the vertices selected for covering. Each vertex $v \in C$ has an associated *loss*, defined as the number of covered edges that would become uncovered if v were removed from C . Conversely, for vertices not in C , a *gain* is calculated based on the number of uncovered edges that would become covered upon their addition. Both loss and gain are used to score vertices, which also have an *age* indicating the time since their status last changed. The iterative process involves swapping vertices in C with those outside of it, a step known as *exchanging step*.

The FASTVC [3] (which in turn is inspired by NUMVC [2]) is a heuristic algorithm, known for its efficiency in handling large graphs, which makes it of practical relevance for real-world applications, where rapid solutions are needed. The algorithm works by following the procedure depicted before, adopting a two-stage exchange method as exchanging step, introduced in the NUMVC algorithm, which consists in firstly removing a vertex from C , then adding a new vertex not in C , updating the scoring properties (loss and gain) at each stage. The great advantage of FASTVC is that to enhance the efficiency and effectiveness of the vertex selection process it mainly relies on the Best from Multiple Selection (BMS) heuristic, which works by generating multiple candidate solutions, choosing the best among them based on a predefined criterion (the one that leads to the smallest vertex cover).

This approach is particularly useful since in problem like MVC the solution space is large and complex, thus direct evaluation of all possible selections would be computationally expensive. By using BMS, FASTVC can more effectively explore the solution space, as it avoids getting trapped in local minima – a common problem in greedy and local search algorithms. It provides a way to balance exploration and exploitation by periodically allowing the algorithm to consider a range of potential moves (vertex additions or removals) rather than being confined to the immediate best move.

More recently, the use of Deep Learning (DL) approaches has been proposed for defining heuristics for NP-hard problems over graphs, like MVC, highlighting that representation learning based approaches are better in building solution to combinatorial optimization over graphs with respect to end-to-end approaches [20]. Some of the most effective research investigation directions are based on using Reinforcement Learning (RL) [13, 1], Graph Neural Networks (GNNs) [16, 14] or the Attention Mechanism [15] by firstly learning a representation of the graph and then by leveraging this representation with autoregressive machine learning-based procedure, local search or greedy search to build the final solution.

3 FastMinTC+

In this section we describe our FASTMINTC+ algorithm, which solves the MINTCOVER problem in an iterative way, following a local search procedure and a heuristic optimization similar to the one adopted by FASTVC [3].

3.1 Overall Algorithm

The FASTMINTC+ overall algorithm (outlined in Algorithm 1) is composed of an *initialization step*, to compute an initial minimal timeline, and an *exchange step*, to reduce the span of the initial computed timeline.

■ **Algorithm 1** FastMinTC+(G , cutoff).

Input: graph $G = (V, E, T)$, *cutoff* time
Output: A minimal timeline cover of G , $\mathcal{T}^* = \{I_u\}_{u \in V}$
 $\mathcal{T}, loss \leftarrow \text{InitializeTC}(G)$
 $gain((v, t)) \leftarrow 0$ for each $(v, t) \notin \mathcal{T}$
 $\mathcal{T}^* \leftarrow \mathcal{T}$
while $elapsed \leq cutoff$ **do**
 if \mathcal{T} covers all edges AND $S(\mathcal{T}) < S(\mathcal{T}^*)$ **then**
 | $\mathcal{T}^* \leftarrow \mathcal{T}$
 end
 $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(v, t) : loss((v, t)) \leq loss((u, t)), \forall (v, t), (u, t) \in \mathcal{T}\}$
 $(v, t) \leftarrow \text{SelectRndVertex}(\mathcal{T}, k)$
 $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(v, t)\}$
 $e \leftarrow$ a random uncovered edge
 $(v, t) \leftarrow$ the endpoint of e with greater gain breaking ties in favor of the one not included in the solution for a larger number of iterations
 $\mathcal{T} \leftarrow \mathcal{T} \cup \{(v, t)\}$
 update loss of vertices $N(v)$ in \mathcal{T} and gain of vertices $N(v)$ not in \mathcal{T}
end

The initialization step is carried out by the *InitializeTC* procedure, which returns an initial timeline which is granted to be minimal, and computes the loss value for each pair $(v, t) \in \mathcal{T}$, which corresponds to the number of edges that would become uncovered, by removing (v, t) from the computed timeline. We remember that the timeline \mathcal{T} is a set of intervals $I_v = [s_v, e_v]$, one for each vertex $v \in V$. Thus, we can consider \mathcal{T} as an ordered list of pairs (v, t) such that, for each vertex $v \in V$, we have exactly two pairs: (v, s_v) , (v, e_v) , with $s_v \leq e_v$. For each vertex in the timeline, thus, the loss is computed only for timestamps

s_v and e_v and not for intermediate timestamps in the interval. After letting computing the initial timeline and the corresponding *loss*, the initialization step computes the *gain*, which is defined for each vertex $(v, t) \notin \mathcal{T}$ as the number of uncovered edges that would become covered by adding (v, t) to the timeline.

The solution is then refined in the exchange step, with a two-stage exchange method, by firstly removing from the timeline the pair (vertex, timestamp) with minimum *loss* from the activity timeline computed and then by randomly reducing the length of the intervals selecting a second pair (vertex, timestamps) to be removed from the activity timeline with the *SelectRndVertex* procedure. Then, the algorithm picks a random uncovered edge e , and chooses the endpoint with greater gain adding it into \mathcal{T} , breaking ties in favor of the endpoint that have not been included in the solution for a larger number of iterations. Note that along with removing or adding a vertex, the *loss* and *gain* values of the vertex and its neighbors are updated accordingly.

3.2 Initialization Algorithm

The initialization procedure is outlined in Algorithm 2 and consists of an *extending phase* and a *shrinking phase*.

■ **Algorithm 2** InitializeTC(G).

```

Input: graph  $G = (V, E, T)$ 
Output: timeline cover of G  $\mathcal{T} = \{I_u\}_{u \in V}$ ,  $loss((v, t))$  for each  $(v, t) \in \mathcal{T}$ 
for  $e \in E$  do
  if  $e$  is uncovered then
    add the timestamp  $t$  to the activity interval of the endpoint of  $e$  with higher
    degree in  $t$ 
  end
end
// initialize loss to 0 (considering only the minimum and maximum  $t$  of each interval
of  $v$  in  $\mathcal{T}$ )
 $loss((v, t)) \leftarrow 0$  for each  $(v, t) \in \mathcal{T}$ 
for  $e \in E$  do
  if only one endpoint of  $e$  belongs to  $\mathcal{T}$  then
    for the endpoint  $(v, t) \in \mathcal{T}$ ,  $loss((v, t)) ++$ 
  end
end
// remove redundant vertices
for  $(v, t) \in \mathcal{T}$  do
  if  $s_v < e_v$  then
    if  $loss((v, t)) = 0$  then
       $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(v, t)\}$ 
      update loss of vertices  $N(v)$  in  $\mathcal{T}$ 
    end
  end
end

```

During the extending phase, the procedure begins with an empty set \mathcal{T} , progressively augmented by evaluating and incorporating edges one at a time. If an edge e is found to be uncovered, the endpoint of e with higher degree is added to \mathcal{T} . It is straightforward that

at the end of this phase we obtain a timeline cover. The shrinking phase consists in first computing the loss value of pairs (v, t) in \mathcal{T} , only for the initial and final timestamps of each interval (thus, we do not compute the loss for the intermediate timestamps in the interval, i.e., $\forall I_u = [s_u, e_u] \in \mathcal{T}$, we compute the loss only for (u, s_u) and (u, e_u)). This phase possibly shrinks an interval length, by removing the initial or final timestamp, if the loss is equal to 0. When computing the *loss* of (v, t) with respect to an edge (u, v, t) , we consider the entire interval on vertex u , i.e., $I_u = [s_u, e_u]$. If it holds that $s_u < t < e_u$, we do not increment the *loss* value of v since the edge (u, v, t) is already covered.

After computing the loss, the algorithm checks if there are vertices in the timeline with *loss* = 0. If it finds a (v, t) with *loss* = 0 and $I_v = [s_v, e_v]$ is such that $s_v < e_v$, then I_v is shrunked with a new timestamp r as follows: if $t = s_v$, s_v is replaced by r , which is the first timestamp in $T(v)$ greater than s_v (which may not necessary be $s_v + 1$ if v is not active in $s_v + 1$); if $t = e_v$, e_v is replaced by r , which is the next timestamp in $T(v)$ smaller than e_v (which may not necessary be $e_v - 1$ if v is not active in $e_v - 1$).

Subsequently *loss* of vertices in $N((v, t))$ is increased by 1 and *loss* of vertex (v, r) is computed considering the vertex in $N((v, r))$.

This algorithm grants to return a *minimal* solution according to Theorem 4 (proof is provided in Appendix B). A timeline cover is minimal if removing any (v, t) would make it not a timeline cover, thus the *loss* $(v, t) > 0$, $\forall (v, t) \in \mathcal{T}$.

► **Theorem 4.** *The timeline $\mathcal{T} = \{I_u\}_{u \in V}$ returned by the *initializeTC* algorithm is a minimal timeline cover.*

3.3 Random Vertex Selection Algorithm

A critical function for FASTMINTC+ is *SelectRndVertex*, outlined in Algorithm 3, which chooses a vertex from the candidate vertex set \mathcal{T} to remove.

■ **Algorithm 3** *SelectRndVertex*(\mathcal{T} , k).

Input: A timeline \mathcal{T} , a parameter k
Output: an element of \mathcal{T}
 $best \leftarrow$ a random vertex (v, t) from \mathcal{T}
for 1 to $k-1$ **do**
 $tmp \leftarrow$ a random vertex (u, r) from \mathcal{T}
 if $loss(tmp) < loss(best)$ **then**
 $best \leftarrow tmp$
 end
end
return $best$

This algorithm follows the structure of the cost-effective BMS heuristic, which picks k elements (where k is a parameter) randomly with replacement from the set \mathcal{T} , and then returns the one with lowest *loss* value. The set \mathcal{T} is composed of all the pairs (v, t) that belong to the activity timeline \mathcal{T} and whose timestamps are an initial or ending timestamp of an interval. In this way, we only remove from the timeline vertices which are at the beginning or end of the interval, with the goal of reducing the length of the activity timelines avoiding producing multiple intervals for the same vertex. From hence, Theorem 5 holds (proof is provided in Appendix C).

► **Theorem 5.** *With $k \geq 50$, the probability that the *SelectRndVertex* algorithm choose a vertex whose loss value is not greater than 90% vertices in \mathcal{T} is greater or equal to 0.9948.*

3.4 Heuristic Complexity Analysis

The overall FASTMINTC+ heuristic (outlined in Algorithm 1), as seen, is composed of two main parts: the initialization step (i.e., the *InitializeTC* procedure outlined in Algorithm 2, followed by the *gain* initialization) and the exchange step (i.e., the iterative cycle carried out for an *elapsed* time smaller or equal to a given *cutoff* time, that uses the *SelectRndVertex* procedure, outlined in Algorithm 3 as two-stage exchange method). Thus, the complexity of the FASTMINTC+ heuristic relies on the complexity of Algorithms 2 and 3.

In the following we denote $n = |V|$, $m = |E|$ and $t = |\mathcal{T}|$.

Let us first focus on the complexity of the *InitializeTC* algorithm. This can be divided in three parts: the extending phase, the initialization of the *loss* values and the shrinking phase. It is straightforward that the complexity of the extending phase is $O(m)$. Indeed, to compute the extending phase, it is necessary to scan the set of edges E one time, adding, for every scanned edge, the endpoint with higher degree. As far as *loss* computation is concerned, it depends on the number of vertices $(v, t) \in \mathcal{T}$. Since we add, for each vertex $v \in V$ exactly two timestamps (the start and end of the activity interval, namely, s_v, e_v), the dimension of the computed activity timeline is exactly $|\mathcal{T}| = 2n$. It follows that the complexity for the initialization of the loss is $O(n)$. In order to update the *loss* values, we must check if an edge is covered by one of two vertices of the timeline; we need to scan the set of edges E , the complexity is $O(m)$. Thus, the overall complexity of *loss* initialization is $O(n + m)$. For the shrinking phase, the complexity depends on the number of updates performed over the *loss* values. Since each vertex is updated at most once for each edge incident in it, the total number of possible updates is bounded to the sum of the global degrees of each vertex, i.e., $\sum_{v \in V} \text{deg}(v) = 2m$, thus the complexity of the shrinking phase is $O(m + n)$. Therefore, the overall complexity of the *InitializeTC* algorithm is $O(m + n)$.

Let's now move to the *SelectRndVertex* procedure. Since the algorithm only performs comparison operation for pair of elements at a time for each iteration, the complexity is $O(k)$, where k is the number of iteration. Since k is constant, the overall complexity is $O(1)$.

Consider now the FASTMINTC+ heuristic. From the previous analysis, it holds that the computational complexity of the initialization phase, i.e., the one derived from the *InitializeTC* algorithm, is $O(m + n)$. For the exchange step, the complexity can be derived by: 1) checking whether \mathcal{T} covers all edges, which can be done summing the *gain* values of $(v, t) \notin \mathcal{T}$ with complexity $O(n)$: if the sum is 0, it means that \mathcal{T} is a minimal cover; 2) getting the vertex (v, t) with lower *loss* value, which can be done with complexity $O(n)$; 3) select a random vertex with the *SelectRndVertex* procedure, which has complexity $O(1)$; 4) extract a random uncovered edge e and add the endpoint of e with greater *gain*, breaking ties in favor of the older one, which has the same complexity as the *loss* update operation, thus $O(m + n)$; 5) update *loss* and *gain* values, which have both complexity $O(m)$. Thus, the overall time complexity of the exchange step is $O(m + n)$, leading to an overall time complexity of the FASTMINTC+ heuristic equals to $O(m + n)$.

4 Performance Evaluation

In this section we provide the results of our experiments on FASTMINTC+. We evaluate it against state-of-the-art approaches on both synthetic and real world graphs.

4.1 Dataset Description

In the current literature, there are no publicly available datasets for the MINTCOVER problem. The sole study addressing experimental analysis on real and synthetic instances is presented in [25]. However, this study has significant limitations: firstly, the method proposed for generating synthetic datasets does not ensure that the computed ground truth is optimal, complicating the assessment of the algorithm’s performance against exact solvers; secondly, the real-world dataset employed is not publicly accessible, precluding reproducibility of the results. To address these gaps, we have developed three distinct datasets, each designed for specific evaluative purposes:

- *Dataset1* consists of 264 synthetically generated instances of sparse temporal graphs, characterized by a low temporal edge-to-vertex ratio ($|E| = O(|V| + |T|)$). These graphs vary considerably, with vertex counts $|V| = [10, 10000]$, timestamp sets $T = [4, 5000]$, and edges $|E| = [10, 1000000]$. This dataset facilitates comparisons between the proposed heuristic and state-of-the-art algorithms across sparse graph scenarios, which reflect conditions found in many real-world datasets. For more granular analysis, instances are categorized as *small* (up to 50 vertices and 20 timestamps), *medium* (up to 500 vertices and 500 timestamps), and *hard* (up to 10000 vertices and 5000 timestamps).
- *Dataset2* includes 195 synthetically generated instances of dense temporal graphs ($|E| \gg O(|V| + |T|)$). These instances also range in size with vertices $|V| = [10, 10000]$, timestamps $T = [2, 5000]$, and edges $|E| = [100, 20000000]$. The purpose of *Dataset2* mirrors that of *Dataset1*, but focuses on denser graphs. Similar to *Dataset1*, it is divided into *small* (up to 30 vertices and 4 timestamps), *medium* (up to 1000 vertices and 100 timestamps), and *hard* categories (up to 10000 vertices and 5000 timestamps) to account for scalability concerns.
- *Dataset3* comprises 25 publicly available instances of temporal benchmark graphs sourced from the DIMACS repository [22]. This dataset was curated to include a diverse array of graphs in terms of edge count $|E|$, vertex count $|V|$ and timestamp range $|T|$, thus resulting in a set of graphs with densities D that ranges from $8,49 \text{ E-}10$ to $4,74 \text{ E-}01$. The primary aim of *Dataset3* is to evaluate the performance of the proposed heuristic against state-of-the-art algorithms on benchmark graphs, thereby assessing their applicability to practical scenarios.

4.2 Experimental Results

To evaluate our heuristic FASTMINTC+ we benchmark it against both an optimal solution derived from the ILP formulation (Equation 1 in Appendix A) and the principal approximation algorithms discussed in Section 2.2.1. The compared algorithms include the iterative method for inner points (INNER) as outlined in [25], and the two-phases approach (2PHASES) introduced in [7]. Our evaluations span three datasets, assessing both the quality (measured by the length of the sum-span of the timelines $S(\mathcal{T})$) and scalability (determined by execution time) of each solution. The algorithmic comparison varies by dataset and instance complexity; for *Dataset1* and *Dataset2*, all four algorithms are evaluated on small instances. However, the ILP solution is omitted from medium instances and both the ILP and 2PHASES are excluded from hard instances due to computational limitations. Only INNER and FASTMINTC+ are analyzed for *Dataset3* due to their superior performance.

In running the test, for the FASTMINTC+ algorithm, we set the parameter $k = 50$ and instead of setting a time limit, we define a number of 2000 iteration for the exchange step. The algorithm is executed 5 times each with a different shuffling of the edges, and the best result is reported. For the INNER algorithm we set the number of iterations to 10 (as suggested by the authors). No parameters have to be set for the 2PHASES approximation.

We implement our algorithm FASTMINTC+ in the Python programming language (version 3.12). The code for INNER is open-source and implemented in Python (version 2) [23]. For the 2PHASES algorithm no implementation were available online, thus, we implement it in Python (version 3.12). Experiments are carried out on a MacBook Pro (2017) under MacOS, using a 16GB RAM and 4 cores of a i7-3,1 GHz CPU.

4.2.1 Sparse Instances Results

The comprehensive results for *Dataset1*, consisting of sparse graphs, are summarized in Table 1. FASTMINTC+ consistently outperforms both INNER and 2PHASES in small instances. Specifically, compared to the sum-span calculated by the ILP and other algorithms, FASTMINTC+ approximates the ILP-derived solution more closely than its competitors. Among the 189 simple instances in *Dataset1*, FASTMINTC+ provides a superior solution (i.e., a shorter sum-span) compared to INNER 177 times and to 2PHASES 154 times. In the subset of 25 medium instances, FASTMINTC+ continues to outperform, invariably offering better solutions than 2PHASES and surpassing INNER in 14 instances. Similar trends are observed in the 50 hard instances, where FASTMINTC+ excels over INNER 34 times.

The fact that 2PHASES’s performance downgrade with larger instances, aligns with the fact that it provides an approximate factor proportional to the number of timestamps.

A notable finding from this experiment pertains to the average execution times. Clearly, the ILP algorithm, while providing optimal solutions, is significantly slower even for basic sparse instances. 2PHASES encounters scalability issues, notably in hard instances where experiments could not be conducted due to prohibitive execution times. Conversely, FASTMINTC+ is faster, even in large sparse instances, typically requiring an order of magnitude less time to compute the solution compared to INNER.

Thus, although FASTMINTC+ generally outperforms INNER and 2PHASES, it may not be invariably the superior choice for sparse graphs, unless graph size increases to a point where INNER becomes computationally impractical.

■ **Table 1** Experimental Results on Dataset1 - small, medium and hard instances. Results show that the proposed heuristic produces for sparse graphs better results with respect to the approximate solutions in a smaller time.

Dataset1	Algorithm	Average Execution Time	Average Sum Span
Small Instances	ILP	1,13 E-01	1,01 E+00
	2Phases	8,68 E-02	3,37 E+00
	Inner	9,37 E-04	7,05 E+00
	FastMinTC+	5,39 E-04	3,02 E+00
Medium Instances	2Phases	3,52 E+01	80,1 E+03
	Inner	1,21 E-01	71,1 E+03
	FastMinTC+	4,32 E-02	70,9 E+03
Hard Instances	Inner	1,75 E+02	16,01 E+06
	FastMinTC+	1,43 E+01	15,95 E+06

4.2.2 Dense Instances Results

The comprehensive results for *Dataset2*, composed of dense graphs, are summarized in Table 2. The findings from evaluations of small, medium, and hard instances are parallel to those observed in *Dataset1*, reinforcing the quality and the scalability of the proposed heuristic.

20:12 FastMinTC+: An Heuristic for the MinTCover Problem

Notably, the difference in average sum-span and execution time between FASTMINTC+ on both approximate and exact solutions is more pronounced here than in *Dataset1*. Indeed, as far as sum-span is concerned, in all 195 test cases assessed, INNER only outperformed FASTMINTC+ once in a small instance scenario, while as far as execution time is concerned, FASTMINTC+ always outperforms INNER, reaching, on average, a larger delta with respect to tests on *Dataset1*, with a maximum delta of 3492,17 seconds on the instance with 10000 vertices and 5000 timestamps. These results validate FASTMINTC+'s capability to effectively scale to larger instances. Moreover, they suggest a distinct advantage in favor of FASTMINTC+ when handling dense graphs, indicating its overall preferable performance relative to INNER in such contexts.

■ **Table 2** Experimental Results on Dataset2 - small, medium and hard instances. Results confirm that the proposed solution produces for dense graphs better results with respect to the approximate solutions in a smaller time.

Dataset2	Algorithm	Average Execution Time	Average Sum Span
Small Instances	ILP	9,62 E+01	8,24 E+00
	2Phases	3,29 E-01	27,85 E+00
	Inner	2,68 E-03	20,39 E+00
	FastMinTC+	5,73 E-04	11,88 E+00
Medium Instances	2Phases	8,54 E+00	28,69 E+03
	Inner	5,81 E-01	28,38 E+03
	FastMinTC+	9,81 E-02	27,08 E+03
Hard Instances	Inner	8,96 E+02	14,74 E+06
	FastMinTC+	1,82 E+01	14,05 E+06

4.2.3 Real World Instances Results

Experiments conducted on real-world graphs corroborate the findings observed in the synthetic test instances too. Detailed results for these real-world instances are presented in Tables 3 and 4. For each graph considered, Table 3 provides the results in terms of sum-span, while Table 4 provides the results in terms of execution time.

Overall, FASTMINTC+ demonstrates superior performance in terms of sum-span compared to INNER on non-sparse graphs. Specifically, in the analysis of 25 real-world instances, FASTMINTC+ achieves a better sum-span in 17 cases. Notably, the instances where INNER outperforms FASTMINTC+ are characterized by particularly low densities, with a maximum density value of 6,27 E-04 and an average density value of 1,09 E-04. In contrast, the instances where FASTMINTC+ outperforms INNER are characterized by a maximum density value of 4,47 E-01 and an average density value of 1,26 E-01. It is also worth noting that there are, albeit rarer, cases of graphs with very low densities (such as *ia-contacts-dublin* with $D = 8.98E - 08$) where FASTMINTC+ outperforms INNER, while there are no cases of dense graphs where INNER outperforms FASTMINTC+. This confirms that, while FASTMINTC+ should be always preferable in the case of dense graphs, it should be taken into account also in the case of sparse graphs, representing a valid alternative to INNER even in these cases.

The results also confirm the best performance in terms of execution time of the FASTMINTC+, which always achieves a lower execution time compared to INNER.

■ **Table 3** Experimental Results on real-world publicly available graphs from Dataset3. For each considered graph we report the information on the size of the network and the results in terms of sum-span $S(\mathcal{T})$ of INNER and FASTMINTC+.

Graph	$ E $	$ V $	$ T $	D	Inner $S(\mathcal{T})$	FMTC+ $S(\mathcal{T})$
aves-sparrow-social	516	52	1	3,89E-01	1,40E+01	1,00E+01
aves-wildbird-network	11900	202	5	1,17E-01	5,88E+02	4,78E+02
copresence-InVS13	394247	95	20128	4,39E-03	1,53E+06	1,49E+06
copresence-InVS15	1283194	219	21535	2,50E-03	3,71E+06	3,64E+06
copresence-LH10	150126	73	12604	4,53E-03	3,97E+05	3,87E+05
copresence-LyonSchool	6594492	242	3123	7,24E-02	7,02E+05	6,64E+05
copresence-SFHH	1417485	403	3148	5,56E-03	9,26E+05	8,92E+05
copresence-Thiers13	18613039	328	8937	3,88E-02	2,57E+06	2,45E+06
email-dnc	39264	1892	19382	1,13E-06	2,54E+05	5,84E+05
fb-wosn-friends	1269502	63731	736674	8,49E-10	8,99E+08	9,83E+08
ia-contacts-dublin	415912	10972	76943	8,98E-08	1,00E+06	9,60E+05
ia-contacts-hypertext2009	20818	113	5245	6,27E-04	3,50E+05	3,93E+05
ia-digg-reply	87627	30398	83942	2,26E-09	1,41E+08	1,76E+08
ia-hospital-ward-proximity	32424	75	9452	1,24E-03	3,26E+05	3,18E+05
ia-primary-school-proximity	125773	242	3099	1,39E-03	6,51E+05	6,30E+05
a-prosper-loans	3394979	89269	1258	6,77E-07	9,78E+05	1,00E+06
a-retweet-pol	61157	18470	60500	5,93E-09	1,06E+07	1,21E+07
insecta-ant-colony1	111578	113	40	4,41E-01	3,46E+03	3,25E+03
insecta-ant-colony3	241280	160	40	4,74E-01	5,07E+03	4,77E+03
insecta-ant-colony5	194317	152	40	4,23E-01	4,36E+03	4,14E+03
mammalia-raccoon-proximity	1997	24	51	1,42E-01	7,73E+02	7,61E+02
rec-amz-Baby	915446	596316	4868	1,06E-09	2,80E+06	2,91E+06
reptilia-tortoise-network-bsv	554	136	3	2,01E-02	9,60E+01	5,00E+01
reptilia-tortoise-network-fi	1713	787	8	6,92E-04	5,5E+02	3,12E+02
SFHH-conf-sensor	70261	403	3508	2,47E-04	7,11E+05	8,27E+05

■ **Table 4** Experimental Results on real-world publicly available graphs from Dataset3. For each considered graph we report the information on the size of the network and the results in terms of execution time of INNER and FASTMINTC+.

Graph	E	V	T	D	Inner elapsed	FMTC+ elapsed
aves-sparrow-social	516	52	1	3,89E-01	0,01	0,00
aves-wildbird-network	11900	202	5	1,17E-01	0,09	0,05
copresence-InVS13	394247	95	20128	4,39E-03	2,52	1,07
copresence-InVS15	1283194	219	21535	2,50E-03	8,50	3,47
copresence-LH10	150126	73	12604	4,53E-03	1,00	0,39
copresence-LyonSchool	6594492	242	3123	7,24E-02	43,65	15,92
copresence-SFHH	1417485	403	3148	5,56E-03	9,59	3,18
copresence-Thiers13	18613039	328	8937	3,88E-02	114,17	43,69
email-dnc	39264	1892	19382	1,13E-06	0,39	0,13
fb-wosn-friends	1269502	63731	736674	8,49E-10	11,36	7,19
ia-contacts-dublin	415912	10972	76943	8,98E-08	3,38	1,12
ia-contacts-hypertext2009	20818	113	5245	6,27E-04	0,13	0,06
ia-digg-reply	87627	30398	83942	2,26E-09	0,98	1,49
ia-hospital-ward-proximity	32424	75	9452	1,24E-03	0,21	0,09
ia-primary-school-proximity	125773	242	3099	1,39E-03	0,80	0,32
a-prosper-loans	3394979	89269	1258	6,77E-07	36,98	23,71
a-retweet-pol	61157	18470	60500	5,93E-09	0,73	0,77
insecta-ant-colony1	111578	113	40	4,41E-01	0,66	0,31
insecta-ant-colony3	241280	160	40	4,74E-01	1,39	0,61
insecta-ant-colony5	194317	152	40	4,23E-01	1,12	0,50
mammalia-raccoon-proximity	1997	24	51	1,42E-01	0,01	0,01
rec-amz-Baby	915446	596316	4868	1,06E-09	18,16	12,53
reptilia-tortoise-network-bsv	554	136	3	2,01E-02	0,14	0,64
reptilia-tortoise-network-fi	1713	787	8	6,92E-04	0,02	0,01
SFHH-conf-sensor	70261	403	3508	2,47E-04	0,45	0,19

5 Conclusion and Future Works

This research tackled the complex problem of summarizing temporal networks, aiming to optimize the timeline cover for entities within a temporal graph. We introduced a novel heuristic approach, FASTMINTC+, which significantly advances the field by offering a computationally feasible solution to the MINTCOVER problem. Our method leverages low-complexity approximate heuristics, shown to be very effective on the related MVC problem on static networks, enabling the effective processing of massive graphs, a notable improvement over existing methodologies.

Experimental results, on both synthetic and real-world datasets, demonstrates that FASTMINTC+ achieves superior performance compared to state-of-the-art algorithms, enhancing computational efficiency and maintaining a high level of accuracy and reliability in identifying sub-optimal timeline covers. These results underscore the potential of our heuristic to facilitate deeper insights into temporal data analysis.

Looking ahead, the integration of DL techniques presents a promising avenue for further enhancing the efficacy of our heuristic. Specifically, exploring the synergy between our heuristic approaches and DL models could yield innovative strategies for tackling the MINTCOVER problem. These developments are poised to redefine the boundaries of temporal network analysis, opening up new possibilities for both theoretical and practical applications in the field.

References

- 1 Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL: <https://openreview.net/forum?id=Bk9mx1SfX>.
- 2 Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 747–753. AAAI Press, 2015. URL: <http://ijcai.org/Abstract/15/111>.
- 3 Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. Numvc: An efficient local search algorithm for minimum vertex cover. *J. Artif. Intell. Res.*, 46:687–716, 2013. doi:10.1613/JAIR.3907.
- 4 Riccardo Dondi. Untangling temporal graphs of bounded degree. *Theor. Comput. Sci.*, 969:114040, 2023. doi:10.1016/J.TCS.2023.114040.
- 5 Riccardo Dondi and Manuel Lafond. An FPT algorithm for temporal graph untangling. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPICs*, pages 12:1–12:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.IPEC.2023.12.
- 6 Riccardo Dondi and Alexandru Popa. Timeline cover in temporal graphs: Exact and approximation algorithms. In Sun-Yuan Hsieh, Ling-Ju Hung, and Chia-Wei Lee, editors, *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7-10, 2023, Proceedings*, volume 13889 of *Lecture Notes in Computer Science*, pages 173–184. Springer, 2023. doi:10.1007/978-3-031-34347-6_15.
- 7 Riccardo Dondi and Alexandru Popa. Exact and approximation algorithms for covering timeline in temporal graphs. *Annals of Operations Research*, April 2024. doi:10.1007/s10479-024-05993-8.

- 8 Vincent Froese, Pascal Kunz, and Philipp Zschoche. Disentangling the computational complexity of network untangling. *CoRR*, abs/2204.02668, 2022. doi:10.48550/arXiv.2204.02668.
- 9 Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.*, 11(3):555–556, 1982. doi:10.1137/0211045.
- 10 Petter Holme and Jari Saramäki. Temporal networks. *CoRR*, abs/1108.1780, 2011. arXiv:1108.1780.
- 11 Yuriy Hulovaty, Huili Chen, and Tijana Milenkovic. Exploring the structure and function of temporal networks with dynamic graphlets. *Bioinform.*, 32(15):2402, 2016. doi:10.1093/BIOINFORMATICS/BTW310.
- 12 George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Trans. Algorithms*, 5(4):41:1–41:8, 2009. doi:10.1145/1597036.1597045.
- 13 Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6348–6358, 2017. URL: <https://proceedings.neurips.cc/paper/2017/hash/d9896106ca98d3d05b8cbdf4fd8b13a1-Abstract.html>.
- 14 Kenneth Langedal, Johannes Langguth, Fredrik Manne, and Daniel Thilo Schroeder. Efficient minimum weight vertex cover heuristics using graph neural networks. In Christian Schulz and Bora Uçar, editors, *20th International Symposium on Experimental Algorithms, SEA 2022, July 25-27, 2022, Heidelberg, Germany*, volume 233 of *LIPICs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SEA.2022.12.
- 15 Giorgio Lazzarinetti, Riccardo Dondi, Sara Manzoni, and Italo Zoppis. An attention-based method for the minimum vertex cover problem on complex networks. *Algorithms*, 17(2):72, 2024. doi:10.3390/A17020072.
- 16 Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 537–546, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/8d3bba7425e7c98c50f52ca1b52d3735-Abstract.html>.
- 17 Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph summarization methods and applications: A survey. *ACM Comput. Surv.*, 51(3):62:1–62:34, 2018. doi:10.1145/3186727.
- 18 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.
- 19 Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. Motifs in temporal networks. *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2016. URL: <https://api.semanticscholar.org/CorpusID:13332080>.
- 20 Yun Peng, Byron Choi, and Jianliang Xu. Graph learning for combinatorial optimization: A survey of state-of-the-art. *Data Sci. Eng.*, 6(2):119–141, 2021. doi:10.1007/S41019-021-00155-3.
- 21 Anna-Kaisa Pietiläinen and Christophe Diot. Dissemination in opportunistic social networks: the role of temporal communities. In *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '12*, pages 165–174, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2248371.2248396.
- 22 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL: <https://networkrepository.com>.
- 23 P. Rozenshtein. the-network-untangling-problem. <https://github.com/polinapolina/the-network-untangling-problem/tree/master>, 2020. [Online; accessed 13-June-2024].

- 24 Polina Rozenshtein, Francesco Bonchi, Aristides Gionis, Mauro Sozio, and Nikolaj Tatti. Finding events in temporal networks: segmentation meets densest subgraph discovery. *Knowl. Inf. Syst.*, 62(4):1611–1639, 2020. doi:10.1007/S10115-019-01403-9.
- 25 Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. The network-untangling problem: from interactions to activity timelines. *Data Min. Knowl. Discov.*, 35(1):213–247, 2021. doi:10.1007/S10618-020-00717-5.
- 26 Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1055–1064. ACM, 2015. doi:10.1145/2783258.2783321.
- 27 Bianca Wackersreuther, Peter Wackersreuther, Annahita Oswald, Christian Böhm, and Karsten M. Borgwardt. Frequent subgraph discovery in dynamic networks. In Ulf Brefeld, Lise Getoor, and Sofus A. Macskassy, editors, *Proceedings of the Eighth Workshop on Mining and Learning with Graphs, MLG '10, Washington, D.C., USA, July 24-25, 2010*, pages 155–162. ACM, 2010. doi:10.1145/1830252.1830272.

A ILP formulation for MinTCover

The MINTCOVER problem can be formulated as an ILP mainly considering a variable $x_{u,t} \in \{0, 1\}$ whose value is 1 if t is included in the activity interval of u , 0 otherwise. With this variable we can formulate the following constraints:

- *Edge Coverage*: For each edge $(u, v, t) \in E$, at least one between u and v must have t in its activity interval. This can be expressed as $x_{u,t} + x_{v,t} \geq 1 \forall (u, v, t) \in E$.
- *Activity Interval definition*: for each vertex u and for each timestamp t , if t is included in the activity interval of u , then the following should hold: $s_u \leq t \leq e_u$. This means that if $x_{u,t} = 1$, then $s_u \leq t$ and $e_u \geq t$. This can be expressed, by adding a constant $M \geq \max(T)$, with two conditions, one over s_u and one over e_u as follows:

1. $s_u \leq t * x_{u,t} + M(1 - x_{u,t})$
2. $e_u \geq t * x_{u,t} - M(1 - x_{u,t})$

Indeed, if $x_{u,t} = 0$, it holds that $M \geq s_u$ and $-M \leq e_u$, otherwise it holds that $s_u \leq t$ and $e_u \geq t$.

- *Length of the Interval*: for each vertex u , the length of the activity interval is by definition $\delta(I_u) = e_u - s_u$. This is the objective function to be minimized. For simplicity, we can define $\delta(I_u) = d_u$.

With these variables and constraints we can formulate the ILP as in Equation 1.

$$\begin{aligned}
 & \text{minimize} && \sum_{u \in V} d_u \\
 & \text{subject to} && x_{u,t} + x_{v,t} \geq 1 && \forall (u, v, t) \in E \\
 & && s_u \leq t * x_{u,t} + M(1 - x_{u,t}) && \forall u \in V, \quad \forall t \in T \\
 & && e_u \geq t * x_{u,t} - M(1 - x_{u,t}) && \forall u \in V, \quad \forall t \in T
 \end{aligned} \tag{1}$$

B InitializeTC's minimality: Proof

Theorem 4 states that the timeline $\mathcal{T} = \{I_u\}_{u \in V}$ returned by the *initializeTC* algorithm, outlined in Algorithm 2, is a minimal timeline cover.

Consider the timeline $\mathcal{T} = \{I_u\}_{u \in V}$ produce as output by the extending phase, which may be modified in the shrinking phase. By construction, at the end of the extending phase \mathcal{T} is a timeline cover. We now prove Theorem 4 by showing that: 1) the timeline \mathcal{T} returned by the procedure is a timeline cover; 2) loss value of any vertex in the timeline \mathcal{T} does not decrease; 3) the timeline \mathcal{T} returned after the shrinking phase is minimal.

Proof.

- 1) Suppose to perform i^{th} iteration of the shrinking phase. Note that, if \mathcal{T} is a timeline cover at the i^{th} iteration, it is a timeline cover also at the $(i+1)^{\text{th}}$ iteration. Indeed, if $\text{loss}(v, t)_{i+1} > 0$, then \mathcal{T} does not change; if $\text{loss}(v, t)_{i+1} = 0$, then (v, t) is removed, but according to the definition of loss , removing such vertex (v, t) would not generate any new uncovered edge. From hence, since \mathcal{T} is a timeline cover at iteration 0, it is a timeline cover also the end of the shrinking phase (i^{th} iteration).
- 2) Notice that during the shrinking phase, the loss value of any vertex (v, t) in \mathcal{T} does not decrease. Indeed, if at the i^{th} iteration $\text{loss}(v, t)_i > 0$ the iteration does nothing, thus its loss value does not decrease; if $\text{loss}(v, t)_i = 0$, then all vertices in $N(v)$ belong to \mathcal{T} , otherwise, if at the i^{th} iteration there exists a vertex $(u, t) : (u, v, t) \in E$ and $(u, t) \notin \mathcal{T}$, by definition $\text{loss}(v, t)_i$ would be at least 1. So, in case $\text{loss}(v, t)_i = 0$, (v, t) is removed and along with that, the loss value of each vertex $u \in N(v)$ is increased by one as, after this iteration, the removal of u would make the edge (u, v, t) from covered to uncovered. The loss of all the vertices not in $N(v)$ do not change.
- 3) Suppose now that after the shrinking phase, there exists a vertex (v, t) in \mathcal{T} whose removal keeps \mathcal{T} a timeline cover. Supposing that the shrinking phase takes i iteration, from the assumption we can say that $\text{loss}(v, t)_i = 0$, by definition of loss . Since the loss value of any vertex (v, t) in \mathcal{T} does not decrease according to point 2 of this proof, the value of $\text{loss}(v, t)$ at the i^{th} iteration is at most 0, thus, since loss values are non-negative it is exactly 0. Therefore, (v, t) would have been removed at the i^{th} iteration. This complete the proof by contradiction. \blacktriangleleft

C SelectRndVertex Quality: Proof

Theorem 5 states that with $k \geq 50$, the probability that the SelectRndVertex algorithm, outlined in Algorithm 3, choose a vertex whose loss value is not grater than 90% vertices in \mathcal{T} is grater or equal to 0.9948.

Proof. Consider a real number $\rho \in (0, 1)$, the probability of the event $E = \{\text{the loss value of the element chosen by the SelectRndVertex algorithm is not greater than } \rho|\mathcal{T}| \text{ elements in the set } \mathcal{T}\}$ is $Pr(E) \geq 1 - (\frac{\rho|\mathcal{T}|-1}{|\mathcal{T}|})^k > 1 - \rho^k$.

Indeed, $\rho|\mathcal{T}|$ is the number of elements in the ρ fraction of \mathcal{T} with the worst losses (i.e., higher loss). Thus, considering that each random selection from \mathcal{T} is independent, $\frac{\rho|\mathcal{T}|}{|\mathcal{T}|}$ represents the chance of selecting one of the worst $\rho|\mathcal{T}|$ elements in one selection. It follows that $\frac{\rho|\mathcal{T}|-1}{|\mathcal{T}|}$ is the probability of selecting an element whose loss value is higher or equal than $(1 - \rho)|\mathcal{T}|$ elements. After k iterations, $(\frac{\rho|\mathcal{T}|-1}{|\mathcal{T}|})^k$ is the probability that all the iterations randomly select elements inside of the worst elements set. From hence $1 - (\frac{\rho|\mathcal{T}|-1}{|\mathcal{T}|})^k > 1 - \rho^k$ is the probability that at least one of the k iterations selects an element from within the top elements set.

This means that for $k = 50$, the probability that the SelectRndVertex algorithm choose a vertex whose loss value is not greater than $\rho = 90\%$ vertices in \mathcal{T} is $Pr(E) \geq 1 - 0.9^{50} > 0.9948$, where we consider the probability of event E to be greater or equal to the computed value, since there might be the case that more than one elements in those $\rho|\mathcal{T}|$ elements have the same loss value, which is the minimum among loss values of all the $\rho|\mathcal{T}|$ element. \blacktriangleleft