29th International Symposium on Temporal Representation and Reasoning

TIME 2022, November 7–9, 2022, Virtual Conference

Edited by

Alexander Artikis
Roberto Posenato
Stefano Tonetta

LIPICS

Editors

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

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

Stefano Tonetta
FBK, Italy
tonettas@fbk.eu

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

ISBN 978-3-95977-262-4

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/978-3-95977-262-4.

Publication date
November, 2022

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

Digital Object Identifier: 10.4230/LIPIcs.TIME.2022.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Alexander Artikis, Roberto Posenato, and Stefano Tonetta 0:vii

TIME Steering Committee
.... 0:ix

Program Committee Members
.... 0:xi

List of Authors
.... 0:xiii

Invited Talks

Linear Temporal Logic: From Infinite to Finite Horizon
Moshe Y. Vardi 1:1–1:1

Visual Analytics Meets Temporal Reasoning: Challenges and Opportunities
Silvia Miksch 2:1–2:1

Getting to the CORE of Complex Event Recognition
Stijn Vansummeren 3:1–3:2

Regular Papers

Early Detection of Temporal Constraint Violations
Isaac Mackey, Raghubir Chimni, and Jianwen Su 4:1–4:16

The Tail-Recursive Fragment of Timed Recursive CTL
Florian Bruse, Martin Lange, and Etienne Lozes 5:1–5:16

Decentralised Runtime Verification of Timed Regular Expressions
Victor Roussanaly and Yliès Falcone 6:1–6:18

Logical Forms of Chronicles
Thomas Guyet and Nicolas Markey 7:1–7:15

Realizability Problem for Constraint LTL
Ashwin Bhaskar and M. Praveen 8:1–8:19

Reasoning on Dynamic Transformations of Symbolic Heaps
Nicolas Peltier 9:1–9:20

Gabbay Separation for the Duration Calculus
Dimitar P. Guelev 10:1–10:14

A Quantitative Extension of Interval Temporal Logic over Infinite Words
Laura Bozzelli and Adriano Peron 11:1–11:16

A Neuro-Symbolic Approach for Real-World Event Recognition from Weak Supervision
Gianluca Apriceno, Andrea Passerini, and Luciano Serafini 12:1–12:19

Neural-Symbolic Temporal Decision Trees for Multivariate Time Series Classification
Giovanni Pagliarini, Simone Scaboro, Giuseppe Serra, Guido Sciavicco, and Ionel Eduard Stan 13:1–13:15

Taming Strategy Logic: Non-Recurrent Fragments
Massimo Benerecetti, Fabio Mogavero, and Adriano Peron 14:1–14:16

Giving Instructions in Linear Temporal Logic
Julian Gutierrez, Sarit Kraus, Giuseppe Perelli, and Michael Wooldridge 15:1–15:14

Preface

Like TIME 2021, the 29th edition of the International Symposium on Temporal Representation and Reasoning (TIME 2022) will take place online, due to the insecurities caused by the pandemic. Since its first edition in 1994, the TIME Symposium is quite unique in the panorama of the scientific conferences as its main goal is to bring together researchers from distinct research areas involving the management and representation of temporal data as well as reasoning about temporal aspects of information. Moreover, the TIME Symposium aims to bridge theoretical and applied research, as well as to serve as an interdisciplinary forum for exchange among researchers from the areas of artificial intelligence, database management, logic and verification and beyond.

The three traditional tracks of TIME concern:

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

This year the authors of the top-ranked papers will be invited to submit an extended version of their contribution to a special issue in Information Systems or Information and Computation.

We received a total of 29 paper submissions representing a wide range of research topics in the areas of artificial intelligence, databases and theoretical computer science. Submissions came from Africa, Asia, Australia, Europe and North America. We would like to thank all the authors of the submitted papers, as they have helped to build a successful TIME 2022 Symposium.

As a result of the review process coordinated by the Program Committee chairs, 12 papers were selected for full presentation at the symposium. The range of the considered topics is wide, including, among others, neuro-symbolic reasoning, complex event recognition and run-time verification. To reach a decision, we discussed the reviews with all three reviewers assigned to a paper. The accepted papers are very interesting and we are confident that we will have lively discussions during the symposium.

We are very pleased to include invited talks by leading scholars in our scientific communities: Moshe Y. Vardi (Rice University, USA), Silvia Miksch (TU Vienna, Austria) and Stijn Vansummeren (University of Hasselt, Belgium). We believe that the invited talks, the selected papers and their presentations will help to stimulate and improve several research efforts in the area of temporal representation and reasoning, and motivate members of under-represented research communities to participate in the TIME Symposia.

We would like to thank all the members of the Program Committee and the additional reviewers, who spent their time and volunteered their expertise to set up the final program. We want also to thank Periklis Mantenoglou for his efforts in maintaining the web page of the symposium. Finally, we would like to acknowledge the generous support of the Department of Computer Science of the University of Verona, Italy, which supported, among others, the open access publication of these proceedings.

Alexander Artikis, University of Piraeus & NCSR Demokritos, Greece
Roberto Posenato, University of Verona, Italy
Stefano Tonetta, FBK, Italy

TIME 2022 Program Committee Co-chairs
September 6, 2022

TIME Steering Committee

Program Committee Members

Alessandro Artale
Free University of Bolzano-Bozen
Italy
artale@inf.unibz.it

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

Krysia Broda
Imperial College
UK
k.broda@imperial.ac.uk

François Bry
Ludwig Maximilian University
Germany
bry@lmu.de

Jaewook Byun
Sejong University
South Korea
bjw0829@kaist.ac.kr

Carlo Combi
Università degli Studi di Verona
Italy
carlo.combi@univr.it

Stéphane Demri
CNRS, LMF, ENS Paris-Saclay
France
demri@lsv.ens-cachan.fr

Clare Dixon
University of Manchester
United Kingdom
clare.dixon@manchester.ac.uk

Alexandre Duret-Lutz
LRDE/EPITA
France
adl@lrde.epita.fr

Johann Eder
Alpen-Adria-Universität Klagenfurt
Austria
Johann.Eder@aau.at

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

Rajeev Gore
The Australian National University
Australia
rajeev.gore@anu.edu.au

Gopal Gupta
The University of Texas at Dallas
USA
gupta@utdallas.edu

Thomas Guyet
Inria
France
thomas.guyet@inria.fr

Sylvain Hallé
Université du Québec à Chicoutimi
France
shalle@acm.org

Luke Hunsberger
Vassar College
USA
hunsberger@vassar.edu

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

Roman Kontchakov
Birkbeck, University of London
United Kingdom
roman@dcs.bbk.ac.uk

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

Stephane Le Roux
ENS Paris-Saclay
France
leroux@lsv.fr

List of Authors

Gianluca Apriceno (12)
University of Trento, Italy;
Fondazione Bruno Kessler, Trento, Italy

Massimo Benerecetti (14)
University of Napoli "Federico II", Italy

Ashwin Bhaskar (8)
Chennai Mathematical Institute, India

Laura Bozzelli (11)
University of Napoli "Federico II", Italy

Florian Bruse (5)
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Raghubir Chimni (4)
University of California,
Santa Barbara, CA, USA

Yliès Falcone (6)
Univ. Grenoble Alpes, Inria, CNRS,
Grenoble INP, LIG, 38000 Grenoble, France

Dimitar P. Guelev (10)
Institute of Mathematics and Informatics,
Bulgarian Academy of Sciences,
Sofia, Bulgaria

Julian Gutierrez (15)
Monash University, Melbourne, Australia

Thomas Guyet (7)
Inria, Lyon, France

Sarit Kraus (15)
Bar-Ilan University, Ramat-Gan, Israel

Martin Lange (5)
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Etienne Lozes (5)
Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis,
Université Côte d'Azur, France

Isaac Mackey (4)
University of California,
Santa Barbara, CA, USA

Nicolas Markey (7)
CNRS, IRISA, Rennes, France

Silvia Miksch (2)
Institute of Visual Computing and
Human-Centered Technology,
TU Wien, Austria

Fabio Mogavero (14)
University of Napoli "Federico II", Italy

Giovanni Pagliarini (13)
Department of Mathematics and Computer Science, University of Ferrara, Italy;
Department of Mathematics, Physics, and Computer Science, University of Parma, Italy

Andrea Passerini (12)
University of Trento, Italy

Nicolas Peltier (9)
Univ. Grenoble Alpes, CNRS, LIG,
Grenoble, France

Giuseppe Perelli (15)
Sapienza University of Rome, Italy

Adriano Peron (11, 14)
University of Napoli "Federico II", Italy

M. Praveen (8)
Chennai Mathematical Institute, India;
CNRS IRL ReLaX, Chennai, India

Victor Roussanaly (6)
Univ. Grenoble Alpes, Inria, CNRS,
Grenoble INP, LIG, 38000 Grenoble, France

Simone Scaboro (13)
Department of Mathematics, Physics, and Computer Science, University of Udine, Italy

Guido Sciavicco (13)
Department of Mathematics and Computer Science, University of Ferrara, Italy

Luciano Serafini (12)
Fondazione Bruno Kessler, Trento, Italy

Giuseppe Serra (13)
Department of Mathematics, Physics, and Computer Science, University of Udine, Italy

Ionel Eduard Stan (13)
Department of Mathematics and Computer Science, University of Ferrara, Italy;
Department of Mathematics, Physics, and Computer Science, University of Parma, Italy

Jianwen Su (4)
University of California,
Santa Barbara, CA, USA

Stijn Vansummeren (3)
Data Science Institute, Applied Computer
Science Laboratory, Hasselt University,
Diepenbeek, Belgium

Moshe Y. Vardi (1)
Rice University, Houston, TX, USA

Michael Wooldridge (15)
University of Oxford, UK

Linear Temporal Logic: From Infinite to Finite Horizon

Moshe Y. Vardi
Rice University, Houston, TX, USA

Abstract

Linear Temporal Logic (LTL), proposed in 1977 by Amir Pnueli for reasoning about ongoing programs, was defined over infinite traces. The motivation for this was the desire to model arbitrarily long computations. While this approach has been highly successful in the context of model checking, it has been less successful in the context of reactive synthesis, due to the chalenging algorithmics of infinite-horizon temporal synthesis. In this talk we show that focusing on finite-horizon temporal synthesis offers enough algorithmic advantages to compensate for the loss in expressiveness. In fact, finite-horizon reasonings is useful even in the context of infinite-horizon applications.

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

Keywords and phrases Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.1

Category Invited Talk

Funding Work supported in part by NSF grants IIS-1527668, CCF-1704883, IIS-1830549, CNS-2016656, DoD MURI grant N00014-20-1-2787, and an award from the Maryland Procurement Office.

References

1 G. De Giacomo and M.Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. 23rd Int'l Joint Conf. on Artificial Intelligence*, pages 854–860, 2013.
2 G. De Giacomo and M.Y. Vardi. Synthesis for LTL and LDL on finite traces. In *Proc. 24th Int'l Joint Conf. on Artificial Intelligence*, pages 1558–1564. AAAI Press, 2015.
3 G. De Giacomo and M.Y. Vardi. LTLf and ldlf synthesis under partial observability. In *Proc. of the 25th Int'l Joint Conf. on Artificial Intelligence*, pages 1044–1050. IJCAI/AAAI Press, 2016.
4 G. De Giacomo, A. Di Stasio, L.M. Tabajara, M.Y. Vardi, and S. Zhu. Finite-trace and generalized-reactivity specifications in temporal synthesis. In *Proc. 30th Int'l Joint Conf. on Artificial Intelligence*, pages 1852–1858. ijcai.org, 2021.
5 S. Xiao, J. Li, S. Zhu, Y. Shi, G. Pu, and M.Y. Vardi. On-the-fly synthesis for LTL over finite traces. In *35th AAAI Conf. on Artificial Intelligence*, pages 6530–6537. AAAI Press, 2021.
6 S. Zhu, L.M. Tabajara, J. Li, G. Pu, and M.Y. Vardi. A symbolic approach to safety LTL synthesis. In *Proc.. 13th Int'l Haifa Verification Conf. on Hardware and Software: Verification and Testing*, volume 10629 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2017.
7 S. Zhu, L.M. Tabajara, J. Li, G. Pu, and M.Y. Vardi. Symbolic LTLf synthesis. In *Proc. 26th Int'l Joint Conf. on Artificial Intelligence*, pages 1362–1369. ijcai.org, 2017.

Visual Analytics Meets Temporal Reasoning: Challenges and Opportunities

Silvia Miksch
Institute of Visual Computing and Human-Centered Technology, TU Wien, Austria

Abstract

Visual Analytics as the science of analytical reasoning facilitated by interactive visual interfaces aims to enable the exploration and the understanding of large, heterogeneous, and complex data sets.

Time is an important data dimension with distinct characteristics.

Intertwining Visual Analytics with time and temporal reasoning introduces outstanding challenges and opportunities, which I will illustrate in this talk.

2012 ACM Subject Classification Human-centered computing → Visualization

Keywords and phrases Visual Analytics, Visualization, Time

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.2

Category Invited Talk

1 Extended Abstract

Visual Analytics integrates the outstanding capabilities of humans in terms of visual information exploration with the enormous processing power of computers to form powerful information and knowledge discovery environments. In other words, Visual Analytics is the science of analytical reasoning facilitated by interactive interfaces and captures the information discovery process keeping the human in the loop as well as gaining deeper insights into huge heterogeneous and complex data sources.

Time is an important data dimension with distinct characteristics. Time is common across many application domains (e.g., medical records, planning, or project management). In contrast to other quantitative data dimensions, which are usually “flat”, time has an inherent semantic structure, which increases time’s complexity substantially. The hierarchical structure of granularities in time (e.g., minutes, hours, days, weeks, or months), is unlike that of most other quantitative dimensions. Specifically, time comprises different forms of divisions (e.g., 60 minutes correspond to one hour, while 24 hours make up one day), and granularities are combined to form calendar systems (e.g., Gregorian, Julian, business, or academic calendars). Moreover, time contains natural cycles and re-occurrences, as for example seasons, but also social (often irregular) cycles, like holidays or school breaks. Therefore, time-oriented data, i.e., data that are inherently linked to time, need to be treated differently than other kinds of data and require appropriate visual, interactive, and analytical methods to explore and analyze them.

In this talk, I will illustrate the concepts of Visualization and Visual Analytics. I will characterize the dimension of time as well as time-oriented data as well as describe tasks that users seek to accomplish using temporal Visual Analytics methods. I will address three key questions: “what” is visualized, “why” is it visualized, and “how” it is visualized. Various examples will illustrate what has been achieved so far and show possible future directions and challenges.

Getting to the CORE of Complex Event Recognition

Stijn Vansummeren
Data Science Institute, Applied Computer Science Laboratory,
Hasselt University, Diepenbeek, Belgium

Abstract

In this talk, I will give an overview of our recent work on complex event recognition.

2012 ACM Subject Classification Information systems → Query languages for non-relational engines; Information systems → Stream management; Theory of computation → Formal languages and automata theory

Keywords and phrases Complex Event Recognition, automata, enumeration-based query processing

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.3

Category Invited Talk

Funding This research was supported by the Bijzonder Onderzoeksfonds (BOF) of Hasselt University (Belgium) under Grant No. BOF20ZAP02.

Acknowledgements The research surveyed in this talk is joint work with Marco Bucchi, Alejandro Grez, Martín Ugarte, Andrés Quintana, and Cristian Riveros.

1 Extended Abstract

Complex Event Recognition (CER for short) refers to the activity of processing high-velocity streams of primitive events by evaluating queries that detect *complex events*: collections of primitive events that satisfy some pattern. In particular, CER queries match incoming events on the basis of their content; where they occur in the input stream; and how this order relates to other events in the stream. CER has been successfully applied in diverse domains such as maritime monitoring, network intrusion detection, industrial control systems and real-time analytics, among others.

In this talk, I will survey our recent work on developing a formal framework for specifying and evaluating CER queries. This framework consist of a formal, core query language called *Complex Event Logic* (CEL) for specifying CER queries [4]. In contrast to previous proposals, CEL has a compositional and denotational semantics, and encompasses all operators that are considered "common base operators" in the literature. Using CEL, we have been able to get a better understanding of the relative expressiveness of these operators as well as the impact of common evaluation heuristics such as selection policies [3, 4].

The framework also consists of an automaton-based formal computational model for CEL, called *Complex Event Automata* (CEA). Using CEA, we have developed a novel evaluation algorithm for CEL that exhibits strong performance guarantees: under data complexity, the algorithm takes only constant time to process each input event [1]. This is in contrast to existing algorithms that take time proportional to the number of previously processed events, or the size of a time window. As I will explain, our algorithm processes each event in constant time by adopting the framework of enumeration-based query evaluation that is receiving increased attention in the database community [5]. Specifically, it maintains a data structure from which at any point in time all found complex events may be enumerated with so-called output-linear delay. This means that the time required to output a new recognized complex event C is linear in the size of C, but independent of the number of already processed events or size of time window, and independent of the number of found complex events.

Our framework is implemented in the CORE complex event recognition engine [2]. CORE's firm formal foundation allows it to exhibit stable query performance, even for long sequence queries and large time windows, and outperform existing systems by up to five orders of magnitude on different workloads [1].

I will discuss the essential ideas behind CORE's query language and evaluation algorithm, as well as their limitations, and from these limitations discuss open questions relevant for the TIME community.

References

1 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. URL: `https://www.vldb.org/pvldb/vol15/p1951-riveros.pdf`.

2 The CORE COmplex event Recognition Engine. `https://github.com/CORE-cer/CORE`.

3 Alejandro Grez, Cristian Riveros, Martín Ugarte, and Stijn Vansummeren. On the expressiveness of languages for complex event recognition. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*, volume 155 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ICDT.2020.15`.

4 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`.

5 Luc Segoufin. Enumerating with constant delay the answers to a query. In *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 10–20. ACM, 2013. `doi:10.1145/2448496.2448498`.

Early Detection of Temporal Constraint Violations

Isaac Mackey
University of California, Santa Barbara, CA, USA

Raghubir Chimni
University of California, Santa Barbara, CA, USA

Jianwen Su
University of California, Santa Barbara, CA, USA

Abstract

Software systems rely on events for logging, system coordination, handling unexpected situations, and more. Monitoring events at runtime can ensure that a business service system complies with policies, regulations, and business rules. Notably, detecting violations of rules *as early as possible* is much desired as it allows the system to reclaim resources from erring service enactments. We formalize a model for events and a logic-based rule language to specify temporal and data constraints. The primary goal of this paper is to develop techniques for detecting each rule violation as soon as it becomes inevitable. We further develop optimization techniques to reduce monitoring overhead. Finally, we implement a monitoring algorithm and experimentally evaluate it to demonstrate our approach to early violation detection is beneficial and effective for processing service enactments.

2012 ACM Subject Classification Information systems → Information systems applications

Keywords and phrases temporal constraints, monitoring, events, early violation detection

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.4

1 Introduction

Events are unorchestrated, asynchronous messages about the states of processes and situations like action and change. Events are a fundamental component in software systems including workflow systems, cyber-physical systems, IOT devices, decision support systems, etc., and a focus of research communities (e.g., [20]). These systems use events to *i*) identify time-critical exceptional situations that need attention, and *ii*) choreograph/orchestrate collaborative systems [5]. This paper studies a technical problem concerning *i*).

In runtime monitoring [2, 12], system policies for exceptional situations, i.e., violations of constraints, are specified in a formal language and algorithms monitor events from the system as they occur to detect and report violations. Violations of system policies can be divided into two categories depending on when the violation is detected: a violation can be detected once the system is finished executing or it can be reported when the system's execution is not yet finished but *as soon as* the violation becomes inevitable; we call the latter *early (violation) detection*.

We investigate the early detection problem in the context of workflow systems, where events report execution of activities in a workflow. In this setting, constraints are set by business rules, organization policies, regulations, and service-level agreements (SLAs) and specify temporal relationships between events in a workflow enactment, with "gap constraints" [24] to restrict time gaps between events. Constraints can also reference and compare data values in events. We call the growing set of events in an enactment a "log". A naive monitoring approach would (re)evaluate constraints over the entire log with each arrival of new events, but this is intractable for large logs, so we evaluate constraints incrementally.

This paper makes the following technical contributions:

- A technique for calculating the earliest time a violation is inevitable (the “deadline”),
- Algorithms and data structures for incrementally maintaining and detecting violations, along with batch algorithms for processing incoming events,
- Optimization techniques for those algorithms, including expiring useless data and improving batch processing, and
- Experimental findings illustrating the benefits and costs of early violation detection.

This paper is organized as follows. Subsection 1.1 discusses related work. Section 2 motivates the early violation detection problem with an example. Section 3 defines the technical framework. Section 4 presents the key techniques for computing “deadlines”, maintaining assignments and relationships between them, and detecting violations. Section 5 presents two optimization techniques. Section 6 presents the findings of an experimental evaluation. Finally, Section 7 concludes the paper.

1.1 Related Work

To identify when a violation is first inevitable, we distinguish between potential violations, which may or may not remain a violation in the future, and permanent violations, which are violations in all possible futures. This distinction is formalized for monitoring LTL formulas in [4], which notes that knowing if a trace satisfies or violates a constraint can be refined by knowing if the satisfaction or violation is permanent. [18] shows that this distinction can be monitored for propositional constraints in the Declare language using an encoding of violation status, i.e., potential or permanent, in states of automata derived from the constraint language. [22] uses a similar classification of violations (potential violations are called pending violations). The status of a violation is represented by a fluent in event calculus (EC) and changes to violations’ status are encoded as EC axioms that initialize and change fluents. We distinguish potential and permanent violations based on satisfiability checking for partial initializations of constraint variables. Partial initialization is not a new technique (e.g., [3, 9]), though [9] does not monitor events with data and neither attempts early violation detection. Our work is more similar to that of [17], where satisfiability checking determines if constraints in MP-Declare (a variant of Declare that supports conditions on data and time) are permanently violated, however we provide an algorithm that calculates deadlines, rather than offloading the calculation to a solver.

Specifying conditions on data carried by events, such as matching the user opening an order to the user charged payment, is an important functionality of monitoring constraints [14]. [21] adds data conditions to Declare and the conditions are incorporated into the EC formalization [22]. Another approach to include data is found in [7, 8], which monitor FO-LTL and Declare constraints, resp., using automata whose states are augmented with data stores, and potential and permanent violations are distinguished in the same manner as [18], but these works assume a fixed, finite domain for data values. Incremental view maintenance for Datalog offers relevant incremental algorithms. [11] maintains non-recursive views but does not have any time or inequality constraints; our language allows timestamps and gap constraints. [23] maintains recursive views; our language assumes a fixed set of atomic events, which does not allow recursion.

[14] also argues that quantitative time constraints are important for compliance specification. LTL, Declare, and their metric extensions [3, 17, 21, 22] can require the gap between a pair of event timestamps to fall in an interval. Our language gives each event atom in a constraint a time variable, thus allowing an unbounded number of gap (in)equalities

and constant offsets between any and all pairs of event timestamps. It is unknown if our constraints can be translated into LTL, though for a subset of our language, specifically dataless, "singly-linked" rules, [15, 16] provide a translation.

Controllability is another approach to manage temporal constraints in workflow enactments. [6] and [13] feature propagation of upper and lower bound constraints similar to our deadline calculation approach, but does not allow comparison of data values in events. [10] applies explicit time variables to the controllability problem for modular process models. Enforcing controllability is a design-time solution, however; we make no assumptions about the control structure of a service in order to afford managers and users maximum flexibility.

2 An Opportunity for Early Violation Detection

We illustrate the problem of detecting violations of business rules for workflows and motivate an approach based on reasoning about constraints. We sketch an example workflow from an Infrastructure-as-a-Service (IaaS) provider. Then, we explain how the constraints on the workflow are evaluated to determine the earliest time violations are permanent.

Consider an IaaS provider that offers high-performance cloud computing rentals. The service is managed by a workflow with the following activities: the user Requests a machine through an account and the provider grants Approval to the user. Then, the user Reserves a machine for their account, makes a Payment with their account and Launches the machine. The completion of each activity generates an event; events for the same rental instance form an *enactment*. Each event has a timestamp, an enactment identifier, and may have additional data, e.g., a user. We view a set of events as a relational database. Fig. 1 shows a database S_9 at time 9, with eight events from two enactments with ids π_1 and π_2. For example, the first row of the Request table indicates a Request event with enactment id π_1 from user Alice with account $a3$ at time 1.

Request

ID	*user*	*account*	ts
π_1	Alice	a3	1
π_1	Alice	a4	3
π_2	Bob	b6	7

Approval

ID	*user*	ts
π_1	Alice	6

Reserve

ID	*user*	*account*	ts
π_1	Alice	a4	8
π_1	Alice	a3	9

Payment

ID	*user*	*account*	ts
π_1	Alice	a3	8
π_1	Alice	a4	9

Launch

ID	*user*	*account*	ts

Figure 1 Database S_9 with events from two enactments π_1 and π_2.

The provider checks each enactment against specified business rules; these may measure service availability, quality, etc. For example, we use a few rules, including: when a user's Request is approved within 7 days and the machine is Reserved within 7 days of Approval by the same account as the request, the user should make a Payment for the machine through that account within 3 days of Approval and Launch it within 7 days of Reserve and 4 days of Payment. In this rule, events generated by either the provider or the user may lead to rule violation. We write this rule as $\varphi \to \psi$ where φ is the rule body and ψ is the rule head:

$$\begin{array}{l}\textsf{Request}(u,a)@x, \textsf{Approval}(u)@y, x\leqslant y\leqslant x+7, \textsf{Reserve}(u,a)@z, y\leqslant z\leqslant\ y+7 \\ \to\ \ \textsf{Payment}(u,a)@w, \textsf{Launch}(u,a)@v, y\leqslant w\leqslant y+3, z\leqslant v\leqslant z+7, v\leqslant w+4\end{array}$$

The core idea of detecting a violation is checking whether each body assignment for the body variables u, a, x, y, z satisfying φ has a matching head assignment for the head variables u, a, w, v satisfying ψ. In order to detect violations incrementally, we build assignments that satisfy the rule's subformulas. Fig. 2(a) lists the partial and complete assignments for φ

Aid	ID	u	a	x	y	z	
μ_1	π_1	Alice	a3	1	-	-	
μ_2	π_1	Alice	a4	3	-	-	
μ_3	π_1	Alice	-	-	6	-	
μ_4	π_1	Alice	a3	1	6	-	$\mu_1 + \mu_3$
μ_5	π_1	Alice	a4	3	6	-	$\mu_2 + \mu_3$
μ_6	π_2	Bob	b6	7	-	-	
μ_7	π_1	Alice	a4	-	-	8	
μ_8	π_1	Alice	a4	3	-	8	$\mu_2 + \mu_7$
μ_9	π_1	Alice	a4	-	6	8	$\mu_3 + \mu_7$
μ_{10}	π_1	Alice	a4	3	6	8	$\mu_2 + \mu_9$
μ_{11}	π_1	Alice	a3	-	-	9	
μ_{12}	π_1	Alice	a3	1	-	9	$\mu_1 + \mu_{11}$
μ_{13}	π_1	Alice	a3	-	6	9	$\mu_3 + \mu_{11}$
μ_{14}	π_1	Alice	a3	1	6	9	$\mu_4 + \mu_{12}$

(a) All body assignments at time 9.

Aid	ID	u	a	x	y	z	
μ_{15}	π_1	Alice	-	-	10	-	
μ_{16}	π_1	Alice	a4	3	10	-	$\mu_2 + \mu_{15}$
μ_{17}	π_2	Bob	-	-	10	-	
μ_{18}	π_2	Bob	b6	7	10	-	$\mu_6 + \mu_{17}$

(b) New body assignments added at time 10.

Figure 2 Assignments for the rule body φ and events S_9, events $S_9 \cup \{e_1, e_2\}$.

and S_9. For example, assignment μ_1 is generated by the first row (event) of the Request table. Assignment μ_3 is generated by the first row of the Approval table, and is combined with μ_1 to form μ_4. Then, μ_{10} and μ_{14} makes φ true.

Suppose two events happen at time 10, e_1:Approval(π_1, [Alice], 10), e_2:Approval(π_2, [Bob], 10). Event e_1 generates a partial assignment μ_{15}, which combines with μ_2 into μ_{16}. Event e_2 yields new assignments μ_{17} and μ_{18}. Fig. 2(b) lists four assignments generated by e_1 and e_2.

ψ has six variables u, a, y, z, w, v, but Payment and Launch events only supply values for the four "event variables" u, a, w, v. We consider assignments for ψ in the same manner as for φ but ignoring y and z. The Payment events at times 8, 9 (Fig. 1) create partial assignments β_1: [π_1, Alice, a3, 8, -] and β_2: [π_1, Alice, a4, 9, -].

One interesting problem is to determine *when* to report rule violations. In Fig. 3, three events create a potential violation μ_{10}. It is natural to report this violation when the *END* of enactment π_1 arrives, after which no more events for π_1 will arrive; if μ_{10} is not extended by a head assignment by that time, it represents a permanent violation. We aim to detect violations as soon as they become permanent, which may be well before the *END* event. Given the rule's constraints $y \leqslant w \leqslant y+3$ and $z \leqslant v \leqslant z+7$, and $\mu_{10}(y) = 6$ and $\mu_{10}(z) = 8$, the violation is known to be permanent at time 9 because no Payment event arrives with a timestamp for w to extend μ_{10}. Note also that 9 is the earliest time we can be certain this is a violation.

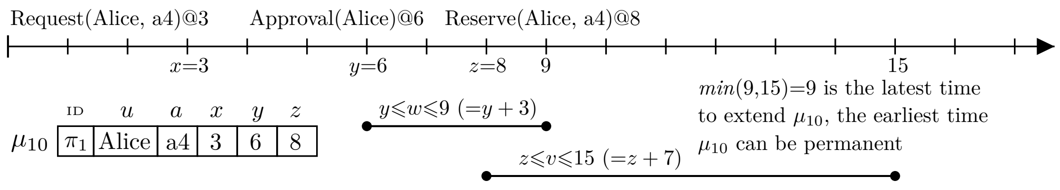


Figure 3 Deadline for extending potential violation μ_{10}.

Fig. 4 shows a Payment event at time 9 that creates the partial assignment β_2.

The main focus of this paper is to calculate these earliest times, which we call "deadlines", and use them in a monitoring algorithm.

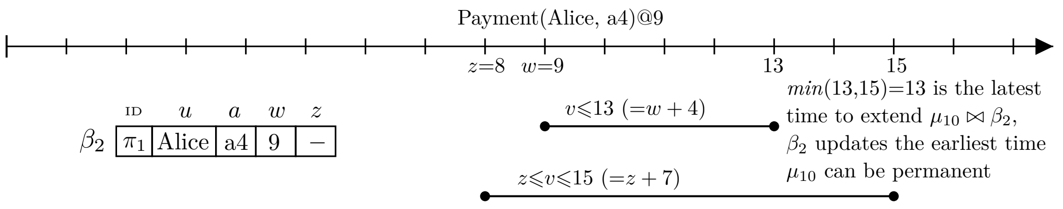

Payment(Alice, a4)@9
z=8
w=9
13
15
ID u a w z
β2 π1 Alice a4 9 –
v⩽13 (=w + 4)
z⩽v⩽15 (=z + 7)
min(13,15)=13 is the latest time to extend μ10 ⋈ β2, β2 updates the earliest time μ10 can be permanent

Figure 4 Deadline for extending β_2 as match for μ_{10}.

3 Rules and the Detection Problem

In this section, we present key notions needed for technical development, including "activities" in workflows, "events" of activities, "enactments", "batches", and "rules".

Activities are atomic units of work in a workflow. Each activity has a name and a set of data attributes. An activity's execution yields an event, which carries values for the data attributes and a timestamp. We use *identifiers* $\mathbf{I}$ (or simply ID's), for (workflow) enactments; each event has an identifier from the workflow instance that generated it. We assume a countably infinite set of timestamps $\mathbf{T}$ with a discrete total order and addition of constants. For technical development, we use natural numbers for timestamps. We also assume a countably infinite set of (data) values $\mathbf{D} = \{a, b, c, ..., a_1, ...\}$ with equality.

▶ **Definition.** *An* event *of an activity $A(\mathrm{C}_1, ..., \mathrm{C}_n)$ is a named tuple $A(\xi, \nu, \tau)$ where ξ is an* ID *from $\boldsymbol{I}$, $\nu : \{\mathrm{C}_1, ..., \mathrm{C}_n\} \to \boldsymbol{D}$ is a mapping from A's attributes to data values, and τ is a timestamp from $\boldsymbol{T}$.*

An instance of a workflow is a finite set η of events called an *enactment*, such that (i) each event has the same enactment ID, (ii) η has exactly one special *START* event that marks its beginning and of workflow enactments and at most one *END* event that marks its completion, (iii) the timestamp of the *START* event is less than that of all other events in η, and (iv) the timestamp of the *END* event, if it occurs, is greater than that of all other events in η. The rows in Fig. 1 with the same ID form an enactment (the *START*/*END* events are not shown).

This paper focuses on monitoring enactments as they are updated by new events. Constraints to be monitored are specified as "rules". In the following, we define and illustrate the notions of a "batch" (new events arriving) and a rule.

▶ **Definition.** *A* batch *for an enactment η is a finite set Δ of events such that (i) all events in Δ have the same timestamp, denoted as* ts_Δ*, greater than the timestamps of all events in η, (ii) for each event e in Δ, the* ID *of e is the* ID *of η, (iii) Δ has a* START *event or η has a* START *event, but not both, and (vi) if an* END *event is in η, no events are in Δ.*

Fig. 1 shows events from two enactments of the workflow in the IaaS example. Suppose that at time 10 exactly two events happened, e_1:$\mathsf{Approval}(\pi_1, [\text{Alice}], 10)$ and e_2:$\mathsf{Approval}(\pi_2, [\text{Bob}], 10)$. Then $\{e_1\}$ is a batch for π_1, $\{e_2\}$ a batch for π_2.

We describe a language for specifying rules, starting with atomic formulas. An *event atom* is an expression "$A(v_1, ..., v_n)@x$" where $A(\mathrm{C}_1, ..., \mathrm{C}_n)$ is an activity, $v_1, ..., v_n, x$ are variables, where x is a *time variable.* A *gap atom* is an expression "$x \pm \epsilon\, \theta\, y$" where x, y are time variables, ϵ (the gap) is a timestamp in $\mathbf{T}$, and $\theta \in \{<, \leqslant, \geqslant, >, =\}$ is an equality or inequality predicate. We denote the variables in a set of gap atoms φ as $var(\varphi)$.

▶ **Definition.** *A* rule *is an expression "$\varphi \to \psi$" where the* body *φ and the* head *ψ are finite sets of event and gap atoms such that each variable in a gap atom in φ occurs in an event atom in φ and each variable in a gap atom in ψ occurs in an event atom in $\varphi \cup \psi$.*

Rule satisfaction is defined as follows: An assignment is a mapping from variables to values in $\mathbf{D} \cup \mathbf{T}$. Time variables take values from $\mathbf{T}$; we use $\mathbb{N}$ as timestamps for technical development. All other variables take values from $\mathbf{D}$. An assignment is *complete* if it is a total mapping for the variables in a given set of atoms, *partial* otherwise. An assignment β extends an assignment α if $\alpha \subseteq \beta$. An enactment η satisfies an event atom $A(v_1, ..., v_n)@x$ for the activity $A(\text{C}_1, ..., \text{C}_n)$ with a complete assignment μ if $A(\eta.\text{ID}, \{\text{C}_1 \mapsto \mu(v_1), \ldots, \text{C}_n \mapsto \mu(v_n)\}, \mu(x))$ is an event in η. An assignment satisfies a gap atom with the obvious interpretation.

An enactment η *satisfies* a set of atoms ϕ with a complete assignment μ if η satisfies every atom in ϕ with μ. An enactment η *satisfies* a rule $r\colon \varphi \to \psi$ if for every complete assignment μ such that η satisfies φ with μ, there is a complete assignment β that extends μ such that η satisfies ψ with β.

▶ **Example 1.** As shown in Fig. 3, the assignment μ_{10} satisfies φ. Then, to satisfy the rule w.r.t. μ_{10}, there must be an assignment extending μ that satisfies ψ; i.e., two events $\text{Payment}(\pi_1, [\text{Alice}, \text{a4}], t_1)$ and $\text{Launch}(\pi_1, [\text{Alice}, \text{a4}], t_2)$ with $6 \leqslant t_1 \leqslant 6+3=9$, $8 \leqslant t_2 \leqslant 8+7=15$, and $t_2 \leqslant t_1+4$ must happen.

An assignment μ is a *potential violation* of a rule $r\colon \varphi \to \psi$ in an enactment η if η satisfies φ with μ and there is no assignment β that extends μ such that η satisfies ψ with β. A *(permanent) violation* μ of a rule $r\colon \varphi \to \psi$ is a potential violation where for every sequence of batches of future events $\Delta_1, ..., \Delta_n$ (where Δ_i is a batch for $\eta \cup (\cup_{j<i} \Delta_j)$ for each $1 \leqslant i \leqslant n$), μ is a violation of r in $\eta \cup (\cup_{i=1}^{n} \Delta_i)$. In the next section, we develop algorithms to identify when violations become permanent.

4 Techniques for Early Violation Detection

In this section, we develop key techniques for early violation detection. First, we define the concept of a "deadline" and present an algorithm to calculate deadlines. Next, we define data structures to store variable assignments and algorithms to create new assignments from arriving events. Finally, we detail how violations are detected. A monitoring algorithm using these techniques was implemented and experimentally evaluated in Sec. 6.

We aim to detect permanent violations as early as possible. Since an enactment is an accumulation of events with increasing timestamps, it may be that a complete body assignment derived from the current enactment can only be extended at or before a specific future time called a deadline. We now formulate the notion of a deadline.

▶ **Definition.** *Let Θ be a set of gap atoms over variables $x_1, ..., x_n$ and μ a (partial) assignment for variables x_i's. We use DEF_μ for the variables μ assigns a value; $\mu(\Theta)$ the gap atoms obtained by replacing each variable $x \in \text{DEF}_\mu$ with $\mu(x)$, and $\max(\mu) = \max\{\mu(x) \mid x \in \text{DEF}_\mu\}$. A timestamp $\tau \in \mathbb{N}$ is the* deadline *for $\Theta, x_1, ..., x_n, \mu$ if (1) $\tau \geqslant \max(\mu)$, and (2) either $\mu(\Theta)$ is unsatisfiable and $\tau = \max(\mu)$ or conditions (i) and (ii) both hold: (i) for each complete extension μ' of μ such that $\mu'(x) > \tau$ for each $x \notin \text{DEF}_\mu$, $\mu'(\Theta)$ is false, and (ii) there is a complete extension μ'' of μ such that $\mu''(\Theta)$ is true.*

▶ **Example 2.** In the running example in Section 2, μ_{10} is created at time 8, where $\mu_{10}(x)=3$, $\mu_{10}(y)=6$, and $\mu_{10}(z)=8$. As shown in Fig. 3, applying μ_{10} to the head atoms yields upper bounds $w \leqslant 9$ $(=y+3)$ and $v \leqslant 15$ $(=z+7)$. From these bounds, it is clear that extensions of μ_{10} must have a Payment event whose time variable w is no later than time 9. Thus, the time 9 is a "deadline" for μ_{10}: the latest time μ_{10} can be extended w.r.t. w, and the earliest time μ_{10} could be recognized as a permanent violation. Fortunately, a Payment event happened

Algorithm 1 Deadline$(\Theta, x_1, ..., x_n, \mu)$.

```
Input: A set of gap atoms $\Theta$ over time variables $x_1, ..., x_n$ and an assignment $\mu$
Output: A timestamp $\tau$
1: if If $\mu(\Theta)$ is unsatisfiable then return $\tau := \max(\mu)$;
                    /* $\max(\mu)$ is the largest timestamp $\mu$ assigns to $x_1, ..., x_n$ */
2: Rewrite each atom in $\mu(\Theta)$ in the form $u \pm k \leqslant v$;
                    /* $u, v$ either a time variable or in $\mathbb{N}$, $k \in \mathbb{Z}$ */
3: Let $UpperBd$ be a map from $x_1, ..., x_n$ to $\{\infty\}$;
4: for each $u \pm k \leqslant v$ in $\mu(\Theta)$ with $v \in \mathbb{N}$ and $u \in \{x_1, ..., x_n\}$ do
5:    $UpperBd(u) := v \mp k$;
6: for $|\Theta|$ iterations do
7:    for each gap atom $u \pm k \leqslant v$ in $\mu(\Theta)$ do
8:       if $UpperBd(v)$ is finite and $UpperBd(u) \pm k > UpperBd(v) \geqslant 0$ then
9:          $UpperBd(u) := UpperBd(v) \mp k$;
10: return $\tau := \min\{UpperBd(x_i) \mid 1 \leqslant i \leqslant n\}$
```

at time 9, which satisfies $w \leqslant 9$. However, v remains unresolved and thus the subsequent deadline to extend μ_{10} is the latest time to observe a value for v: $v \leqslant 13$ $(=w+4)$ and $v \leqslant 15$ $(=z+7)$, so the deadline to extend μ_{10} is changed to 13.

We compute deadlines with function Deadline (Alg. 1). Deadline determines for each x_i the least τ_i such that $\mu(\Theta) \to x_i \leqslant \tau_i$, and the deadline τ is the least of τ_i's. First, if $\mu(\Theta)$ is unsatisfiable, μ is a violation at the time of its creation, i.e., at its largest timestamp. Otherwise, an array $UpperBd$ is initialized with constants (Lines 3-5), then tightened with the initial bounds and the gap atoms in Θ: a gap atom $u \pm k \leqslant v$ indicates $UpperBd(v) \mp k$ is an upper bound for u. For each gap atom $u \pm k \leqslant v$ for which $UpperBd(v)$ is defined, we update $UpperBd(u)$ as $\max(UpperBd(v) \mp k, UpperBd(u))$ (Lines 7-9).

The Deadline function (Alg. 1) can compute deadlines for complete body assignments and for complete body assignments with matching partial head assignments. For a complete body assignment μ and a partial head assignment β, we compute the latest time $\mu \cup \beta$ can be extended. This time is, in fact, the earliest time μ becomes a permanent violation. In the following lemma, we state a property of deadlines for a complete body assignment and partial head assignment.

▶ **Lemma 3.** *Let $r: \varphi \to \psi$ be a rule, φ_g, ψ_g the gap atoms in φ, ψ (resp.), μ a complete body assignment such that $\mu(\varphi_g)$ is true, β an incomplete head assignment extending μ such that $\beta(\mu(\psi_g))$ is satisfiable, and U the variables in ψ_g undefined by β. Let $\tau = \text{Deadline}(\psi_g, var(\varphi_g \cup \psi_g), \mu \cup \beta)$. The following hold:*

1. *If $\tau \in \mathbb{N}$, then there is a complete head assignment β' extending $\mu \cup \beta$ such that $\min(\beta'(U)) \leqslant \tau$ and $\beta'(\psi_g)$ is true,*
2. *If $\tau \in \mathbb{N}$, then for all complete head assignments β' extending $\mu \cup \beta$ such that $\min(\beta'(U)) > \tau$, $\beta'(\psi_g)$ is false, and*
3. *If $\tau = \infty$, then for all timestamps n in $\mathbb{N}$, there is a complete head assignment β' extending $\mu \cup \beta$ such that $\max(\beta'(U)) > n$ and $\beta'(\psi_g)$ is true.*

A sketch of the proof is given in Appendix A. The key idea is that for the combined assignment $\mu \cup \beta$ and atoms ψ, either for some time variable z and timestamp τ, $\mu(\beta(\psi)) \wedge (z \geqslant \tau')$ is unsatisfiable (so τ is a deadline) or no such time variable z and timestamp τ exists (there is no deadline). Lemma 3 is applied in the following way: for a complete body assignment μ, we try to extend μ with each partial head assignment β when it is created. For each pair μ and β, we calculate a deadline using μ, β, and the rule head. According to Lemma 3, the output of Deadline is the time after which β cannot extend μ.

The discussions in Section 2 also suggest maintaining partial and complete assignments for variables. We define three tables BA_r for body assignments, HA_r for head assignments, and EXT_r (extensions) to track pairings of body and head assignments. BA_r and HA_r consist of the following columns: (*i*) one column for the assignment identifier (*Aid*) from **I**, (*ii*) one column for the enactment identifier (ID) from **I**, (*iii*) one column in BA_r for each variable in φ and one column in HA_r for each event variable in ψ (resp.) (a variable in the head ψ is an *event variable* if it occurs in an event atom in ψ.) to hold a value from **D** or **T**, and (*iv*) one column for gap atoms in φ and ψ (resp.) simplified with the assigned values as possible. Additionally, BA_r has one more column (*v*) *match?* indicating with *yes* or *no* the presence or absence, resp., of a complete head assignment extending the complete body assignment. For convenience, we refer to rows in these two tables as assignments. EXT_r has three columns: (*i*) one column for a body *Aid* from BA_r, (*ii*) one column for a head *Aid* from HA_r that extends the row's body assignment, and (*iii*) one column for the *deadline*, calculated using the row's assignments and the head gap atoms as inputs for Deadline.

For each enactment η, $\mathrm{BA}_r(\eta)$ and $\mathrm{HA}_r(\eta)$ store all assignments that can be generated from η and satisfy φ and ψ (resp.). Specifically, for a rule $r\colon \varphi \to \psi$ and an enactment η, $\mathrm{BA}_r(\eta)$ contains every assignment μ such that for a non-empty subset P of the event atoms in φ, μ is defined for all variables in P, $\mu(P) \subseteq \eta$, and η satisfies all atoms in φ having only variables in P with μ. $\mathrm{HA}_r(\eta)$ is similar, using ψ instead of φ. Fig. 5(a) shows the assignments inserted into BA_r table at time 10 (those from Fig. 2(b)) with columns for gap atoms and the possibility of matching. $\mathrm{EXT}_r(\eta)$ stores each pair of assignments from $\mathrm{BA}_r(\eta)$ and $\mathrm{HA}_r(\eta)$, resp., such that the body assignment can be extended by the head assignment only at or before the row's deadline.

Aid	u	a	x	y	z	gap atoms	match?
μ_{10}	Alice	a4	3	6	8	-	No
μ_{11}	Alice	a3	-	-	9	$x \leqslant y \leqslant x+7$, $y \leqslant 9 \leqslant y+7$	No
μ_{12}	Alice	a3	1	-	9	$1 \leqslant y \leqslant 8$, $y \leqslant 9 \leqslant y+7$	No
μ_{13}	Alice	a3	-	6	9	$x \leqslant 6 \leqslant x+7$	No
μ_{14}	Alice	a3	1	6	9	-	No

(a) Some assignments in $\mathrm{BA}_r(\pi_1)$ (Fig.2(a)) at $\mathsf{ts} = 9$.

Aid	u	a	w	v	gap atoms
β_1	Alice	a3	8	-	$v \leqslant 12$
β_2	Alice	a4	9	-	$v \leqslant 13$
β_3	Alice	a3	-	12	$8 \leqslant w$
β_4	Alice	a3	8	10	-

(b) Some assignments in $\mathrm{HA}_r(\pi_1)$ (Fig.2(b)) at $\mathsf{ts} = 10$.

Figure 5 Body and Head Table Examples.

body Aid	head Aid	deadline
μ_{10}	-	9
μ_{10}	β_2	13
μ_{14}	-	9
μ_{14}	β_1	12
μ_{14}	β_3	12
μ_{14}	β_4	-

Figure 6 Extensions of μ_{10} and μ_{14} in $\mathrm{EXT}_r(\pi_1)$ at $\mathsf{ts} = 13$.

We next present an algorithm called Update to create and combine assignments as batches of events arrive. This algorithm maintains BA and HA incrementally without accessing enactments directly; Update (Alg. 2) does not take an enactment as input. This is important since enactments may be very large.

We now outline the behavior of Update. Given atoms Θ (here, the head of a rule), a batch Δ, and either BA_r or HA_r for an enactment η, Lines 2-6 generate assignments from the events in Δ and Θ, adding them to the table if they are satisfiable (extendible to

Algorithm 2 Update($\Theta, \Delta, T(\eta)$).

```
Input: A set of atoms $\Theta$, a batch $\Delta$, a table $T(\eta)$ ($T$ is $\text{BA}_r$ or $\text{HA}_r$ for enactment $\eta$)
Output: Updated table $T(\eta \cup \Delta)$ for the new enactment $\eta \cup \Delta$
 1: $\Gamma := T(\eta)$ ;
 2: for each event $e \in \Delta$ do
 3:    for each event atom $\gamma$ in $\Theta$ with the same activity as $e$ do
 4:       Create a (partial) assignment $\mu$ from $e, \gamma$ such that $\mu(\gamma) = e$ ;
 5:       if $\mu(\Theta)$ is satisfiable then
 6:          Add to $\Gamma$ the row $s = \langle a, e.\text{ID}, \mu(v_1), ..., \mu(v_n), \text{B}, (\text{no})\rangle$,
               where $a$ is a fresh assignment identifier, $v_1, ..., v_n$ are the event variables
               in $\Theta$, and B the gap atoms in $\Theta$, evaluated and simplified under $\mu$;
 7: while $\Gamma$ changes do
 8:    for each pair of unique and consistent rows $\mu_1$ and $\mu_2$ in $T$ do
 9:       $\mu := \text{MERGE}(\mu_1, \mu_2)$ ;          /* consistent, MERGE explained in the text */
10:       Add to $\Gamma$ the row: $s = \langle a, \mu_1.\text{ID}, \max(t_1, t_2), \mu(v_1), ..., \mu(v_n), \text{B}, (\text{no})\rangle$
            where $a$ is a fresh assignment identifier and
            B is the union of gap atoms in $\mu_1, \mu_2$, evaluated with $\mu$ ;
11: output $\Gamma$
```

complete assignments). The **while** loop in Lines 7-10 searches for pairs of *consistent* partial assignments, Two assignments are *consistent* if they agree on the variables for which they are both defined, e.g., in Fig. 2 μ_1 and μ_2 agree on u but not on a. If two assignments are consistent and satisfy the necessary gap atoms, a new assignment is created with MERGE, which combines their variable mappings and gap atoms and recomputes their deadline. For example, assignment μ_5 in Fig. 2 is the *merge* of μ_2 and μ_3. The loop only creates assignments whose data values are pre-existing in Γ or the batch Δ, i.e., it doesn't introduce new data values, so the **while** loop terminates.

▶ **Example 4.** For the enactment and rule in Section 2, consider the enactment's event Request(π_1, [Alice, a3], 1) and the rule's atom Request(user u, account a)@x. The mapping [ID $\mapsto \pi_1$, $u \mapsto$ Alice, $v \mapsto$ a3, $x \mapsto 1$] maps the atom to this event; the assignment corresponding to this mapping is added to BA_r as μ_1 in Fig. 2(a). For the same example in Section 2 and Fig. 2(a), assignments μ_2: [π_1, Alice, a4, 3, -, -, $\{3 \leqslant y \leqslant 10, y \leqslant z \leqslant y+7\}$] and μ_3: [π_1, Alice, -, -, 6, -, $\{x \leqslant 6 \leqslant x+7,\ 6 \leqslant z \leqslant 13\}$] are in $\text{BA}_r(\pi_1)$ at ts $= 9$ and agree on u. Their combination MERGE(μ_2, μ_3) satisfies $x \leqslant 6 \leqslant x+7$ and $3 \leqslant y \leqslant 10$, so a row corresponding to MERGE(μ_2, μ_3) is added to BA_r as μ_5.

The following lemma states that Update refreshes the body and head tables by adding the partial and complete assignments with values from Δ as expected.

▶ **Lemma 5.** *Let $r: \varphi \to \psi$ be a rule, η an enactment, and Δ a batch for η.* Update($\varphi, \Delta, \text{BA}_r(\eta)$) *(or* Update($\psi, \Delta, \text{HA}_r(\eta)$)*) computes $\text{BA}_r(\eta \cup \Delta)$ (resp. $\text{HA}_r(\eta \cup \Delta)$).*

A sketch of the proof is given in Appendix A. The key idea is that for an assignment in $\text{BA}_r(\eta \cup \Delta)$, some data values may come from events in η so they will be in $\text{BA}_r(\eta)$ and some may come from events in Δ, in which case they will be introduced in Line 4 of Alg. 2 and merged with other assignments in the loop of Line 7 of Alg. 2. The proof is similar for $\text{HA}_r(\eta \cup \Delta)$.

The EXT table pairs complete body assignments with partial and complete head assignments along with a deadline. When a batch arrives, Update-E (Alg. 3) adds new complete body assignments to EXT (Lines 2-3), and then adds pairs using head assignments (Lines 4-8), computing a deadline for each pair (Line 8). Line 9 checks if there is a match between complete body and head assignments, and updates BA if so.

Algorithm 3 Update-E($\Delta, \textsc{ext}_r(\eta), \textsc{ba}_r(\eta\cup\Delta), \textsc{ha}_r(\eta\cup\Delta)$).

```
Input: A batch $\Delta$, un-updated table $\textsc{ext}_r(\eta)$,
       updated tables $\textsc{ba}_r(\eta\cup\Delta)$ and $\textsc{ha}_r(\eta\cup\Delta)$ for an enactment $\eta$
Output: Updated table $\textsc{ext}_r(\eta\cup\Delta)$
 1: $\Gamma := \textsc{ext}_r(\eta)$ ;
 2: for each complete body assignment $\mu$ in $\textsc{ba}_r(\eta\cup\Delta)$ do
 3:   if $\max(\mu) = \mathsf{ts}_\Delta$ then Add $\langle \mu, \text{-}, \mathsf{Deadline}(\psi, var(\psi), \mu)\rangle$ to $\Gamma$ ;
 4: for each assignment $\gamma$ in $\textsc{ha}_r(\eta\cup\Delta)$ do
 5:   if $\max(\gamma) = \mathsf{ts}_\Delta$ then
 6:     for each row $\langle \mu, \beta, d\rangle$ in $\Gamma$ do
 7:       if $\gamma$ extends $\mu\cup\beta$ and $\gamma(\mu(\psi))$ is satisfiable then
 8:         Add $\langle \mu, \gamma, \mathsf{Deadline}(\psi, var(\psi), \mu\cup\gamma)\rangle$ to $\Gamma$ ;
 9:         if $\gamma$ is complete then Update $\textsc{ba}_r(\eta\cup\Delta)$ to indicate $\mu$ has a match ;
10: output $\Gamma$ ;                                        /* $= \textsc{ext}_r(\eta\cup\Delta)$ */
```

For all complete body assignments, EXT stores each head assignment that extends it and indicates the latest time the pair can be further extended. The following lemma characterizes the conditions and time whereby a violation can be detected using EXT.

▶ **Lemma 6.** *Let η be an enactment with no* END *event, r a rule, τ a timestamp, and μ a complete body assignment for r. Then, μ is a permanent violation of r in η at τ iff μ occurs in $\textsc{ext}_r(\eta)$ but no rows in $\textsc{ext}_r(\eta)$ pairs μ with a complete head assignment, and each row in $\textsc{ext}_r(\eta)$ with μ has a deadline no greater than τ.*

A sketch of the proof is given in Appendix A. The key idea is that by Lemma 3, the largest deadline τ for μ and a partial match β in $\textsc{ext}_r(\eta)$ represent the time beyond which any assignment extending β derived from a future event will have a timestamp that is inconsistent with ψ. Thus, μ must be extended on or before time τ in order to be matched with β.

▶ **Example 7.** In Section 2, μ_{10} satisfies φ and must be extended no later than 9. Then, the deadline for matching the unpaired μ_{10} in $\textsc{ext}_r(\eta_{\leqslant 9})$ is 9. At time 9, a Payment event creates β_2 (Fig. 5), and μ_{10} and β_2 are inserted into $\textsc{ext}_r(\eta_{\leqslant 9})$ with deadline 13 because $\beta_2(w) = 9$ and ψ contains $v \leqslant w + 4$. Assuming no matching Launch event arrives, μ_{10} can be reported as a violation at time 13.

We now present the algorithm Detect (Algorithm 4) that detects permanent violations. These are unmatched body assignments in EXT (1) whose largest deadline is less than or equal to the current time or (2) whose enactments have ended.

Algorithm 4 Detect($\Delta, \textsc{ext}_r(\eta\cup\Delta)$).

```
Input: A batch $\Delta$, updated $\textsc{ext}_r(\eta\cup\Delta)$
Output: A set of assignments indicating rule violations
 1: $Violations := \{\}$;
 2: for each complete body assignment $\mu$ in $\textsc{ext}_r(\eta\cup\Delta)$ do
 3:   if $\mu$ is not extended by any complete head assignment then
 4:     if $\Delta$ contains an END event $e$ with $e.\textsc{id} = \mu.\textsc{id}$ then
 5:       Add $\mu$ to $Violations$ ;
 6:     Let $\tau$ be the maximum deadline for the rows in $\textsc{ext}_r(\eta\cup\Delta)$ with $\mu$;
 7:     if $\mathsf{ts}_\Delta \geqslant \tau > max(\eta)$ then
 8:       Add $\mu$ to $Violations$ ;
 9: output Violations ;
```

In the following theorem, we assert that applying Algorithm 4 reports rule violations at the earliest possible time.

▶ **Theorem 8.** *Let r be a rule, η an enactment, and Δ a batch for η. Then, μ is a violation in $\eta \cup \Delta$ but not in η iff* Detect*$(\Delta, \textsc{ext}_r(\eta \cup \Delta))$ reports μ.*

A sketch of the proof is given in Appendix A. The key idea is that for a given body assignment μ in $\textsc{ext}_r(\eta \cup \Delta)$, by Lemma 6, if μ is a violation, it will be in exclusively unmatched rows in $\textsc{ext}_r(\eta \cup \Delta)$ with a deadline of at most ts_Δ. Then, when Δ is processed, μ can be recognized and reported.

From Theorem 8, we see that our monitoring algorithm reports exactly the set of violations in the enactment as soon as they are permanent. This concludes the presentation of the data structures and sub-routines used in our monitoring algorithm.

5 Optimizations

While the algorithms presented in Section 4 handle the monitoring task, their time and space complexities can be improved. We present one optimization to remove useless assignments using a similar reasoning to deadline calculation, another to avoid repeated computation by tracking which data is new. We report their improvement of relevant algorithms as a factor of the log size $|L|$, the batch size $|\Delta|$, the number of active enactments as approximated by $|\Delta|$, and the number of event atoms in the rule body or head e.

Expiring partial assignments. Early violation detection motivates a similar technique for discarding useless assignments. Partial assignments in BA and HA are *expired* (i.e., useless) if (1) they can no longer be extended because their timestamps and unresolved gap atoms are inconsistent with all possible future assignments, or (2) they are derived from an enactment that has ended. It is much desired to remove expired assignments, and thus reduce the sizes of BA and HA. Calculating expiration times resembles deadline calculation; in fact, the Deadline function is reused. To incorporate expiration time, we augment BA and HA (resp.) with an *expiration* column as new tables BAE and HAE, requiring that incomplete assignments in BAE and HAE be extendable by future events to complete assignments. To maintain this property, Deadline calculates its expiration time for each incomplete assignment with respect to its unresolved gap atoms. Removing expired assignments reduces the size of the BAE and HAE tables from $O(|L|^e)$ to $O(|\Delta|^e)$, which benefits the algorithms in § 4 by reducing the number of computations in Update from $O(|L|^{2e})$ to $O(|\Delta|^{2e})$, and that in Update-E from $O(|L|^e)$ to $O(|\Delta|^e)$. It also improves Update-E by decreasing the number of assignments checked for insertion into EXT (Lines 2 and 4), from $O(|L|^e)$ to $O(|\Delta|^e)$.

Semi-naive MERGE **of assignments.** We can also decrease the number of computations in the Update algorithm by tracking which data generated by the most recent batch. The **while** loop (Lines 7-10) in Update tests pairs of assignments to merge. For each batch Δ, we only need to try pairs that have at least one assignment added from events in Δ, because all other pairs of assignments were considered before Δ arrived. To make Update to reflect this, we use a queue Γ_{new} to hold new assignments generated at Line 6. We exchange the **for** loop in Update (Lines 8-10) to a doubly nested for-loop that iterates through each assignment μ_n in Γ_{new} (outer loop) and each row μ_o in Γ (inner loop), adding the new assignment to the queue Γ_{new}, moving μ_n from Γ_{new} to Γ after processing μ_n. This resembles "semi-naive" evaluation of Datalog programs [1] and reduces the search for matching assignments from considering $O(|L|^{2e}|)$ pairs to only pairs involving some new data: $O(|L|^e|\Delta|^e)$ pairs.

6 Experimental Evaluation

We implemented (Python 3.8.2) a monitoring algorithm using the data structures and algorithms in Section 4 and optimizations in Section 5. Moreover, our implementation handles multiple enactments simultaneously. Using this implementation, we experimentally evaluated the benefits and costs of early violation detection (EVD) and the overall batch processing times. We used logs created by simulating workflow models of the IaaS application in Section 2 with a simulator [25], varying the size of enactments from *normal* enactments (10 events per enactment) to *large* enactments (100 events per enactment) and using batch sizes of 100, 1,000, and 10,000 events. We used logs with an average of 100 concurrent enactments and monitored both *simple* rules (1-2 body atoms, 1-2 head atoms) and *complex* rules (2-4 body atoms, 2-4 head atoms). Our test data is motivated by discovering the feasible ranges for monitoring for enactment and batch size in five target applications areas: (1) healthcare information systems that manage medical services for compliance with patients' medical history, (2) drone management services that enforce geographic fencing and limits on flight time, (3) college admissions portals that manage application due dates and admission decisions, (4) IaaS providers, as illustrated above, and (5) retail websites where customers' orders must be paid for, filled, and delivered in a timely manner. For all experiments, we used a Mac laptop (MacOS Big Sur 12.2.1) with a 3.2 GHz, 8-core Apple M1 processor with 8GB memory.

Our experimental results indicate that early violation detection yields a significant resource savings (Finding 1) with a negligible overhead (Finding 2), and is feasible for enactments with up to 100 events and batches up to 10,000 events (Finding 3). Additionally, we can conclude that our algorithms are appropriate for some application areas of business services.

Finding 1. 16% of events in normal-length violating enactments and 66% of events in large violating enactments may be ignored.

First, we examine how soon violations could be detected with respect to each enactment's events. We report the average percentage of events observed in violating enactments before and after their first reported violation. This number represents the percentage of events that could be ignored, or even prevented, in the case that detecting a violation early halts the enactment's execution. This finding is partially dependent on the percentage of enactments that are violating and the size of gaps in rules as a proportion of enactment duration; future work could analyze these dimensions as factors of the potential savings. Fig. 7 shows the percentage of events observed in violating enactments before and after their first detected violation.

	normal-length enactments		large enactments	
rules	% events before first violation	% after	% events before first violation	% after
simple	74.9	25.1	33.5	66.5
complex	83.7	16.3	69.4	30.6

Figure 7 Percentages of events observed before and after the first detected violation.

Finding 2. The overhead of detecting violations early is $\leqslant$15% compared with the overall processing time, even for large enactments and rules with up to 8 atoms.

The benefits of early violation detection could be nullified if the time to calculate deadlines and find matches is a significant percentage of the overall processing time. As a baseline, we used an algorithm that does not calculate deadlines, and instead, detects and reports violations only once the enactment's *END* event arrives. Fig. 8 compares our monitoring algorithm with EVD to the baseline algorithm (without EVD). The increase in processing time with EVD for normal enactments ($\leqslant$2%) is less than the increase with EVD for large enactments ($\leqslant$15%). This is attributed to the higher number of events with matching data values in large enactments, which increases the number of assignment pairs, thus more deadlines are calculated in Lines 3 and 8 of Algorithm 3.

	normal-length enactments		large enactments	
rules	without EVD	with EVD	without EVD	with EVD
simple	4.27×10^{-2}	4.73×10^{-2} (+0.2%)	6.65×10^{-2}	7.60×10^{-2} (+14.3%)
complex	9.19×10^{-2}	9.31×10^{-2} (+1.3%)	1.840×10^{-1}	2.084×10^{-1} (+13.3%)

Figure 8 Batch processing times (seconds) with and without early violation detection.

Finding 3. Monitoring is feasible for enactments with up to 100 events, and batches of up to 10,000 events, with an arrival rate of 1 second.

We report the average batch processing time for normal and large enactments, simple and complex rules, and batches of 100, 1,000, and 10,000 events. Logs with larger batches were not obtained due to limitations of the simulator. We assume a batch arrival interval of 1 second, thus an average processing time $\leqslant$1 second indicates monitoring is feasible for some application areas, because each batch can (on average) be processed before the following batch arrives, thus no backlog of events accumulates over time. Fig. 9 shows that the average processing time is $\leqslant$1 second for all trials.

As the batch size grows, the number of events processed by Algorithm 2 grows proportionally. Given that most events in a batch are from different enactments, larger batches do not have proportionally more pairs of assignments to compare in Line 8 of Algorithm 2, so these times grow linearly with the batch size as expected. As the enactment length grows, the number of compatible events, and thus partial assignment pairs, grows, increasing the number of matches in Line 8 of Algorithm 2 and the number of updates to the EXT table in Line 4 of Algorithm 3. Then, enactment length accounts for the increase in processing time.

Lastly, we place the results in context for the five application areas. Given that the batch processing times in Finding 3 for enactments with 100 events, batches of 10,000 events, and rules with 8 atoms are below our assumed batch interval of 1 second, applying our algorithms to applications in areas (1) and (2), which feature similar dimensions for enactments and constraints, is feasible. It is also feasible for small applications in areas (3), (4), and (5), though monitoring larger applications with hundreds of thousands of concurrent users or enactments with thousands of events may not be possible. Additionally, Finding 2 suggests whenever monitoring is feasible, early violation detection is also feasible, as it has negligible computational overhead.

	enactment length			
	normal	large	normal	large
batch size	simple rules		complex rules	
100	4.55×10^{-4}	6.19×10^{-4}	7.74×10^{-4}	1.363×10^{-3}
1,000	4.330×10^{-3}	6.177×10^{-3}	7.534×10^{-3}	1.3509×10^{-2}
10,000	4.2414×10^{-2}	6.0769×10^{-2}	7.4925×10^{-2}	1.35218×10^{-1}

Figure 9 Batch processing times (seconds) for different enactments and rules.

7 Conclusions

Techniques for event monitoring are increasing in demand as more software systems generate and/or rely on events. This paper contributes monitoring and violation detection techniques for temporal constraints in workflow systems. More study is needed of the trade-offs of expressiveness of temporal constraints, specifically a comparison of our language's gap atoms with LTL and MTL, as well with extensions of our language with negation for modeling the absence of events. Additionally, it remains to be seen if early violation detection is possible, and then more effective and efficient, with respect to sets of rules, where deadlines may appear earlier due to interactions of "conflicting" constraints, as in [19]. Also, our techniques only consider whether or not a violation is certain; it may be useful to reason about violations probabilistically, which could allow them to be anticipated farther in advance and thus better mitigated.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Howard Barringer, Ylies Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann. *Runtime Verification: First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, volume 6418. Springer, 2010.

3 David Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. Monitoring metric first-order temporal properties. *Journal of the ACM (JACM)*, 62(2):1–45, 2015.

4 Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing ltl semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

5 Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. On enabling time-aware consistency of collaborative cross-org. business processes. In *ICSOC 2014*, pages 351–358. Springer, 2014.

6 Carlo Combi and Roberto Posenato. Towards temporal controllabilities for workflow schemata. In *TIME 2010*, pages 129–136. IEEE, 2010.

7 Riccardo De Masellis, Fabrizio M Maggi, and Marco Montali. Monitoring data-aware business constraints with finite state automata. In *Proceedings of the 2014 International Conference on Software and System Process*, pages 134–143, 2014.

8 Riccardo De Masellis and Jianwen Su. Runtime enforcement of first-order ltl properties on data-aware business processes. In *International Conference on Service-Oriented Computing*, pages 54–68. Springer, 2013.

9 Christophe Dousson and Pierre Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *IJCAI*, volume 7, pages 324–329, 2007.

10 Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of business processes with temporal variables. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 40–47, 2019.

11 Ashish Gupta, Inderpal Singh Mumick, and Venkatramanan Siva Subrahmanian. Maintaining views incrementally. *ACM SIGMOD Record*, 22(2):157–166, 1993.

12 Klaus Havelund and Grigore Rosu. Monitoring programs using rewriting. In *ASE 2001*, pages 135–143. IEEE, 2001.

13 Luke Hunsberger and Roberto Posenato. Sound-and-complete algorithms for checking the dynamic controllability of conditional simple temporal networks with uncertainty. In *25th International Symposium on Temporal Representation and Reasoning (TIME 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

14 Linh Thao Ly, Fabrizio Maria Maggi, Marco Montali, Stefanie Rinderle-Ma, and Wil MP Van Der Aalst. Compliance monitoring in business processes: Functionalities, application, and tool-support. *Information systems*, 54:209–234, 2015.

15 Isaac Mackey and Jianwen Su. Mapping business rules to ltl formulas. In *ICSOC 2019*, pages 563–565, 2019.
16 Isaac Mackey and Jianwen Su. Mapping singly-linked, acyclic rules to linear temporal logic formulas. In submission, 2022.
17 Fabrizio Maria Maggi, Marco Montali, and Ubaier Bhat. Compliance monitoring of multi-perspective declarative process models. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 151–160. IEEE, 2019.
18 Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil MP Van Der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *International Conference on Business Process Management*, pages 132–147. Springer, 2011.
19 Fabrizio Maria Maggi, Michael Westergaard, Marco Montali, and Wil MP van der Aalst. Runtime verification of ltl-based declarative process models. In *International Conference on Runtime Verification*, pages 131–146. Springer, 2011.
20 Alessandro Margara, Emanuele Della Valle, Alexander Artikis, Nesime Tatbul, and Helge Parzyjegla, editors. *International Conference on Distributed and Event-Based Systems*. ACM, ACM, 2021.
21 Marco Montali, Federico Chesani, Paola Mello, and Fabrizio M Maggi. Towards data-aware constraints in declare. In *Proceedings of the 28th annual ACM symposium on applied computing*, pages 1391–1396, 2013.
22 Marco Montali, Fabrizio M Maggi, Federico Chesani, Paola Mello, and Wil MP van der Aalst. Monitoring business constraints with the event calculus. *ACM TIST 2014*, 5(1):1–30, 2014.
23 Milos Nikolic, Mohammad Dashti, and Christoph Koch. How to win a hot dog eating contest: Distributed incremental view maintenance with batch updates. In *Proceedings of the 2016 International Conference on Management of Data*, pages 511–526, 2016.
24 Peter Z Revesz. A closed-form evaluation for datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.
25 Gabriel Siqueria. Log generator. `https://github.com/GabrielSiq/LogGenerator`, 2020.

A Proof Sketches of Lemmas 3, 5, 6 and Theorem 8

▶ **Lemma 3.** *Let $r: \varphi \to \psi$ be a rule, φ_g, ψ_g the gap atoms in φ, ψ (resp.), μ a complete body assignment such that $\mu(\varphi_g)$ is true, β an incomplete head assignment extending μ such that $\beta(\mu(\psi_g))$ is satisfiable, and U the variables in ψ_g undefined by β. Let $\tau = \mathsf{Deadline}(\psi_g, var(\varphi_g \cup \psi_g), \mu \cup \beta)$. The following hold:*

1. *If $\tau \in \mathbb{N}$, then there is a complete head assignment β' extending $\mu \cup \beta$ such that $\min(\beta'(U)) \leqslant \tau$ and $\beta'(\psi_g)$ is true,*
2. *If $\tau \in \mathbb{N}$, then for all complete head assignments β' extending $\mu \cup \beta$ such that $\min(\beta'(U)) > \tau$, $\beta'(\psi_g)$ is false.*
3. *If $\tau = \infty$, then for all timestamps n in $\mathbb{N}$, there is a complete head assignment β' extending $\mu \cup \beta$ such that $\max(\beta'(U)) > n$ and $\beta'(\psi_g)$ is true.*

Proof Sketch for Lemma 3. To show (1), assume there is no complete head assignment β' extending $\mu \cup \beta$ such that $\min(\beta'(U)) \leqslant \tau$ and $\beta'(\psi_g)$ is *true*. Then, $(\mu \cup \beta)(\psi) \wedge (z = \tau)$ is not satisfiable. Then, there is a gap atom in $\mu \cup \beta(\psi)$ that provides an upper bound for z below τ. Then, τ is not the minimum of the upper bounds in *UpperBd*. Thus Algorithm 1 on $\mu \cup \beta$ and ψ should not output τ. This is a contradiction. To show (2), assume some complete head assignment β' extends $\mu \cup \beta$ such that $\min(\beta'(U)) > \tau$ and $\beta'(\psi_g)$ is true. Then, $(\mu \cup \beta)(\psi) \wedge (z = \tau')$ is satisfiable for some z in $var(\psi)$. Then, $\mu(\psi)$ does not imply $z_i \leqslant \tau$ for all variables z_i. Thus Algorithm 1 on $\mu \cup \beta$ and ψ should not output τ. This is a contradiction. To show (3), assume $\tau = \infty$. Algorithm 1 only produces ∞ when $\mu(\psi)$ is satisfiable and for some variable z_i and for all $n \in \mathbb{N}$, $\mu(\psi) \not\to (z_i \leqslant n)$ Then, for all timestamps n in $\mathbb{N}$, there is some complete assignment that extends μ, satisfies ψ, and uses some n' larger than n. Then, μ can be extended arbitrary far in the future. ◀

▶ **Lemma 5.** *Let $r\colon \varphi \to \psi$ be a rule, η an enactment, and Δ a batch for η.* $\mathsf{Update}(\varphi, \Delta, \textsc{ba}_r(\eta))$ *(or* $\mathsf{Update}(\psi, \Delta, \textsc{ha}_r(\eta))$*) computes $\textsc{ba}_r(\eta \cup \Delta)$ (resp. $\textsc{ha}_r(\eta \cup \Delta)$).*

Proof Sketch for Lemma 5. We argue this for $\textsc{ba}_r(\eta)$; adapting this argument for $\textsc{ha}_r(\eta)$ is trivial. For an assignment μ in $\textsc{ba}_r(\eta \cup \Delta)$ created by $\mathsf{Update}(\varphi, \Delta, \textsc{ba}_r(\eta))$, some events C in η and some D in Δ provide values for μ. Then an assignment μ_C for C is present in $\textsc{ba}_r(\eta)$ and Lines 2–5 of Alg. 2 generates $|D|$ assignments for each event in D. Next, these $|D| + 1$ assignments will merge with each other in the loop of Line 7 of Alg. 2 until μ is created and added to Γ. Alternatively, consider any assignment μ that is not in $\textsc{ba}_r(\eta \cup \Delta)$ after Algorithm 2. Then, no subset of events in $\eta \cup \Delta$ can create μ on Line 4 or μ is inconsistent with the rule body or head and will not proceed past Lines 5 or 8 of Alg. 2. ◀

▶ **Lemma 6.** *Let η be an enactment with no* END *event, r a rule, τ a timestamp, and μ a complete body assignment for r. Then, μ is a violation of r in η iff μ occurs in $\textsc{ext}_r(\eta)$ but no rows in $\textsc{ext}_r(\eta)$ pairs μ with a complete head assignment, and each row in $\textsc{ext}_r(\eta)$ with μ has a deadline no greater than τ.*

Proof Sketch for Lemma 6. Let τ be the largest timestamp in η. $\textsc{ext}_r(\eta)$ contains all possible pairs for μ and head assignments from $\textsc{ha}_r(\eta)$, so if μ is unmatched in $\textsc{ba}_r(\eta)$, there is no assignment with $min(\beta) \leqslant \tau$ that extends μ and satisfies ψ. Alternatively, let τ be the largest deadline for μ in $\textsc{ext}_r(\eta)$, by Lemma 3, for all rows with μ and β in $\textsc{ext}_r(\eta)$, for all complete head assignments β' that extend $\mu \cup \beta$, such that $min(\beta'(U)) > \tau$, $\beta'(\psi)$ is inconsistent. Thus, no future (i.e., with a value greater than τ) complete head assignment can extend μ and satisfy ψ. Then, μ will never be extended by a complete head assignment that satisfies ψ, so μ is a violation for η. ◀

▶ **Theorem 8.** *Let r be a rule, η be an enactment, and Δ a batch for η. Then, μ is a violation in $\eta \cup \Delta$ but not in η iff* Detect*$(\Delta, \textsc{ext}_r(\eta \cup \Delta))$ reports μ.*

Proof Sketch for Theorem 8. Let μ be a violation in $\eta \cup \Delta$. $\eta \cup \Delta$ may contain an *END* event and will have no later events, in which case, η*.END* is in Δ and μ will be added to *Violations* on Line 5 of Algorithm 4. Otherwise, by Lemma 6, μ is complete and in exclusively unmatched rows in $\textsc{ext}_r(\eta \cup \Delta)$ with a deadline of, at most, ts_Δ. Then, μ will be added to *Violations* on Line 8 of Algorithm 4.

Conversely, if $\mathsf{Detect}(\Delta, \textsc{ext}_r(\eta \cup \Delta))$ reports μ, then μ is added to *Violations* on Line 5 or Line 8 of Algorithm 4. Given Line 2 of the algorithm, μ must be a complete assignment in $\textsc{ext}_r(\eta \cup \Delta)$ that is not extended by any complete head assignment. Then, either (1) η*.END* in Δ or (2) ts_Δ is greater than or equal to the deadline for μ in all rows in $\textsc{ext}_r(\eta \cup \Delta)$. If (1), then μ is a violation because $\eta \cup \Delta$ will have no later events. If (2), μ is a violation in $\eta \cup \Delta$ by Lemma 6. ◀

The Tail-Recursive Fragment of Timed Recursive CTL

Florian Bruse
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Martin Lange
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Etienne Lozes
Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis,
Université Côte d'Azur, France

Abstract

Timed Recursive CTL (TRCTL) was recently proposed as a merger of two extensions of the well-known branching-time logic CTL: Timed CTL on one hand is interpreted over real-time systems like timed automata, and Recursive CTL (RecCTL) on the other hand obtains high expressiveness through the introduction of a recursion operator. Model checking for the resulting logic is known to be 2-EXPTIME-complete.

The aim of this paper is to investigate the possibility to obtain a fragment of lower complexity without losing too much expressive power. It is obtained by a syntactic property called "tail-recursiveness" that restricts the way that recursive formulas can be built. This restriction is known to decrease the complexity of model checking by half an exponential in the untimed setting. We show that this also works in the real-time world: model checking for the tail-recursive fragment of TRCTL is EXPSPACE-complete. The upper bound is obtained by a standard untiming construction via region graphs, and rests on the known complexity of tail-recursive fragments of higher-order modal logics. The lower bound is established by a reduction from a suitable tiling problem.

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

Keywords and phrases formal specification, temporal logic, real-time systems

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.5

1 Introduction

Models of systems that incorporate real-time aspects play an important role in the specification and verification of the behaviour of embedded systems. Correct functioning of such systems often depends on the satisfaction of constraints that involve concrete times like "*the wing flaps are adjusted within 5msec of a change in vertical angle reported by the gyrometer sensor.*"

Timed automata [3] are a standard model for the abstraction of the behaviour of real-time systems which has been studied well, including ways to extend their expressiveness, cf. [4, 7]. The desired behaviour of such dynamic systems is typically specified using temporal logics that formalise statements about the evolution of such a system's behaviour in time. For example, the above property in a formal syntax yields a formula like $\mathtt{AG}(\mathtt{chng} \rightarrow \mathtt{AF}_{\leq 5}\mathtt{adj})$.

This formula belongs to the real-time temporal logic known as Timed Computation Tree Logic (TCTL) [2]. It extends the well-known simple branching-time temporal logic CTL – essentially a language to formalise nested reachability queries – with the ability to make assertions about the duration of time that passes along the runs of the system. The model checking problem for TCTL (over systems specified as timed automata) is PSPACE-complete [2], i.e. more difficult than the polynomial-time model checking for CTL (over finite transition systems) [15]. Its expressive power in terms of structural properties is limited: similar to CTL, properties specifiable in TCTL are nested reachability queries expressed by a combination of a universal or existential quantification over an execution path and a simple temporal property of the form "something happens eventually / always / until something else happens".

For many verification tasks, such limited expressiveness far below regularity (i.e. definable in MSO) is sufficient. Yet in practice, for example when systems are composed of parallel and interacting components, additional expressiveness raised to full regularity may be needed [23]. In other situations, even full regularity is not enough as there is no formula of a temporal logic of regular expressiveness that e.g. describes the lack of underflows in FIFO or LIFO buffers of unbounded size [21]. One may argue that in practice, a buffer is always bounded but this makes the correctness property depend on the implementation. It should be clear that correctness properties should be formalisable independently of the system that is supposed to satisfy them, for otherwise formal verification could easily be achieved in general.

In order to extend the applicability of formal verification for real-time systems in situations where correctness is a structurally more complex property than what fits into TCTL, we have recently proposed Timed Recursive CTL (TRCTL) [12] which merges two extensions of CTL: the aforementioned one by real-time aspects that lifts CTL to TCTL is combined by an extension to properties that are specifiable using recursive property transformers (in the form of first-order functions[1]). This is taken from the untimed world where logics of high expressiveness have been studied for the same reasons as laid out here [18]. The high expressiveness comes at a price, both in terms of computational complexity as well as pragmatics. The syntax of logics like HFL [24] is based on the modal μ-calculus and a typed λ-calculus, and is therefore fairly inaccessible to non-experts and thus not usable at the forefront in system design and verification. Recursive CTL (RecCTL) has therefore been proposed to allow for a reasonable extension of expressive power beyond regularity whilst retaining as much intuitive syntax from CTL as possible.

The model checking problem for TRCTL over timed automata is 2EXPTIME-complete [12]. This may seem odd because that of TCTL is "only" PSPACE-complete, and the extension facilitated by recursive predicates (as least fixpoints of first-order functions) should raise the complexity intuitively by one exponential and result in EXPSPACE-completeness. However, the recursion operator lifts the restriction to a fixed system of nested reachability queries built into CTL's syntax. Hence, said recursion operator not only introduces predicates with unbounded recursion in the world of first-order functions, but also for ordinary predicates in temporal formulas. Thus, adding the recursion operator implicitly turns the base logic from TCTL into a timed variant of the modal μ-calculus. Such temporal logics which can express unbounded recursion – here in the form of least and greatest fixpoints of predicates – typically have model checking complexities that are complete for time classes.

The timed μ-calculus does not feature as prominently in the literature as its untimed counterpart, the modal μ-calculus [17], probably due to the combination of real-time operators and explicit fixpoint operators which may be unsellable to an ordinary user in formal verification. There are also syntactic variants in the literature that could be covered by the generic term *timed μ-calculus* [2, 16]. Going further into this is beyond the scope of this paper; we simply note that "the" timed μ-calculus (obtained from TRCTL as the restriction without first-order elements) has an EXPTIME-complete model checking problem [1] and can therefore be seen as a fragment of TRCTL of lower complexity, yet a regular one.

[1] See the formal definition of the syntax in Sect. 2.2 for an explanation of what "first-order" means here.

In this paper we investigate the question after the existence of a fragment of TRCTL whose expressiveness remains reasonably beyond regularity and whose model checking complexity is genuinely lower than that of full TRCTL (up to the current knowledge in complexity theory). We employ a syntactic restriction called tail-recursiveness which limits the ways that recursive properties can be defined. In untimed logics, tail-recursiveness leads to lower complexity [13], and it is characteristic of space rather than time complexity. The result in this paper therefore fits into what can be expected from previous work on timed and non-regular specification languages: the model checking problem for tail-recursive TRCTL is "only" EXPSPACE-complete, i.e. exactly one exponential worse than that of its untimed counterpart. The upper bound is established making use of the well-known region-graph abstraction [3]. The lower bound is established by a reduction from a suitable tiling problem.

The paper is organised as follows. In Sect. 2 we recall preliminaries on timed automata and TRCTL. In Sect. 3 we introduce tail-recursiveness, define the fragment under consideration here and argue why it can be model checked in exponential space. In Sect 4 we present the more elaborate lower bound construction. Sect. 5 contains some remarks on further work.

2 Preliminaries

2.1 Timed Automata

Timed Transition Systems. A *timed labelled transition system* (TLTS) over a finite set *Prop* of atomic propositions (and a single, anonymous[2] action) is a $\mathcal{T} = (\mathcal{S}, \rightarrow, s_0, \lambda)$ s.t.

- $\mathcal{S}$ is a set of *states* containing a designated starting state s_0,
- $\rightarrow \subseteq \mathcal{S} \times \mathcal{S} \cup \mathcal{S} \times \mathbb{R}^{\geq 0} \times \mathcal{S}$ is the transition relation, consisting of two kinds:
  - *discrete transitions* of the form $s \rightarrow t$ for $s, t \in \mathcal{S}$, and
  - *delay transitions* of the form $s \xrightarrow{d} t$ for $s, t \in \mathcal{S}$ and $d \in \mathbb{R}^{\geq 0}$, satisfying $s \xrightarrow{0} t$ iff $s = t$ for any $s, t \in \mathcal{S}$, and

$$\forall d, d_1, d_2 \in \mathbb{R}^{\geq 0}, \forall s, t \in \mathcal{S} : d = d_1 + d_2 \text{ and } s \xrightarrow{d} t \Leftrightarrow \exists u \in \mathcal{S} \text{ s.t. } s \xrightarrow{d_1} u \text{ and } u \xrightarrow{d_2} t\,,$$

- $\lambda : \mathcal{S} \rightarrow 2^{Prop}$ labels each state with the set of atomic propositions that hold true in it.

The *extended transition relations* $\overset{d}{\Longrightarrow}$, $d \in \mathbb{R}^{\geq 0}$, are obtained by padding discrete transitions with delays:

$$s \overset{d}{\Longrightarrow} t \quad \text{iff} \quad \exists d_1, d_2 \in \mathbb{R}^{\geq 0}, s', t' \in \mathcal{S} \text{ s.t. } s \xrightarrow{d_1} s', s' \rightarrow t', t' \xrightarrow{d_2} t \text{ and } d = d_1 + d_2$$

A *trace* is a sequence $\pi = s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} \ldots$

An (untimed) labeled transition system (LTS) is a TLTS over an empty delay transition relation. It is *finite* if the set of its states is finite.

Clock Constraints. Let $\mathcal{X} = \{\mathtt{x}, \mathtt{y}, \ldots\}$ be a set of $\mathbb{R}^{\geq 0}$-valued variables called *clocks*. By $CC(\mathcal{X})$ we denote the set of *clock constraints* over $\mathcal{X}$ which are conjunctive formulas of the form $\top$ or $\mathtt{x} \oplus c$ for $\mathtt{x} \in \mathcal{X}$, $c \in \mathbb{N}$ and $\oplus \in \{\leq, <, \geq, >, =\}$. We write $\mathtt{x} \in [c, c']$ for $\mathtt{x} \geq c \wedge \mathtt{x} \leq c'$, and similarly for open interval bounds.

A *clock evaluation* is an $\eta : \mathcal{X} \rightarrow \mathbb{R}^{\geq 0}$. A clock constraint φ is interpreted in a clock evaluation η in the obvious way:

[2] CTL-based logics are usually oblivious to action labels, whence we restrict ourselves to a single action.

- $\eta \models \top$ holds for any η,
- $\eta \models \varphi_1 \wedge \varphi_2$ if $\eta \models \varphi_1$ and $\eta \models \varphi_2$,
- $\eta \models \mathtt{x} \oplus c$ if $\eta(\mathtt{x}) \oplus c$ for $\oplus \in \{\leq, <, \geq, >, =\}$.

Given a clock evaluation η, $d \in \mathbb{R}^{\geq 0}$ and a set $R \subseteq \mathcal{X}$, we write $\eta{+}d$ for the clock evaluation that is defined by $(\eta{+}d)(\mathtt{x}) = \eta(\mathtt{x}) + d$ for any $\mathtt{x} \in \mathcal{X}$, and $\eta|_R$ for the clock evaluation that is defined by $\eta|_R(\mathtt{x}) = 0$ for $\mathtt{x} \in R$ and $\eta|_R(\mathtt{x}) = \eta(\mathtt{x})$ otherwise.

Timed Automata. Again, since the CTL-based logics considered here are oblivious of action names, we introduce *timed automata* (TA) over a single anonymous action. Such a TA over *Prop* is an $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$ where

- L is a finite set of so-called *locations* containing a designated *initial* location $\ell_0 \in L$,
- $\mathcal{X}$ is a finite set of clocks,
- $\iota : L \to CC(\mathcal{X})$ assigns a clock constraint, called *invariant*, to each location,
- $\delta \subseteq L \times CC(\mathcal{X}) \times 2^{\mathcal{X}} \times L$ is a finite set of transitions. We write $\ell \xrightarrow{g,R} \ell'$ instead of $(\ell, g, R, \ell') \in \delta$. In such a transition, g is called the *guard*, and $R \subseteq \mathcal{X}$ are the *reset* clocks of this transition,
- $\lambda : L \to 2^{Prop}$ labels each location with the set of atomic propositions that hold true in it.

The *index* $m(\mathcal{A})$ of $\mathcal{A}$ is the largest constant occurring in its invariants or guards. Its *size* is

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

Note that the size is only logarithmic in the value of constants used in clock constraints as they can be represented in binary notation for instance.

TA are models of state-based real-time systems. The semantics, resp. behaviour of a TA $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$ is given by a TLTS $\mathcal{T}_{\mathcal{A}}$ over the time domain $\mathbb{R}^{\geq 0}$ as follows.

- The state set is $\mathcal{S} = \{(\ell, \eta) \mid \ell \in L, \eta \in (\mathcal{X} \to \mathbb{R}^{\geq 0})$ such that $\eta \models \iota(\ell)\}$, consisting of pairs of locations and clock evaluations that satisfy the location's invariant.
- The initial state is $s_0 = (\ell_0, \eta_0)$ where $\eta_0(\mathtt{x}) = 0$ for all $\mathtt{x} \in \mathcal{X}$.
- Delay transitions retain the location and (possibly) advance the value of clocks in a state: for any $(\ell, \eta) \in \mathcal{S}$ and $d \in \mathbb{R}^{\geq 0}$ we have $(\ell, \eta) \xrightarrow{d} (\ell, \eta{+}d)$ if $\eta{+}d' \models \iota(\ell)$ for all $d' \leq d$.
- Discrete transitions possibly change the location and reset clocks: for any $(\ell, \eta) \in \mathcal{S}$, $\ell' \in L$ and $R \subseteq \mathcal{X}$ we have $(\ell, \eta) \to (\ell', \eta|_R)$ if there is $g \in CC(\mathcal{X})$ such that $(\ell, g, R, \ell') \in \delta$ and $\eta \models g$ as well as $\eta|_R \models \iota(\ell')$.
- The propositional label of a state is that of its underlying location: $\lambda(\ell, \eta) = \lambda(\ell)$.
- Clock constraints hold in a state if they hold for its clocks: $(\ell, \eta) \models \chi$ iff $\eta \models \chi$.

In other words, a TA finitely represents a TLTS. However, not every TLTS is finitely representable. For a detailed introduction to timed automata we refer to the literature [3, 6]. Henceforth, we will only consider TLTS that arise from a TA. Consequently, we can always assume that the interpretation of clock constraints like $\mathtt{x} \leq 4$ in a TLTS is well-defined.

The Region Abstraction. There is a well-known abstraction of a TLTS $\mathcal{T}_{\mathcal{A}}$ into a finite LTS known as the *region graph* $\mathcal{R}_{\mathcal{A}}$ [3], used in decidability proofs for decision problems on TA.

In the following we only consider TLTS $\mathcal{T}_{\mathcal{A}}$ that arise from some TA $\mathcal{A} = (L, \mathcal{X}, \ell_0, \iota, \delta, \lambda)$. The region abstraction is a mapping of such $\mathbb{R}^{\geq 0}$-TLTS into finite LTS. It is based on an equivalence relation $\simeq_m$, for $m \in \mathbb{N}$, on clock evaluations defined as follows.

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

Here, $\mathit{frac}(r)$ denotes the fractional part of a real number. It is easy to see that $\simeq_m$ is indeed an equivalence relation for any m. It is lifted to states of the TLTS $\mathcal{T}_\mathcal{A}$ in the most straight-forward way: $(\ell, \eta) \simeq_m (\ell', \eta')$ iff $\ell = \ell'$ and $\eta \simeq_m \eta'$.

We write $[\eta]_m$ for the equivalence class of η under $\simeq_m$ and likewise for $[(\ell, \eta)]_m$. When m is clear from the context we may also drop it and simply write $[\eta]$, resp. $[(\ell, \eta)]$.

Note that $\simeq_m$ is a bisimulation on the state space of $\mathcal{T}_\mathcal{A}$ w.r.t. the labelling and discrete and delay transitions: if $(\ell, \eta) \simeq_m (\ell', \eta')$ then we have $\lambda([(\ell, \eta)]) = \lambda([(\ell', \eta')])$ and for every ℓ'', η'': $[(\ell, \eta)] \to [(\ell'', \eta'')]$ iff $[(\ell', \eta')] \to [(\ell'', \eta'')]$. This is what makes it usable for an abstraction of the uncountable state space of $\mathcal{T}_\mathcal{A}$ into a finite discrete state space as follows.

The *region graph* $\mathcal{R}_\mathcal{A}$ of the TA $\mathcal{A}$ is the LTS $(\mathcal{S}, \to, s_0, \lambda)$ obtained as the quotient of $\mathcal{T}_\mathcal{A}$ under $\simeq_m$ with $m := m(\mathcal{A})$, together with an additional collapse of delay transitions for different delays into a single "*some-delay*" value τ. Its components are as follows.

- $\mathcal{S} = \{[(\ell, \eta)]_m \mid \ell \in L, \eta \in (\mathcal{X} \to \mathbb{R}^{\ge 0}), \eta \models \iota(\ell)\}$, and $s_0 = [(\ell_0, \eta_0)]_m$.
- Discrete transitions from one state to another are obtained by possibly delaying, then performing a discrete transition, then possibly delaying again afterwards. We have

  $$[(\ell, \eta)]_m \to [(\ell', \eta')]_m \quad \text{if there are } d, d' \in \mathbb{R}^{\ge 0}, \hat{\eta}, \hat{\eta}' \text{ s.t. } (\ell, \eta) \xrightarrow{d_1} (\ell, \hat{\eta}) \to (\ell', \hat{\eta}') \xrightarrow{d_2} (\ell', \eta')$$

  for any $\ell, \ell' \in L$, $\eta, \eta' \in \mathcal{X} \to \mathbb{R}^{\ge 0}$.
- The propositional labelling is given as $\lambda([(\ell, \eta)]_m) = \lambda(\ell, \eta) = \lambda(\ell)$.

▶ **Proposition 1** ([3]). *Let $\mathcal{A}$ be a TA over n clocks with ℓ locations and of index m. Then $\mathcal{R}_\mathcal{A}$ is an (untimed) LTS of size $\ell \cdot 2^{\mathcal{O}(n(\log n + \log m))}$, i.e. exponential in $|\mathcal{A}|$, and there is a path $s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} \dots$ in $\mathcal{T}_\mathcal{A}$ iff there is a path $[s_0] \to [s_1] \to \dots$ in $\mathcal{R}_\mathcal{A}$.*

2.2 Timed Recursive Computation-Tree Logic

TRCTL incorporates the two extensions from CTL to TCTL introducing real-time and to RecCTL introducing recursive predicates.

Syntax. Let *Prop* be a set of atomic propositions. Let $\mathcal{V}_1 = \{x, y, \dots\}$ be a set of propositional variables and $\mathcal{V}_2 = \{\mathcal{F}, \dots\}$ be a set of so-called *recursion* variables. Formulas of TRCTL are given by the following grammar.

$$\begin{aligned}\varphi &::= q \mid x \mid \chi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathtt{E}(\varphi\, \mathtt{U}_J\, \varphi) \mid \mathtt{A}(\varphi\, \mathtt{U}_J\, \varphi) \mid \Phi(\varphi, \dots, \varphi) \\ \Phi &::= \mathcal{F} \mid \mathtt{rec}\, \mathcal{F}(x_1, \dots, x_k).\, \varphi\end{aligned}$$

where $q \in \mathit{Prop}$, J denotes an interval in $\mathbb{R}^{\ge 0}$ with rational bounds, χ is a clock constraint, and $x, x_i, y_i \in \mathcal{V}_1$, $\mathcal{F} \in \mathcal{V}_2$.

The (sub-)formulas derived from φ are called *propositional*, those derived from Φ are called *first-order*. We allow further Boolean operators like $\mathtt{tt}, \mathtt{ff}, \to$, etc. and temporal operators like $\mathtt{EF}, \mathtt{EG}, \mathtt{AF}, \mathtt{AG}$, etc. through their standard abbreviations. We also avoid parentheses through the standard precedence rules and remove empty tuples, i.e. we write $\mathtt{rec}\, \mathcal{F}.\varphi$ instead of $(\mathtt{rec}\, \mathcal{F}().\varphi)()$. We also consider $\mathtt{rec}\, \mathcal{F}.\varphi$ as a propositional rather than a first-order formula, because it results from the application of a first-order formula to zero arguments.

The grammar above allows non-well-formed formulas to be constructed, too. These need to be excluded using a stronger mechanism than context-free grammars. We refer to [12] for a formal definition of a (simple but tedious) typing system for well-formedness and introduce this notion intuitively instead: a well-formed formula obeys the following two restrictions.

- The numbers of formal parameters and arguments coincide, i.e. in a formula of the form $(\texttt{rec}\,\mathcal{F}(x_1,\dots,x_n).\varphi)(\psi_1,\dots,\psi_m)$ we must have $n = m$. However, arguments can also be passed to a first-order formula of the form $\mathcal{F}$ where the parameters are not visible. We assume that each recursion variable is bound by the recursion operator at most once, whence, we can associate with each (bound) $\mathcal{F}$ an arity given by the number of parameters in its definition. This must then also match the number of arguments passed to it.
- The recursion operator is explained semantically via least fixpoints in function lattices. For this to be well-defined, each recursive call must occur positively in its defining body. Violations occur, e.g., in $\mathcal{F}().\neg\mathcal{F}$ or in $\texttt{rec}\,\mathcal{F}.(\texttt{rec}\,\mathcal{G}(x).\neg x)(\mathcal{F})$ where both recursion variables $\mathcal{F}$ and $\mathcal{G}$ appear to be used positively only, resp. not at all. Hence, a simple criterion like occurrence under an even number of negation symbols does not capture well-definedness as negative occurrences can be hidden in function applications.

Semantics. Formulas of TRCTL are interpreted over timed transition systems $\mathcal{T} = (\mathcal{S}, \rightarrow, s_0, \lambda)$ (as arising from TA for example). A propositional formula φ denotes a set of states $[\![\varphi]\!]^{\mathcal{T}} \subseteq \mathcal{S}$, while a first-order formula with k formal parameters denotes a function $[\![\Phi]\!]^{\mathcal{T}} : (2^{\mathcal{S}})^k \rightarrow 2^{\mathcal{S}}$ that maps k sets of such states to a set of states. The semantics is given inductively, which is why environments α are needed in order to explain the meaning of free variables. Formally, α maps propositional variables x to sets of states and first-order variables $\mathcal{F}$ to functions as stated above. The semantics is then defined via

$$[\![q]\!]^{\mathcal{T}}_\alpha := \{s \mid q \in \lambda(s)\} \qquad [\![x]\!]^{\mathcal{T}}_\alpha := \alpha(x) \qquad [\![\chi]\!]^{\mathcal{T}}_\alpha = \{s \mid s \models \chi\}$$
$$[\![\neg\varphi]\!]^{\mathcal{T}}_\alpha := \mathcal{S} \setminus [\![\varphi]\!]^{\mathcal{T}}_\alpha \qquad [\![\varphi \vee \varphi]\!]^{\mathcal{T}}_\alpha := [\![\varphi]\!]^{\mathcal{T}}_\alpha \cup [\![\psi]\!]^{\mathcal{T}}_\alpha \qquad [\![\varphi \wedge \varphi]\!]^{\mathcal{T}}_\alpha := [\![\varphi]\!]^{\mathcal{T}}_\alpha \cap [\![\psi]\!]^{\mathcal{T}}_\alpha$$

and

$$\begin{aligned}
[\![\texttt{E}(\varphi\,\texttt{U}_J\,\varphi)]\!]^{\mathcal{T}}_\alpha &:= \{s \mid \text{ there is a path } \pi = s,\dots \text{ s.t. } \mathcal{T},\pi \models_\alpha \varphi\,\texttt{U}_J\,\psi\} \\
[\![\texttt{A}(\varphi\,\texttt{U}_J\,\varphi)]\!]^{\mathcal{T}}_\alpha &:= \{s \mid \text{ for all paths } \pi = s,\dots \text{ we have } \mathcal{T},\pi \models_\alpha \varphi\,\texttt{U}_J\,\psi\} \\
[\![\Phi(\varphi_1,\dots,\varphi_n)]\!]^{\mathcal{T}}_\alpha &:= [\![\Phi]\!]^{\mathcal{T}}_\alpha([\![\varphi_1]\!]^{\mathcal{T}}_\alpha,\dots,[\![\varphi_n]\!]^{\mathcal{T}}_\alpha) \\
[\![\mathcal{F}]\!]^{\mathcal{T}}_\alpha &:= \alpha(\mathcal{F}) \\
[\![\texttt{rec}\,\mathcal{F}(x_1,\dots,x_n).\,\varphi]\!]^{\mathcal{T}}_\alpha &:= \bigsqcap\{f : (2^{\mathcal{S}})^n \rightarrow 2^{\mathcal{S}} \mid \text{ for all } T_1,\dots,T_n \subseteq \mathcal{S} \text{ we have} \\
&\qquad\qquad [\![\varphi]\!]^{\mathcal{T}}_{\alpha[\mathcal{F}\mapsto f, x_1\mapsto T_1,\dots x_n\mapsto T_n]} \subseteq f(T_1,\dots,T_n)\}
\end{aligned}$$

where $(\bigsqcap_{i\in I} f_i)(T_1,\dots,T_n) := \bigcap_{i\in I} f_i(T_1,\dots,T_n)$ and the satisfaction of a $\texttt{U}$-property by a non-Zeno path $\pi = s_0 \overset{d_0}{\Longrightarrow} s_1 \overset{d_1}{\Longrightarrow} s_2 \overset{d_2}{\Longrightarrow} \dots$ in the TLTS is given as follows. We have $\pi \models_\alpha \varphi\,\texttt{U}_J\,\psi$ iff

$$\exists i \geq 0, \exists d \in [0, d_i], \exists s' \text{ s.t. } s_i \overset{d}{\Longrightarrow} s' \text{ and } (\sum_{h=0}^{i} d_i) + d \in J \text{ and } s' \in [\![\psi]\!]^{\mathcal{T}}_\alpha \text{ and}$$
$$\forall j < i, \forall d' \in [0, d_j], \forall s' \text{ s.t. } s_j \overset{d'}{\Longrightarrow} s' \text{ we have } s' \in [\![\varphi \vee \psi]\!]^{\mathcal{T}}_\alpha \text{ and}$$
$$\forall d' \in [0, d), \forall s' \text{ s.t. } s_i \overset{d'}{\Longrightarrow} s' \text{ we have } s' \in [\![\varphi \vee \psi]\!]^{\mathcal{T}}_\alpha.$$

This may seem odd at first glance but it is in fact standard to interpret an Until operator in this way in the real-time setting, cf. [16, 6]. The sum on the right-hand side is simply used to express the reaching of some state in the future by delay steps along multiple transitions, hence the time that passes up to this step is being added up. The other possibly unintuitive feature is the seemingly weak assertion on s' (under the universal quantification) to satisfy φ *or* ψ, where one may assume it to have to satisfy φ. First note that allowing such "earlier" moments to also satisfy ψ instead of φ is not harmful to the intuitive meaning of $\varphi \, \mathtt{U} \, \psi$: if ψ holds at some point but also earlier as well, then it still holds at some point. In a discrete-time setting there is always a first moment at which ψ holds, and it suffices when all moments before that satisfy φ. However, in the real-time setting there may not be a first moment for ψ to hold. So it is in fact necessary to allow these earlier moments to also satisfy the Until's right argument. This ensures, for example, that a formula like $\mathtt{E}(\mathtt{x}{=}0 \, \mathtt{U} \, \mathtt{x}{>}0)$ is satisfiable. Note that, after a moment satisfying $\mathtt{x} = 0$, there is no first moment satisfying $\mathtt{x} > 0$, but intuitively the formula should be satisfiable.

For a closed formula φ and arbitrary $s \in \mathcal{S}$ we write $\mathcal{T}, s \models \varphi$ if $s \in [\![\varphi]\!]^{\mathcal{T}}$ for arbitrary $s \in \mathcal{S}$, and also $\mathcal{T} \models \varphi$ if $\mathcal{T}, s_0 \models \varphi$.

Examples. TRCTL is able to express structurally complex properties of real-time systems. We give two examples which show how the recursion operator can be used to create combinations of temporal formulas that could not be expressed in logics of regular expressiveness only.

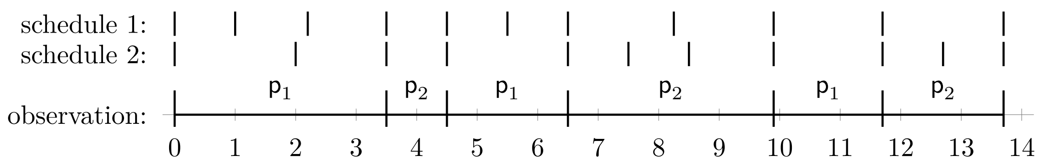
▶ **Example 2.** Consider a TLTS over propositions including $\{\mathsf{r}, \mathsf{g}\}$ which signal the request of a resource respectively the granting of such a request. The TRCTL formula

$$\big(\mathtt{rec}\, \mathcal{F}(x,y).(x \to y) \wedge \mathcal{F}(\mathtt{EF}^{\leq 2} x, \mathtt{EF}^{\leq 3} y)\big)(\mathsf{r}, \mathsf{g})$$

then states "whenever a request is issued after at most $2n$ time units, then a grant is issued after at most $3n$ time units (for the same n)". To see that this is indeed expressed by the formula one only needs three principles: (I) unfolding of recursive definitions and (II) replacement of parameter variables by arguments, and (III) the temporal simplification rule $\mathtt{EF}^{\leq c}\mathtt{EF}^{\leq d}\psi \equiv \mathtt{EF}^{\leq c+d}\psi$. To keep the calculation short we identify $\mathcal{F}$ with $\mathtt{rec}\, \mathcal{F}(x,y).(x \to y) \wedge \mathcal{F}(\mathtt{EF}^{\leq 2} x, \mathtt{EF}^{\leq 3} y)$. Then we have

$$\begin{aligned}
\mathcal{F}(\mathsf{r},\mathsf{g}) &\equiv (\mathsf{r} \to \mathsf{g}) \wedge \mathcal{F}(\mathtt{EF}^{\leq 2}\mathsf{r}, \mathtt{EF}^{\leq 3}\mathsf{g}) \\
&\equiv (\mathsf{r} \to \mathsf{g}) \wedge (\mathtt{EF}^{\leq 2}\mathsf{r} \to \mathtt{EF}^{\leq 3}\mathsf{g}) \wedge \mathcal{F}(\mathtt{EF}^{\leq 2}\mathtt{EF}^{\leq 2}\mathsf{r}, \mathtt{EF}^{\leq 3}\mathtt{EF}^{\leq 3}\mathsf{g}) \\
&\equiv (\mathsf{r} \to \mathsf{g}) \wedge (\mathtt{EF}^{\leq 2}\mathsf{r} \to \mathtt{EF}^{\leq 3}\mathsf{g}) \wedge \mathcal{F}(\mathtt{EF}^{\leq 4}\mathsf{r}, \mathtt{EF}^{\leq 6}\mathsf{g}) \\
&\equiv (\mathsf{r} \to \mathsf{g}) \wedge (\mathtt{EF}^{\leq 2}\mathsf{r} \to \mathtt{EF}^{\leq 3}\mathsf{g}) \wedge (\mathtt{EF}^{\leq 4}\mathsf{r} \to \mathtt{EF}^{\leq 6}\mathsf{g}) \wedge \mathcal{F}(\mathtt{EF}^{\leq 6}\mathsf{r}, \mathtt{EF}^{\leq 9}\mathsf{g}) \\
&\equiv \;\cdots\; \equiv \bigwedge_{n \geq 0} \mathtt{EF}^{\leq 2n}\mathsf{r} \to \mathtt{EF}^{\leq 3n}\mathsf{g}.
\end{aligned}$$

▶ **Example 3.** Take a timed system in which a scheduler governs the execution of two different processes. We assume that proposition p_i, $i \in \{1,2\}$ holds whenever process i is active, that at any moment exactly one of them holds, and that the execution of a process takes between 1 and 2 time units. A possible trace of such a system w.r.t. only the two propositions p_1 and p_2 in real time is represented by the bottom line in this picture:



The trace divides the real line into intervals during which either of the two propositions in question holds. Clearly, from this trace we can derive that the scheduler finished an execution of process 1 and started an execution of process 2 at time point 3.5 for example. Only the switches from one process to another are visible. The finishing of process i and subsequent rescheduling of it cannot be inferred from these propositions alone. All that is known, for example, is that between time moments 0 and 3.5, there must have been at least 2 and at most 3 executions of process 1. Thus, such a trace can result from several different schedulings; two of these are depicted here on top. In schedule 1, three instances of process 1 have been completed before time point 3.5, in schedule 2 only two of them have been run, etc.

Now suppose that there is an additional constraint stating that at any moment, process 2 may never have been scheduled more often than process 1. Note that schedule 1 satisfies this property but schedule 2 does not since, at time point 9.9, only three instances of process 1 have been completed but already four instances of process 2 are done.

The TRCTL formula

$$\neg\Big(\big(\texttt{rec}\,\mathcal{F}(x).\texttt{E}(\texttt{p}_2\,\texttt{U}^{\geq 1}\,x) \vee \texttt{E}(\texttt{p}_1\,\texttt{U}^{\leq 2}\,\mathcal{F}(\mathcal{F}(x)))\big)(\texttt{tt})\Big)$$

guarantees the absence of such faulty schedulings. It states that it is not possible to find a path and its division into intervals of length at least 1, resp. at most 2, depending on whether $\texttt{p}_1$ or $\texttt{p}_2$ holds at the moment, such that at some point the number of $\texttt{p}_2$-intervals has exceeded the number of $\texttt{p}_1$-intervals seen so far.

To understand how this is expressed it is probably best to remember that the context-free grammar $F \to b \mid aFF$ generates all (minimal) words w s.t. $|w|_b > |w|_a$ but $|v|_b \leq |v|_a$ for any prefix v of w. Note how the formula above follows exactly this structure. By unfolding the recursion and replacing arguments successively, $\mathcal{F}(\texttt{tt})$ can be seen to be equivalent to a disjunction of nested EU-formulas like

$$\texttt{E}(\texttt{p}_1\,\texttt{U}^{\leq 2}\,\texttt{E}(\texttt{p}_1\,\texttt{U}^{\leq 2}\,\texttt{E}(\texttt{p}_2\,\texttt{U}^{\geq 1}\,\texttt{E}(\texttt{p}_1\,\texttt{U}^{\leq 2}\,\texttt{E}(\texttt{p}_2\,\texttt{U}^{\geq 1}\,\texttt{E}(\texttt{p}_2\,\texttt{U}^{\geq 1}\,\texttt{E}(\texttt{p}_2\,\texttt{U}^{\geq 1}\,\texttt{tt})))))))$$

which is satisfied by the trace presented above as schedule 2 shows. Each such disjunct demands one more occurrence of $\texttt{p}_2$- than $\texttt{p}_1$-intervals.

3 The Tail-Recursive Fragment of Timed Recursive CTL

3.1 The Syntactical Restriction

In functional programming, a definition of a recursive function is tail recursive if the return value of a function is the return value of a recursive call without alterations. Tail-recursive functions are often more efficient to evaluate since one does not need to remember intermediate variable bindings. This concept also yields improved efficiency in the evaluation of formulas defined via fixpoints, cf. e.g. [13, 9]. For this, it is crucial that the interplay between the fixpoints and the recursive definitions is not too complex.

$$\frac{}{\emptyset \vdash_{\mathsf{tr}} p} \qquad \frac{}{\emptyset \vdash_{\mathsf{tr}} x} \qquad \frac{}{\emptyset \vdash_{\mathsf{tr}} \chi} \qquad \frac{}{\{\mathcal{F}\} \vdash_{\mathsf{tr}} \mathcal{F}} \qquad \frac{\emptyset \vdash_{\mathsf{tr}} \varphi}{\emptyset \vdash_{\mathsf{tr}} \neg\varphi} \qquad \frac{\mathcal{V} \vdash_{\mathsf{tr}} \varphi_1 \qquad \mathcal{V}' \vdash_{\mathsf{tr}} \varphi_2}{\mathcal{V} \cup \mathcal{V}' \vdash_{\mathsf{tr}} \varphi_1 \lor \varphi_2}$$

$$\frac{\emptyset \vdash_{\mathsf{tr}} \varphi_1 \qquad \mathcal{V} \vdash_{\mathsf{tr}} \varphi_2}{\mathcal{V} \vdash_{\mathsf{tr}} \varphi_1 \land \varphi_2} \qquad \frac{\emptyset \vdash_{\mathsf{tr}} \varphi_1 \qquad \mathcal{V} \vdash_{\mathsf{tr}} \varphi_2}{\mathcal{V} \vdash_{\mathsf{tr}} \mathtt{E}(\varphi_1 \, \mathtt{U}_J \, \varphi_2)} \qquad \frac{\emptyset \vdash_{\mathsf{tr}} \varphi_1 \qquad \emptyset \vdash_{\mathsf{tr}} \varphi_2}{\emptyset \vdash_{\mathsf{tr}} \mathtt{A}(\varphi_1 \, \mathtt{U}_J \, \varphi_2)}$$

$$\frac{\mathcal{V} \vdash_{\mathsf{tr}} \Phi \qquad \emptyset \vdash_{\mathsf{tr}} \varphi_1 \qquad \cdots \qquad \emptyset \vdash_{\mathsf{tr}} \varphi_n}{\mathcal{V} \vdash_{\mathsf{tr}} \Phi(\varphi_1, \ldots, \varphi_n)} \qquad \frac{\mathcal{V} \vdash_{\mathsf{tr}} \varphi}{\mathcal{V} \setminus \{\mathcal{F}\} \vdash_{\mathsf{tr}} \mathtt{rec}\, \mathcal{F}(x_1, \ldots, x_m).\varphi}$$

Figure 1 Derivation rules for establishing tail-recursiveness. The sets $\mathcal{V}$ and $\mathcal{V}'$ denote the set of free fixpoint variables of the formula in question.

In our setting this means that recursion variables cannot appear in an operand setting, which is the equivalent to the above stipulation that the return value of a function is the return value of the recursive call, emphwithout alterations. Moreover, we also can have (almost) no branching introduced by boolean alternation, so at most one subformula of a formula of the form $\varphi_1 \land \varphi_2$ can have free recursion variables. If this is satisfied, the formula without free variables can be evaluated first in a suitable model-checking procedure (cf. [13]) until the recursion resumes in the other subformula. Note that U-formulas introduce hidden branching through their definition. The reason for this stipulation is that nondeterminism and universal nondeterminism introduce branching in the flow of a program if both branches contain recursive calls. However, it is well-known that space complexity classes from PSPACE and upwards admit free nondeterminism via Savitch's Theorem [20], hence one kind of nondeterminism can be mixed with recursive calls. Boolean alternation, however, can not be mixed with recursion without breaking this property. Since we make use of Savitch's Theorem to relax the requirements on disjunctions, we restrict boolean alternation. Also, this means that we have to restrict the use of negation, which would turn nondeterministic branching into universal branching. The above considerations are condensed into the derivation system in Fig. 1, giving the following definition:

▶ **Definition 4.** *A* TRCTL *formula φ is called* tail recursive *if the statement $\mathcal{V} \vdash_{\mathsf{tr}} \varphi$ for some, potentially empty, set of recursion variables $\mathcal{V}$ can be derived in the derivation system given in Fig. 1. We write* trTRCTL *for the fragment of all tail-recursive formulas.*

The derivation system in Fig. 1 is to be understood as follows: The set $\mathcal{V}$ in front of the $\vdash$ simply collects the set of free recursion variables of the subformula in question. For example, the subformula p has no free recursion variables, since it is a proposition, and neither does the subformula x, since x is not a recursion variable. The system also enforces the above stipulations: a formula directly under a negation can have no free recursion variables, but note that something like $\neg\mathcal{F}.\ p \lor \mathtt{E}(q \, \mathtt{U}_J \, \mathcal{F})$ is permitted. Disjunctions can contain free recursion variables on both sides, while conjunctions, including those introduced by U formulas, may contain free variables only on one side of the conjunction. The reason for this is that the subformula that is closed w.r.t. recursion can be evaluated first in a non-recursive fashion, and then recursion can proceed in a tail-recursive fashion on the other side. Hence, $\mathtt{rec}\,\mathcal{F}.\mathtt{E}(\mathtt{z} < 3 \, \mathtt{U} \, \mathcal{F})$ is tail-recursive, but $\mathtt{rec}\,\mathcal{F}(x).\mathcal{F}(p) \land \mathcal{F}(x \lor \mathtt{z} \le 2)$ is not since it contains the recursion variable $\mathcal{F}$ on both sides of the conjunction.

Finally, applications may not contain subformulas with free recursion variables on the operand side. Hence, $\mathtt{rec}\, \mathcal{F}(x).x \wedge \mathcal{F}(\mathcal{F}(\mathtt{E}(\mathtt{z} \leq 3)\, \mathtt{U}\, x))$ is not tail recursive, and neither is the formula constructed in Ex. 3. However, the one from Ex. 2 is tail recursive.

3.2 Model Checking in Exponential Space

Tail recursiveness can be applied to the untimed logic RecCTL resulting in the fragment trRecCTL. Using the untiming construction we can then reduce the model checking problem for trTRCTL over TA to that of trRecCTL (over an exponentially larger LTS) whose complexity is not difficult to estimate.

▶ **Theorem 5.** *Model checking* trRecCTL *is in PSPACE.*

Proof. The model checking problem for (untimed, non-tail-recursive) RecCTL is known to be EXPTIME-complete [11]. The argument for the upper bound uses a conceptually simple polynomial translation into HFL^1, the first-order fragment of Higher-Order Fixpoint Logic whose model checking problem is known to be EXPTIME-complete [5]. The translation from RecCTL to HFL^1, when applied to a tail-recursive formula, also produces a formula of tail-recursive HFL^1. The model checking problem for this is known to be easier, namely only PSPACE-complete [13], which establishes the claim. ◀

We now lift this to an upper bound for trTRCTL via standard constructions.

▶ **Theorem 6.** *The* trTRCTL *model checking problem over TA is decidable in* EXPSPACE*.*

Proof. Let $\varphi \in$ trTRCTL and $\mathcal{A}$ be a TA not using the clock $\mathtt{z}$. We construct an LTS $\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}}$ by extending the original region graph $\mathcal{R}_{\mathcal{A}}$ for $\mathcal{A}$ to make clock values visible to the formula.

- For each state $[(\ell,\eta)]$ and each $c \leq m(\varphi)$, add the proposition $p_{\mathtt{z}\oplus c}$ to $\lambda([(\ell,\eta)])$ if $\eta \models \mathtt{z} \oplus c$ for $\oplus \in \{\leq,<,\geq,>,=\}$.
- For each state $[(\ell,\eta)]$ introduce a new state $s_{[(\ell,\eta)]}$ with the sole label $\{r_{\mathtt{z}}\}$, and add transitions $[(\ell,\eta)] \to s_{[(\ell,\eta)]} \to [(\ell,\eta|_{\{\mathtt{z}\}})]$.

The formula $\varphi^{\mathtt{z}}$ results from φ by replacing each subformula of the form

- χ by p_{χ},
- $\mathtt{E}(\psi_1\, \mathtt{U}_{[c,d]}\, \psi_2)$ by $\mathtt{EX}(r_{\mathtt{z}} \wedge \mathtt{EXE}((\neg r_{\mathtt{z}} \wedge \psi_1)\, \mathtt{U}\, (\neg r_{\mathtt{z}} \wedge p_{\mathtt{z}\in[c,d]} \wedge \psi_2)))$,
- $\mathtt{A}(\psi_1\, \mathtt{U}_{[c,d]}\, \psi_2)$ by $\mathtt{EX}(r_{\mathtt{z}} \wedge \mathtt{EXA}((\neg r_{\mathtt{z}} \to \psi_1)\, \mathtt{U}\, (\neg r_{\mathtt{z}} \to p_{\mathtt{z}\in[c,d]} \wedge \psi_2)))$.

For open intervals on one side, the p-propositions are amended accordingly to $p_{\mathtt{z}>c}$ etc.

We observe that $\varphi^{\mathtt{z}}$ is a formula of (untimed) tail-recursive RecCTL that is constructible in time $\mathcal{O}(|\varphi|)$, and $\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}}$ is an (untimed) LTS of size at most (singly) exponential in $|\mathcal{A}|$ and $m(\varphi)$ and also constructible in such time. It is then standard to show, by induction on the structure of φ, that $\mathcal{T}_{\mathcal{A}} \models \varphi$ iff $\mathcal{R}^{\varphi}_{\mathcal{A}^{\mathtt{z}}} \models \varphi^{\mathtt{z}}$. This establishes an exponential reduction from trTRCTL model checking to trRecCTL model checking and, thus, an EXPSPACE bound on the former due to Thm. 5. ◀

4 An Exponential Space Lower Bound for Model Checking

The aim of this section is to provide a lower bound on model checking trTRCTL, matching the exponential-space upper bound in Thm. 6. For this, we first introduce the *exponential corridor tiling problem* ExpTiling, known to be EXPSPACE-complete.

4.1 Exponential Space Complexity

A *tiling system* is a $\mathcal{W} = (T, H, V, t_I, t_F)$ s.t. T is a finite set of tile types, $H, V \subseteq T \times T$ are two binary relations on T called *horizontal*, resp. *vertical matching relation*, and t_0 and t_{fin} are two designated so-called *initial* and *final* tiles.

A $C \subseteq \mathbb{N} \times \mathbb{N}$ is called *closed*, if for all $(i, j) \in C$ we have:

- if $i > 0$ then $(i-1, j) \in C$, and
- if $j > 0$ then $(i, j-1) \in C$.

A (valid) *$\mathcal{W}$-tiling* of such a closed subspace (for the tiling system $\mathcal{W}$ above) is a $\tau : C \to T$ that satisfies the following properties.

- $\tau(0,0) = t_I$,
- for all $(i+1, j) \in C$ we have $(\tau(i,j), \tau(i+1,j)) \in H$,
- for all $(i, j+1) \in C$ we have $(\tau(i,j), \tau(i,j+1)) \in V$, and
- there are i, j s.t. $\tau(i,j) = t_F$.

The *exponential corridor tiling problem* ($\mathsf{ExpTiling}$) is the following.

given:	a tiling system system $\mathcal{W} = (T, H, V, t_I, t_F)$ and a number n encoded unarily
decide:	is there an m and a valid $\mathcal{W}$-tiling τ of the space $[2^n] \times [m]$?

In this case, we simply also say that there is a valid $\mathcal{W}$-tiling on the 2^n-corridor.

Note that $|T|^{2^n}$ is an upper bound on the minimal m witnessing the existence of a $\mathcal{W}$-tiling. Also, we can always assume that t_{fin} is placed in the final row $m-1$, as any closed subspace of a correctly tiled space which includes t_{fin} can also be given a valid $\mathcal{W}$-tiling.

Intuitively, the problem $\mathsf{ExpTiling}$ asks for the existence of a run of a nondeterministic, exponential-space bounded Turing Machine such that the configurations are abstractly represented as rows of tiles of width 2^n. The vertical matching relation in the tiling assures that each following configuration, resp. row, matches the one below according to a finite set of rules (which can be used to model the local rewriting behaviour of a Turing Machine). The horizontal matching relation is needed in order to assure that local transformations in a Turing Machine configuration only happen in a single place, namely where the tape head is located. A more detailed exposition and explanation of the connection between Turing Machine runs and tilings can be found in [14, Sect. 11.1].

▶ **Proposition 7** ([22]). *The problem ExpTiling is* EXPSPACE*-complete.*

We remark that $\mathsf{ExpTiling}$ is also EXPSPACE-hard when n is given in binary coding but the upper bound would not hold anymore. Moreover, the reduction to model checking trTRCTL presented below relies on the ability to write down clock constraints like $\mathtt{x} \leq 2^n - 1$ in polynomial time using binary encoding. This would be rather difficult for n encoded binarily.

4.2 The Reduction

Given a tiling system $\mathcal{W} = (T, H, V, t_I, t_F)$ with designated initial and final tiles $t_I, t_F \in T$, and a number $n \in \mathbb{N}$ encoded unarily, we construct – in time polynomial in $|\mathcal{W}|$ and n – a timed automaton $\mathcal{A}_{\mathcal{W}}$ with some location t_0, and a trTRCTL formula $\varphi_{\mathcal{W},n}$ s.t.

$$\mathcal{T}_{\mathcal{W}}, [(t_0, \mathtt{x} \mapsto 0)] \models \varphi_{\mathcal{W},n} \qquad \text{iff} \qquad \text{there is a valid } \mathcal{W}\text{-tiling on the } 2^n\text{-corridor}$$

where $\mathcal{T}_{\mathcal{W}}$ is the TLTS associated with the timed automaton $\mathcal{A}_{\mathcal{W}}$. The single clock $\mathtt{x}$ involved in this construction is never used in the TA's transitions or locations. Instead it only occurs in $\varphi_{\mathcal{W},n}$ in order to state that something happens along runs during the first $2^n - 1$ units.

In order to keep $\varphi_{\mathcal{W},n}$ tail-recursive, we cannot mimic the reduction from a similar problem showing 2-EXPTIME-hardness of model checking the full logic TRCTL. In that reduction, the constructed formula employs a recursion subformula $\texttt{rec}\,\mathcal{F}(s,t).\dots$ with two parameters s and t that encode, respectively, the indices of the coordinates of a tile. But then we would have to state that the tile located at (s,t) matches its right neighbour horizontally *and* its neighbour above vertically which leads to a body of the recursion formula, roughly containing something like $\mathcal{F}(s',t) \land \mathcal{F}(s,t')$ which renders the entire formula non-tail-recursive.

Instead, the trick is to construct $\varphi_{\mathcal{W},n}$ such that it uses unary recursion formulas of the form $\texttt{rec}\,\mathcal{F}(r)$ with a single propositional variable r only, interpreted as a set of states of $\mathcal{T}_{\mathcal{W}}$, encoding an entire row of a possible $\mathcal{W}$-tiling. For this we simply let $\mathcal{A}_{\mathcal{W}}$ consist of $|T|$ many locations, arranged in a full clique such that, at any moment in time, a transition from any t to any t' (including t itself) is possible. We assume that $T = \{t_0,\dots,t_{k-1}\}$ for some $k \in \mathbb{N}$, as we will need a total order on T later on. We also use T as atomic propositions and let each location t satisfy the proposition t uniquely. Invariants or guards are not needed. Hence, a run through $\mathcal{T}_{\mathcal{W}}$ can freely traverse through the locations in T and change between them at any moment in time.

The crucial intuition for this reduction is the following: a valid $\mathcal{W}$-tiling of the $[2^n] \times [m]$-corridor exists for some $m \geq 1$, iff there is a sequence $R_0,\dots,R_{m-1}$ of *rows*, i.e. $\mathcal{W}$-tilings of the $[2^n] \times [1]$-corridor each, such that vertical matching is guaranteed between them. I.e. if $R_i = t_{i,0},\dots,t_{i,2^n-1}$ and $R_{i+1} = t_{i+1,0},\dots,t_{i+1,2^n-1}$ then $(t_{i,j},t_{i+1,j}) \in V$ for all $j \in [2^n]$. Within each row, the horizontal matching relation needs to be obeyed of course.

The existence of such a sequence can be expressed by a recursion formula that, intuitively, takes an initial row and, for as long as the current row is not final, generates vertically matching successor rows and continues the search with one of them. For this we need to encode such rows as formulas. Propositional formulas are interpreted as sets of states in $\mathcal{T}_{\mathcal{W}}$, i.e. objects of the form $[(t, \mathtt{x} \mapsto v)]$, since the locations are just the tiles from T and the only clock that is used here is $\mathtt{x}$. This gives rise to a canonical representation of such a row r_i as the set containing exactly the pairs (t_{i_j}, j) for $j = 0,\dots,2^n-1$. For simplicity we write (t,x) instead of $[(t,\mathtt{x} \mapsto x)]$. In the following, x will implicitly be understood as a value of clock $\mathtt{x}$. Moreover, we will write $[\![\cdot]\!]^\alpha$ for $[\![\cdot]\!]^\alpha_{\mathcal{T}_{\mathcal{W}}}$.

▶ **Definition 8.** *A propositional formula ψ is said to be a (representation of) a* row candidate *(under α) if there are $t_{i_0},\dots,t_{i_{2^n-1}} \in T$ s.t. $[\![\psi]\!]_\alpha = \{(t_{i_0},0),(t_{i_1},1),\dots,(t_{i_{2^n-1}},2^n-1)\}$.*

A row *is such a row candidate that additionally satisfies: $(t_{i_j},t_{i_{j+1}}) \in H$ for all $j \in [2^n-1]$.*

Let *first* $:= t_0 \land \mathtt{EF}^*_{=1}(\mathtt{x} = 2^n-1)$ where $\mathtt{EF}^*_{=1}\psi := \texttt{rec}\,\mathcal{G}.\psi \lor \mathtt{EF}_{=1}\mathcal{G}$ expresses that some state satisfying ψ can be reached in an integer interval of time. It is satisfied by a state (t,x) iff $t = t_0$ and $x \in [2^n]$. The second conjunct ensures that x must be an integer value. Hence, any state in the set defined by *first* must combine these two properties. Since exactly the states $(t_0,0),\dots,(t_0,2^n--1)$, satisfy both properties, *first* defines the set $\{(t_0,0),\dots,(t_0,2^n-1)\}$ or, likewise, it represents the row candidate $t_0,\dots,t_0$.

A row is a row candidate in which adjacent tiles match w.r.t. H. This is also easy to express:

$$row(r) := \mathtt{AG}_{\leq 2^n-1}\big(r \rightarrow \bigwedge_{(t,t')\notin H} t \rightarrow \mathtt{AG}_{=1}(r \rightarrow \neg t')\big)$$

Here we use that $\mathcal{A}_{\mathcal{W}}$ forms a clique and uses no clocks. Hence, any state (t',x') of $\mathcal{T}_{\mathcal{W}}$ is reachable from any state (t,x) for as long as $x \leq x'$.

▶ **Lemma 9.** *Let α be an interpretation mapping the propositional variable r to a row candidate $\alpha(r)$. Then $(t_0, 0) \in [\![row(r)]\!]_\alpha$ iff $\alpha(r)$ is in fact a row.*

There is no reason other than notational canonicity to choose t_0 as the location in which $row(r)$ is being evaluated. The statement also holds for any other location. What is important for the following arguments is that it is evaluated in a state with clock value 0. In a state (t, x) with $x > 0$ one simply cannot access all the tiles contained in a row candidate because time only moves forward and states (t', x') with $x' < x$ are not reachable from (t, x).

The next construction is more involved. Ultimately, we want to enumerate all possible row candidates in order to choose a next row in the iterative process described above. There is a standard way of getting from one row candidate to a canonical next one. Starting with the row candidate represented by *first*, we obtain the next one by incrementing a number represented in base-$|T|$ coding, making use of the total order on T given by the indices which makes a row candidate $t_{i_0}, \dots, t_{i_{2^n-1}}$ as a base-$|T|$ number with 2^n many digits. However, here we assume that the least significant digit is on the right, i.e. it is the one indexed $2^n - 1$. The reason for this is that the value of a digit in an incremented number depends on the values of the digits of lesser significance. In trTRCTL and a TA with no clock resets we can only access the future, yet not the past. Letting earlier time moments represent digits of higher significance allows for a simpler encoding of a base-$|T|$ increment operation. Given any row candidate $R_i = t_{i,0}, \dots, t_{i,2^n-1}$, the next one R_{i+1} is obtained using the well-known mechanism of incrementing a number represented in base $|T|$:

$$t_{i+1,j} = \begin{cases} t_0 & \text{, if } t_{i,h} = t_{k-1} \text{ for all } h = j, \dots, 2^n - 1, \\ t_{m+1} & \text{, if } t_{i,j} = t_m, m < k-1 \text{ and } t_{i,h} = t_{k-1} \text{ for all } h = j+1, \dots, 2^n - 1, \\ t_{i,j} & \text{, if there is } h > j \text{ s.t. } t_{i,h} \neq t_{k-1}. \end{cases}$$

This can straightforwardly be formalised as follows.

$$\begin{aligned} next(r) := {} & \big(t_0 \wedge \mathtt{AG}^*_{=1}(r \to t_{k-1})\big) \vee \Big(\bigvee_{m=0}^{k-2} t_{m+1} \wedge \mathtt{EF}_{=0}(r \wedge t_m)\Big) \\ & \vee \Big(\bigvee_{m=0}^{k-1} t_m \wedge \mathtt{EF}_{=0}(r \wedge t_m)\Big) \wedge \mathtt{EF}^+_{=1}(r \wedge \neg t_{k-1}) \end{aligned}$$

where $\mathtt{EF}^+_{=1}\psi := \mathtt{EF}_{=1}\mathtt{EF}^*_{=1}\psi$ and $\mathtt{AG}^*_{=1}\psi := \neg\mathtt{EF}^*_{=1}\neg\psi$.

▶ **Lemma 10.** *Define a sequence of sets of states in $\mathcal{T}_\mathcal{W}$ as follows: $R_0 := [\![first]\!]$, $R_{i+1} := [\![next(r)]\!]_{[r \mapsto R_i]}$.*

a) *R_i is a row candidate for all $i \geq 0$.*

b) *Let $m := |T|^{2^n}$. The sets $R_0, \dots, R_{m-1}$ are pairwise different. Consequently, the sequence $R_0, R_1, \dots$ constitutes an enumeration of all possible row candidates for the given $\mathcal{W}$.*

Using this we can facilitate a search for a row (satisfying some formula $\psi(r)$) by enumerating all row candidates in a recursive iteration and terminating it when a proper row r satisfying $\psi(r)$ has been found.

$$\exists^{\mathsf{row}} r.\psi(r) := \Big(\mathtt{rec}\,\mathcal{G}(r).row(r) \wedge (\psi(r) \vee \mathcal{G}(next(r)))\Big)(first)$$

▶ **Lemma 11.** *Let $\psi(r)$ be a formula. We have $(t_0, 0) \in [\![\exists^{\mathsf{row}} r.\psi(r)]\!]$ iff there is a (representation of a) row R such that $(t_0, 0) \in [\![\psi(r)]\!]_{[r \mapsto R]}$, i.e. that satisfies ψ. Moreover, $\exists^{\mathsf{row}} r.\psi(r)$ is tail-recursive if $\psi(r)$ is so.*

A valid $\mathcal{W}$-tiling is comprised of a sequence of rows, starting with an initial one and ending in a final one. The initial one is such that its first tile, i.e. in position 0, is t_I; a final row is one that contains the final tile t_F. Both are easily specified as follows.

$$\mathit{init}(r) := \mathtt{AG}_{=0}(r \to t_I) \qquad\qquad \mathit{final}(r) := \mathtt{EF}^*_{=1}(r \wedge t_F)$$

▶ **Lemma 12.** *Let R be a (representation of a) row.*

a) $(t_0, 0) \in [\![\mathit{init}(r)]\!]_{[r \mapsto R]}$ *iff* $(t_I, 0) \in R$*, i.e. R starts with the initial tile.*

b) $(t_0, 0) \in [\![\mathit{final}(r)]\!]_{[r \mapsto R]}$ *iff there is* $i \in [2^n - 1]$ *s.t.* $(t_F, i) \in R$*, i.e. R contains the final tile.*

We now construct the overall formula $\varphi_{\mathcal{W},n}$ such that, procedurally thinking, it facilitates a search through the space of rows in order to decide the existence of a valid $\mathcal{W}$-tiling. It starts with some initial row, generates successors (which need to match vertically in all positions of these rows), until a final row has been found. All that is needed at this point is a formula that takes two rows r, r' and decides whether r' can be placed above r in a valid $\mathcal{W}$-tiling, i.e. in any position the tile in t vertically matches the one in t' in this position.

$$\mathit{match}(r, r') := \mathtt{AG}^*_{=1}\big(\bigwedge_{(t,t') \notin V} r \wedge t \to \mathtt{AG}_{=0}(r' \to \neg t')\big)$$

Note that the right part of the implication in this formula asserts that, if a state (ℓ, η) is contained in r, then any state (ℓ', η), i.e. one with the same clock value, is such that the locations ℓ and ℓ' are tiles matching vertically.

▶ **Lemma 13.** *Let* $R = t_0^R, \dots, t_{2^n-1}^R$ *and* $R' = t_0^{R'}, \dots, t_{2^n-1}^{R'}$ *be two rows. We have* $(t_0, 0) \in [\![\mathit{match}(r, r')]\!]_{[r \mapsto R, r' \mapsto R']}$ *iff* $(t_i^R, t_i^{R'}) \in V$ *for all* $i \in [2^n]$*, i.e. R' matches vertically onto R.*

We can then put this all together as follows.

$$\varphi_{\mathcal{W},n} := \exists^{\mathsf{row}} r_0 . \mathsf{init}(r_0) \wedge \Big(\big(\mathtt{rec}\, \mathcal{F}(r) . \mathit{final}(r) \vee \exists^{\mathsf{row}} r' . \mathit{match}(r, r') \wedge \mathcal{F}(r') \big)(r_0) \Big) \tag{1}$$

▶ **Theorem 14.** *The model checking problem for* trTRCTL *over TA is* EXPSPACE*-hard.*

Proof. By reduction from ExpTiling. From given $\mathcal{W} = (T, H, V, t_I, t_F)$ and unary $n \in \mathbb{N}$ we construct $\mathcal{A}_{\mathcal{W}}$ and $\varphi_{\mathcal{W},n}$ as described above. It is not hard to see that this can be done in time polynomial in $|W|$ and n, as the clock constraints of the form $\mathtt{x} = 2^n - 1$ etc. can be written in binary. This is where unary encoding of the parameter n in ExpTiling is needed.

Moreover, $\varphi_{\mathcal{W},n}$ is easily seen to be tail recursive as each first-order fixpoint variable – the $\mathcal{F}$ that is visible in (1) and the two $\mathcal{G}$'s that are implicitly present in the definition of the operator $\exists^{\mathsf{row}}$ – only occurs once in its corresponding body. Note that other variables, like those occurring in the macros $\mathtt{EF}^*_{=1}$ etc., are propositional only. They also occur tail recursively only but this is indeed not required for falling into the fragment trTRCTL.

At last, it remains to argue that the reduction is correct. Indeed, by Lemmas 9–13 we have $\mathcal{T}_{\mathcal{W}}, (t_0, 0) \models \varphi_{\mathcal{W},n}$ iff there is some $m \geq 1$ and a sequence of rows $R_0, \dots, R_{m-1}$ s.t. R_0 is initial, R_{m-1} is final, and R_{i+1} matches vertically onto R_i for all $i = 0, \dots, m-2$. In other words, there is a valid $\mathcal{W}$ tiling for the $[2^n] \times [m]$-corridor. ◀

5 Conclusion & Further Work

We have introduced trTRCTL, the tail-recursive fragment of TRCTL, and shown that its model-checking problem is EXPSPACE-complete. This reinforces the observation made in [10] that the complexity of TRCTL is dominated by the higher-order effects, since the lower bounds are achieved using one clock only. Hence, adding real time to RecCTL simply adds one exponential. Restricting the way recursion works reduces the complexity, while the number of clocks has no impact beyond the first one. This is notably different for TCTL [19].

We have extablished EXPSPACE hardness for the combined complexity of the trTRCTL model checking problem. However, we conjecture that the hardness result already holds for the data complexity.

Given that we now have a sound understanding of the theoretical constraints w.r.t. TRCTL and its derivatives, further research should be focused on practical applications or adding expressive power. The first aspect concerns trTRCTL in particular, as the restriction to tail recursive definitions opens up techniques like local model checking, cf. [9]. The second aspect may include making even more aspects of clock values visible to the logic, for example via so-called diagonal constraints. cf. [8].

References

1 L. Aceto and F. Laroussinie. Is your model checker on time? On the complexity of model checking for timed modal logics. *J. Log. Algebraic Methods Program*, 52-53:7–51, 2002.

2 R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th Ann. IEEE Symp. on Logic in Computer Science, LICS'90*, pages 414–427. IEEE Computer Society Press, 1990. doi:10.1109/LICS.1990.113766.

3 R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

4 R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *Formal Methods for the Design of Real-Time Systems: Revised Lectures of the International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, pages 1–24. Springer, 2004. doi:10.1007/978-3-540-30080-9_1.

5 R. Axelsson, M. Lange, and R. Somla. The complexity of model checking higher-order fixpoint logic. *Log. Meth. in Comp. Sci.*, 3:1–33, 2007. doi:10.2168/LMCS-3(2:7)2007.

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

7 P. Bouyer. Timed automata. In *Handbook of Automata Theory*, pages 1261–1294. European Mathematical Society Publishing House, 2021. doi:10.4171/Automata-2/12.

8 P. Bouyer, F. Laroussinie, N. Markey, J. Ouaknine, and J. Worrell. Timed temporal logics. In *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, volume 10460 of *LNCS*, pages 211–230. Springer, 2017. doi:10.1007/978-3-319-63121-9_11.

9 F. Bruse, J. Kreiker, M. Lange, and M. Sälzer. Local higher-order fixpoint iteration. In *Proc. 11th Int. Symp. on Games, Automata, Logics, and Formal Verification, GandALF'20*, volume 326 of *EPTCS*, pages 97–113, 2020. doi:10.4204/EPTCS.326.7.

10 F. Bruse and M. Lange. Model checking timed recursive CTL. Submitted to Inf. and Comp.

11 F. Bruse and M. Lange. Temporal logic with recursion. In *Proc. 27th Int. Symp. on Temporal Representation and Reasoning, TIME'20*, volume 178 of *LIPIcs*, pages 6:1–6:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.TIME.2020.6.

12 F. Bruse and M. Lange. Model checking timed recursive CTL. In *Proc. 28th Int. Symp. on Temporal Representation and Reasoning, TIME'21*, volume 206 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

13 F. Bruse, M. Lange, and E. Lozes. Space-efficient fragments of higher-order fixpoint logic. In M. Hague and I. Potapov, editors, *Proc. 11th Int. Workshop on Reachability Problems, 2017, London, UK*, volume 10506 of *LNCS*, pages 26–41. Springer, 2017. doi:10.1007/978-3-319-67089-8_3.

14 S. Demri, V. Goranko, and M. Lange. *Temporal Logics in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. doi:10.1017/CBO9781139236119.

15 E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982. doi:10.1016/0167-6423(83)90017-5.

16 T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

17 D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

18 M. Lange. Specifying program properties using modal fixpoint logics: A survey of results. In *Proc. 8th Indian Conf. on Logic and Its Applications, ICLA'19*, volume 11600 of *LNCS*, pages 42–51. Springer, 2019.

19 F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking timed automata with one or two clocks. In *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, volume 3170 of *LNCS*, pages 387–401. Springer, 2004. doi:10.1007/978-3-540-28644-8_25.

20 W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

21 A. P. Sistla, E. M. Clarke, N. Francez, and A. R. Meyer. Can message buffers be axiomatized in linear temporal logic? *Information and Control*, 63(1/2):88–112, 1984.

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

23 M. Y. Vardi. From Church and Prior to PSL. In *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *LNCS*, pages 150–171. Springer, 2008.

24 M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, volume 3170 of *LNCS*, pages 512–528. Springer, 2004. doi:10.1007/978-3-540-28644-8_33.

Decentralised Runtime Verification of Timed Regular Expressions

Victor Roussanaly
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France

Yliès Falcone
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France

Abstract

Ensuring the correctness of distributed cyber-physical systems can be done at runtime by monitoring properties over their behaviour. In a decentralised setting, such behaviour consists of multiple local traces, each offering an incomplete view of the system events to the local monitors, as opposed to the standard centralised setting with a unique global trace. We introduce the first monitoring framework for timed properties described by timed regular expressions over a distributed network of monitors. First, we define functions to rewrite expressions according to partial knowledge for both the centralised and decentralised cases. Then, we define decentralised algorithms for monitors to evaluate properties using these functions, as well as proofs of soundness and eventual completeness of said algorithms. Finally, we implement and evaluate our framework on synthetic timed regular expressions, giving insights on the cost of the centralised and decentralised settings and when to best use each of them.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Regular languages; Theory of computation → Rewrite systems; Theory of computation → Automata over infinite objects; Computer systems organization → Real-time system specification

Keywords and phrases Timed expressions, Timed properties, Monitoring, Runtime verification, Decentralized systems, Asynchronous communication

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.6

Funding *Victor Roussanaly*: H2020- ECSEL-2018-IA call – Grant Agreement number 826276 (CPS4EU).
Yliès Falcone: H2020- ECSEL-2018-IA call – Grant Agreement number 826276 (CPS4EU), Région Auvergne-Rhône- Alpes – “Pack Ambition Recherche” programme, French ANR project ANR-20-CE39-0009 (SEVERITAS), LabEx PERSYVAL-Lab (ANR- 11-LABX-0025-01).

Introduction

Modern systems tend to be more distributed and interconnected. Automated verification methods are required to ensure those systems behave as they should. Moreover, their interactions with their environment are getting increasingly unpredictable, which tends to hinder static verification methods such as model checking, as they require a model of the verified system and do not scale well. On the other hand, runtime verification [12, 4, 13] requires no model. In this area, several monitoring methods detect if a system violates its given specification based on events observed at runtime. In order to express finer properties over more complex behaviour, several algorithms for monitoring properties based on real continuous time have been proposed in [16] and [15]. However, these algorithms assume a central observation point in the system, which might be less robust to an architecture change, more vulnerable to outside attacks, or less compatible with the system's architecture. For this purpose, decentralised monitoring algorithms account for the absence of a central observation point, e.g., [6] and [10]. Existing decentralised monitoring algorithms consider linear discrete time and synchronous communication between the system components. In contrast, we address the verification of timed properties of continuous time in an asynchronous setting.

We introduce a decentralised monitoring algorithm that uses local knowledge of the global behaviour to express a verdict about a verified global timed property. In a synchronous setting, while a component lacks information on what events happened in other components, it can use a round-based approach to consider only a finite number of possibilities about global behaviour. In contrast, in an asynchronous setting, for a given time interval, there are no bounds on the number of events that can happen on another component, meaning that a local monitor should take into account an infinite number of scenarios for the events that it has not seen. Another challenge is that a monitor can be notified of past events from another component and has to update its local knowledge accordingly. Indeed, a method is proposed in [8] to account for a partial view of a global behaviour in a timed context, but it assumes that events that have not been seen cannot be seen afterwards. In our case, we assume that events that are not seen yet can still be seen in the future, and the local knowledge should be updated accordingly.

In this paper, we introduce several algorithms for monitoring timed properties in a decentralised setting, as well as an implementation to simulate and evaluate these algorithms. In Sec. 1, we present timed regular expressions, which we use to specify timed properties, and we explain how we evaluate them on a timed trace. After formally defining the decentralised monitoring problem (Sec. 2), we define a progression function that updates timed regular expression based on the local knowledge, first for the centralised setting and then for the decentralised one (Sec. 3). Afterwards (Sec. 4), we define two algorithms for decentralised monitoring of timed regular expressions, using the progression function mentioned above. Finally (Sec. 5), we present our implementation and experimental results. We compare with related work in Sec. 6 and conclude in Sec. 7.

1 Timed Words and Timed Regular Expressions

We recall the basic notions related to timed words, timed regular expressions, and regular languages in Sec. 1.1, using the same formalism as in [3]. In Sec. 1.2, we introduce reduced timed regular expressions and how to transform timed regular expressions into reduced ones, as they will serve in our monitoring framework. In Sec. 1.3, we transpose the semantics of timed regular expressions, from timed words to timed traces. We also propose new operators for timed traces.

We start by defining some basic notation: $\mathcal{I}$ denotes the set of intervals in $\mathbb{R}^+$ and for S a set, $\mathcal{P}(S)$ denotes the set of subsets of S. We also use $\cdot$ to denote the concatenation between two words.

1.1 Timed Regular Expressions and Timed Regular Languages [3]

We recall the syntax and semantics of timed words and timed regular expressions. Let Σ be an alphabet. A timed word (called *time-event sequence* in [3]) over the alphabet Σ is a word of $\mathbb{R}^+ \cup \Sigma$, composed of events in Σ and numerical values that represent delays, that is, the time between two consecutive events. As such, two consecutive delays can be added, which means that for two timed words u and v and for two delays x and y, $u \cdot x \cdot y \cdot v = u \cdot (x + y) \cdot v$. For example, $0.4 \cdot 1.1 \cdot b \cdot a \cdot 0.1 \cdot 0.2 \cdot c \cdot 2.1$ and $1.5 \cdot b \cdot a \cdot 0.3 \cdot c \cdot 2.1$ represent the same timed word. We denote by $\mathcal{T}(\Sigma)$ the set of timed words over Σ and by ϵ the empty word. A subset of $\mathcal{T}(\Sigma)$ is called *timed language*. A function $\theta : \Sigma_1 \to \Sigma_2 \cup \{\epsilon\}$ is called a *renaming*. We consider its natural extensions $\Sigma_1^* \to \Sigma_2^*$ and $\mathcal{T}(\Sigma_1) \to \mathcal{T}(\Sigma_2)$, and we use the same symbol θ to denote them.

We use the classical word concatenation denoted by $\cdot$, but we also need an *absorbing concatenation* denoted by the operator $\circ$. Let us denote by $\delta : \mathcal{T}(\Sigma) \to \mathbb{R}^+$ the function that returns the sum of delays in a timed word. For two timed words u and v, if there exists a word w such that $\delta(u) \cdot w = v$, then we can define $u \circ v = u \cdot w$. This means that $u \circ v$ is defined if and only if v starts with a delay greater than the sum of delays in u, and if that is the case, then we remove this delay from the front of v before concatenating it to u. For example, $(a \cdot 2 \cdot b) \circ (3 \cdot c) = a \cdot 2 \cdot b \cdot 1 \cdot c$ while $(a \cdot 2 \cdot b) \circ (1 \cdot c)$ is not defined. Concatenation operators are extended to timed languages in the classical way. Moreover, for $n \in \mathbb{N}$ and $n \geq 2$, L^n and $L^{\circ n}$ respectively denote the language obtained by concatenating L with itself using operators $\cdot$ and $\circ$, respectively; while $L^0 = L^{\circ 0} = \{\epsilon\}$.

▶ **Definition 1** (Syntax of timed regular expressions). Timed regular expressions *over* Σ *are defined inductively by the following grammar where* $I \subseteq \mathcal{I}$*,* $a \in \Sigma$*,* L' *a timed regular expression over* Σ' *and* $\theta : \Sigma' \to \Sigma \cup \{\epsilon\}$ *a renaming:*

$$L := \epsilon \mid \underline{a} \mid \langle L \rangle_I \mid L \wedge L \mid L \vee L \mid L \cdot L \mid L \circ L \mid \theta(L') \mid L^\star \mid L^\circledast$$

Intuitively, a timed regular expression can be, respectively, the empty word, the letter a at any time, a language limited to time interval I, the conjunction ($\wedge$), disjunction ($\vee$), concatenation ($\cdot$), and absorbing concatenation ($\circ$) of two timed regular expressions, the renaming obtained through function θ, as well as the Kleene star ($\star$) and the Kleene star using the absorbing concatenation ($\circledast$) applied to a timed regular expression. The set of timed regular expressions obtained as above is denoted by $\mathcal{E}(\Sigma)$.

▶ **Definition 2** (Semantics of timed regular expressions). *The semantics of a timed regular expression is the timed language defined inductively by function* $[\![\cdot]\!] : \mathcal{E}(\Sigma) \to \mathcal{P}(\mathcal{T}(\Sigma))$*:*

- $[\![\epsilon]\!] = \{\epsilon\}$,
- $[\![\underline{a}]\!] = \{r \cdot a \mid r \in \mathbb{R}^+\}$,
- $[\![L_1 \wedge L_2]\!] = [\![L_1]\!] \cap [\![L_2]\!]$,
- $[\![L_1 \vee L_2]\!] = [\![L_1]\!] \cup [\![L_2]\!]$,
- $[\![L_1 \cdot L_2]\!] = [\![L_1]\!] \cdot [\![L_2]\!]$,
- $[\![L_1 \circ L_2]\!] = [\![L_1]\!] \circ [\![L_2]\!]$,
- $[\![\langle L \rangle_I]\!] = \{u \in [\![L_1]\!] \mid \delta(u) \in I\}$,
- $[\![L^\star]\!] = \bigcup_{i=0}^{\infty} [\![L^i]\!]$,
- $[\![L^\circledast]\!] = \bigcup_{i=0}^{\infty} [\![L^{\circ i}]\!]$,
- $[\![\theta(L)]\!] = \{\theta(u) \mid u \in [\![L]\!]\}$.

We call *timed regular languages* the languages defined by timed regular expressions.

▶ **Example 3** (Timed regular expressions). Let us consider some alphabet $\{a, b\}$:

- $(\underline{a} \vee \underline{b})^\star \cdot (\underline{a} \circ \langle \underline{b} \rangle_{[0;1]}) \cdot (\underline{a} \vee \underline{b})^\star$ denotes the language of timed words where at some point b occurs within one time unit after some a;
- $\langle \underline{a} \rangle_{[0;1]}^\star$ denotes the language of timed words composed of a's where each event occurs within one time unit from the previous one;
- $\langle \underline{a} \rangle_{[0;1]}^\circledast$ denotes the language of timed words where all events occur within the first time unit. Note that this is semantically equivalent to $\langle \underline{a}^\circledast \rangle_{[0;1]}$ and $\langle \underline{a}^\star \rangle_{[0;1]}$.

Two timed regular expressions L and L' are *equivalent* if their language is the same, that is, $[\![L]\!] = [\![L']\!]$.

1.2 Reduced Timed Regular Expressions

First, let us introduce reduced timed regular expressions.

▶ **Definition 4** (Reduced timed regular expression). *A* reduced timed regular expression *is a timed regular expression where the time constraining operator $\langle\cdot\rangle_I$ is applied only to singular events.*

Any timed regular expression can be rewritten into an equivalent reduced timed regular expression that defines the same language.

▶ **Proposition 5.** *Let L, L_1, L_2 be some timed regular expressions over Σ. We have:*

- $[\![\langle L_1 \vee L_2\rangle_I]\!] = [\![\langle L_1\rangle_I \vee \langle L_2\rangle_I]\!]$,
- $[\![\langle L_1 \wedge L_2\rangle_I]\!] = [\![\langle L_1\rangle_I \wedge \langle L_2\rangle_I]\!]$,
- $[\![\langle L_1 \circ L_2\rangle_I]\!] = [\![L_1 \circ \langle L_2\rangle_I]\!]$,
- $[\![\langle L_1 \cdot L_2\rangle_I]\!] = [\![L_1 \cdot L_2 \wedge \langle \Sigma^{\circledast}\rangle_I]\!]$,
- $[\![\theta(L)\rangle_I]\!] = [\![\theta(\langle L\rangle_I)]\!]$,
- $[\![\langle\epsilon\rangle_I]\!] = \{\epsilon\}$, *if* $0 \in I$, $\emptyset$ *otherwise,*
- $[\![\langle L^{\circledast}\rangle_I]\!] = [\![\langle\epsilon\rangle_I \vee \langle L^{\circledast} \circ L\rangle_I]\!] = [\![\langle\epsilon\rangle_I \vee L^{\circledast} \circ \langle L\rangle_I]\!]$,
- $[\![\langle L^{\star}\rangle_I]\!] = [\![\langle\epsilon\rangle_I \vee (\langle L^{\star} \cdot L\rangle_I]\!] = [\![\langle\epsilon\rangle_I \vee (L^{\star} \cdot L \wedge \langle \Sigma^{\circledast}\rangle_I)]\!]$.

In the remainder, we only consider reduced timed regular expressions.

1.3 Semantics of Timed Regular Expressions over Timed Traces

In the context of decentralised monitoring, the monitor uses a trace as a sequence of time-stamped events. Formally, a *timed trace* over the alphabet Σ is a finite word over $\Sigma \times \mathbb{R}^+$ such that for two consecutive letters (α_1, t_1) and (α_2, t_2), we have $t_1 \leq t_2$. A timed trace is a sequence of events from Σ where each event is paired with the time at which it occurs. For example, $(a, 1.5) \cdot (b, 3.1) \cdot (a, 3.1) \cdot (c, 3.4)$ is a timed trace. Additionally, we also consider (ϵ, t) where ϵ is the empty word. Although this does not represent an observed event, it can be used to represent the absence of such an event. It can be simplified if there is an element following it with $(\epsilon, t) \cdot (\alpha, t') = (\alpha, t')$ for $\alpha \in \Sigma$. We denote by $\mathcal{R}(\Sigma)$ the set of timed traces over Σ.

For $\pi = (\alpha_1, t_1) \cdots (\alpha_n, t_n)$ a timed trace, let us denote by $\tau_{\text{first}}(\pi) = t_1$ (resp. $\tau_{\text{last}}(\pi) = t_n$) the time at which the first (resp. last) event of π occurs. We denote by $L{\downarrow_t}$ the language represented by $\theta_\epsilon(\langle \underline{x}\rangle_{[t,t]})$ where θ_ϵ is the renaming that maps everything to ϵ. Intuitively, $L{\downarrow_t}$ is the language of timed words $\{t \cdot u \mid u \in [\![L]\!]\}$. Similarly, we define $L{\uparrow^t}$ by shifting all the time constraints that appear before the first concatenation $(\cdot)$ in L by subtracting t. It can be defined inductively as such:

- $\underline{a}{\uparrow^t} = \underline{a}$,
- $(\langle L\rangle_I){\uparrow^t} = \langle L{\uparrow^t}\rangle_{I-t}$
- $(L_1 \wedge L_2){\uparrow^t} = L_1{\uparrow^t} \wedge L_2{\uparrow^t}$,
- $L^{\star}{\uparrow^t} = L{\uparrow^t} \vee L^{\star}$,
- $(L_1 \vee L_2){\uparrow^t} = L_1{\uparrow^t} \vee L_2{\uparrow^t}$,
- $L^{\circledast}{\uparrow^t} = (L{\uparrow^t})^{\circledast}$,
- $(L_1 \cdot L_2){\uparrow^t} = L_1{\uparrow^t} \cdot L_2$,
- $\theta(L){\uparrow^t} = \theta(L{\uparrow^t})$.
- $(L_1 \circ L_2){\uparrow^t} = L_1{\uparrow^t} \circ L_2{\uparrow^t}$,

▶ **Definition 6** (Semantics of timed regular expressions over timed traces). *The semantics of a timed regular expression is the set of timed traces defined inductively by function $[\![\cdot]\!]_{\text{tr}}$:*

- $[\![\epsilon]\!]_{\text{tr}} = \{\epsilon\}$
- $[\![\langle L\rangle_I]\!]_{\text{tr}} = \{\pi \in [\![L_1]\!]_{\text{tr}} \mid \tau_{\text{last}}(u) \in I\}$
- $[\![\underline{a}]\!]_{\text{tr}} = \{(a,t) \mid t \in \mathbb{R}^+\}$
- $[\![L^{\star}]\!]_{\text{tr}} = \bigcup_{i=0}^{\infty} [\![L^i]\!]_{\text{tr}}$
- $[\![L_1 \wedge L_2]\!]_{\text{tr}} = [\![L_1]\!]_{\text{tr}} \cap [\![L_2]\!]_{\text{tr}}$
- $[\![L_1 \vee L_2]\!]_{\text{tr}} = [\![L_1]\!]_{\text{tr}} \cup [\![L_2]\!]_{\text{tr}}$
- $[\![L^{\circledast}]\!]_{\text{tr}} = \bigcup_{i=0}^{\infty} [\![L^{\circ i}]\!]_{\text{tr}}$
- $[\![\theta(L)]\!]_{\text{tr}} = \{\theta(u) \mid u \in [\![(L)]\!]_{\text{tr}}\}$
- $[\![L_1 \circ L_2]\!]_{\text{tr}} = \{u \cdot v \mid u \in [\![L_1]\!]_{\text{tr}}, v \in [\![L_2]\!]_{\text{tr}}\}$
- $[\![L_1 \cdot L_2]\!]_{\text{tr}} = \{u \cdot v \mid u \in [\![L_1]\!]_{\text{tr}}, v \in [\![L_2{\downarrow_{\tau_{\text{last}}(u)}}]\!]_{\text{tr}}\}$

Let us denote by $\omega : \mathcal{R}(\Sigma)\mathcal{T}(\Sigma)$ the function that takes a timed trace $(\alpha_1, t_1) \cdot (\alpha_2, t_2) \cdots (\alpha_n, t_n)$ and returns the time word $t_1 \cdot \alpha_1 \cdot (t_2 - t_1) \cdot \alpha_2 \cdot (t_3 - t_2) \cdots (t_n - t_{n-1}) \cdot \alpha_n$. This function converts a timed trace into a timed word that represents the same sequence of events over physical time.

▶ **Proposition 7.** *For $u \in \mathcal{R}(\Sigma)$ and $L \in \mathcal{E}(\Sigma)$, $u \in [\![L]\!]_{\text{tr}}$ if and only if $\omega(u) \in [\![L]\!]$.*

Note that the timed words produced by function ω do not have a delay at the end. That is why we denote by $\sim$ the relation defined by $u \sim v$ if and only if there exists $x \in \mathbb{R}$ such that $v = u \cdot x$ or $u = v \cdot x$. We can show that every equivalence class in $\mathcal{T}(\Sigma)/\!\sim$ has only one representative that is an image of a timed trace by ω and this equivalence class has one unique reverse image by ω^{-1}.

▶ **Corollary 8.** *For $u \in \mathcal{T}(\Sigma)$ and $L \in \mathcal{E}(\Sigma)$, $u \in [\![L]\!]$ if and only if there exists $v \in \mathcal{T}(\Sigma)$ such that $v \sim u$ and $\omega^{-1}(v) \in [\![L]\!]_{\text{tr}}$.*

This means that the language obtained by applying ω^{-1} to the language of timed words of an expression is exactly the language of timed traces of that expression.

1.4 Global operators

Using the timed traces semantics, we introduce two new operators $\lfloor . \rfloor_I$ and $\lceil \cdot \rceil^I$. $\lceil \cdot \rceil^I$ is similar to the $\langle \cdot \rangle_I$ operators, except that it refers to global time. In this case, global does not refer to the distributed nature of the systems we study, it means that it is based on the absolute time of the events, and not the relative time between two events. Of course, this operator would make no sense in the timed words definition of the semantics, as the concatenation of two words change the value of this global time for the word on the right side of the concatenation. This is why we express the semantics of these operators by extending $[\![\cdot]\!]_{\text{tr}}$.

- $[\![\lceil L \rceil^I]\!] = \{u \in [\![L]\!]_{\text{tr}} \mid \tau_{\text{last}}(u) \in I\}$
- $[\![\lfloor L \rfloor_I]\!] = \{u \in [\![L]\!]_{\text{tr}} \mid \tau_{\text{first}}(u) \in I\}$

We call *global timed regular expressions* an expression that contains these operators and denote the set of expressions over Σ by $\mathcal{GE}(\Sigma)$.

▶ **Proposition 9.** *For all $L \in \mathcal{GE}(\Sigma)$, there exists $L' \in \mathcal{E}(\Sigma)$ such that $[\![L]\!]_{\text{tr}} = [\![L']\!]_{\text{tr}}$.*

This can be proven through direct construction or by using the fact that timed regular expressions are semantically equivalent to timed automata, and as such adding global constraints does not add to the expressiveness of timed automata.

▶ **Example 10.** The expression $\langle \underline{a} \rangle_{[0;3]} \cdot (\langle \underline{b} \rangle_{[0;2]})^{\circledast} \cdot \lfloor (\underline{a} \cdot \underline{b})^{+} \rfloor_{[4;5]}$ is semantically equivalent to $\langle \langle \underline{a} \rangle_{[0;3]} \cdot (\langle \underline{b} \rangle_{[0;2]})^{\circledast} \cdot \underline{a} \rangle_{[4;5]} \cdot \underline{b} \cdot (\underline{a} \cdot \underline{b})^{\star}$.

2 The Decentralised Timed Monitoring Problem

We formally state the decentralised timed monitoring problem as well as its objectives.

Context and notations. Let us suppose that the system at hand consists of n independent components denoted by C_i, for $0 < i \leq n$. Each component C_i emits local events over a local alphabet Σ_i. Let $\Sigma = \bigcup_{i \in [1,n]} \Sigma_i$ be the global alphabet of events. We assume that the local alphabets form a partition of Σ, which means that if $i \neq j$ then $\Sigma_i \cap \Sigma_j = \emptyset$. Let p_i be the projection of a timed trace of $\mathcal{R}(\Sigma)$ onto $\mathcal{R}(\Sigma_i)$ and denote by p the function defined by $p(\sigma) = (p_1(\sigma), p_2(\sigma), ..., p_n(\sigma))$. As such, after observing our local traces $\sigma_1, \sigma_2, ..., \sigma_n$, we can apply p^{-1} to obtain the possible global traces. We also note $\sigma|_t$ as the prefix of σ, which contains all events that occur before t. To each component C_i is attached a monitor M_i, which can observe a trace over Σ_i. We denote the verdict of a monitor by $\mathrm{Verdict}_i : \mathbb{R} \to \{\mathtt{bad}, \mathtt{inconclusive}\}$. Monitors are purposed to detect violations.

Assumptions. Our assumptions on the system are as follows.

- Messages can be exchanged between any pair of monitors.
- The monitors only observe their local trace and the messages they receive.
- If a message is sent, it is eventually received, which means that there is no loss of messages.
- The communication is done through FIFO channels, meaning that the messages sent from one monitor to another are received in the order they were sent.
- All monitors share the same global clock, meaning that there are no clock drifts.

Objectives. Given some timed regular expression L, and a global trace σ, our goal is to find an algorithm for the monitors such that the following properties hold.

▶ **Definition 11** (Definitive Verdict). *If there is a time t for which there is a verdict* $\mathrm{Verdict}_i(t) = \mathtt{bad}$*, then for all $t' \geq t$,* $\mathrm{Verdict}_i(t') = \mathtt{bad}$*.*

▶ **Definition 12** (Soundness). *If there is a time t for which there is a verdict* $\mathrm{Verdict}_i(t) = \mathtt{bad}$ *then $\sigma|_t$ is a bad prefix of L, i.e. for all $\sigma' \in \mathcal{R}(\Sigma)$, if $\sigma|_t$ is a prefix of σ' then $\sigma' \notin [\![L]\!]_{\mathrm{tr}}$.*

▶ **Definition 13** (Eventual completeness). *If there is a time t for which $\sigma|_t$ is a bad prefix of L, then there is a delay $t' \geq 0$ such that* $\mathrm{Verdict}_i(t + t') = \mathtt{bad}$*.*

The goal is for one monitor, using its partial knowledge, to deduce whether or not its partial vision of the trace can be completed to be $[\![L]\!]_{\mathrm{tr}}$.[1] Moreover, we also aim to minimise the time it takes to detect such a violation of the expression. Finally, to limit the communication overhead, we also aim to limit the number of messages sent, as well as the total size of the messages exchanged.

3 Progression for Timed Regular Expressions

We consider decentralised monitors that observe only their local events, indexed with the time at which they happen. For a monitor of index i, we define a *decentralised progression function* $\mathrm{Pr}_i : \mathcal{GE}(\Sigma) \to (\Sigma \times \mathbb{R}^+) \to \mathcal{GE}(\Sigma)$, meaning a function that takes an event locally observed (α, t) and a specification as a timed regular expression L, and returns an expression that represents the new specification now that the monitor knows that (α, t) happened. In a centralised case, where there is only one monitor observing the events in the same order they occur, we denote such function Pr_0 and it should satisfy the following property:

▶ **Definition 14** (Centralised progression function). $\mathrm{Pr}_0 : \mathcal{GE}(\Sigma) \to (\Sigma \times \mathbb{R}^+) \to \mathcal{GE}(\Sigma)$ *is a centralised progression function iff for any expression $L \in \mathcal{GE}(\Sigma)$ and any trace event (α, t),* $\mathrm{Pr}_0(L)(\alpha, t) = \{u \mid (t \cdot \alpha) \circ u \in L\}$*.*

[1] Note that we do not consider the alternative problem of determining that the trace will always be in $[\![L]\!]_{\mathrm{tr}}$ for any possible future completion. This problem is, however, harder, as it requires testing whether or not an expression denotes the universal language, which is undecidable.

In other words, it is a left derivative. If a centralised monitor applies this function successively as it observes the events, and the resulting expression represents the empty language, then the monitor can produce a verdict. Whereas in the decentralised case where a monitor observes a local event (α, t), there might have been other events happening in another component at a time before t. In other words, for the decentralised case, Pr_i must satisfy the following property:

▶ **Definition 15** (Decentralised progression function). $\mathrm{Pr}_i : \mathcal{GE}(\Sigma) \to (\Sigma \times \mathbb{R}^+) \to \mathcal{GE}(\Sigma)$ *is a decentralised progression function iff for any expression* $L \in \mathcal{GE}(\Sigma)$ *and any trace event* (α, t)*, we have:* $[\![\mathrm{Pr}_i(L)(\alpha, t)]\!]_{\mathrm{tr}} = \{u \cdot v \in \mathcal{R}(\Sigma) \mid u \cdot (\alpha, t) \cdot v \in [\![L]\!]_{\mathrm{tr}} \wedge u \in \mathcal{R}(\Sigma \setminus \Sigma_i)\}$.

The above function is not a left derivative, as it allows events that are not from Σ_i to occur before the observed event. We now propose a decentralised progression function and detail its inductive definition. To define such a function, we define a filter function $\Phi : \mathcal{GE}(\Sigma) \to \mathcal{P}(\Sigma) \to \mathcal{GE}(\Sigma)$. The proper definition is given in the appendix. This function is defined such that it removes events of a given alphabet from an expression.

▶ **Proposition 16.** *For* $L \in \mathcal{GE}(\Sigma)$ *and* $\Sigma' \subseteq \Sigma$, $[\![\Phi(L)(\Sigma')]\!] = [\![L \wedge (\Sigma \setminus \Sigma')^\star]\!]$.

In other words, function Φ takes an expression $L \in \mathcal{GE}(\Sigma)$ and an alphabet $\Sigma' \subseteq \Sigma$ and returns an expression that represents the restriction of L to $\Sigma \setminus \Sigma'$. Let us define a progression function Pr_i for component C_i by looking at each case. First, let us look at the base cases:

- $\mathrm{Pr}_i(\underline{a})(\alpha, t) = \epsilon_t$, if $\alpha = a$
- $\mathrm{Pr}_i(\underline{a})(\alpha, t) = \emptyset$, if $\alpha \neq a$
- $\mathrm{Pr}_i(\langle \underline{a} \rangle_I)(\alpha, t) = \emptyset$, if $t \notin I$
- $\mathrm{Pr}_i(\langle \underline{a} \rangle_I)(\alpha, t) = \mathrm{Pr}_i(\underline{a})(\alpha, t)$, if $t \in I$
- $\mathrm{Pr}_i(L_1 \vee L_2)(\alpha, t) = \mathrm{Pr}_i(L_1)(\alpha, t) \vee \mathrm{Pr}_i(L_2)(\alpha, t)$
- $\mathrm{Pr}_i(L_1 \wedge L_2)(\alpha, t) = \mathrm{Pr}_i(L_1)(\alpha, t) \wedge \mathrm{Pr}_i(L_2)(\alpha, t)$

For those cases, the decentralised progression is straightforward, and if the event has been observed, it is removed from the formula. Then, let us look at the absorbing concatenation:

- $\mathrm{Pr}_i(L_1 \circ L_2)(\alpha, t) = (\mathrm{Pr}_i(L_1)(\alpha, t) \circ \lfloor L_2 \rfloor_{[t;\infty[}) \vee \lceil \Phi(L_1)(\Sigma_i) \rceil^{[0;t]} \circ \mathrm{Pr}_i(L_2)(\alpha, t)$
- $\mathrm{Pr}_i(L^{\circledast})(\alpha, t) = \lceil \Phi(L)(\Sigma_i)^{\circledast} \rceil^{[0;t]} \circ \mathrm{Pr}_i(L)(\alpha, t) \circ \lfloor L^{\circledast} \rfloor_{[t;\infty[}$

For the absorbing concatenation $\circ$, we examine whether (α, t) corresponds to an event on each side of the concatenation. If (α, t) is an event on the left side, then events on the right side must have occurred later. Otherwise, if (α, t) is an event on the right side, then the events on the left side must have happened earlier. Since they happened before the current time, it means that the current monitor can not have seen them, so they cannot be events from the monitor of index i.

Let us consider the progression for the global operators. For these cases, we use additional notation. First, for I an interval and $t \in \mathbb{R}$, we write $t < I$ (resp. $t > I$) if and only if for all $t' \in I, t < t'$ (resp. $t > t'$). We also denote by $I\!\downarrow$ the interval obtained by changing the lower bound of I to 0 inclusive, and by $I\!\uparrow$ the interval obtained by changing the upper bound of I to ∞. Hence, we have:

- $\mathrm{Pr}_i(\lceil L \rceil^I)(\alpha, t) = \lceil \mathrm{Pr}_i(L)(\alpha, t) \rceil^{I\downarrow}$, if $t \in I$
- $\mathrm{Pr}_i(\lfloor L \rfloor_I)(\alpha, t) = \lfloor \mathrm{Pr}_i(L)(\alpha, t) \rfloor_{I\uparrow}$, if $t \in I$
- $\mathrm{Pr}_i(\lceil L \rceil^I)(\alpha, t) = \lceil \mathrm{Pr}_i(L)(\alpha, t) \rceil^{I}$, if $t < I$
- $\mathrm{Pr}_i(\lfloor L \rfloor_I)(\alpha, t) = \lfloor \mathrm{Pr}_i(L)(\alpha, t) \rfloor_{I}$, if $t > I$
- $\mathrm{Pr}_i(\lceil L \rceil^I)(\alpha, t) = \emptyset$, otherwise
- $\mathrm{Pr}_i(\lfloor L \rfloor_I)(\alpha, t) = \emptyset$, otherwise

For $\lceil L \rceil^I$, we note that if (α, t) happened and $t \in I$, then it means that we can apply the progression to L provided that all other events occur before t or between t and the upper bound of I, which means that they occur in $I\!\downarrow$. On the other hand, if $t < I$, then it cannot be the last event seen, and we are still waiting for an event in I. We have a similar approach to $\lfloor L \rfloor_I$.

Finally, let us consider the non-absorbing concatenation. We want something similar to $\circ$ for $\cdot$ where we consider whether the event occurred on the right or left side. The problem here is that if the event happened on the right-hand side, then the result of our progression function depends on the time of the last event on the left-hand side since the concatenation "resets" the time at which we interpret the expression. Here we use the fact that the function $t' \to \mathrm{Pr}_i(L{\downarrow_{t'}})(\alpha, t)$ is a piece-wise constant function. This means that we can look at a finite number of possible intervals at which the last element of the left-hand side occurs. By building those intervals and computing the expression associated with each one, we can build a function $\Delta_i : (\mathcal{GE}(\Sigma) \times \mathcal{I}) \to (\Sigma \times \mathbb{R}^+) \to \mathcal{P}(\mathcal{I} \times \mathcal{GE}(\Sigma))$. A more detailed definition of such a function is provided in the appendix.

- $\mathrm{Pr}_i(L_1 \cdot L_2)(\alpha, t) = \bigvee_{(I,L') \in \Delta_i(L_2, \mathbb{R}^+)(\alpha,t)} \lceil \Phi(L_1)(\Sigma_i) \rceil^I \cdot L' \vee \mathrm{Pr}_i(L_1)(\alpha, t) \cdot \lfloor L_2 \rfloor_{[t;\infty[}$
- $\mathrm{Pr}_i(L^\star)(\alpha, t) = \bigvee_{(I,L') \in \Delta_i(L, \mathbb{R}^+)(\alpha,t)} \lceil \Phi(L)(\Sigma_i)^\star \rceil^I \cdot L' \cdot \lfloor L^\star \rfloor_{[t;\infty[}$
- $\mathrm{Pr}_i(\theta(L))(\alpha, t) = \bigvee_{\alpha' \in u^{-1}(\alpha)} \theta(\mathrm{Pr}_i(L)(\alpha', t))$

▶ **Example 17.** Let us consider the language $L = (\underline{a} \vee \underline{b})^{\circledast} \cdot \langle \underline{a} \rangle_{[0;1]}$ with $\Sigma_1 = \{a\}, \Sigma_2 = \{b\}$. Then $\mathrm{Pr}_1(L)(a, 2) = (\lceil \underline{b}^{\circledast} \rceil^{[0;2]} \circ \lfloor (\underline{a} \vee \underline{b})^{\circledast} \cdot \langle \underline{a} \rangle_{[0;1]} \rfloor_{[2;\infty[}) \vee \lceil \underline{b}^{\circledast} \rceil^{[1;2]}$.
This means that, either the a observed was on the left of the concatenation, and in that case we have the language defined by L where before the global time $t = 2$, there can only be b events. Or the a observed was the one on the right side, so we can only observe events b that preceded it, with the last b between the global time $t = 1$ and $t = 2$.

We prove in appendix that this is a decentralised progression function Pr_i following Definition 15. As a consequence, we can prove the following theorem.

▶ **Theorem 18.** *For all i, j, $i \neq j$, for all $a \in \Sigma_i$ and $b \in \Sigma_j$, for all $t_a, t_b \in \mathbb{R}^+$ for all $L \in \mathcal{GE}(\Sigma)$, $[\![\mathrm{Pr}_j((\mathrm{Pr}_i(L)(a, t_a))(b, t_b)]\!]_{\mathrm{tr}} = [\![\mathrm{Pr}_i((\mathrm{Pr}_j(L)(b, t_b))(a, t_a)]\!]_{\mathrm{tr}}$.*

This means that the order at which we observe two events from two different monitors does not matter, and we always obtain an equivalent expression. Note that in the case $\Sigma_i = \Sigma$, then Pr_i also satisfies Definition 14 and we denote it Pr_0, which means that we also have a centralised progression function, defined as a specific case of our decentralised progression function.

Let us inductively define function Pr^* such that for (α, t) a trace event of Σ_i and π a sequence of timed trace events, $\mathrm{Pr}^*(L)((\alpha, t) \cdot \pi) = \mathrm{Pr}^*(\mathrm{Pr}_i(L)(\alpha, t))(\pi)$ and $\mathrm{Pr}^*(L)(\epsilon) = L$. Note that we do not require π to be a timed trace, meaning that the events are not necessarily ordered by ascending time.

▶ **Corollary 19.** *For all $L \in \mathcal{GE}(\Sigma)$, for all π, π' such that for all i, $p_i(\pi) = p_i(\pi')$, $[\![\mathrm{Pr}^*(L)(\pi)]\!]_{\mathrm{tr}} = [\![\mathrm{Pr}^*(L)(\pi')]\!]_{\mathrm{tr}}$*

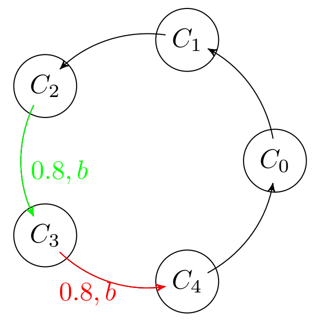
Corollary 19 implies that the order at which the events are observed does not change the language we obtain, provided that the local order on each component is preserved.

4 Decentralised Monitoring

We define two algorithms to achieve decentralised monitoring for timed regular expressions. Both algorithms allow detecting a violation of the specification at runtime. The first algorithm simulates centralised monitoring, while the second algorithm leverages the decentralised progression function.

```
$L_i$ := specification for the system
when an internal event $(t, \alpha)$ is observed do
    Send $(C_i, \alpha, t)$ to $C_{i+1}$
when a message $(C, \alpha, t)$ is received do
    if $k \neq i$ then
        add $(\alpha, t)$ to the memory
        Send $(C, \alpha, t)$ to $C_{i+1}$
    else
        foreach $(\alpha', t')$ in the memory s.t.
          $t' \leq t$ (ordered by ascending $t'$) do
            $L_i := \mathrm{Pro}_0(L)(\alpha', t')$
            Remove $(\alpha', t')$ from memory
        $L_i := \mathrm{Pro}_0(L)(\alpha, t)$
        if $[\![L_i]\!]_{\mathrm{tr}} = \emptyset$ then
            return bad verdict
```

(a) Centralised progression algorithm for C_i.



(b) C_3 sends its message when it observes b at $t = 0.8$. When it receives it later (green message), then it knows for sure that it has seen all events up to $t = 0.8$.

Figure 1 Algorithm for decentralised monitoring using centralised progression.

4.1 Simulating Centralised Monitoring

We place ourselves under the assumptions we described in Sec. 2. That is to say that when a message is sent, it is eventually received with no loss, and we assume *sequential consistency*, meaning that when one component sends multiple messages to another component, then they are received in the same order as they were sent. We also discard the possibility of two events happening at the exact same time in two different components. Since we consider real time, this is not something that would likely occur, but if it happened, we would not know in which order to consider them. But it can also be said that if the order of two simultaneous events on two different components mattered, then the specification would not be adapted to the system either. Finally, we also consider that the components are connected in a ring, as depicted in Figure 1b.

With these assumptions, let us consider that all components apply the algorithm shown in Figure 1a. It means that a component sends events that it sees to its successor. When it receives a message that it did not originate, it saves it into its memory and forwards it to its successor. Using sequential consistency, the following property holds:

▶ **Proposition 20.** *If C_i observes (α, t) and $C_{i'}$ observes (α', t') with $t < t'$, then $C_{i'}$ receives the message (C_i, α, t) before $(C_{i'}, \alpha', t')$.*

From this, we can deduce that when a monitor receives a message it originated, then it saw all the events that happened in the system up to the time when this message was first created. So it can simulate centralised monitoring on its memory up to that point. Using Proposition 20 and Definition 14 we can show the following.

▶ **Theorem 21.** *Let $\sigma \in \mathcal{R}(\Sigma)$ be the global timed trace, L the initial expression, and let (α, t) be the last event of this trace, with $\alpha \in \Sigma_i$. After C_i receives its last message (C_i, α, t), we have $[\![L_i]\!]_{\mathrm{tr}} = \{\sigma' \in \mathcal{R}(\Sigma) \mid \sigma \cdot \sigma' \in L\}$.*

Algorithm 1 Algorithm for C_i.

```
if $i = 1$ then
    $L_i :=$ specification
else
    $L_i := waiting$
when an internal event $(\alpha, t)$ is observed do
    if $L_i = waiting$ then
        add $(\alpha, t)$ to the memory
    else
        $L_i := \mathrm{Pr}_i(L)(e, t)$
        if $[\![L_i]\!]_{\mathrm{tr}} = \emptyset$ then
            return bad verdict
when an expression $L$ is received do
    $L_i = L$ foreach $(\alpha, t)$ in the memory do
        $L_i := \mathrm{Pr}_i(L)(\alpha, t)$
        Remove $(\alpha, t)$ from the memory
        if $[\![L_i]\!]_{\mathrm{tr}} = \emptyset$ then
            return bad verdict
when urgent$(L_i)$ do
    send $L_i$ to target$(L_i)$
    $L_i := waiting$
```

Emptiness checking

We know that, if at some point $[\![L_i]\!]_{\mathrm{tr}} = \emptyset$ for some i, then the specification is violated. This entails testing the emptiness of an expression. One way to do so is to build a timed automaton that recognise the same language, using the construction shown in [3], as we know that the problem of knowing whether or not the language of a timed automaton is empty is PSPACE [1] and there are several tools that solve that problem. While this construction ensures an emptiness check, it is costly, and it is desirable to avoid it at runtime. Instead, we simplify the expression between each progression to detect when the denoted language is empty. The simplification used in this work is very simple. First we simplify every ϵ, ϵ_t or $\emptyset$ that appear in the expression. Then we find nested time constraints in order to merge them and simplify them as much as possible. This could be improved by simplifying conjunctions and disjunctions, but simplification of timed regular expressions is not well documented, and that is why we limit ourselves to these naive simplifications.

One could also notice that if $[\![L_i]\!]_{\mathrm{tr}} = [\![\Sigma^\star]\!]_{\mathrm{tr}}$ then it means that from this point onward the specification will always be satisfied. This means that we could possibly detect whether or not the specification is sure to be satisfied if we can test whether or not the language associated with the expression is universal. Unfortunately, this problem is not decidable in the general case [2], and that is why we only propose verdict `bad`.

4.2 Using Decentralised Progression

We now consider another approach where monitors do not exchange observed events, but instead there is one running expression passed along the monitors and that is updated with their progression function. In that case, at any given time, there is at most one monitor holding the expression and being marked as active. Monitors that are not active are only observing local events and recording them in their local memory. The active monitor updates a running expression using decentralised progression, based on its local observations. At some point, it passes this expression to another monitor and becomes not active. The monitor that receives that expression updates it with its own memory of observed events and becomes active. This is shown in Algorithm 1. Hence, in that case, we do not need sequential consistency, nor do we need the assumption of the components communicating in a ring. This algorithm uses two functions, urgent(L_i) and target(L_i). These functions decide when we want to pass the expression and to whom we want to pass it. There are several possible ways to implement them, as long as the expression is eventually passed to every component.

In that case, the verdict reached by this algorithm is eventually the same as the centralised monitoring as a direct consequence of Corollary 19. We propose the following implementation of these functions. Function urgent(L) replaces every occurrence of $\lceil L' \rceil^I$ by $\emptyset$ and checks whether the resulting expression represents the empty language. If it is not empty, then it means that the specification can still be satisfied, even if nothing has been seen by the other monitors, and the active monitor returns `false`. Otherwise it means that we want to know what the other monitors observed and it returns `true`. The function target(L_i) can be implemented in multiple ways. First, we consider choosing target$(L_i) = i + 1 \bmod n$, which gives us something similar to the centralised approach, where the messages are sent to the successor along the ring. We also propose another implementation where we try to detect which monitor is the most relevant for the past constraints $\lceil L \rceil^I$ present in the formula and choose it as the target.

Note that this approach has several advantages. The main one being that a monitor can decide that the specification has been violated even if it has only a partial view and has not communicated with all the other monitors. This also means that less messages can be exchanged, so the impact of communication delays is reduced. The next section describes experiments with these approaches.

5 Simulation and Benchmarks

In this section, we validate the effectiveness of our decentralised monitoring approaches by showing the benefits over an approach that simulates centralised monitoring. For this, we first briefly describe the implementation of our monitoring algorithms (Sec. 5.1). Then, we detail our experimental setup (Sec. 5.2) and obtained results, as well as some of the conclusions drawn (Sec. 5.3).

5.1 Implementation as an OCaml Benchmark

We implemented a simulation environment for the methods described in the previous sections. For this, we extended DecentMon [6, 7], an OCaml benchmark for decentralised monitoring in the discrete-time setting. We extended it to our timed setting and added support for regular timed expressions and timed traces. We implemented progression-based monitoring for the two methods. The extension for the timed setting consists of 1,400 LLOC. For the approach based on decentralised progression, we consider the following two alternatives for the target component chosen by a monitor when sending a message: (i) a dynamic approach where the target component is chosen based on the current expression, and (ii) a static approach where the target is chosen as the successor in a directed ring, as explained in the previous section.

5.2 Experimental Setup

We test and compare the three approaches for decentralised monitoring of timed regular expressions: (i) simulating centralised, (ii) decentralised with a dynamic target, and (iii) decentralised with a static ring target. We consider a network of 10 monitors, each of them capable of observing a different event. We consider the communication delay when a message is sent as a random value between 0 and 40 units of time. Timed regular expressions are not commonly used, since people prefer timed automata that are as expressive as timed regular expressions. This means that one approach could have been to take a benchmark of timed automata, compute a set of equivalent expressions and use them as a benchmark.

Table 1 Summary of the experimental results.

Algorithm	**Target**	**messages**		**# progressions**	**decision time**
		# sent	total size		
Centralised	–	278.50	10305	278.50	257.99
Decentralised	dynamic	5.91	539426	10.46	142.71
	static	6.70	376944	11.78	143.78

But this poses a problem as the performance would be affected by our choice of equivalent expression. Another difficulty is the absence of reference benchmark for timed decentralised monitoring. That is why in this context, we choose to generate random timed regular expressions. We generate 100 expressions of a fixed size, for each size between 2 and 8. For the time constraints that may appear, we randomly chose the lower bound between 0 and 30, and the upper bound is ∞ or a sum between the lower bound and an integer chosen randomly between 0 and 30. For each of these expressions, we generate 100 timed traces of length 200, with a delay between two consecutive events chosen randomly between 0 and 10. Note that a randomly generated trace will most likely not respect a random specification, and the progression will find an empty language after a couple of steps. In order to avoid those cases, we ensure that for an expression of size k, the progressed expression is not empty after observing the first $5 \times k$ events. To do so, we discard expressions where we cannot find such a trace in a reasonable time. For each trace, we perform monitoring according to the three aforementioned approaches.

During each experiment, we record the number of messages sent, the total size of the messages sent, the number of progressions performed, and the time taken to reach a verdict. This results in 2,900,000 tests for each of the three algorithms.

We evaluate the algorithms along three dimensions that are relevant for decentralised monitoring.

- Communication (messages). We measure the total number and size of the messages exchanged by the monitor. Since there are between 8 and 16 operators, they are encoded over 4 bits. The time values in timed constraints are encoded over 32 bits. There are 10 possible events encoded over 4 bits
- Computation (progressions). We measure the total number of calls to the progression functions defined in the previous sections, including the recursive calls.
- Delay (decision time). We measure the time it takes for one of the monitors to detect that the current global trace violates the monitored timed regular expression.

Note that here we do not consider the computation time associated with each progression computation. We consider that this time should be negligible compared to the communication delays.

5.3 Results and Discussion

Tab. 1 reports the results of our experiments, reporting the average values for each of the metrics. In the following, we discuss the results and conclude.

First, let us compare the two decentralised approaches. Choosing a dynamic target gives us slightly better results on all metrics, except for the size of the messages. The difference remains marginal, and it seems that in our experiments, dynamically choosing the target does not give a significantly shorter decision time. Of course, this stems from the choice function, which can be improved, as it showed poor results in some specific cases. Although these are seldom, their impact on the average values is noticeable. We believe that a more in-depth analysis of these cases can give us a better choice of target.

However, the performance of the decentralised approach is significantly better than that of the centralised approach, where the number of messages and progressions is higher. Indeed, in the centralised approach, every monitor records each event and sends a message for each one, meaning that each monitor applies many progressions. This is not the case in the decentralised approach, where only a few messages are seen by each monitor. It is also shown that the time required to pass observations along all monitors in the centralised approach is much longer. Indeed, in order to apply the progression function on a local event it has observed, a monitor has to wait for it to circulate along all the other monitors; with 10 monitors and an average communication delay of 20 time units, it means that the monitor has to wait on average for 200 time units.

Of course, the trade-off can be seen in the size of the sent messages. This is because the centralised approach sends timed events, while the decentralised approach sends timed expressions. Those expressions are written with timed constraints, that tend to pile up after several progressions, and that increase the size of the expression. However, we can imagine two ways to alleviate this issue. The first would be to improve the simplification of expressions to reduce the size of expressions, possibly by simplifying some global time constraints or some conjunctions.

Another idea would be to send either a history of the timed events or an expression depending on which one is smaller. Indeed, each monitor could compute the expression if it received some history of the timed events. This approach is hard to implement, as it would mean that monitors should remember what other monitors have seen so far.

Another data that we did not show in the table is the impact of the communication delay relative to the delay between consecutive events. In fact, with higher communication delays, the simulated centralised approach has a decision time that increases much more than the decentralised approach. This is because the centralised method requires the message to travel along the entirety of the ring before taking any decision, which slows down the decision time proportionality to both the number of monitors and the delay of communication. This is less of a problem for the decentralised approach, which can decide by exchanging fewer messages between a few monitors. In our tests, we have even seen that in many cases, the decentralised approach decided with fewer messages as we increased the communication delay. This happens because when a message is received after being in transit for a long time, the receiving monitor has had enough time to build a long history that would violate the property. So we can deduce that when the communication delay is high, the decentralised approach should be the main option.

6 Related Work

As this paper introduces decentralised runtime verification for timed properties described by timed regular expressions [3], the related approaches consist of those for monitoring timed properties and those decentralising the monitoring process. In monitoring timed specifications, research efforts have mainly focused on synthesising decision procedures (monitors) for timed properties. Bauer et al. [5] introduce a variant of Timed Linear-time Temporal Logic (TLTL), a timed extension of Linear-time Temporal Logic, with a semantics tailored for runtime verification defined on finite traces. They additionally synthesise finite-trace monitors from TLTL formulas. Metric Temporal Logic (MTL) is another extension of LTL, with dense time. Nickovic et al. [16] translate MTL formulas into timed automata and Thati et al. [18] use a tailored progression function to evaluate formulas at runtime. More recently, Grez et al. [15] consider the monitoring problem for timed automata by introducing a data structure that allows the monitoring of a non-deterministic 1-clock automaton. Pinisetty et al. [17] introduce a predictive setting for runtime verification of timed properties leveraging reachability analysis to anticipate the detection of verdicts. Specific to timed regular expressions [3], Montre [21] is a tool monitoring using timed pattern matching. The mentioned approaches consider that the monitored system is centralised, and the decision procedure is fed with a unique trace containing complete observations.

Decentralised runtime verification has been introduced in [6], see [10] for a recent overview. Approaches in decentralised runtime verification take as input Linear-time Temporal Logic formulas such as [6, 7, 14] or finite-state automata such as in [11, 9]. All these approaches monitor specifications of discrete time, which is much simpler and does not account for the physical time that impacts the evaluation of the specification as well as the moment at which monitors perform their evaluation.

Finally, we note that decentralised runtime verification resembles diagnosis [22, 23], which tries to detect the occurrence of a fault after a finite number of discrete steps and the component responsible for the fault. Our approach differs from diagnosis, as we assume that monitors' (combined) local information suffices to detect violations. However, in diagnosis, the model of the system is taken as input, and a central decision-making point is assumed. Similar to diagnosis, there is decentralised observability [19, 20] that combines the state of local observers with locally or globally bounded or unbounded memory to get a truthful verdict. While [19] requires a central decision-making point, the recent approach in [20] introduces the "at least one can tell" condition, which characterises when local agents can evaluate the global behaviour. While this approach, like ours, does not require a central observation point, it does not allow monitoring of the membership to an arbitrary timed regular expression.

7 Conclusion and Perspectives

We have introduced centralised and decentralised progression for timed regular expressions and have shown how it can be used to implement several algorithms to achieve decentralised monitoring of timed properties described by timed regular expressions. While several approaches exist for decentralised monitoring of untimed properties and centralised monitoring of timed properties, this is the first realisation of decentralised monitoring of timed properties. We have implemented the decentralised monitoring algorithms and evaluated their runtime behaviour costs in metrics relevant to decentralised monitoring.

These results give insights and research directions to improve these methods, such as using better simplification rules or having a better choice of targets when the expression is passed between monitors. Alternatively, decentralised monitoring approaches can be designed for properties described by timed automata, as they are well adopted in the community. Since there is an equivalence between timed regular expressions and timed automata, one could simply implement the transformation from automata to expressions. However, we believe that finding an analogue of progression for timed automata seems promising, as it could outperform the methods shown in this paper using knowledge of the states and transitions, as in [11] with finite-state automata. Finally, another perspective is to define progression for Metric Temporal Logic (MTL). Indeed, if we find a progression that satisfies the same properties as those shown in this paper, the same algorithms can be applied.

References

1 R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *[1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 414–425, 1990. doi:10.1109/LICS.1990.113766.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

3 Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002. doi:10.1145/506147.506151.

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

5 Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, September 2011. doi:10.1145/2000799.2000800.

6 Andreas Klaus Bauer and Yliès Falcone. Decentralised LTL monitoring. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, volume 7436 of *Lecture Notes in Computer Science*, pages 85–100. Springer, 2012. doi:10.1007/978-3-642-32759-9_10.

7 Christian Colombo and Yliès Falcone. Organising LTL monitors over distributed systems with a global clock. *Formal Methods Syst. Des.*, 49(1-2):109–158, 2016. doi:10.1007/s10703-016-0251-x.

8 Daniel de Leng and Fredrik Heintz. Approximate stream reasoning with metric temporal logic under uncertainty. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33012760.

9 Antoine El-Hokayem and Yliès Falcone. On the monitoring of decentralized specifications: Semantics, properties, analysis, and simulation. *ACM Trans. Softw. Eng. Methodol.*, 29(1):1:1–1:57, 2020. doi:10.1145/3355181.

10 Yliès Falcone. On decentralized monitoring. In Ayoub Nouri, Weimin Wu, Kamel Barkaoui, and ZhiWu Li, editors, *Verification and Evaluation of Computer and Communication Systems - 15th International Conference, VECoS 2021, Virtual Event, November 22-23, 2021, Revised Selected Papers*, volume 13187 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2021. doi:10.1007/978-3-030-98850-0_1.

11 Yliès Falcone, Tom Cornebize, and Jean-Claude Fernandez. Efficient and generalized decentralized monitoring of regular languages. In Erika Ábrahám and Catuscia Palamidessi, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 34th IFIP WG 6.1 International Conference, FORTE 2014, Held as Part of the 9th International Federated Conference on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014. Proceedings*, volume 8461 of *Lecture Notes in Computer Science*, pages 66–83. Springer, 2014. doi:10.1007/978-3-662-43613-4_5.

12 Yliès Falcone, Klaus Havelund, and Giles Reger. A tutorial on runtime verification. In Manfred Broy, Doron A. Peled, and Georg Kalus, editors, *Engineering Dependable Software Systems*, volume 34 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 141–175. IOS Press, 2013. doi:10.3233/978-1-61499-207-3-141.

13 Yliès Falcone, Srdan Krstic, Giles Reger, and Dmitriy Traytel. A taxonomy for classifying runtime verification tools. *Int. J. Softw. Tools Technol. Transf.*, 23(2):255–284, 2021. doi:10.1007/s10009-021-00609-z.

14 Florian Gallay and Yliès Falcone. Decentralized LTL enforcement. In Pierre Ganty and Davide Bresolin, editors, *Proceedings 12th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2021, Padua, Italy, 20-22 September 2021*, volume 346 of *EPTCS*, pages 135–151, 2021. doi:10.4204/EPTCS.346.9.

15 Alejandro Grez, Filip Mazowiecki, Michal Pilipczuk, Gabriele Puppis, and Cristian Riveros. The monitoring problem for timed automata. *CoRR*, abs/2002.07049, 2020. arXiv:2002.07049.

16 Dejan Nickovic and Oded Maler. AMT: a property-based monitoring tool for analog systems. In Jean-François Raskin and P. S. Thiagarajan, editors, *Proceedings of the 5th International Conference on Formal modeling and analysis of timed systems (FORMATS 2007)*, volume 4763 of *Lecture Notes in Computer Science*, pages 304–319. Springer-Verlag, 2007.

17 Srinivas Pinisetty, Thierry Jéron, Stavros Tripakis, Yliès Falcone, Hervé Marchand, and Viorel Preoteasa. Predictive runtime verification of timed properties. *J. Syst. Softw.*, 132:353–365, 2017. doi:10.1016/j.jss.2017.06.060.

18 Prasanna Thati and Grigore Rosu. Monitoring algorithms for metric temporal logic specifications. *Electronic Notes in Theoretical Computer Science*, 113:145–162, 2005. doi:10.1016/j.entcs.2004.01.029.

19 Stavros Tripakis. Decentralized observation problems. In *44th IEEE IEEE Conference on Decision and Control and 8th European Control Conference Control, CDC/ECC 2005, Seville, Spain, 12-15 December, 2005*, pages 6–11. IEEE, 2005. doi:10.1109/CDC.2005.1582122.

20 Stavros Tripakis and Karen Rudie. Decentralized observation of discrete-event systems: At least one can tell. *IEEE Control. Syst. Lett.*, 6:1652–1657, 2022. doi:10.1109/LCSYS.2021.3130887.

21 Dogan Ulus. Montre: A tool for monitoring timed regular expressions. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 329–335. Springer, 2017. doi:10.1007/978-3-319-63387-9_16.

22 Yin Wang, Tae-Sic Yoo, and Stéphane Lafortune. Diagnosis of discrete event systems using decentralized architectures. *Discret. Event Dyn. Syst.*, 17(2):233–263, 2007. doi:10.1007/s10626-006-0006-8.

23 Xiang Yin and Stéphane Lafortune. Codiagnosability and coobservability under dynamic observations: Transformation and verification. *Autom.*, 61:241–252, 2015. doi:10.1016/j.automatica.2015.08.023.

A Definition of Φ

- $\Phi(\underline{a})(\Sigma') = \underline{a}$ if $a \notin \Sigma'$
- $\Phi(\langle \underline{a} \rangle_I)(\Sigma') = \langle \underline{a} \rangle_I$ if $a \notin \Sigma'$
- $\Phi(L^{\circledast})(\Sigma') = \Phi(L)(\Sigma')^{\circledast}$
- $\Phi(\lceil L \rceil^I)(\Sigma') = \lceil \Phi(L)(\Sigma') \rceil^I$
- $\Phi(\theta(L))(\Sigma') = \theta(\Phi(L)(\theta^{-1}(\Sigma')))$
- $\Phi(L_1 \vee L_2)(\Sigma') = \Phi(L_1)(\Sigma') \vee \Phi(L_2)(\Sigma')$
- $\Phi(L_1 \wedge L_2)(\Sigma') = \Phi(L_1)(\Sigma') \wedge \Phi(L_2)(\Sigma')$
- $\Phi(L_1 \circ L_2)(\Sigma') = \Phi(L_1)(\Sigma') \circ \Phi(L_2)(\Sigma')$
- $\Phi(L_1 \cdot L_2)(\Sigma') = \Phi(L_1)(\Sigma') \cdot \Phi(L_2)(\Sigma')$
- $\Phi(\underline{a})(\Sigma') = \emptyset$ if $a \in \Sigma'$
- $\Phi(\langle \underline{a} \rangle_I)(\Sigma') = \emptyset$ if $a \in \Sigma'$
- $\Phi(L^{\star})(\Sigma') = \Phi(L)(\Sigma')^{\star}$
- $\Phi(\lfloor L \rfloor_I)(\Sigma') = \lfloor \Phi(L)(\Sigma') \rfloor_I$

B Definition of Δ_i

For I, I' two intervals, we denote $I \lhd I' = \{x \in I \mid \forall t \in I', x < t\}$.
And $I \rhd I' = \{x \in I \mid \forall t \in I', x > t\}$ such that $I \lhd I'$, $I \rhd I'$ and $I \cap I'$ are always a partition of I. We define Δ_i as follows:

- $\Delta_i(\underline{a}, I)(\alpha, t) = \{(I, \epsilon)\}$ if $\alpha = a$.
- $\Delta_i(\underline{a}, I)(\alpha, t) = \{(I, \emptyset)\}$ otherwise.
- $\Delta_i(\langle \underline{a} \rangle_{I'}, I)(\alpha, t) = \{(I \rhd I', \emptyset); (I \lhd I', \emptyset); (t - I' \cap I, \epsilon)\}$ if $\alpha = a$.
- $\Delta_i(\langle \underline{a} \rangle_{I'}, I)(\alpha, t) = \{(\emptyset, I)\}$ otherwise.
- $\Delta_i(L_1 \vee L_2, I)(\alpha, t) =$
  $\{(I_1 \cap I_2, L_1' \vee L_2') \mid (I_1, L_1') \in \Delta_i(L_1, I)(\alpha, t), (I_2, L_2') \in \Delta_i(L_2, I)(\alpha, t)\}$
- $\Delta_i(L_1 \wedge L_2, I)(\alpha, t) =$
  $\{(I_1 \cap I_2, L_1' \wedge L_2') \mid (I_1, L_1') \in \Delta_i(L_1, I)(\alpha, t), (I_2, L_2') \in \Delta_i(L_2, I)(\alpha, t)\}$
- $\Delta_i(L_1 \circ L_2, I)(\alpha, t) = \{(I_1 \cap I_2, L_1' \circ \lfloor L_2 \rfloor_{[t;\infty[} \vee \lceil \Phi(L_1)(\Sigma_i) \rceil^{[0;t]} \circ L'2) \mid (I_1, L_1') \in \Delta_i(L_1, I)(\alpha, t), (I_2, L_2') \in \Delta_i(L_2, I)(\alpha, t)\}$
- $\Delta_i(L_1^{\circledast}, I)(\alpha, t) =$
  $\{(I_1, \lceil Past(L_1)(\Sigma_i)^{\circledast} \rceil^{[0;t]} \circ L'1 \circ \lfloor L_1^{\circledast} \rfloor_{[t;\infty[}) \mid (I_1, L_1') \in \Delta_i(L_1, I)(\alpha, t)\}$
- $\Delta_i(L_1 \cdot L_2, I)(\alpha, t) =$
  $\{(I_1, (\bigvee_{(I_2, L_2') \in \Delta_i(L_2, I)(\alpha, t)} \lceil \Phi(L_1)(\Sigma_i)) \cdot L_2' \rceil^{I_2} \vee L_1' \circ \lfloor L_2 \rfloor_{[t;\infty[}) \mid (I_1, L_1') \in \Delta_i(L_1, I)(\alpha, t)\}$
- $\Delta_i(L_1^{\star}, I)(\alpha, t) = \{(I_1, \bigvee_{(I_1, L_1') \in \Delta_i(L_1, I)(\alpha, t)} \lceil \Phi(L_1)(\Sigma_i)^{\star} \rceil^{I_1} \cdot L_1' \cdot \lfloor L_1^{\star} \rfloor_{[t;\infty[})\}$

  This function satisfies the following properties.

$$\forall t, I, L, \forall (I', L') \in \Delta_i(L, I)(\alpha, t), \forall t' \in I', L' = \mathrm{Pr}_i(L \downarrow_{t'})(\alpha, t)$$
$$\forall t, I, L, \forall (I_1, L_1), (I_2, L_2) \in \Delta_i(L, I)(\alpha, t), (I_1, L_1) \neq (I_2, L_2) \Leftrightarrow I_1 \cap I_2 = \emptyset$$
$$\forall t, I, L, \bigcup_{(I', L') \in \Delta_i(L, I)(\alpha, t)} I' = I$$

C Proof that Pr_i is a decentralised progression function

Proof. Let us prove this by induction. We will consider the cases that are not immediately apparent :

- Let $L = L_1 \circ L_2$. Let us prove the double inclusion.
  1. Assume $u \in [\![\mathrm{Pr}_i(L)(\alpha, t)]\!]_{\mathrm{tr}}$. There are then two possible cases.
     - First case: $u \in [\![\mathrm{Pr}_i(L_1)(\alpha, t) \circ \lfloor L_2 \rfloor_{[t;\infty[}]\!]_{\mathrm{tr}}$ which means that $u = u_1 \cdot u_2$ with $u_1 \in [\![\mathrm{Pr}_i(L_1)(\alpha, t)]\!]_{\mathrm{tr}}$ and $u_2 \in [\![L_2]\!]_{\mathrm{tr}}$ and $\tau_{\mathrm{first}}(u_2) \geq t$. Using our induction hypothesis $u_1 = v_1 \cdot v_2$ with $v_1 \cdot (\alpha, t) \cdot v_2 \in L_1$ and $v_1 \in \mathcal{R}(\Sigma \setminus \Sigma_i)$. Therefore, we proved $v_1 \cdot (\alpha, t) \cdot v_2 \cdot u_2 \in [\![L]\!]_{\mathrm{tr}}$.
     - Second case $u \in [\![\lceil \Phi(L_1)(\Sigma_i) \rceil^{[0;t]} \circ \mathrm{Pr}_i(L_2)(\alpha, t)]\!]_{\mathrm{tr}}$, then it means that $u = u_1 \cdot u_2$ with $u_1 \in [\![\Phi(L_1)(\Sigma_i)]\!]_{\mathrm{tr}}$ and $u_2 \in [\![\mathrm{Pr}_i(L_2)(\alpha, t)]\!]_{\mathrm{tr}}$. This means that $u_1 \in [\![L_1]\!]_{\mathrm{tr}} \cap \mathcal{R}(\Sigma \setminus \Sigma_i)$. Using our induction hypothesis, we have $u_2 = v_1 \cdot v_2$ with $v_1 \cdot (\alpha, t) \cdot v_2 \in [\![L_2]\!]_{\mathrm{tr}}$ and $v_1 \in \mathcal{R}(\Sigma \setminus \Sigma_i)$. We can then deduce $u_1 \cdot v_1 \cdot (\alpha, t) \cdot v_2 \in [\![L]\!]_{\mathrm{tr}}$ and $(u_1 \cdot v_1) \in \mathcal{R}(\Sigma \setminus \Sigma_i)$

     This means that in both cases $u \in \{u' \cdot v' \mid u' \cdot (\alpha, t) \cdot v' \in [\![L]\!]_{\mathrm{tr}}$ and $u' \in \mathcal{R}(\Sigma \setminus \Sigma_i)\}$
  2. Let us prove the other side of the inclusion. Let us assume u and v such that $u \cdot (\alpha,) \cdot v \in [\![L]\!]_{\mathrm{tr}} \wedge u \in \mathcal{R}(\Sigma \setminus \Sigma_i)\}$. This means that $u \cdot (\alpha, t) \cdot v = u' \cdot v'$ with $u' \in [\![L_1]\!]_{\mathrm{tr}}$ and $v' \in [\![L_2]\!]_{\mathrm{tr}}$. We can then deduce that we either have $u' = u \cdot (\alpha, t) \cdot v_1$ or have $v' = u_1 \cdot (\alpha, t) \cdot v$. In other words, we have either $u \cdot v_1 \in [\![\mathrm{Pr}_i(L_1)(\alpha, t)]\!]_{\mathrm{tr}}$ with $v' \in [\![\lfloor L_2 \rfloor_{[t,\infty[}]\!]_{\mathrm{tr}}$ or $u_1 \cdot v \in [\![\mathrm{Pr}_i(L_2)(\alpha, t)]\!]_{\mathrm{tr}}$ with $u' \in L_1$ and in the last case, every element before (α, t) cannot contain elements of Σ_i, therefore it must be true for u' which means $u' \in [\![\lceil \Phi_i(L_1)(\Sigma_i) \rceil^{[0;t]}]\!]_{\mathrm{tr}}$. This means that in both cases $u \cdot v \in [\![\mathrm{Pr}_i(L)(\alpha, t)]\!]_{\mathrm{tr}}$

- If $L = L_1^{\circledast}$.
  1. Let us assume $u \in [\![\mathrm{Pr}_i(L)(\alpha,t)]\!]_{\mathrm{tr}}$. This means that $u = u_1 \cdot u_2 \cdot u_3$ with $u_1 \in [\![\lceil \Phi(L_1)(\Sigma_i)^{\circledast} \rceil^{[0;t]}]\!]_{\mathrm{tr}}$, $u_3 \in [\![\lfloor L_1^{\circledast} \rfloor_{[t;\infty[}]\!]_{\mathrm{tr}}$ and $u_2 \in [\![\mathrm{Pr}_1(L_1)(\alpha,t)]\!]_{\mathrm{tr}}$. Therefore, we know that $u_1 \in [\![L_1^{\circledast}]\!]_{\mathrm{tr}}$, $u_1 \in \mathcal{R}(\Sigma \setminus \Sigma_i)$, and $u_2 = v_1 \cdot v_2$ with $v_1 \cdot (\alpha,t) \cdot v_2 \in [\![L_1]\!]_{\mathrm{tr}}$. Hence $u_1 \cdot v_1 \cdot (\alpha,t) \cdot v_2 \cdot v_3 \in [\![L_1^{\circledast}]\!]_{\mathrm{tr}}$ with $u_1 \cdot v_1 \in \mathcal{R}(\Sigma \setminus \Sigma_i)$
  2. Now to prove the other inclusion. Let us assume $u \cdot v$ such that $w = u \cdot (\alpha,t) \cdot v \in [\![L]\!]_{\mathrm{tr}}$. Since w is not empty, it is equal to $w_1 \cdot w_2 \cdot .... \cdot w_n$, with $w_i \in [\![L_1]\!]_{\mathrm{tr}}$. Because we know that w contains (α,t), we can denote by k the index of the w_i containing this event. In other words, we have $w_k = u' \cdot (\alpha,t) \cdot v' \in [\![L_1]\!]_{\mathrm{tr}}$, which means that $u' \cdot v' \in [\![\mathit{Prog}_i(L_1)(\alpha,t)]\!]_{\mathrm{tr}}$. This proves that $u \cdot v = w_0 \cdot ... \cdot w_{k-1} \cdot u' \cdot v' \cdot w_{k+1} \cdot ...w_n \in [\![\lceil \Phi(L_1)(\Sigma_i)^{\circledast} \rceil^{[0;t]} \circ \mathrm{Pr}_i(L_1)(\alpha,t) \circ \lfloor L_1^{\circledast} \rfloor_{[t;\infty[}]\!]_{\mathrm{tr}}$
- Let $L = L_1 \cdot L_2$. Let us prove the double inclusion.
  1. Assume $u \in [\![\mathrm{Pr}_i(L)(\alpha,t)]\!]_{\mathrm{tr}}$. There are then two possible cases.
     - First case: $u \in [\![\mathrm{Pr}_i(L_1)(\alpha,t) \circ \lfloor L_2 \rfloor_{[t;\infty[}]\!]_{\mathrm{tr}}$ then means that $u = u_1 \cdot u_2$ with $u_1 \in [\![\mathrm{Pr}_i(L_1)(\alpha,t)]\!]_{\mathrm{tr}}$ and $u_2 \in [\![L_2 \downarrow_{\tau_{\mathrm{last}}(u_1)}]\!]_{\mathrm{tr}}$ and $\tau_{\mathrm{first}}(u_2) \geq t$. Using our induction hypothesis $u_1 = v_1 \cdot v_2$ with $v_1 \cdot (\alpha,t) \cdot v_2 \in L_1$ and $v_1 \in \mathcal{R}(\Sigma \setminus \Sigma_i)$. This allows us to conclude that $v_1 \cdot (\alpha,t) \cdot v_2 \cdot u_2 \in [\![L]\!]_{\mathrm{tr}}$.
     - Second case: There is $(I, L') \in \Delta_i(L_2, \mathbb{R}^+)(\alpha,t)$ such that $u \in [\![\lceil \Phi(L_1)(\Sigma_i) \rceil^I \cdot L'(\alpha,t)]\!]_{\mathrm{tr}}$. It means that $u = u_1 \cdot u_2$ with $u_1 \in [\![\Phi(L_1)(\Sigma_i)]\!]_{\mathrm{tr}}$ and $u_2 \in [\![L'(\alpha,t)]\!]_{\mathrm{tr}}$. This means that $u_1 \in [\![L_1]\!]_{\mathrm{tr}} \cap \mathcal{R}(\Sigma \setminus \Sigma_i)$. Taking into account the first property of Δ_i, we can also see that $u_2 \in [\![\mathrm{Pr}_i(L_2 \downarrow_{\tau_{\mathrm{last}}(u_1)})(\alpha,t)]\!]_{\mathrm{tr}}$. Using our induction hypothesis, we have $u_2 = v_1 \cdot v_2$ with $v_1 \cdot (\alpha,t) \cdot v_2 \in [\![L_2 \downarrow_{\tau_{\mathrm{last}}(u_1)}]\!]_{\mathrm{tr}}$ and $v_1 \in \mathcal{R}(\Sigma \setminus \Sigma_i)$. Therefore, we have $u_1 \cdot v_1 \cdot (\alpha,t) \cdot v_2 \in [\![L]\!]_{\mathrm{tr}}$ and $(u_1 \cdot v_1) \in \mathcal{R}(\Sigma \setminus \Sigma_i)$
  2. Let us prove the other side of the inclusion. Assume u and v such that $u \cdot (\alpha,) \cdot v \in [\![L]\!]_{\mathrm{tr}} \wedge u \in \mathcal{R}(\Sigma \setminus \Sigma_i)\}$. This means that $u \cdot (\alpha,t) \cdot v = u' \cdot v'$ with $u' \in [\![L_1]\!]_{\mathrm{tr}}$ and $v' \in [\![L_2 \downarrow_{\tau_{\mathrm{last}}(u')}]\!]_{\mathrm{tr}}$. We can deduce that we have $u' = u \cdot (\alpha,t) \cdot v_1$ or we have $v' = u_1 \cdot (\alpha,t) \cdot v$.
     In other words, we have $u \cdot v_1 \in [\![\mathrm{Pr}_i(L_1)(\alpha,t)]\!]_{\mathrm{tr}}$ with $v' \in [\![\lfloor L_2 \rfloor_{[t,\infty[}]\!]_{\mathrm{tr}}$ or we have $u_1 \cdot v \in [\![\mathrm{Pr}_i(L_2 \downarrow_{\tau_{\mathrm{last}}(u')})(\alpha,t)]\!]_{\mathrm{tr}}$ with $u' \in L_1$. In the first case, we have $u \cdot v \in [\![\mathrm{Pr}_i(L)(\alpha,t)]\!]_{\mathrm{tr}}$.
     Let us look at the other case. Every element before (α,t) cannot contain elements of Σ_i, therefore it must be true of u', which means $u' \in [\![\Phi_i(L_1)(\Sigma_i)]\!]_{\mathrm{tr}}$. Using the second and third properties of Δ_i, we can see that there is one and only one $(I, L') \in \Delta_i(L_2, \mathbb{R}^+)(\alpha,t)$ such that $\tau_{\mathrm{last}}(u') \in I$. That means we have $u' \in [\![\lceil \Phi_i(L_1)(\Sigma_i) \rceil^I]\!]_{\mathrm{tr}}$ and $u_1 \cdot v \in [\![L']\!]_{\mathrm{tr}}$. We then proved in both cases that $u \cdot v \in [\![\mathrm{Pr}_i(L)(\alpha,t)]\!]_{\mathrm{tr}}$.
- If $L = L_1^{\star}$.
  1. Assume $u \in [\![\mathrm{Pr}_i(L)(\alpha,t)]\!]_{\mathrm{tr}}$. This means that there is $(I, L') \in \Delta_i(L_1, \mathbb{R}^+)(\alpha,t)$ such that $u = u_1 \cdot u_2 \cdot u_3$ with $u_1 \in [\![\lceil \Phi(L_1)(\Sigma_i)^{\star} \rceil^I]\!]_{\mathrm{tr}}$ and $u_3 \in [\![\lfloor L_1^{\star} \rfloor_{[t;\infty[} \downarrow_{\tau_{\mathrm{last}}(u_2)}]\!]_{\mathrm{tr}}$ as well as $u_2 \in [\![L']\!]_{\mathrm{tr}}$. Using the first property of Δ_i, this means $u_2 \in [\![\mathrm{Pr}_i(L_1 \downarrow_{\tau_{\mathrm{last}}(u_1)})(\alpha,t)]\!]_{\mathrm{tr}}$. In other words, we know that $u_1 \in [\![L_1^{\star}]\!]_{\mathrm{tr}}$, $u_1 \in \mathcal{R}(\Sigma \setminus \Sigma_i)$ and $u_2 = v_1 \cdot v_2$ with $v_1 \cdot (\alpha,t) \cdot v_2 \in [\![L_1 \downarrow_{\tau_{\mathrm{last}}(u_1)}]\!]_{\mathrm{tr}}$. We can then deduce that $u_1 \cdot v_1 \cdot (\alpha,t) \cdot v_2 \cdot v_3 \in [\![L_1^{\star}]\!]_{\mathrm{tr}}$ with $u_1 \cdot v_1 \in \mathcal{R}(\Sigma \setminus \Sigma_i)$
  2. Now to prove the other inclusion. Assume $u \cdot v$ such that $w = u \cdot (\alpha,t) \cdot v \in [\![L]\!]_{\mathrm{tr}}$. Since w is not empty, $w = w_1 \cdot w_2 \cdot .... \cdot w_n$, with $w_i \in [\![L_1 \downarrow_{\tau_{\mathrm{last}}(w_{i-1})}]\!]_{\mathrm{tr}}$. Because we know that w contains (α,t), we can denote by k the index of w_i that contains this event. In other words, we have $w_k = u' \cdot (\alpha,t) \cdot v' \in [\![L_1 \downarrow_{\tau_{\mathrm{last}}(w_{k-1})}]\!]_{\mathrm{tr}}$, which means that $u' \cdot v' \in [\![\mathit{Prog}_i(L_1 \downarrow_{\tau_{\mathrm{last}}(w_{i-1})})(\alpha,t)]\!]_{\mathrm{tr}}$. Using the second and third properties of Δ_i we deduce that there is one and only one $(I, L') \in \Delta_i(L_1, \mathbb{R}^+)(\alpha,t)$ such that $\tau_{\mathrm{last}}(w_{k-1}) \in I$ and in that case we have $w_k \in [\![L']\!]_{\mathrm{tr}}$. This shows that $u \cdot v = w_0 \cdot ... \cdot w_{k-1} \cdot u' \cdot v' \cdot w_{k+1} \cdot ...w_n \in [\![\lceil \Phi(L_1)(\Sigma_i)^{\star} \rceil^I \cdot L' \cdot \lfloor L_1^{\star} \rfloor_{[t;\infty[}]\!]_{\mathrm{tr}}$. ◀

Logical Forms of Chronicles

Thomas Guyet
Inria, Lyon, France

Nicolas Markey
CNRS, IRISA, Rennes, France

Abstract

A chronicle is a temporal model introduced by Dousson et al. for situation recognition. In short, a chronicle consists of a set of events and a set of real-valued temporal constraints on the delays between pairs of events. This work investigates the relationship between chronicles and classical temporal-model formalisms, namely TPTL and MTL. More specifically, we answer the following question: *is it possible to find an equivalent formula in such formalisms for any chronicle?* This question arises from the observation that a single chronicle captures complex temporal behaviours, without imposing a particular order of the events in time.

For our purpose, we introduce the subclass of *linear chronicles*, which set the order of occurrence of the events to be recognized in a temporal sequence. Our first result is that any chronicle can be expressed as a disjunction of linear chronicles. Our second result is that any linear chronicle has an equivalent TPTL formula. Using existing expressiveness results between TPTL and MTL, we show that some chronicles have no equivalent in MTL. This confirms that the model of chronicle has interesting properties for situation recognition.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Timed and hybrid models

Keywords and phrases temporal logics, temporal models

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.7

1 Introduction

The problem of representing complex behaviours has a lot of applications to monitor dynamic systems based on their functioning traces. Detecting complex behaviours in these traces is useful to identify a faulty state of a system or to label traces with higher-level events.

One possible application of this problem is the improvement of health-care systems. A major issue is to evaluate the incidence of a disease or a treatment in a real-world population. This can be done through the analysis of Electronic Health Records (EHR). EHR are health-administrative databases that gather all delivered cares at hospital. This gives a longitudinal view – or a temporal trace – of patients' treatments and their responses. The difficulty with EHR data is that they do not necessarily hold the desired medical information (such as the patient's medical status). This requires to infer the medical status of a patient from observable information.

A practical solution is to define a proxy of a status by a complex situation to match with patient's care pathways. Such proxy is called a *computable-phenotype* or a *phenotype* [8]. The more expressive the phenotype language, the more accurate the evaluation of incidence.

Then, we advocate for the primary importance of the temporal dimension to accurately represent phenotypes. Indeed, systems such as GLARE [25] emphasize management of temporal knowledge to formalize clinical guidelines, including comprehensive treatment of temporal constraints. But, contrary to clinical guidelines that uses complex reasoning on few care pathways (e.g., guideline compliance [5]), our objective is to find the occurrences of a specified complex situation in a large set of patients (i.e., millions of patients). This requires computational efficiency. To sum up, we need a temporal query language to specify phenotypes, i.e. complex situations, and that can efficiently be matched in large collections of temporal sequences.

Behind this problem lies the classical tradeoff of computer science: the expressiveness of temporal queries *vs.* their computational efficiency. Researchers aim to find computational models that achieve the best compromise between these two opposite objectives. This compromise also depends on the context of usage, and it is not unique. Thus it has been addressed in a wide range of research fields: knowledge reasoning, temporal logic, model checking, temporal databases, complex-event processing, ... Representing complex situations is studied from the origin of logic-based artificial intelligence to represent and reason about temporal facts. Indeed, the representation of actions and formalisms/logics to reason with them are very central to AI. Many knowledge-reasoning formalisms have been proposed: temporal logic of action [19], situation calculus [20], event calculus [21], ... They are very expressive but knowledge-reasoning tools are not efficient enough for being practically applied on massive data.

As the problem of specifying situations is of particular interested for monitoring dynamic or reactive systems, dedicated formalisms attracted a lot of interest at the crossroad of model checking and temporal logic. We can mention Linear Temporal Logic (LTL) [23] when dealing with discrete time, or Metric Temporal Logic (MTL) [18] for real-valued time. Their success comes from their expressiveness and their clear semantics. Many temporal systems are based on these logics. This is especially the case of temporal databases, which extend the principles of relational databases to timed records. A query language such as TSQL2 [4] combines relational operators and LTL operators. DatalogMTL [27] combines datalog language and MTL operators and has been used to query log data [7]. Finally, temporal models have also been developed in the field of complex-event processing and stream reasoning to address the specific questions of recognition efficiency. Kervac and Piel [17] survey such techniques including ETALIS language [2] or chronicles [13]. These tools provide expressive temporal models suitable to efficiently recognize temporal patterns in real-valued timed sequences.

In the present article, we focus on the notion of chronicle. A chronicle is a temporal model introduced by Dousson et al. [13] for situation recognition. In short, a chronicle consists of a set of events and a set of real-valued temporal constraints on the delays between pairs of events. It describes situations that can be recognized within a temporal sequence, i.e., a sequence of timestamped events (with no durations). Chronicles are close to, but not equivalent to, temporal constraint networks [12]. They have the following interesting characteristics:

- they are user-friendly. Their graphical representation makes them attractive for a wide range of applications where temporal patterns have to be analyzed by domain experts;
- they are used with computational efficiency in a wide range of tasks: planning, diagnosis, system modelling, and also data mining. In 1999, Dousson et al. [14] proposed an algorithm to discover chronicles from a set of temporal sequences. In this latter context, chronicles form a very expressive class of models, and many works have been proposed in the field of pattern mining to extract frequent or discriminant chronicles [10, 11]. Chronicle recognition algorithms are also computationally efficient [13].
- they are *a priori* expressive. Despite their apparent simplicity, a single chronicle model captures a wide range of practical temporal situations. For instance, they do not imposes a strict order of appearance on the events. A chronicle involving a and b event types can match sequence that contains a and b whatever their order of appearance.

These characteristics make chronicles a first-class citizen to represent complex situations to match in temporal sequences.

A natural question then arises from the remark about the expressiveness of chronicles: what is the relationship between chronicles and classical temporal-model formalisms? The intuition that chronicles are expressive is based on practical uses but, to the best of our knowledge, their expressiveness has not been compared to that of alternative temporal models. A situation can also be seen as a property of a reactive system. Recognizing a situation is similar to matching the property of a reactive system on a single trace [26]. Temporal logics such as Linear Temporal Logic (LTL) [23], Metric Temporal Logic (MTL) [18] or Timed Propositional Temporal Logic (TPTL) [1], have been widely studied to specify temporal properties of reactive systems, and more specifically for the pattern-recognition task. Contrary to LTL or CTL (Computational Tree Logic [9]), which deal with sequences of events in time, MTL and TPTL deal with events having real-valued timestamps. Surprisingly, there is no existing result stating the relationship between temporal logics and chronicles. It is worth noting that chronicles are already equipped with efficient recognition algorithms and that the purpose of the comparisons is to evaluate the expressive power of chronicles but not to gain efficiency with the use TPTL or MTL tools.

This article compares the expressiveness of chronicles with two temporal logics, $\mathsf{TPTL}_{\Diamond}$ [1, 6] and MTL. The main results are that chronicles can be expressed with $\mathsf{TPTL}_{\Diamond}$ formulas (in the pointwise semantics [6]), but in general they cannot be expressed with MTL formulas. To obtain this result, we first introduce a notion of *linear chronicles*, and show that any chronicle is equivalent to a finite conjunction of linear chronicles. We then propose a transformation of linear chronicles into $\mathsf{TPTL}_{\Diamond}$. The impossibility to express chronicles in MTL is then obtained by adapting a result from Bouyer et al. [6].

2 Chronicles

In this section, we introduce the basic notions and notations of chronicles. We start by introducing the definitions of temporal sequences and chronicles, and give their semantics through the definition of an *occurrence* of a chronicle in a temporal sequence. Then, we introduce the subclass of *linear* chronicles. This subclass highlights the specificity of the chronicle model, which allows to leave the order of occurrence of some events unspecified. A first result is that any chronicle has an equivalent disjunctive set of linear chronicle. This result is our first step further toward a translation into $\mathsf{TPTL}_{\Diamond}$.

2.1 Syntax and semantics

Given a finite alphabet Σ of event types, we first introduce the notion of timed sequence over Σ, and then we define chronicles. We let $\leq_\Sigma$ denote a total order on the elements of Σ. In the following, event types are capital letters and $\leq_\Sigma$ is the alphabetic order. For $m \in \mathbb{N}$, we write $[m]$ for the set $\{i \in \mathbb{N} \mid 1 \leq i \leq m\}$.

▶ **Definition 1** (Timed sequence). *A* ***timed sequence*** *of length n over a finite alphabet Σ is a finite sequence $\rho = \langle(\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)\rangle$ in $(\Sigma \times \mathbb{R}_{\geq 0})^n$ where for all $1 \leq i < n$, it holds $\tau_i \leq \tau_{i+1}$.*

Using the notations of e.g. Ouaknine *et al.* [22] or Bouyer *et al* [6], a timed sequence $\langle(\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)\rangle$ corresponds to the timed word $\langle \sigma = \sigma_1 \cdots \sigma_n,\ \tau = \tau_1 \cdots \tau_n\rangle$. We may identify timed sequences and their corresponding timed words in the sequel, as long as no ambiguity arises.

Let us now define the notion of chronicle introduced by Ghallab [15]. Our definition of chronicle is borrowed from Besnard and Guyet [3].

▶ **Definition 2** (Chronicle). *A **chronicle** is a pair $(\mathcal{E}, \mathcal{T})$ where*

- *$\mathcal{E}$ is a **multiset** over Σ, i.e. $\mathcal{E}$ is of the form $\{\!\{c_1, \dots, c_m\}\!\}$ (in which repetitions are allowed) such that $c_i \in \Sigma$ for $i = 1, \dots, m$ and $c_1 \leq_\Sigma \dots \leq_\Sigma c_m$. We impose the latter condition for technical reasons explained at Remark 6;*
- *$\mathcal{T}$ is a set of **temporal constraints**, i.e. expressions of the form $(c, o_c)[t^-, t^+](c', o_{c'})$ such that*
  1. *$c, c' \in \mathcal{E}$ and*
  2. *$t^-, t^+ \in \mathbb{Q} \cup \{-\infty, +\infty\}$ and*
  3. *$o_c, o_{c'} \in [m]$ and $o_c < o_{c'}$ and*
  4. *$c_{o_c} = c$ and $c_{o_{c'}} = c'$.*

The size of a chronicle $(\mathcal{E}, \mathcal{T})$ is the size m of its multiset $\mathcal{E}$.

▶ **Example 3.** Let $\Sigma = \{A, B, C\}$; The pair

$$\left(\{\!\{A, B, B, C\}\!\}, \left\{\begin{array}{ll} (A,1)[-3.5,2](B,2), & (A,1)[-2,2.3](C,4), \\ (B,2)[-1,5](C,4), & (B,2)[0.1,2](B,3), \\ (B,3)[-1,5](C,4) & \end{array}\right\}\right)$$

is a chronicle. It involves two occurrences of event type B, and one occurrence of event types A and C. It has no direct temporal constraints between $(A, 1)$ and $(B, 3)$.

Such a chronicle can be represented as a directed graph, with one vertex per event in the multiset, and edges labelled with the temporal constraints. Figure 1 represents the chronicle above.

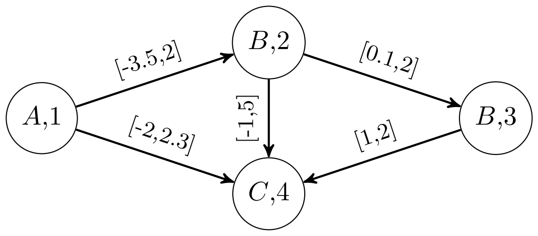


Figure 1 Graphical representation of the chronicle in Example 3.

We now define the semantics of chronicles, via the notion of occurrence of a chronicle in a timed sequence:

▶ **Definition 4** (Occurrence of a chronicle). *Let $\boldsymbol{s} = \langle(\sigma_1, \tau_1), (\sigma_2, \tau_2), \dots, (\sigma_n, \tau_n)\rangle$ be a timed sequence of length n, and $\mathscr{C} = (\mathcal{E} = \{\!\{c_1, \dots, c_m\}\!\}, \mathcal{T})$ be a chronicle of size m. Chronicle $\mathscr{C}$ is said to occur in $\boldsymbol{s}$ if, and only if, there exists an injective function $\varepsilon\colon [m] \to [n]$, hereafter called* embedding, *such that:*

1. *for all $1 \leq i < m$, $\tau_{\varepsilon(i)} < \tau_{\varepsilon(i+1)}$ whenever $c_i = c_{i+1}$,*[1]
2. *for all $1 \leq i \leq m$, $\sigma_{\varepsilon(i)} = c_i$,*
3. *for all $1 \leq i, j \leq m$, $\tau_{\varepsilon(j)} - \tau_{\varepsilon(i)} \in [t^-, t^+]$ whenever $(c_i, i)[t^-, t^+](c_j, j) \in \mathcal{T}$.*

Then, $\tilde{\boldsymbol{s}} = \{(\sigma_{\varepsilon(1)}, \tau_{\varepsilon(1)}), \dots, (\sigma_{\varepsilon(m)}, \tau_{\varepsilon(m)})\}$ is an occurrence *of $\mathscr{C}$ in $\boldsymbol{s}$.*

Chronicle $\mathscr{C}$ is said to match the sequence $\boldsymbol{s}$, denoted $\mathscr{C} \Subset \boldsymbol{s}$, if, and only if, there is at least one occurrence of $\mathscr{C}$ in $\boldsymbol{s}$.

[1] This condition is not always required for occurrences of chronicles, but it appears for instance in [3]. As we explain later, our results also holds when this condition is lifted. Similarly, in some settings it might be convenient to remove the injectiveness condition of the embedding functions, and again this would be easy to deal with in our translations.

The temporal constraints of a chronicle are conjunctive: an embedding of a chronicle has to satisfy all of them. As a consequence, a chronicle with two temporal constraints $(c, o_c)[t^-, t^+](c', o_{c'})$ and $(c, o_c)[u^-, u^+](c', o_{c'})$ relating the same pair of events is equivalent (w.r.t. occurrences) to a single temporal constraint with the interval $[t^-, t^+] \cap [u^-, u^+]$. It follows that in any chronicle, $\mathcal{T}$ can be assumed to contain at most one temporal constraint per pair of events. For any two indices i and j in $[m]$ with $i < j$, we write $t^-_{i,j}$ and $t^+_{i,j}$ for the rational values such that $(c_i, i)[t^-_{i,j}, t^+_{i,j}](c_j, j)$ is the (unique) temporal constraint between (c_i, i) and (c_j, j).

On a similar note, chronicles do not allow temporal constraints on pairs of events (c_i, i) and (c_j, j) when $i \geq j$; such constraints are trivial when $i = j$, while when $i > j$, the constraint $(c_i, i)[t^-, t^+](c_j, j)$ is equivalent (w.r.t. occurrences) to the constraint $(c_j, j)[-t^+, -t^-](c_i, i)$, which in turn can be intersected with the other constraints relating (c_i, i) and (c_j, j).

A chronicle that occurs in no sequences in said *inconsistent*. This is in particular the case of chronicles with unsatisfiable temporal constraints. For instance, the chronicle $(\{\!\{A, B, C\}\!\}, \{(A, 1)[1, 2](B, 2), (B, 2)[3, 4](C, 3), (A, 1)[-2, -1](C, 3)\})$ is inconsistent. Indeed, because of the first two temporal constraints, C must occur after A (B after A and C after B), but the third constraint enforces A to occur before C.

In this article, we do not bother about the minimality or the satisfiability of the temporal constraints. Dechter *et al.* [12] proposed reasoning techniques about temporal constraints to narrow the intervals of temporal constraints (w.r.t. some equivalence) or identify insatisfiable temporal constraints. Such techniques may apply for chronicles also, but they are not required for the results presented in this article.

The problem we address in this paper is whether chronicles can be represented by equivalent temporal logic formulas. In such a case, theoretical results and algorithms could be used to better understand chronicles, and possibly improve existing chronicle-matching algorithms [16].

▶ **Example 5.** Consider again the chronicle depicted at Fig. 1. This chronicle can be seen to occur in the following timed sequences:

- $\langle (A, 1.8), \boldsymbol{(A, 3.5)}, \boldsymbol{(B, 3.9)}, \boldsymbol{(B, 4.1)}, (C, 4.2), \boldsymbol{(C, 5.7)} \rangle$
- $\langle \boldsymbol{(B, 0.2)}, \boldsymbol{(B, 0.9)}, \boldsymbol{(C, 2.5)}, (B, 3.2), \boldsymbol{(A, 3.7)}, (A, 4.7) \rangle$

Events in bold are the events that form the embedding of the chronicle. The second temporal sequence illustrates that the chronicle can have occurrences with different orders of event types. This is possible thanks temporal constraints allowing negative delays.

▶ Remark 6. Notice that in a timed sequence, several events may occur at the same time, and such "simultaneous" events can appear in an occurrence of a chronicle. However, because the embeddings ε are required to be injective, a single event in a timed sequence cannot be used to match different copies of the same event in a chronicle. For instance, chronicle $(\{\!\{A, A\}\!\}, (A, 1)[-2, 2](A, 2))$ cannot occur in a timed sequence containing a single event A.

3 Disjunction of linear chronicles

We have seen that a chronicle expresses only conjunctive temporal constraints. This is obviously limiting to represent complex temporal behaviours for which there are possible alternative situations to represent. A natural solution is to use several chronicles, one for each situation and to define the situation recognition as a disjunctive matching of the chronicles.

▶ **Definition 7** (Occurrence of (disjunctive) collection of chronicles). *Let $\mathcal{C} = \{(\mathcal{E}_1, \mathcal{T}_1), \ldots, (\mathcal{E}_h, \mathcal{T}_h)\}$ be a collection of h chronicles and $\boldsymbol{s}$ be a timed sequence. A subsequence $\tilde{\boldsymbol{s}}$ of $\boldsymbol{s}$ is an* ***occurrence*** *of $\mathcal{C}$ if, and only if, there exists a chronicle $(\mathcal{E}, \mathcal{T}) \in \mathcal{C}$ such that $\tilde{\boldsymbol{s}}$ is an occurrence of $(\mathcal{E}, \mathcal{T})$. A collection $\mathcal{C}$ of chronicles matches a sequence $\boldsymbol{s}$, denoted $\mathcal{C} \Subset \boldsymbol{s}$, whenever there is at least one occurrence of $\mathcal{C}$ in $\boldsymbol{s}$.*

We now consider a variation of chronicles called *linear chronicles* (LC for short). A linear chronicle is a chronicle equipped with a permutation of the events in its multiset, prescribing the order of occurrence of those events. This is detailed more formally in the following definition.

▶ **Definition 8** (Linear chronicle). *A linear chronicle is a triple $\mathscr{L} = (\{\!\{c_1, \ldots, c_m\}\!\}, \mathcal{T}, \pi)$, where $(\{\!\{c_1, \ldots, c_m\}\!\}, \mathcal{T})$ is a chronicle and π is a permutation of $[m]$.*

A linear chronicle $\mathscr{L} = (\{\!\{c_1, \ldots, c_m\}\!\}, \mathcal{T}, \pi)$ occurs in a timed sequence $\boldsymbol{s} = \langle(\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots, (\sigma_n, \tau_n)\rangle$ whenever there exists an embedding $\varepsilon\colon [m] \to [n]$ witnessing that the chronicle $(\{\!\{c_1, \ldots, c_m\}\!\}, \mathcal{T})$ occurs in $\boldsymbol{s}$, and such that $\varepsilon \circ \pi$ is increasing.

Intuitively, a linear chronicle is a chronicle for which the order of the events is fixed (via π): in any occurrence, event $c_{\pi(i)}$ always occurs before event $c_{\pi(j)}$ when $i < j$.

▶ Remark 9. Condition 1 in Def. 4 states that for any two identical events c_i and c_j in a chronicle with $i < j$, the time at which c_i is matched must be strictly earlier than the time at which c_j is matched. In particular, for any embedding ε, we must have $\varepsilon(i) < \varepsilon(j)$. Assume that $\pi^{-1}(j) < \pi^{-1}(i)$; since $\varepsilon \circ \pi$ is increasing, we would have $\varepsilon(\pi(\pi^{-1}(j))) < \varepsilon(\pi(\pi^{-1}(i)))$, i.e., $\varepsilon(j) < \varepsilon(i)$. Then no embeddings would exist, and the linear chronicle is inconsistent. As a consequence, for a linear chronicle to be consistent, if two identical events c_i and c_j are such that $i < j$, we must have $\pi^{-1}(i) < \pi^{-1}(j)$; in other terms, π has to preserve the order of identical events.

The occurrence of a disjunctive collection of linear chronicles is defined in the very same way as for disjunctive collections of plain chronicles. Two collections of (possibly linear) chronicles are said *equivalent* if any occurrence of one of them is also an occurrence of the other one. As a special case, this defines an equivalence relation between single chronicles and collections of linear chronicles. We use this notion in the following proposition:

▶ **Proposition 10.** *For any chronicle $\mathscr{C} = (\mathcal{E}, \mathcal{T})$, there exists an equivalent disjunctive collection of linear chronicles.*

Proof. Write $\mathcal{E} = \{\!\{c_1, \ldots, c_m\}\!\}$. For any occurrence $\boldsymbol{s}$ of $\mathscr{C}$, with embedding ε, there is a permutation π of $[m]$ such that $\varepsilon(\pi(j)) < \varepsilon(\pi(j+1))$. We can thus write $\mathscr{C}$ as the disjunction, over all permutations π of $[m]$, of the linear chronicles obtained from $\mathscr{C}$ by adding the ordering π.

That the resulting disjunction of linear chronicles is equivalent to the original chronicle is straightforward: on the one hand, all linear chronicles are obtained from the original one by imposing an order of the events, so that any occurrence of any of the linear chronicles is an occurrence of the original chronicle; on the other hand, as already argued above, any occurrence of the original chronicle satisfies a specific order defined by some permutation π, so that it is an occurrence of one of the linear chronicles. ◀

▶ Remark 11. The number of linear chronicles in the disjunction obtained in our proof can be bounded by $m!$ (the factorial of m). Chronicles without any temporal constraints are easily seen to achieve this bound. By Remark 9, when several copies of the same event appear in the multiset, their order has to be preserved by π, which would reduce the number of permutations to consider.

As can be expected, linear chronicles can be turned in a special form where the timing constraints reflect the order of events:

▶ **Proposition 12.** *For any linear chronicle $\mathscr{L} = (\mathcal{E}, \mathcal{T}, \pi)$, there exists an equivalent linear chronicle $\mathscr{L}' = (\mathcal{E}, \mathcal{T}', \pi)$ such that each interval occurring in $\mathcal{T}'$ is either a subset of $\mathbb{R}_-$ or a subset of $\mathbb{R}_+$.*

Proof. Consider a temporal constraint $(c_{\pi(i)}, \pi(i))[t^-, t^+](c_{\pi(j)}, \pi(j))$ in $\mathcal{T}$. First assume that $i < j$ (the other case is symmetric). For any sequence $\boldsymbol{s} = \langle(\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)\rangle$ matched by $\mathscr{L}$, for any embedding $\varepsilon\colon [m] \to [n]$, we must have $\varepsilon(\pi(i)) < \varepsilon(\pi(j))$ (by definition of a matching for a linear chronicle), i.e., $\varepsilon(\pi(j)) - \varepsilon(\pi(i)) \in [0, +\infty)$. Under this requirement, the temporal constraint which imposes that $\tau_{\varepsilon(\pi(j))} - \tau_{\varepsilon(\pi(i))} \in [t^-, t^+]$ is then equivalent to the temporal constraint $\tau_{\varepsilon(\pi(j))} - \tau_{\varepsilon(\pi(i))} \in [0, t^+]$, so that the temporal constraint $(c_{\pi(i)}, \pi(i))[t^-, t^+](c_{\pi(j)}, \pi(j))$ in $\mathcal{T}$ can be replaced by the temporal constraint $(c_{\pi(i)}, \pi(i))[0, t^+](c_{\pi(j)}, \pi(j))$, while preserving the same set of occurrences. ◀

▶ **Example 13** (Equivalent collection of linear chronicles)**.** Figure 2 represents chronicle $\mathscr{C} = (\{\!\{A, B, C\}\!\}, \{(A, 1)[-2, 1](C, 3), (B, 2)[3, 4](C, 3)\})$ and an equivalent collection of three linear chronicles.

The timing constraint $(B, 2)[3, 4](C, 3)$ imposes that B must occur before C, but A can occur either before B, or between B and C, or after C.

We can verify that the temporal constraints of $\mathscr{C}$ forbids the other orders. Then, we can derive one linear from each order and from the temporal constraints of $\mathscr{C}$. This leads to the three linear chronicles depicted in Figure 2.

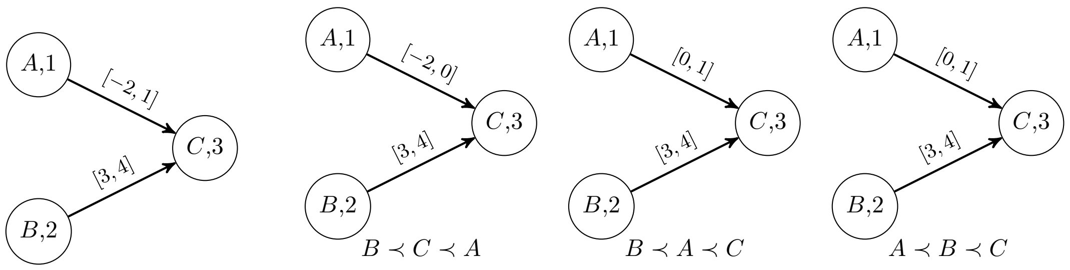


Figure 2 On the left: a chronicle $\mathscr{C}$. On the right: collection of three linear chronicles equivalent to $\mathscr{C}$. The order of event at the bottom of each linear chronicle illustrates their π; Thus, the two rightmost linear chronicles are distinct.

▶ Remark 14. As claimed above, Prop. 10 extends to the setting where we remove Condition 1 in Def. 4. Indeed, this condition imposes the order of occurrence of identical events in a matching; our translation into a disjunction of linear chronicles can thus easily be adapted by dropping all permutations that do not satisfy this condition.

Similarly, Prop. 10 extends to the setting where embeddings are allowed not to be injective: for this it suffices to transform chronicles into disjunctions of (still exponentially many) linear chronicles in which some of the identical events are merged.

4 Chronicles and $\mathsf{TPTL}_\Diamond$

The objective of this section is to establish a first result between chronicle and a temporal logic, namely the $\mathsf{TPTL}_\Diamond$. We start by recalling the syntax and the semantics of TPTL and $\mathsf{TPTL}_\Diamond$, then we propose the construction of a $\mathsf{TPTL}_\Diamond$ formula equivalent to any given linear chronicle. With the result of the previous section, the main result is that an equivalent $\mathsf{TPTL}_\Diamond$ formula can be constructed for any chronicle.

4.1 Timed Propositional Temporal Logic (TPTL)

The Timed Propositional Temporal Logic (TPTL) is a timed extension of LTL. It uses clock variables to explicitly represent timing constraints in formulae. Below, we define the syntax and semantics of TPTL borrowed from Bouyer et al. [6]. Formulae of TPTL are built from letters in Σ, Boolean connectives, *Until* operators ($\mathcal{U}$), clock resets and clock constraints:

$$\mathsf{TPTL} \ni \varphi ::= \sigma \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid x.\varphi \mid x \sim c$$

where σ is a letter in Σ, x is a clock variable drawn from a finite set X, $c \in \mathbb{Q}$ is a rational number, and $\sim \in \{\leq, <, =, >, \geq\}$.

We are interested in the pointwise semantics, which interprets TPTL over timed sequences. More precisely, models are (finite) sequences $\rho = (\sigma_i, \tau_i)_{i\in[n]}$. The satisfaction of a formula at a position i of such a sequence depends on the values of the clock variables that appear in the formula. Writing $v\colon X \rightharpoonup \mathbb{R}^+$ for a partial valuation of the clock variables, the satisfaction relation can then be inductively defined as follows:

$(\rho, i, v) \models \sigma$	if, and only if,	$\sigma = \sigma_i$
$(\rho, i, v) \models \varphi_1 \wedge \varphi_2$	if, and only if,	$(\rho, i, v) \models \varphi_1$ and $(\rho, i, v) \models \varphi_2$
$(\rho, i, v) \models \neg\varphi$	if, and only if,	it is not the case that $(\rho, i, v) \models \varphi$
$(\rho, i, v) \models \varphi_1 \mathcal{U} \varphi_2$	if, and only if,	there exists $j > i$ such that $(\rho, j, v) \models \varphi_2$ and for all $i < k < j$, $(\rho, k, v) \models \varphi_1$
$(\rho, i, v) \models x.\varphi$	if, and only if,	$(\rho, i, v[x \mapsto \tau_i]) \models \varphi$
$(\rho, i, v) \models x \sim c$	if, and only if,	$v(x)$ is defined, and $\tau_i - v(x) \sim c$

As can be observed in this definition, formulas of the form $x.\varphi$ (called *clock resets*) have the effect of storing the current time τ_i in clock x; at any later time τ_j, the value $\tau_j - v(x)$ corresponds to the amount of time that elapsed since the last reset of clock x: this justifies the semantics of formulas of the form $x \sim c$ (*clock constraints*).

▶ **Example 15.** Formula $x.[\alpha\,\mathcal{U}(\beta \wedge x \leq 10)]$ states that event β has to occur within 10 time units, and that only event α is allowed to occur in the meantime. Similarly, $x.[(\alpha \wedge x \leq 10)\,\mathcal{U}\,\beta]$ also states that eventually β must occur and that only events of type α can occur in the meantime, but additionally all of these events α must occur within the first 10 time units.

TPTL comes with several classical shorthands: conjunctions $\varphi_1 \vee \varphi_2$ are obtained as $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$; $\top$ stands for $\sigma \vee \neg\sigma$ (for some fixed $\sigma \in \Sigma$); $\Diamond\varphi$ stands for $\top\,\mathcal{U}\,\varphi$ (and means that φ eventually holds in a strict future), and $\Box\varphi$ stands for $\neg\Diamond\neg\varphi$ (and means that φ always holds in the strict future).

$\mathsf{TPTL}_\Diamond$ denotes the fragment of TPTL that uses only the $\Diamond$ modality (and limitations on the use of negation):

$$\mathsf{TPTL}_\Diamond \ni \varphi ::= \sigma \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond\varphi \mid x.\varphi \mid x \sim c.$$

▶ **Example 16.** The $\mathsf{TPTL}_\Diamond$ formula $x.\Diamond(\alpha \wedge \Diamond(\beta \wedge x \leq 10))$ states that (at least) one occurrence of α and one occurrence of β have to occur in that order within the next 10 time units.

4.2 $\mathsf{TPTL}_\Diamond$ formulae for linear chronicles

Contrary to TPTL operators, temporal constraints in chronicles do not impose an order on the occurrence of events. But linear chronicles do impose such an order; relying on Prop. 10, we present an encoding of chronicles in $\mathsf{TPTL}_\Diamond$.

Let $\mathscr{L} = (\mathcal{E} = \{\!\{c_1, \ldots, c_m\}\!\}, \mathcal{T}, \pi)$ be a linear chronicle of size m. We characterize $\mathscr{L}$ by a $\mathsf{TPTL}_\Diamond$ formula $\varphi_\mathscr{L}$ over $\mathcal{E}$ (seen as a finite set) and using a set $X = \{x_i \mid 1 \leq i \leq m-1\}$ of $m-1$ clocks; formula $\varphi_\mathscr{L}$ is obtained through the inductive definition of a collection of $\mathsf{TPTL}_\Diamond$ formulae $(\varphi^i_\mathscr{L})_{1\leq i\leq m}$, such that $\varphi_\mathscr{L} = \bar{\Diamond}\varphi^1_\mathscr{L}$, where $\bar{\Diamond}\phi$ stands for $\Diamond\phi \vee \phi$ (and means that ϕ holds now or at some point in the future).

The collection of formulae $(\varphi^i_\mathscr{L})_{i=1\ldots m}$ is defined as follows: if $m = 1$, then $\varphi^1_\mathscr{L} = c_{\pi(1)}$; otherwise,

$$\varphi^1_\mathscr{L} = \big(c_{\pi(1)} \wedge x_{\pi(1)}.\Diamond\varphi^2_\mathscr{L}\big) \tag{1}$$

and for all $2 \leq i \leq m-1$,

$$\varphi^i_\mathscr{L} = c_{\pi(i)} \wedge \mathscr{T}_i(\mathscr{L}) \wedge x_{\pi(i)}.\Diamond\varphi^{i+1}_\mathscr{L} \tag{2}$$

and finally

$$\varphi^m_\mathscr{L} = c_{\pi(m)} \wedge \mathscr{T}_m(\mathscr{L}) \tag{3}$$

where

$$\mathscr{T}_i(\mathscr{L}) = \bigwedge_{\substack{(c_{\pi(k)},\pi(k))[l,u](c_{\pi(i)},\pi(i))\in\mathcal{T}\\ \pi(k)<\pi(i)}} (l \leq x_{\pi(k)} \leq u) \wedge \\ \big((\pi(i) > 1 \wedge c_{\pi(i)} = c_{\pi(i)-1}) \to x_{\pi(i)-1} > 0\big) \tag{4}$$

Formula $\varphi^m_\mathscr{L}$ has size linear in m. Notice that, in the last conjunct of Eq. (4), the condition to the left of the implication is *static* (it only depends on the chronicle $\mathscr{L}$), and its role is simply to decide whether the condition on $x_{\pi(i)-1}$ has to be imposed.

We prove that this construction correctly encodes chronicles:

▶ **Proposition 17.** *For any linear chronicle $\mathscr{L}$ and any sequence ρ, it holds $\mathscr{L} \Subset \rho$ if, and only if, $\rho \models \varphi_\mathscr{L}$.*

Proof. Let $\mathscr{L} = (\{\!\{c_1, \ldots, c_m\}\!\}, \mathcal{T}, \pi)$ be a linear chronicle of size m, and $\rho = \langle(\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots, (\sigma_n, \tau_n)\rangle$ be a sequence. If $m = 1$, the chronicle has no timing constraints, and the result is straightforward. We now assume that $m > 1$.

We begin with the direct implication, assuming that $\mathscr{L} \Subset \rho$. By Def. 8, this means that there exists $\varepsilon\colon [m] \to [n]$ such that:

1. $\tau_{\varepsilon(i)} < \tau_{\varepsilon(i+1)}$ whenever $c_i = c_{i+1}$ for all $1 \leq i < m$,
2. $\sigma_{\varepsilon(i)} = c_i$ for all $1 \leq i \leq m$,
3. $\tau_{\varepsilon(j)} - \tau_{\varepsilon(i)} \in [l, u]$, whenever $(c_i, i)[l, u](c_j, j) \in \mathcal{T}$ for all $i < j$.

For all $1 \leq i \leq m$, we define $v_i\colon \{x_{\pi(j)} \mid 1 \leq j < i\} \to \mathbb{R}_+$ by $v_i(x_{\pi(j)}) = \tau_{\varepsilon(j)}$. By Property 2 of ε, we have that $\rho, \varepsilon(\pi(i)), v_i \models c_{\pi(i)}$ for all $i \in [m]$. By Property 3, we have that $\tau_{\varepsilon(\pi(i))} - v_i(x_{\pi(k)}) = \tau_{\varepsilon(\pi(i))} - \tau_{\varepsilon(\pi(k))} \in [l, u]$ for any $(c_{\pi(k)}, \pi(k))[l, u](c_{\pi(i)}, \pi(i)) \in \mathcal{T}$

with $i, k \in [m]$ such that $\pi(k) < \pi(i)$. Moreover, by Property 1, if $c_{\pi(i)-1} = c_{\pi(i)}$ (and $\pi(i) > 1$), then $\tau_{\varepsilon(\pi(i)-1)} < \tau_{\varepsilon(\pi(i))}$. It follows that $\tau(\varepsilon(\pi(i))) - v_i(x_{\pi(i)-1}) > 0$, which means that $\rho, \varepsilon(\pi(i)), v_i \models x_{\pi(i)-1} > 0$. In the end, we have shown that $\rho, \varepsilon(\pi(i)), v_i \models \mathscr{T}_i(\mathscr{L})$ for all $i \in [m]$.

We now prove by downward induction that $\rho, \varepsilon(\pi(i)), v_i \models \varphi^i_{\mathscr{L}}$. By the two properties above, we have $\rho, \varepsilon(\pi(m)), v_m \models c_{\pi(m)} \wedge \mathscr{T}_m(\mathscr{C})$, which proves our base case. Assuming that $\rho, \varepsilon(\pi(i+1)), v_{i+1} \models \varphi^{i+1}_{\mathscr{L}}$ for some $i \in [m-1]$, we prove that $\rho, \varepsilon(\pi(i)), v_i \models \varphi^i_{\mathscr{L}}$. Again, from the above two remarks, we have $\rho, \varepsilon(\pi(i)), v_i \models c_{\pi(i)} \wedge \mathscr{T}_i(\mathscr{L})$. It remains to prove that $\rho, \varepsilon(\pi(i)), v_i \models x_{\pi(i)}.\ \Diamond\varphi^{i+1}_{\mathscr{L}}$. This directly follows from the following facts: $\rho, \varepsilon(\pi(i+1)), v_{i+1} \models \varphi^{i+1}_{\mathscr{L}}$, and $\varepsilon(\pi(i)) < \varepsilon(\pi(i+1))$ (since $\varepsilon \circ \pi$ is increasing). By induction, we get $\rho, \varepsilon(\pi(1)), v_1 \models \varphi^1_{\mathscr{L}}$, which entails $\rho \models \varphi_{\mathscr{L}}$.

We now assume that $\rho \models \varphi_{\mathscr{L}}$. We construct an embedding ε of $\mathscr{L}$ in ρ inductively: formally, at each step j, we define $\varepsilon(\pi(j))$ so that

1. $\varepsilon(\pi(j))$ satisfies the conditions of Def. 4;
2. letting $w_j(x_{\pi(k)}) = \tau_{\varepsilon(\pi(k))}$ for all $k \in [j]$, we have $\rho, \varepsilon(\pi(j)), w_j \models \Diamond\varphi^{j+1}_{\mathscr{L}}$.

We initialize the induction as follows: since $\rho \models \bar{\Diamond}\varphi^1_{\mathscr{L}}$, there exists $\varepsilon(\pi(1))$ such that $\rho, \varepsilon(\pi(1)), \emptyset \models \varphi^1_{\mathscr{L}}$. By definition of $\varphi^1_{\mathscr{L}}$, this entails that

1. $\rho, \varepsilon(\pi(1)), \emptyset \models c_{\pi(1)}$, hence $\sigma_{\varepsilon(\pi(1))} = c_{\pi(1)}$; the other two conditions for being an embedding hold vacuously;
2. $\rho, \varepsilon(\pi(1)), \emptyset \models x_{\pi(1)}.\Diamond\varphi^2_{\mathscr{L}}$, which entails $\rho, \varepsilon(\pi(1)), w_1 \models \Diamond\varphi^2_{\mathscr{L}}$, where w_1 is the partial embedding defined only for $x_{\pi(1)}$ with $w_1(x_{\pi(1)}) = \tau_{\varepsilon(\pi(1))}$.

Now, assume that the result holds up to some step $\pi(j)$ for some $j < m$; we extend it to $\pi(j+1)$. Since $\rho, \varepsilon(\pi(j)), w_j \models \Diamond\varphi^{j+1}_{\mathscr{L}}$, there exists $\varepsilon(\pi(j+1)) > \varepsilon(\pi(j))$ for which $\rho, \varepsilon(\pi(j+1)), w_j \models \varphi^{j+1}_{\mathscr{L}}$. By definition of $\varphi^{j+1}_{\mathscr{L}}$, we get:

1. $\rho, \varepsilon(\pi(j+1)), w_j \models c_{\pi(j+1)}$, so that $\sigma_{\varepsilon(\pi(j+1))} = c_{\pi(j+1)}$. Additionally, $\rho, \varepsilon(\pi(j+1)), w_j \models \mathscr{T}_{j+1}(\mathscr{L})$, so that for each $(c_{\pi(k)}, \pi(k))[l, u](c_{\pi(j+1)}, \pi(j+1)) \in \mathcal{T}$, we have $l \leq \tau_{\varepsilon(\pi(j+1))} - \tau_{\varepsilon(\pi(k))} \leq u$. Finally, if $\pi(j+1) > 1$ and $c_{\pi(j+1)} = c_{\pi(j+1)-1}$, then $\rho, \varepsilon(\pi(j+1)), w_j \models x_{\pi(j+1)-1} > 0$, which means $\tau_{\varepsilon(\pi(j+1))} - \tau_{\varepsilon(\pi(j+1)-1)} > 0$, i.e., $\tau_{\varepsilon(\pi(j+1)-1)} < \tau_{\varepsilon(\pi(j+1))}$.
2. $\rho, \varepsilon(\pi(j+1)), w_j \models x_{\pi_{j+1}}\Diamond\varphi^{j+2}_{\mathscr{L}}$ (unless $j+1 = m$), which entails $\rho, \varepsilon(\pi(j+1)), w_{j+1} \models \Diamond\varphi^{j+2}_{\mathscr{L}}$.

In the end, we have built an embedding ε witnessing the fact that $\mathscr{L} \Subset \rho$. ◀

Our main result follows:

▶ **Theorem 18.** *Any chronicle $\mathscr{C}$ admits an equivalent* TPTL$_\Diamond$ *formula $\varphi_{\mathscr{C}}$.*

▶ Remark 19. Again, notice that our construction easily extends to the case where Condition 1 in Def. 4 is removed: it suffices to drop the last part of the definition of $\mathscr{T}_i(\mathscr{L})$ (Eq. (4)).

▶ Remark 20. In our definition of timed words and TPTL, we have taken the approach of seeing events as letters: if several events can take place at the same date, they are encoded as several consecutive letters in the timed word, all having the same timestamp; for instance, $w = (A, 3.2)(B, 3.2)(C, 4.1)$ corresponds to two events A and B occurring at the same date, and C occurring later.

Another approach would consist in seeing events as atomic propositions: in that setting, each timestamp would be unique, and would be associated with a (non-empty) set of events. The timed word w above would then correspond to $w' = (\{A, B\}, 3.2)(\{C\}, 4.1)$. Notice that this would not allow to have two occurrences of the same event at the same time; because of Condition 1 in Def. 4, removing multiple simultaneous copies of the same event preserves occurrences of chronicles.

Conceptually, our translation would still work, but we would have to allow the operator $\Diamond$ to also consider the present position; in other terms, we would have to replace each occurrence of $\Diamond$ in $\varphi^m_{\mathscr{L}}$ with $\bar{\Diamond}$. Strictly speaking, this would involve an exponential blow-up of the $\mathsf{TPTL}_\Diamond$ formula, since $\bar{\Diamond}\phi$ rewrites as $\phi \vee \Diamond\phi$, which requires duplicating formula ϕ.

▶ **Example 21** ($\mathsf{TPTL}_\Diamond$ formula for linear chronicles)**.** This example illustrates the construction of a $\mathsf{TPTL}_\Diamond$ formula for the linear chronicle depicted at Fig. 3; the order $B \prec A \prec C_3 \prec C_4$ for the events corresponds to the permutation π defined by $\pi(1) = 2$, $\pi(2) = 1$, $\pi(3) = 3$, $\pi(4) = 4$. We denote this linear chronicle by $\mathscr{L}$.

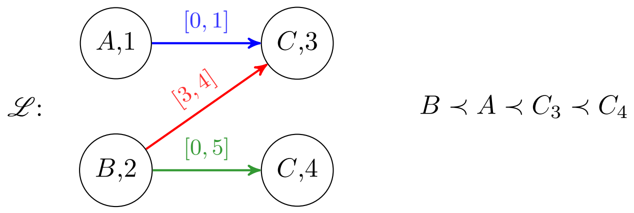


Figure 3 An example of a chronicle.

To construct a $\mathsf{TPTL}_\Diamond$ formula corresponding to $\mathscr{L}$, each event except the last one (according to π) is associated to a clock (x_1 for event $(A, 1)$, x_2 for event $(B, 2)$, x_3 for event $(C, 3)$).

Let us then define the temporal constraints according to Eq. (4):

$$\begin{aligned}
\mathscr{T}_1 &= \emptyset \\
\mathscr{T}_2 &= \emptyset \\
\mathscr{T}_3 &= x_1 \le 1 \wedge x_1 \ge 0 \wedge x_2 \le 4 \wedge x_2 \ge 3 \\
&= x_1 \le 1 \wedge x_2 \le 4 \wedge x_2 \ge 3 \\
\mathscr{T}_4 &= x_2 \le 5 \wedge x_2 \ge 0 \wedge x_3 > 0 \\
&= x_2 \le 5 \wedge x_3 > 0
\end{aligned}$$

The colors in formula match the color in Fig. 3. Formula $\mathscr{T}_2$ is empty because A is the first event in the chronicle multiset. Formula $\mathscr{T}_1$ is also empty, but for a different reason: there are no temporal constraints between A and B. It is also worth noticing that the constraints $x_3 > 0$ has been added to the temporal constraints of $\mathscr{T}_4$ because the implicit order between chronicle events having the same event type. The inequality is strict because of the injectiveness condition (see Remark 6).

Then, we construct the formula by induction following the order defined by π. It adds temporal constraints on the clocks marking the occurrence of previous events (relatively to time instants of the current event occurrence).

Thus, we obtain the following $\mathsf{TPTL}_\Diamond$ formula equivalent to chronicle $\mathscr{L}$:

$$\varphi_{\mathscr{L}} \quad = \quad \bar{\Diamond}(B \wedge x_2.\Diamond(A \wedge \mathscr{T}_2 \wedge x_1.\Diamond(C \wedge \mathscr{T}_3 \wedge x_3.\Diamond(C \wedge \mathscr{T}_4))))$$

The brown part of the formula is φ^4, which is the final case in the inductive construction of the formula. The black part is the initial case of the formula, which starts by the B event (the first event, according to π). The $\bar{\Diamond}$ operator catches the case of a sequence starting with a B. The remaing parts (orange for φ^3 and purple for φ^2) are the regular induction cases.

This formula can be rewritten after simplification and use of classical operators as follows:

$$\begin{aligned}\varphi_{\mathscr{L}} = {} & \Diamond(B \land x_2.\Diamond(A \land x_1.\Diamond(C \land x_1 \leq 1 \land x_2 \leq 4 \land x_2 \geq 3 \land x_3.\Diamond(C \land x_2 \leq 5 \land x_3 > 0)))) \\ & \lor (B \land x_2.\Diamond(A \land x_1.\Diamond(C \land x_1 \leq 1 \land x_2 \leq 4 \land x_2 \geq 3 \land x_3.\Diamond(C \land x_2 \leq 5 \land x_3 > 0))))\end{aligned}$$

▶ Remark 22. Notice that while each single linear chronicle is translated into a linear-size $\mathsf{TPTL}_\Diamond$ formula, the translation of a plain (non-linear) chronicle into a $\mathsf{TPTL}_\Diamond$ formula generally involves an exponential blow-up. This is not a concern for this paper, but proving that this blow-up cannot be avoided is an interesting direction for future work.

5 Metric temporal logic and chronicles

Metric Temporal Logic (MTL) is another timed extension of LTL that could be suited to encode chronicles. We first define its syntax and semantics, before dealing with the problem of encoding chronicles.

Given a finite alphabet Σ of atomic events, formulae of MTL are built up from Σ by Boolean connectives and time-constrained versions of the temporal operator *Until* as follows:

$$\mathsf{MTL} \ni \varphi ::= \sigma \mid \neg\varphi \mid \varphi_1 \lor \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2$$

where σ ranges over Σ, and $I = [l, u]$ is a temporal interval with l and u in $\mathbb{Q}$.

As for TPTL, in the pointwise semantics, MTL is evaluated at a position $i \in \mathbb{N}$ along a timed word $\rho = (\sigma_i, \tau_i)_{i\in[n]}$ as follows:

$$\begin{array}{lll} (\rho, i) \models \sigma & \text{if, and only if,} & \sigma = \sigma_i \\ (\rho, i) \models \varphi_1 \land \varphi_2 & \text{if, and only if,} & (\rho, i) \models \varphi_1 \text{ and } (\rho, i) \models \varphi_2 \\ (\rho, i) \models \neg\varphi & \text{if, and only if,} & \text{it is not the case that } (\rho, i) \models \varphi \\ (\rho, i) \models \varphi_1 \mathcal{U}_I \varphi_2 & \text{if, and only if,} & \exists j \geq i \text{ s.t. } (\rho, j) \models \varphi_2 \text{ and } \tau_j - \tau_i \in I \\ & & \text{ and } \forall i < k < j,\ (\rho, k) \models \varphi_1 \end{array}$$

As previously, we use the classical shorthands such has $\Diamond_I\varphi$, which stands for $\top\mathcal{U}_I\varphi$, and $\Box_I\varphi$, which stands for $\neg\Diamond_I\neg\varphi$.

It is not hard to observe that any MTL formula can be expressed in TPTL: formula $\phi\mathcal{U}_I\,\psi$ can be expressed as $x.(\phi\mathcal{U}(\psi \land x \in I))$. It was shown in Bouyer et al. [6] that TPTL is strictly more expressive than MTL in the general case. An example of a TPTL formula that has no equivalent MTL formula in the pointwise semantics is:

$$\phi = x.\Diamond(b \land \Diamond(c \land x \leq 2)) \tag{5}$$

Base on this counterexample of the equivalence between MTL and TPTL, we exhibit a chronicle that has no equivalent in MTL. Consider the following chronicle:

$$\mathscr{C} = (\{\!\{A, B, C\}\!\}, \{(A, 1)[0, +\infty](B, 2), (B, 2)[0, +\infty](C, 3), (A, 1)[0, 2](C, 3)\})\,.$$

The first two constraints impose that A, B and C must appear in that order (or possibly with the same timestamp). The last temporal constraint states that C must occur within two time units after A.

Using our transformation, the TPTL formula representing chronicle $\mathscr{C}$ is a disjunction of

$$\varphi^0_{\mathscr{C}} = \bar{\Diamond}(a \land x.\Diamond(b \land x \geq 0 \land y.\Diamond(c \land x \leq 2 \land y \geq 0))),$$

and of formulas of the form

$$\bar{\Diamond}(b \wedge y.\Diamond(a \wedge y \leq 0 \wedge x.\Diamond(c \wedge x \leq 2))$$

(one for each non-trivial permutation). Formula $\varphi^0_{\mathscr{C}}$ simplifies as

$$\bar{\Diamond}(a \wedge x.\Diamond(b \wedge \Diamond(c \wedge x \leq 2))),$$

which contains formula ϕ of Eq. (5) as a subformula.

Of course, the fact that ϕ occurs as a subformula of $\varphi^0_{\mathscr{C}}$ does not imply that $\mathscr{C}$ cannot be characterized in MTL. In order to formally prove this fact, we rely on the proof of [6] that some TPTL formulas have no MTL equivalent: consider the timed sequences depicted on Fig. 4: first notice that, for any integer p, chronicle $\mathscr{C}$ occurs in $\mathcal{A}_p$, while it does not occur in $\mathcal{B}_p$. On the other hand, Lemma 8 in [6] states that no MTL formula involving constants that are integer multiple of $\frac{1}{p}$ can distinguish between $\mathcal{A}_p$ and $\mathcal{B}_p$.

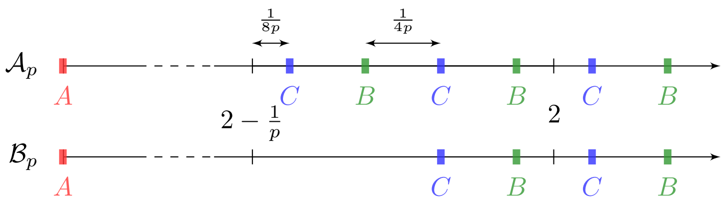


Figure 4 Two timed sequences not distinguishable by MTL with constants that are multiple of $1/p$.

Now, if there were an MTL formula $\psi_{\mathscr{C}}$ characterizing exactly chronicle $\mathscr{C}$, then for some integer p_0, this formula would involve constants that are integer multiple of p_0, and this formula would take the same value on $\mathcal{A}_{p_0}$ and $\mathcal{B}_{p_0}$, contradicting the fact that it exactly corresponds to $\mathscr{C}$.

It follows:

▶ **Theorem 23.** *There exist chronicles (with only three events) that admit no equivalent* MTL *formula.*

It is interesting to notice that the counterexample is a very simple chronicle: it only has three events and one useful temporal constraint. In addition, the temporal constraints do not straddle zero. They are all included in $\mathbb{R}_+$. Our initial intuition was that such temporal constraints would be a problem for translating chronicle in MTL. But the problem does not come from them. Intuitively, MTL formula does not need memory to be recognized but the recognition of a chronicle requires to store the position of all the occurrences of multiset events to check the temporal constraints. If chronicles of size 2 have equivalent formulae in MTL, the counterexample above illustrates a chronicle of size 3 – requiring to store the position of two events – a non-equivalent formula in MTL.

6 Conclusion

This work started from our need to specify complex situations to recognize disease or treatment from patient care pathways. In this medical context, specifying temporal arrangements of events is of particular interest to have accurate specifications. Then, our problem is to find a formalism that is both expressive and efficient to recognize complex situations in large datasets of patients.

In this article we investigated the temporal model of chronicle and compared its expressiveness to two temporal logics TPTL and MTL. Chronicle is a temporal model that emerged in the field of complex event processing. It can be used to efficiently monitor a stream of events. Despite its seeming simplicity, the temporal constraints of a chronicle allows to specify situations without presupposition on the events order. Then, such temporal models were intuitively qualified as *highly* expressive but, to the best of our knowledge, no formal comparison with other temporal formalisms were made. Then, our objective was to better qualify the expressiveness of chronicles through their comparison against MTL.

In this article we have shown that any chronicle as an equivalent formula in $\mathsf{TPTL}_{\Diamond}$, but some chronicles have no equivalent formula in MTL. This confirms that chronicles have an interesting expressiveness.

This first result opens interesting perspectives: the formulation of TPTL equivalent to chronicle can be a cornerstone for new results with other temporal models, and more especially other models from the field of event processing. It also raises the question of the equivalence with TPTL: would it be possible to find an equivalent collection of chronicles for a TPTL formula? If it is possible, chronicles may be a new approach for recognition of TPTL formulae.

The negative result we had with MTL also opens new questions. We identify a plausible reason for non-equivalence that guides us toward other logics that would be also interesting to compare chronicles. Our first target is MTL with past operators [24]. We conjecture there is no equivalence with chronicles.

References

1 Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–203, 1994.

2 Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic web*, 3(4):397–407, 2012.

3 Philippe Besnard and Thomas Guyet. *Chronicles.* under submission, 2022.

4 Michael H Böhlen, Jan Chomicki, Richard T Snodgrass, and David Toman. Querying TSQL2 databases with temporal logic. In *Proceedings of the International Conference on Extending Database Technology*, pages 325–341, 1996.

5 Alessio Bottrighi, Laura Giordano, Gianpaolo Molino, Stefania Montani, Paolo Terenziani, and Mauro Torchio. Adopting model checking techniques for clinical guidelines verification. *Artificial Intelligence in Medicine*, 48(1):1–19, 2010.

6 Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. *Information and Computation*, 208(2):97–116, 2010.

7 Sebastian Brandt, Elem Güzel Kalaycı, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. Querying log data with metric temporal logic. *Journal of Artificial Intelligence Research*, 62:829–877, 2018.

8 Martin Chapman, Luke V Rasmussen, Jennifer A Pacheco, and Vasa Curcin. Phenoflow: A microservice architecture for portable workflow-based phenotype definitions. *Procedings of the Summits on Translational Science*, 2021:142, 2021.

9 Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71, 1981.

10 Damien Cram, Benoît Mathern, and Alain Mille. A complete chronicle discovery approach: application to activity analysis. *Expert Systems*, 29(4):321–346, 2012.

11 Yann Dauxais, Thomas Guyet, David Gross-Amblard, and André Happe. Discriminant chronicles mining. In *Proceedings of the Conference on Artificial Intelligence in Medicine in Europe (AIME)*, pages 234–244, 2017.

12 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1–3):61–95, 1991.

13 Christophe Dousson, Paul Gaborit, and Malik Ghallab. Situation recognition: representation and algorithms. In *Proceedings of the international joint conference on Artifical intelligence (IJCAI)*, pages 166–172, 1993.

14 Christophe Dousson and Thang Vu Duong. Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In *Proceedings of the international joint conference on Artifical intelligence (IJCAI)*, pages 620–629, 1999.

15 Malik Ghallab. On chronicles: Representation, on-line recognition and learning. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 597–606, 1996.

16 Thomas Guyet, Philippe Besnard, Ahmed Samet, Nasreddine Ben Salha, and Nicolas Lachiche. Énumération des occurrences d'une chronique. In *Extraction et Gestion des Connaissances*, pages 253–260, 2020.

17 Romain Kervac and Ariane Piel. A survey on chronicles and other behavior detection techniques. *Journal of Aerospace Lab*, 15, 2020.

18 Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

19 Leslie Lamport. The temporal logic of actions. *Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, 1994.

20 Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the situation calculus. *Linköping Electronic Articles in Computed and Information Science*, 3(18), 1998.

21 Erik T Mueller. Event calculus. *Foundations of Artificial Intelligence*, 3:671–708, 2008.

22 Joël Ouaknine and James Worrell. On metric temporal logic and faulty turing machines. In *Proceedings of the International Conference on Foundations of Software Science and Computation Structures*, pages 217–230, 2006.

23 Amir Pnueli. The temporal logic of programs. In *Proceedings of the Annual Symposium on Foundations of Computer Science (SFCS)*, pages 46–57, 1977.

24 Pavithra Prabhakar and Deepak D'Souza. On the expressiveness of MTL with past operators. In *Proceedings of Formal Modeling and Analysis of Timed Systems*, pages 322–336, 2006.

25 Paolo Terenziani, Gianpaolo Molino, and Mauro Torchio. A modular approach for representing and executing clinical guidelines. *Artificial intelligence in medicine*, 23(3):249–276, 2001.

26 Dogan Ulus. *Pattern Matching with Time: Theory and Applications.* PhD thesis, Université Grenoble Alpes, 2018.

27 Przemyslaw A. Wałęga, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. DatalogMTL: Computational complexity and expressive power. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1886–1892, 2019.

Realizability Problem for Constraint LTL

Ashwin Bhaskar
Chennai Mathematical Institute, India

M. Praveen
Chennai Mathematical Institute, India
CNRS IRL ReLaX, Chennai, India

Abstract

Constraint linear-time temporal logic (CLTL) is an extension of LTL that is interpreted on sequences of valuations of variables over an infinite domain. The atomic formulas are interpreted as constraints on the valuations. The atomic formulas can constrain valuations over a range of positions along a sequence, with the range being bounded by a parameter depending on the formula. The satisfiability and model checking problems for CLTL have been studied by Demri and D'Souza. We consider the realizability problem for CLTL. The set of variables is partitioned into two parts, with each part controlled by a player. Players take turns to choose valuations for their variables, generating a sequence of valuations. The winning condition is specified by a CLTL formula – the first player wins if the sequence of valuations satisfies the specified formula. We study the decidability of checking whether the first player has a winning strategy in the realizability game for a given CLTL formula. We prove that it is decidable in the case where the domain satisfies the completion property, a property introduced by Balbiani and Condotta in the context of satisfiability. We prove that it is undecidable over $(\mathbb{Z}, <, =)$, the domain of integers with order and equality. We prove that over $(\mathbb{Z}, <, =)$, it is decidable if the atomic constraints in the formula can only constrain the current valuations of variables belonging to the second player, but there are no such restrictions for the variables belonging to the first player. We call this single-sided games.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking; Theory of computation → Automata over infinite objects; Theory of computation → Tree languages

Keywords and phrases Realizability, constraint LTL, Strategy trees, Tree automata

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.8

Related Version *Full Version*: `https://arxiv.org/abs/2207.06708`

Funding *M. Praveen*: This author is partially supported by the Infosys foundation.

1 Introduction

Propositional linear temporal logic (LTL) and related automata theoretic models have been extended in various ways to make it more expressive. Prompt-LTL [18], Constraint LTL [13], LTL with freeze operators [12], temporal logic of repeating values [11, 24], finite memory automata [16], data automata [8] are all examples of this. Prompt-LTL is concerned with bounding wait times for formulas that are intended to become true eventually, while other extensions are concerned with using variables that range over infinite domains in place of Boolean propositions used in propositional LTL. Variables ranging over infinite domains are a natural choice for writing specifications for systems that deal with infinite domains. For example, constraint LTL has been used for specifications of cloud based elastic systems [6], where the domain of natural numbers is used to reason about the number of resources that are being used by cloud based systems.

An orthogonal development in formal verification is synthesis, that is concerned with automatically synthesizing programs from logical specifications. The problem was identified by Church [10] and one way to solve it is by viewing it as the solution of a two person game. For specifications written in propositional LTL, the worst case complexity of the realizability problem is doubly exponential [23]. However, efficient algorithms exist for fragments of LTL. The algorithms are efficient enough and the fragments are expressive enough to be used in practice, for example to synthesize robot controllers [17], data buffers and data buses [22].

This paper is in an area that combines both developments mentioned in the above paragraphs. We consider constraint LTL (CLTL) and partition the set of variables into two parts, each being owned by a player in a two player game. The players take turns to choose a valuation for their variables over an infinite domain. The game is played forever and results in a sequence of valuations. The first player tries to ensure that the resulting sequence satisfies a specified CLTL formula (which is the winning condition) and the second player tries to foil this. We study the decidability of checking whether the first player has a winning strategy, called the realizability problem in the sequel. CLTL is parameterized by a constraint system, that can have various relations over the infinite domain. The atomic formulas of CLTL can compare values of variables in different positions along a range of positions, using the relations present in the constraint system. The range of positions is bounded and depends on the formula. E.g., an atomic formula can say that the value of x at a position is less than the value of y in the next position, in the domain of integers or real numbers with linear order. Decidability of the CLTL realizability problem depends on the constraint system. It also depends on whether the atomic formulas can compare values at different positions of the input, as opposed to comparing values of different variables at the same position of the input. If the former is allowed only for variables belonging to one of the players, they are called single-sided games. This is illustrated next.

For instance in cloud based elastic systems [6], the number of resources allocated and the number of virtual machines running are tracked. One desirable property is that if the number of virtual machines increases, the number of resources allocated also increase. Typically the number of resources allocated is controlled by the system and the number of virtual machines is controlled by the environment. Let x be a variable that keeps track of the number of resources allocated and let y denote the number of virtual machines. Specifying this property will require comparing the value of x at the current position with the value of x at the next position. We may also compare the current value of y with its value at the next position, but this will need both the system and the environment to be able to compare the values of their variables at different positions. Instead, if we restrict the environment to only decide whether a new virtual machine request is raised at the current position, the environment need not compare the value of y with its value at the next position. Hence only system compares the values of its variables across different positions and thus, the game will be single-sided.

Contributions. We prove that the realizability problem for CLTL is

1. 2EXPTIME-complete for constraint systems that satisfy a so-called completion property,
2. undecidable for integers with linear order and equality and
3. 2EXPTIME-complete for single-sided games on integers with linear order and equality.

The third result above is the main one and is inspired by concepts used in satisfiability [13]. In satisfiability, this technique is based on patterns that repeat in ultimately periodic words. It requires new insights to make it work in trees that we use to represent strategies here.

Related works. Two player games on automata models and logics dealing with infinite domains have been studied before [28, 14]. The techniques involved are similar to those used here in the sense that instead of reasoning about sequences of values from an infinite domain, sequences of elements from a finite abstraction are considered. Single-sided games are considered in [14], like we do here, but for register automata specifications. Their result subsumes ours, since register automata are more expressive than CLTL. In register automata, values can be compared even if they occur far apart in the input sequence, but in CLTL, values can only be compared if they occur within a bounded distance. For this reason, CLTL can be handled with simpler arguments, resulting in some differences in technical details, which we will highlight later in this paper. This can potentially speed up procedures in case the specifications only need CLTL and not the full power of register automata[1]. Similar single-sided games are also considered in [25], for an extension of LTL incomparable with CLTL. There, single-sided games are reduced to energy games [2] to get decidability.

Church Synthesis problem for a restriction of First Order Logic over parametrized alphabets has been studied in [5]. The parametrized alphabet reflects the number of processes. Similar to the results in our paper, the synthesis problem in [5] is undecidable in the general case but turns decidable when the number of processes that are controllable by the environment is bounded, while the number of system processes remains unbounded. In [26], a parametrized extension of the Church Synthesis Problem of MSO Logic over $(\mathbb{N}, <)$ is considered. The decidability result in this paper extensively uses the idea of patterns repeating in ultimately periodic words [26, Proposition 3.4] as is the case in our work too.

2 Preliminaries

Let $\mathbb{Z}$ be the set of integers and $\mathbb{N}$ be the set of non-negative integers. We denote by $\lceil i \rceil_k$ the number i ceiled at k: $\lceil i \rceil_k = i$ if $i \leq k$ and $\lceil i \rceil_k = k$ otherwise. If m is any mapping and S is a subset of the domain of m, we denote by $m \upharpoonright S$ the mapping m restricted to the domain S. For a sequence of mappings $m_1 \cdot m_2 \cdots$, we write $m_1 \cdot m_2 \cdots \upharpoonright S$ for $m_1 \upharpoonright S \cdot m_2 \upharpoonright S \cdots$. For integers n_1, n_2, we denote by $[n_1, n_2]$ the set $\{n \in \mathbb{Z} \mid n_1 \leq n \leq n_2\}$.

We recall the definitions of constraint systems and constraint LTL (CLTL) from [13]. A constraint system $\mathcal{D}$ is of the form $(D, R_1, \ldots, R_n, \mathcal{I})$, where D is a non-empty set called the domain. Each R_i is a predicate symbol of arity a_i, with $\mathcal{I}(R_i) \subseteq D^{a_i}$ being its interpretation.

Let V be a set of variables, partitioned into the sets V^a, V^b of look-**a**head and future-**b**lind variables. A look-ahead term is of the form $X^i y$, where y is a look-ahead variable, $i \geq 0$ and X is a symbol intended to denote "next". For $k \geq 0$, we denote by $T^a[k]$ the set of all look-ahead terms of the form $X^i y$, where $i \in [0, k]$ and y is a look-ahead variable. A constraint c is of the form $R(t_1, \ldots, t_n)$, where R is a predicate symbol of arity n and $t_1, \ldots, t_n$ are all future-blind variables or they are all look-ahead terms. The syntax of CLTL is given by the following grammar, where c is a constraint as defined above.

$$\phi ::= c \mid \neg\phi \mid \phi \vee \phi \mid X\phi \mid \phi U \phi$$

The semantics of CLTL is defined over sequences σ (also called concrete models in the following); for every $i \geq 0$, $\sigma(i) \colon V \to D$ is a mapping of the variables. Given, $x_1, \ldots, x_n \in V^a$ and $i_1, \ldots, i_n \in \mathbb{N}$, the i^{th} position of a concrete model σ satisfies the constraint

[1] This does need a detailed study, which we defer to future work.

$R(X^{i_1}x_1, \ldots, X^{i_n}x_n)$ (written as $\sigma, i \models R(X^{i_1}x_1, \ldots, X^{i_n}x_n)$) if $(\sigma(i+i_1)(x_1), \ldots, \sigma(i+i_n)(x_n)) \in \mathcal{I}(R)$. If the constraint is of the form $R(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n \in V^b$, then $\sigma, i \models R(x_1, \ldots, x_n)$ if $(\sigma(i)(x_1), \ldots, \sigma(i)(x_n)) \in \mathcal{I}(R)$. The semantics is extended to the rest of the syntax similar to the usual propositional LTL. We use the standard abbreviations $F\phi$ (resp. $G\phi$) to mean that ϕ is true at some position (resp. all positions) in the future. The X-length of a look-ahead term X^iy is i. We say that a formula is of X-length k if it uses look-ahead terms of X-length at most k. The constraint system $(\mathbb{Z}, <, =)$ (resp. $(\mathbb{N}, <, =)$) has the domain $\mathbb{Z}$ (resp. $\mathbb{N}$) and $<, =$ are interpreted as the usual linear order and equality relations. The formula $G(x < Xy)$ will be true in the first position of a concrete model if in all positions, the value of x is less than the value of y in the next position. Recall the example of cloud based elastic systems described in the introduction. Variable x denoted the number of resources allocated and it was possible to compare its values across different positions, hence making it a look-ahead variable. Whereas, under the restrictions on the environment, the value of the variable y was not allowed to be compared with the values of variables at other positions. This makes y a future-blind variable.

We adapt the concept of realizability games [23] to CLTL. There are two players `system` and `environment`. The set of variables V is partitioned into two parts SV, EV owned by `system`, `environment`, respectively. The `environment` begins by choosing a mapping $em_0 \colon EV \to D$, to which `system` responds by choosing a mapping $sm_0 \colon SV \to D$. This first round results in the mapping $em_0 \oplus sm_0$. This notation is used to define the function such that $em_0 \oplus sm_0(x) = em_0(x)$ if $x \in EV$ and $em_0 \oplus sm_0(x) = sm_0(x)$ if $x \in SV$. In the next round, the two players chose mappings em_1, sm_1. Both players continue to play forever and the play results in a concrete model $\sigma = (em_0 \oplus sm_0)(em_1 \oplus sm_1)\cdots$. The winning condition is specified by a CLTL formula ϕ. `System` wins this play of the game if $\sigma, 0 \models \phi$.

Let M (resp. EM,SM) be the set of all mappings of the form $V \to D$ (resp. $EV \to D$, $SV \to D$). For a concrete model σ and $i \geq 0$, let $\sigma \upharpoonright i$ denote the prefix of σ of length i (for $i = 0$, $\sigma \upharpoonright i$ is the empty sequence ϵ). An `environment` strategy is a function $et \colon M^* \to EM$ and a `system` strategy is a function $st \colon M^* \cdot EM \to SM$. We say that the `environment` plays according to the strategy et if the resulting model $\sigma = (em_0 \oplus sm_0)(em_1 \oplus sm_1)\cdots$ is such that $em_i = et(\sigma \upharpoonright i)$ for all $i \geq 0$. `System` plays according to the strategy st if the resulting model $\sigma = (em_0 \oplus sm_0)(em_1 \oplus sm_1)\cdots$ is such that $sm_i = st(\sigma \upharpoonright i \cdot em_i)$ for all $i \geq 0$. We say that st is a winning strategy for `system` if she wins all plays of the game played according to st, irrespective of the strategy used by `environment`. For example, let us consider a CLTL game with $V = V^a = \{x, y\}, EV = \{x\}, SV = \{y\}$, over the constraint system $(\mathbb{Z}, <, =)$ with winning condition $G((y > Xy) \wedge \neg((X^2x > y) \wedge (X^2x < Xy)))$. For `system` to win, the sequence of valuations for y should form a descending chain, and at any position, the value of x should be outside the interval defined by the previous two values of y. `System` has a winning strategy in this game: it can choose y to be $-i$ in the i^{th} round and the `environment` cannot choose its x to be strictly between the previous two values of y in any round. `System` does not have a winning strategy in the same game when it is considered over $(\mathbb{N}, <, =)$, as there is no infinite descending sequence of natural numbers. `System` does not have a winning strategy over dense domains, since `environment` can choose the third value of x to be strictly between the first two values of y, violating the winning condition. Given a CLTL formula ϕ, the **realizability problem** is to check whether `system` has a winning strategy in the CLTL game whose winning condition is ϕ.

We now state an important result.

▶ **Theorem 1.** *The realizability problem for CLTL over $(\mathbb{Z}, <, =)$ and $(\mathbb{N}, <, =)$ is undecidable.*

This can be proved by a reduction from the repeated control state reachability problem for 2-counter machines, which is known to be undecidable [3]. The main idea of the reduction is that one of the players simulates the counter machine and the other player catches mistakes, like other similar reductions for games [1]. For a detailed proof of this result, please refer to the arXiv version of this paper [7].

Some proofs and technical details in the subsequent sections are moved to the appendix due to space constraints.

3 Symbolic Models

The models of CLTL are infinite sequences over infinite alphabets. Frames, introduced in [13], abstract them to finite alphabets. We adapt frames to constraint systems of the form $(D, <, =)$. Conceptually, frames and symbolic models as we will define here are almost the same as introduced in [13], where the authors used these notions to solve the satisfiability problem for CLTL. For the purpose of CLTL games, we use slightly different definitions and notations, as this makes it easier to present game-theoretic arguments. For the rest of the paper, we shall assume that the set of variables V is finite. Also, unless mentioned otherwise, we shall assume that D is $\mathbb{Z}$, $\mathbb{N}$ or a domain that satisfies a so-called completion property.

Suppose that the first player owns the variables x, z. The second player owns y and wants to ensure that $x < y \ \wedge \ y < z$ over the domain of integers. It depends on whether the gap between the values assigned by the first player to x and to z, is large enough for the second player to push y in between.

▶ **Definition 2** (gap functions). *Given a mapping $m\colon V^b \to D$, we associate with it a gap function $gp\colon V^b \to \mathbb{N}$ as follows. Arrange V^b as $x_0, x_1, \ldots$ such that $m(x_0) \leq m(x_1) \leq \cdots$. Define the function gp such that $gp(x_0) = 0$ and $gp(x_{l+1}) = gp(x_l) + \lceil m(x_{l+1}) - m(x_l) \rceil_{|V^b|-1}$ for all $l < |V^b| - 1$.*

The left hand side of the above equation denotes the gap between x_l and x_{l+1} according to the gp function. The right hand side denotes the gap between the same variables according to the mapping m, ceiled at $|V^b| - 1$. Since V^b is finite, the set of gap functions is also finite. We use gap functions only for future-blind variables V^b, only for the domains $\mathbb{Z}$ or $\mathbb{N}$. Hence, the minus sign '$-$' in the definition of gap functions is interpreted as the usual subtraction over $\mathbb{Z}$ or $\mathbb{N}$.

The following definition formalizes how a frame captures information about orders and gaps for s successive positions.

▶ **Definition 3** (Frames). *Given a number $s \geq 1$, an s-frame f is a pair $(\leq_f, gp_f)$, where $\leq_f$ is a total pre-order*[2] *on the set of look-ahead terms $T^a[s-1]$ and $gp_f\colon V^b \times [0, s-1] \to \mathbb{N}$ is a function such that for all $i \in [0, s-1]$, $\lambda x.gp_f(x, i)$*[3] *is a gap function.*

In the notation s-frame, s is intended to denote the size of the frame – the number of successive positions about which information is captured. The current position and the following $(s-1)$ positions are considered, for which the look-ahead terms in $T^a[s-1]$ are needed. We denote by $<_f$ and $\equiv_f$ the strict order and equivalence relation induced by $\leq_f$: $x <_f y$ iff $x \leq_f y$ and $y \not\leq_f x$ and $x \equiv_f y$ iff $x \leq_f y$ and $y \leq_f x$.

[2] a reflexive and transitive relation such that for all x, y, either $x \leq_f y$ or $y \leq_f x$

[3] Note that we could have used a function $h_i(x) = gp_f(x, i)$ instead of using the lambda notation. But this introduces a new notation – the function h_i, which will not be used anywhere else.

We will deal with symbolic models that constitute sequences of frames. An s-frame will capture information about the first s positions of a model. If this is followed by a $(s+1)$-frame, it will capture information about the first $(s+1)$ positions of the model. Both frames capture information about the first s positions, so they must be consistent about the information they have about the shared positions. Similarly, an s-frame meant for positions i to $i+s-1$ may be followed by another s-frame meant for positions $i+1$ to $i+s$. The two frames must be consistent about the positions $i+1$ to $i+s-1$ that they share. The following definition formalizes these requirements.

▶ **Definition 4** (One-step compatibility). *For $s \geq 1$, an s-frame f and an $(s+1)$-frame g, the pair (f,g) is one-step compatible if the following conditions are true.*

- *For all terms $t_1, t_2 \in T^a[s-1]$, $t_1 \leq_f t_2$ iff $t_1 \leq_g t_2$.*
- *For all $j \in [0, s-1]$ and all variables $x \in V^b$, $gp_f(x,j) = gp_g(x,j)$.*

For $s \geq 2$ and s-frames f, g, the pair (f,g) is one-step compatible if:

- *For all terms $t_1, t_2 \in T^a[s-2]$, $Xt_1 \leq_f Xt_2$ iff $t_1 \leq_g t_2$ and*
- *for all $j \in [0, s-2]$ and all variables $x \in V^b$, $gp_f(x, j+1) = gp_g(x,j)$.*

Fix a number $k \geq 0$ and consider formulas of X-length k. A symbolic model is a sequence ρ of frames such that for all $i \geq 0$, $\rho(i)$ is an $\lceil i+1 \rceil_{k+1}$-frame and $(\rho(i), \rho(i+1))$ is one-step compatible. CLTL formulas can be interpreted on symbolic models, using symbolic semantics $\models_s$ as explained next. To check if the i^{th} position of ρ symbolically satisfies the atomic constraint $t_1 < t_2$ (where t_1, t_2 are look-ahead terms), we check whether $t_1 < t_2$ according to the i^{th} frame $\rho(i)$. In formal notation, this is written as $\rho, i \models_s t_1 < t_2$ if $t_1 <_{\rho(i)} t_2$. For future-blind variables x, y, $\rho, i \models_s x < y$ if $gp_{\rho(i)}(x,0) < gp_{\rho(i)}(y,0)$. The symbolic satisfaction relation $\models_s$ is extended to all CLTL formulas of X-length k by induction on the structure of the formula, as done for propositional LTL. To check whether $\rho, i \models_s t_1 < t_2$ in this symbolic semantics, we only need to check $\rho(i)$, the i^{th} frame in ρ, unlike the CLTL semantics, where we may need to check other positions also. In this sense, the symbolic semantics lets us treat CLTL formulas as if they were formulas in propositional LTL and employ techniques that have been developed for propositional LTL. But to complete that task, we need a way to go back and forth between symbolic and concrete models.

Given a concrete model σ, we associate with it a symbolic model $\mu(\sigma)$ as follows. Imagine we are looking at the concrete model through a narrow aperture that only allows us to view $k+1$ positions of the concrete model, and we can slide the aperture to view different portions. The i^{th} frame of $\mu(\sigma)$ will capture information about the portion of the concrete model visible when the right tip of the aperture is at position i of the concrete model (so the left tip will be at $i - \lceil i \rceil_k$). Formally, the total pre-order of the i^{th} frame is the one induced by the valuations along the positions $i - \lceil i \rceil_k$ to i of the concrete model. For every $j \in [0, \lceil i \rceil_k]$, the function $\lambda x. gp_f(x,j)$ of the i^{th} frame is the gap function associated with the mapping $\sigma(i - \lceil i \rceil_k + j) \upharpoonright V^b$.

For every concrete model, there is an associated symbolic model, but the converse is not true. E.g., if every frame in a symbolic model requires $Xx < x$, the corresponding concrete model needs to have an infinite descending chain, which is not possible in the constraint system $(\mathbb{N}, <, =)$. We say that a symbolic model ρ admits a concrete model if there exists a concrete model σ such that $\rho = \mu(\sigma)$.

▶ **Lemma 5** ([13, Lemma 3.1]). *Let ϕ be a CLTL formula of X-length k. Let σ be a concrete model and $\rho = \mu(\sigma)$. Then $\sigma, 0 \models \phi$ iff $\rho, k \models_s \phi$.*

4 Decidability Over Domains Satisfying the Completion Property

In this section, we prove that the CLTL realizability problem is decidable if the domain satisfies a so called completion property. Let C be a set of constraints over a constraint system $\mathcal{D}$. We call C satisfiable if there is a valuation satisfying all the constraints in C. For a subset $U \subseteq V$ of variables, $C \restriction U$ is the subset of C consisting of those constraints that only use terms built with variables in U. A partial valuation v' is a valuation for the terms occurring in $C \restriction U$. We say $\mathcal{D}$ has the completion property if for every satisfiable set of constraints C and every subset $U \subseteq V$, every partial valuation v' satisfying $C \restriction U$ can be extended to a valuation v satisfying C. An example of a constraint system which does not satisfy the completion property is $(\mathbb{Z}, <, =)$, since for the set of constraints $C = \{x < y, x < z, z < y\}$ over the set of variables $V = \{x, y, z\}$, the partial valuation $v\colon x \mapsto 0, y \mapsto 1$ satisfies the constraints in C involving x and y, but cannot be extended to a valuation which satisfies the constraints $x < z$ and $z < y$ in C. The constraint systems $(\mathbb{Q}, <, =)$ and $(\mathbb{R}, <, =)$ satisfy the completion property. Also, one can easily see that for every infinite domain D, the constraint system $(D, =)$ always satisfies the completion property.

It is known that CLTL satisfiability is decidable for constraint systems that satisfy the completion property [13, 4]. The completion property of a constraint system is closely related to the denseness of the underlying domain. A constraint system satisfies the completion property if and only if the underlying domain is dense and open [13, Lemma 5.3].

Consider an example of a controller system that controls the temperature of water in a water tank. Let x be a variable that denotes the temperature of the water. The controller may be required to guarantee, for instance, that $20 \leq x \leq 100$. In principle, the temperature of water can be any real number. Hence x comes from a dense domain. So the domain over which properties of such a system are specified satisfies the completion property (refer to [27] for a detailed explanation of such a controller). In contrast, consider cloud-based elastic systems [6], which we briefly described in the introduction. It is clear that both–the number of resources allocated and the number of virtual machines running are natural numbers. As the domain of natural numbers is not dense, we can conclude that constraint systems used to model these cloud-based elastic systems do not satisfy the completion property.

Now we prove that for constraint systems of the form $(D, <, =)$ that satisfy the completion property, the CLTL realizability problem is decidable. This holds even when both players have look-ahead variables, so we don't need to treat future-blind variables separately. Hence, we set V^b to be empty and ignore gap functions in frames.

We reduce CLTL games to parity games on finite graphs, which are known to be decidable (see, e.g., [19]). In a CLTL game, `environment` chooses a valuation for EV, which we track in our finite graph by storing the positions of the new values relative to the values chosen in the previous rounds. We do this with partial frames, which we define next.

▶ **Definition 6** (Partial frames and compatibility). *For $s \geq 1$, a partial s-frame pf is a total pre-order $\leq_{pf}$ on the set of terms $T^a[s-2] \cup \{X^{s-1}y \mid y \in EV\}$. For $s \geq 0$, an s-frame f and an $(s+1)$-partial frame pf, the pair (f, pf) is one step compatible if for all $t_1, t_2 \in T^a[s-1]$, $t_1 \leq_f t_2$ iff $t_1 \leq_{pf} t_2$. For $s \geq 2$, an s-frame f and an s-partial frame pf, the pair (f, pf) is one-step compatible if for all $t_1, t_2 \in T^a[s-2]$, $Xt_1 \leq_f Xt_2$ iff $t_1 \leq_{pf} t_2$. For $s \geq 2$, an s-partial frame pf and an s-frame f, (pf, f) is one step compatible if for all $t_1, t_2 \in T^a[s-2] \cup \{X^{s-1}y \mid y \in EV\}$, $t_1 \leq_{pf} t_2$ iff $t_1 \leq_f t_2$.*

In the set of terms $T^a[s-2] \cup \{X^{s-1}y \mid y \in EV\}$ used in partial frames, the terms in the first set represent values chosen in the previous rounds and the terms in the second set represent values chosen by `environment` for EV in the current round.

Note that a partial s-frame is a total pre-order on the set of terms $T^a[s-2] \cup \{X^{s-1}y \mid y \in EV\}$ and an s-frame is a total pre-order on the set of terms $T^a[s-1]$. Let pf be an s-partial frame and let f be an s-frame such that (pf, f) is one-step compatible. Suppose $C_1 = \{t_1 = t_2 \mid t_1 \equiv_f t_2\} \cup \{t_1 < t_2 \mid t_1 <_f t_2\}$ and $C_2 = \{t_1 = t_2 \mid t_1 \equiv_{pf} t_2\} \cup \{t_1 < t_2 \mid t_1 <_{pf} t_2\}$. Clearly, C_2 is a subset of C_1 skipping all those constraints that contain `system` variables corresponding to the s^{th} position. If a finite sequence of mappings $(em_1 \oplus sm_1)...(em_{s-1} \oplus sm_{s-1})em_s$ satisfies the pre-order $\leq_{pf}$ then it satisfies the constraints in C_2. Since the constraint system satisfies the completion property, there must exist a `system` mapping sm_s for the `system` variables at position s such that the sequence of mappings $(em_1 \oplus sm_1)...(em_s \oplus sm_s)$ satisfies the constraints in C_1 and hence, also satisfies the pre-order $\leq_f$. Thus, we have the following proposition:

▶ **Proposition 7.** *Given $s \geq 1$, suppose $(em_1 \oplus sm_1)...(em_i \oplus sm_i)em$ is a sequence of mappings, where $em_1, \dots, em_i, em \in EM$, $sm_1, \dots, sm_i \in SM$, pf is the s-partial frame induced by em and the previous $(s-1)$ mappings in the sequence, and f is an s-frame such that (pf, f) is one-step compatible (where $i \geq s$). If the constraint system satisfies the completion property, then em can be extended to a mapping $em \oplus sm$ such that f is the s-frame associated with $em \oplus sm$ and the previous $(s-1)$ mappings in the sequence.*

We know that any LTL formula ϕ can be converted to an equivalent non-deterministic Büchi automaton with an exponential number of states in the size of ϕ in EXPTIME [30]. Now, every non-deterministic Büchi automaton B with n states can be converted to a deterministic parity automaton [15, Chapter 1] with number of states exponential in n and number of colours polynomial in n [21, Theorem 3.10]. Using these results, it is easy to see that given a CLTL formula ϕ, we can construct a deterministic parity automaton A_ϕ with set of states Q and with number of colours d, accepting the set of all sequences of frames that symbolically satisfy ϕ, such that $|Q|$ is double exponential in the size of ϕ and d is exponential in the size of ϕ. Now we design parity games to simulate CLTL games.

▶ **Definition 8.** *Let ϕ be the CLTL formula defining the winning condition for a CLTL game and k be its X-length. Let $\mathcal{F}$ denote the set of all s-frames for $s \in [0, k]$. Let A_ϕ be a deterministic parity automaton accepting the set of all sequences of frames that symbolically satisfy ϕ, with Q being the set of states, $q_I \in Q$ being the initial state and d being the number of colours. We define a parity game with* `environment` *vertices $V_e = \{(f, q_I) \mid f$ is an s-frame$, 0 \leq s \leq k\} \cup \{(f, q) \mid f$ is a $(k+1)$-frame$, q \in Q\}$. The set of* `system` *vertices is $V_s = \{(f, q_I, pf) \mid f$ is an s-frame$, 0 \leq s \leq k, pf$ is an $(s+1)$-partial frame$\} \cup \{(f, q, pf) \mid f$ is a $(k+1)$-frame$, pf$ is a $(k+1)$-partial frame$\}$. There is an edge from (f, q) to (f, q, pf) if (f, pf) is one-step compatible, f is an s-frame for some s and pf is a partial $\lceil s+1 \rceil_{(k+1)}$-frame. There is an edge from (f, q_I, pf) to (g, q_I) if (pf, g) is one step compatible and g is an s-frame for $s \in [1, k]$. There is an edge from (f, q, pf) to (g, q') if (pf, g) is one-step compatible, g is a $(k+1)$-frame and A_ϕ goes from q to q' on reading g. Vertices (f, q) and (f, q, pf) get the same colour as q in the parity automaton A_ϕ. The initial vertex is $(\perp, q_I)$, where $\perp$ is the trivial 0-frame.*

The edges of the parity game above are from V_s to V_e or vice-versa. They are designed such that q_I is the only state used for the first k rounds, where the frames will be of size at most k (this is because for the `system` to win in a play of the parity game generating a frame sequence ρ, we only require that the sequence $\boldsymbol{\rho[k, \infty)}$ symbolically satisfy ϕ, according to Lemma 5). For the first $(k+1)$ frame, an edge from a `system` vertex of the form (f, q_I, pf)

to an `environment` vertex of the form (g, q') is taken and from then on, we track the state of the parity automaton as it reads the sequence of frames contained in the sequence of vertices that are chosen by the players in the game.

▶ **Lemma 9.** *For a CLTL game over a constraint system satisfying the completion property with winning condition given by a formula ϕ, `system` has a winning strategy iff she has a positional winning strategy in the parity game given in Definition 8.*

Proof idea. For every play in the CLTL game, there is a corresponding play in the parity game, but the converse is not true in general, since only the order of terms are tracked in the parity game and not the actual values. For constraint systems satisfying the completion property, Proposition 7 implies that there exist valuations corresponding to all possible orderings of terms, so the converse is also true. ◀

▶ **Theorem 10.** *The CLTL realizability problem over constraint systems that satisfy the completion property is 2EXPTIME-complete.*

Proof. From Lemma 9, this is effectively equivalent to checking the existence of a winning strategy for `system` in a game. Now, checking if `system` has a winning strategy in the parity game (constructed using A_ϕ) can be achieved in $O(n^{\log d})$ time where n is the number of states in the game graph [9]. Now, by our construction, $n = |Q| \times |\mathcal{F}|$. We know, $|\mathcal{F}|$ is the number of total pre-orders on V, for which $2^{(k.|V|)^2}$ is a crude upper bound. This means that $|\mathcal{F}|$ is exponential in the size of ϕ and hence, overall we get a 2EXPTIME upper bound for our realizability problem. We also know that the realizability problem for LTL is complete for 2EXPTIME [23]. Thus, the CLTL realizability problem over constraint systems satisfying the completion property is also 2EXPTIME-complete. ◀

We know that a positional winning strategy in the parity game for a player, if it exists, can be implemented by a deterministic finite state transducer. Since $\mathcal{D}$ satisfies the completion property, consider a resource-bounded Turing machine M, which can, given an environment mapping *em* as described in Proposition 7, extend it to a mapping $em \oplus sm$ such that the order f imposed by the $em \oplus sm$ and the previous $s - 1$ mappings over the set of all terms extends the order pf imposed by *em* and the previous $s-1$ mappings. Now, for implementing the winning strategy for a player in a CLTL game, we use the deterministic finite state transducer corresponding to the parity game given in Definition 8. For every input of a partial frame pf by `environment` in a round, the transducer returns a frame f for `system` that extends pf. The transducer along with the machine M implements the winning strategy for `system` in a given CLTL game, if it exists.

Note that as we saw above, the constraint systems $(\mathbb{N}, =)$ and $(\mathbb{Z}, =)$ (with just equality and no linear order) also satisfy the completion property. So, it follows that the CLTL realizability problem over these constraint systems is also decidable.

5 Decidability of single-sided CLTL games over $(\mathbb{Z}, <, =)$

We consider games where `environment` has only future-blind variables, while the `system` has both future-blind and look-ahead variables. We call this single-sided CLTL games. So, in a single-sided game, $EV = EV^b$ and $SV = SV^b \cup SV^a$. Given a CLTL formula ϕ, the **single-sided realizability problem** is to check whether `system` has a winning strategy in the single-sided CLTL game whose winning condition is ϕ. We only consider the constraint system $(\mathbb{Z}, <, =)$ and show that the single-sided realizability problem is decidable over $(\mathbb{Z}, <, =)$. We do this in two stages. In the first stage, we reduce it to the problem of checking the non-emptiness of a set of trees satisfying certain properties. These trees represent `system` strategies. In the second stage, we show that non-emptiness can be checked using tree automata techniques.

Let G be the set of gap functions associated with mappings of the form $EV^b \to \mathbb{Z}$. For $s \geq 1$, an s-frame g and a function $gp \in G$, the pair (gp, g) is gap compatible if for all $x, y \in EV^b$, $gp(x) - gp(y) = gp_g(x, s-1) - gp_g(y, s-1)$. Intuitively, the gaps that frame g imposes between EV^b variables in its last position are the same as the gaps imposed by gp. We now have the following proposition (refer to the arXiv version of the paper for the proof).

▶ **Proposition 11** (gap compatibility). *For $s \geq 1$, an s-frame g and a function $gp \in G$, suppose the pair (gp, g) is gap compatible. If gp is the gap function associated with a mapping $em\colon EV^b \to \mathbb{Z}$, it can be extended to a mapping $em \oplus sm\colon V^b \to \mathbb{Z}$ such that $\lambda x.gp_g(x, s-1)$ is the gap function associated with $em \oplus sm$.*

Let ϕ be the CLTL formula defining the winning condition of a single-sided CLTL game and let k be its X-length. Let $\mathcal{F}$ be the set of all s-frames for $s \in [0, k]$. For technical convenience, we let $\mathcal{F}$ include the trivial 0-frame $\perp = (\leq_\perp, gp_\perp)$, where $\leq_\perp$ is the trivial total pre-order on the empty set and $gp_\perp$ is the trivial function on the empty domain.

▶ **Definition 12** (Winning strategy trees). *A strategy tree is a function $T\colon G^* \to \mathcal{F}$ such that for every node $\eta \in G^*$, $T(\eta)$ is a $\lceil|\eta|\rceil_{k+1}$-frame and for every $gp \in G$, $(T(\eta), T(\eta \cdot gp))$ is one-step compatible and $(gp, T(\eta \cdot gp))$ is gap compatible. A function L is said to be a labeling function if for every node $\eta \in G^*$, $L(\eta)\colon V \to \mathbb{Z}$ is a mapping of the variables in V. For an infinite path π in T, let $T(\pi)$ (resp. $L(\pi)$) denote the infinite sequence of frames (resp. mappings) labeling the nodes in π, except the root node ϵ. A winning strategy tree is a pair (T, L) such that T is a strategy tree and L is a labelling function satisfying the condition that for every infinite path π, $T(\pi) = \mu(L(\pi))$ and $T(\pi), k \models_s \phi$.*

The last condition above means that $T(\pi)$ is the symbolic model associated with the concrete model $L(\pi)$ and that it symbolically satisfies the formula ϕ.

Two concrete models may have the same symbolic model associated with them, if they differ only slightly, as explained next. Two concrete models σ_1, σ_2 are said to coincide on V^a if $\sigma_1(i) \upharpoonright V^a = \sigma_2(i) \upharpoonright V^a$ for all $i \geq 0$. They are said to coincide on V^b up to gap functions if for every $i \geq 0$, the same gap function is associated with $\sigma_1(i) \upharpoonright V^b$ and $\sigma_2(i) \upharpoonright V^b$. The following result follows directly from definitions.

▶ **Proposition 13** (similar concrete models have the same symbolic model). *If two concrete models coincide on V^a and they coincide on V^b up to gap functions, then they have the same symbolic model associated with them.*

The following result accomplishes the first stage of the decidability proof, reducing the existence of winning strategies to non-emptiness of a set of trees. A detailed proof of this result can be found on the arXiv version of the paper with the same title.

▶ **Lemma 14** (strategy to tree). `System` *has a winning strategy in the single-sided CLTL game with wining condition ϕ iff there exists a winning strategy tree.*

Proof idea. If `environment` chooses a mapping $em\colon EV^b \to \mathbb{Z}$ in the CLTL game, the corresponding choice in the tree T is to go to the child gp, the gap function associated with em. `System` responds with the mapping $L(gp) \upharpoonright SV^a$ for the look-ahead variables. For the future-blind variables SV^b, `system` chooses a mapping that ensures compatibility with the frame $T(gp)$. This will ensure that `system`'s response and L coincide on V^a and coincide on V^b up to gap functions, so Proposition 13 ensures that both have the same symbolic model. The symbolic model symbolically satisfies ϕ by definition of wining strategy trees and Lemma 5 implies that the concrete model satisfies ϕ. ◀

Given a tree $G^* \to \mathcal{F}$, a tree automaton over finite alphabets can check whether it is a strategy tree or not, by allowing transitions only between one-step and gap compatible frames. However, to check whether it is a winning strategy tree, we need to check whether there exists a labeling function L, which is harder. One way to check the existence of such a labeling function is to start labeling at the root and inductively extend to children. Suppose there are two variables x, y at some node and we have to label them with integers. There may be many variables in other nodes whose labels should be strictly between those of x, y in the current node. So our labels for x, y in the current node should leave a gap large enough to accommodate others that are supposed to be in between. Next we introduce some orderings we use to formalize this.

A node variable in a strategy tree T is a pair (η, x) where η is a node and $x \in V^a$ is a look-ahead variable. The tree induces an order on node variables as follows. Suppose η is a node, $T(\eta)$ is an s-frame for some s and η_a is an ancestor of η such that the difference in height $h = |\eta| - |\eta_a|$ between the descendant and ancestor is at most $s-1$. For look-ahead variables $x, y \in V^a$, recall that the term $X^{s-1}x$ represents the variable x in the last position of the frame $T(\eta)$, and $X^{s-1-h}y$ represents the variable y at h positions before the last one. We say $(\eta, x) \sqsubseteq_T (\eta_a, y)$ (resp. $(\eta_a, y) \sqsubseteq_T (\eta, x)$) if $X^{s-1}x \leq_{T(\eta)} X^{s-1-h}y$ (resp. $X^{s-1-h}y \leq_{T(\eta)} X^{s-1}x$). In other words, for the variables and positions captured in the frame $T(\eta)$, $\sqsubseteq_T$ is same as the total pre-order $\leq_{T(\eta)}$. We define $(\eta, x) \sqsubset_T (\eta_a, y)$ (resp. $(\eta_a, y) \sqsubset_T (\eta, x)$) if $(\eta, x) \sqsubseteq_T (\eta_a, y)$ and $(\eta_a, y) \not\sqsubseteq_T (\eta, x)$ (resp. $(\eta_a, y) \sqsubseteq_T (\eta, x)$ and $(\eta, x) \not\sqsubseteq_T (\eta_a, y)$). We define $\sqsubseteq_T^*$ to be the reflexive transitive closure of $\sqsubseteq_T$ and $\sqsubset_T^+$ to be the transitive closure of $\sqsubset_T$. Note that $\sqsubseteq_T^*$ and $\sqsubset_T^+$ can compare node variables that are in different branches of the tree also, though they are not total orders. We write $(\eta_1, x) \sqsubseteq_T^* (\eta_2, y)$ (resp, $(\eta_1, x) \sqsubset_T^+ (\eta_2, y)$) equivalently as $(\eta_2, y) \sqsupseteq_T^* (\eta_1, x)$ (resp. $(\eta_1, x) \sqsupset_T^+ (\eta_2, y)$). By definition, $(\eta_1, x) \sqsubset_T^+ (\eta_2, y)$ (resp.$(\eta_2, y) \sqsubset_T^+ (\eta_1, x)$) if $(\eta_1, x) \sqsubseteq_T^* (\eta_2, y)$ and $(\eta_2, y) \not\sqsubseteq_T^* (\eta_1, x)$ (resp. $(\eta_2, y) \sqsubseteq_T^* (\eta_1, x)$ and $(\eta_1, x) \not\sqsubseteq_T^* (\eta_2, y)$). $\sqsubset^+$ is irreflexive and transitive.

▶ **Definition 15** (Bounded chain strategy trees)**.** *Suppose T is a strategy tree, η, η' are two nodes and $x, y \in V^a$ are look-ahead variables such that $(\eta, x) \sqsubset_T^+ (\eta', y)$. A chain between (η, x) and (η', y) is a sequence $(\eta_1, x_1)(\eta_2, x_2) \cdots (\eta_r, x_r)$ such that $(\eta, x) \sqsubset_T^+ (\eta_1, x_1) \sqsubset_T^+ (\eta_2, x_2) \sqsubset_T^+ \cdots \sqsubset_T^+ (\eta_r, x_r) \sqsubset_T^+ (\eta', y)$. We say r is the length of the chain. The strategy tree T is said to have bounded chains if for any two node variables (η, x) and (η', y), there is a bound N such that any chain between (η, x) and (η', y) is of length at most N.*

▶ **Lemma 16.** *A strategy tree T has a labeling function L such that (T, L) is a winning strategy tree iff T has bounded chains.*

The above lemma characterizes those strategy trees that are winning strategy trees. This is the main technical difference between CLTL games and games with register automata specifications [28, 14]. Since register automata can compare values that are arbitrarily far apart, the corresponding characterization of symbolic structures that have associated concrete structures is more involved compared to Lemma 16 above.

Detecting unbounded chains is still difficult for tree automata – to find longer chains, we may have to examine longer paths. This difficulty can be overcome if we can show that longer chains can be obtained by repeatedly joining shorter ones. We now introduce some notation and results to formalize this. For a node η and an ancestor η_a, $\hat{T}(\eta_a, \eta)$ is the sequence of frames $T(\eta_a) \cdots T(\eta)$ labeling the path from η_a to η. A node η_1 is said to occur within the influence of (η_a, η) if η_1 occurs between η_a and η or η_1 is an ancestor of η_a and $|\eta_a| - |\eta_1| \leq s - 1$, where s is the size of the frame $T(\eta_a)$. The following result follows directly from definitions.

▶ **Proposition 17** (Identical paths induce identical orders). *Suppose nodes η, η' and their ancestors η_a, η'_a respectively are such that $\hat{T}(\eta_a, \eta) = \hat{T}(\eta'_a, \eta')$. Suppose η_1, η_2 occur within the influence of (η_a, η) and η'_1, η'_2 occur within the influence of (η'_a, η') such that $|\eta| - |\eta_1| = |\eta'| - |\eta'_1|$ and $|\eta| - |\eta_2| = |\eta'| - |\eta'_2|$. For any look-ahead variables x, y, $(\eta_1, x) \sqsubset^*_T (\eta_2, y)$ (resp. $(\eta_1, x) \sqsubseteq_T (\eta_2, y)$) iff $(\eta'_1, x) \sqsubset^*_T (\eta'_2, y)$ (resp. $(\eta'_1, x) \sqsubseteq_T (\eta'_2, y)$).*

For a node η, the subtree T_η rooted at η is such that for all η', $T_\eta(\eta') = T(\eta \cdot \eta')$. A tree T is called *regular* if the set $\{T_\eta \mid \eta \in G^*\}$ is finite, i.e., there are only finitely many subtrees up to isomorphism. Two nodes η, η' are said to be isomorphic if $T_\eta = T_{\eta'}$.

▶ **Lemma 18** (Pumping chains in regular trees). *Suppose T is a regular tree. Then T has unbounded chains iff there exists an infinite path containing two infinite sequences $(\eta_1, x), (\eta_2, x), (\eta_3, x) \ldots$ and $(\eta'_1, y), (\eta'_2, y), (\eta'_3, y) \ldots$ such that η_{i+1} (resp. η'_{i+1}) is a descendant of η_i (resp. η'_i) for all $i \geq 1$ and satisfy one of the following conditions.*

$$\begin{array}{ccccccccccccccc}
(\eta_1, x) & \sqsubset^+_T & (\eta_2, x) & \sqsubset^+_T & (\eta_3, x) \sqsubset^+_T \cdots & & & (\eta_1, x) & \sqsupset^+_T & (\eta_2, x) & \sqsupset^+_T & (\eta_3, x) \sqsupset^+_T \cdots \\
\rotatebox{-90}{$\sqsubseteq_T$} & & \rotatebox{-90}{$\sqsubseteq_T$} & & \rotatebox{-90}{$\sqsubseteq_T$} & & \textit{or} & \rotatebox{90}{$\sqsubseteq_T$} & & \rotatebox{90}{$\sqsubseteq_T$} & & \rotatebox{90}{$\sqsubseteq_T$} \\
(\eta'_1, y) & \sqsupset^*_T & (\eta'_2, y) & \sqsupset^*_T & (\eta'_3, y) \sqsupset^*_T \cdots & & & (\eta'_1, y) & \sqsubset^*_T & (\eta'_2, y) & \sqsubset^*_T & (\eta'_3, y) \sqsubset^*_T \cdots
\end{array}$$

Proof idea. We can choose a chain that is long enough to contain two isomorphic nodes. The path between them can be repeated infinitely. Proposition 17 will imply that this infinite path contains an infinite chain as required. ◀

Lemma 18 says that if a regular tree has unbounded chains, it will have an infinite path containing an infinite chain. The infinite sequence of the first (resp. second) kind given in Lemma 18 is called an infinite forward (resp. backward) chain. Now we design a tree automaton $\mathcal{A}_\phi$ whose language $\mathcal{L}(\mathcal{A}_\phi)$ is an approximation of the set $\mathcal{T} = \{T \mid \exists L, (T, L)$ is a winning strategy tree$\}$ such that $\mathcal{L}(\mathcal{A}_\phi)$ is non-empty iff $\mathcal{T}$ is. Hence, the single-sided CLTL realizability problem is equivalent to checking the non-emptiness of $\mathcal{L}(\mathcal{A}_\phi)$. The tree automaton $\mathcal{A}_\phi$ is the intersection of three automata $\mathcal{A}^{\text{str}}_\phi$, $\mathcal{A}^{\text{symb}}_\phi$ and $\mathcal{A}^{\text{chain}}_\phi$, all of which read $|G|$-ary trees labeled with letters from $\mathcal{F}$. The automaton $\mathcal{A}^{\text{str}}_\phi$ accepts the set of all strategy trees, $\mathcal{A}^{\text{symb}}_\phi$ accepts the set of all trees each of whose paths symbolically satisfies the formula ϕ and $\mathcal{A}^{\text{chain}}_\phi$ accepts the set of all trees that do not have any infinite forward or backward chains. Construction of these automata are explained in detail in Appendix B.

▶ **Lemma 19.** *The* `system` *player has a winning strategy in the single-sided CLTL$(\mathbb{Z}, <, =)$ game with winning condition ϕ iff $\mathcal{L}(\mathcal{A}_\phi)$ is non-empty.*

Proof. Suppose there is a winning strategy for the `system` player in single-sided CLTL$(\mathbb{Z}, <, =)$ game with winning condition ϕ. By Lemma 14, there exists a winning strategy tree, say (T, L). Since, T is a strategy tree, $T \in \mathcal{L}(\mathcal{A}^{\text{str}})$. We know that every branch of T must symbolically satisfy ϕ and hence, $T \in \mathcal{L}(\mathcal{A}^{\text{symb}}_\phi)$. Further, since T has the labelling function L, Lemma 16 implies that T has bounded chains and thus, it cannot have any infinite forward or backward chains. So $T \in \mathcal{L}(\mathcal{A}^{\text{chain}})$. Thus, $T \in \mathcal{L}(\mathcal{A}_\phi)$.

Conversely, suppose $\mathcal{A}_\phi$ accepts a tree T. It is known that if the language of a tree automaton is non-empty, it contains a regular tree [20, Corollary 8.20]. Although this result holds for tree automata that read infinite binary trees as inputs, the proofs can be suitably modified to work for tree automata that read $|G|$-ary trees. Hence we can conclude that $\mathcal{A}_\phi$ must accept a regular tree T'. Since, $T' \in \mathcal{L}(\mathcal{A}_\phi)$, every branch of T' must symbolically satisfy ϕ, T' must be a strategy tree and it cannot have any infinite forward or backward chains. Thus, by Lemma 18, T' must have bounded chains and hence by Lemma 16, T' must have a labelling function L' such that (T', L') is a winning strategy tree. Hence, by Lemma 14 the `system` player has a winning strategy in the single-sided CLTL$(\mathbb{Z}, <, =)$ game. ◀

Note that $\mathcal{L}(\mathcal{A}_\phi)$ is not equal to the set $\mathcal{T} = \{T \mid \exists L, (T, L)$ is a winning strategy tree$\}$ in general. As seen in the above proof, we can only guarantee that the regular trees in $\mathcal{L}(\mathcal{A}_\phi)$ are in $\mathcal{T}$. The non-regular trees in $\mathcal{L}(\mathcal{A}_\phi)$ need not be in $\mathcal{T}$.

Using the previous lemma, we get the following decidability result.

▶ **Theorem 20.** *The single-sided realizability problem for CLTL over* $(\mathbb{Z}, <, =)$ *is 2EXPTIME -complete.*

Proof. Given a formula ϕ, Lemma 19 implies that it is enough to construct the tree automaton $\mathcal{A}_\phi$ and check it for non-emptiness. From the description of the construction in Appendix B, we can see that $\mathcal{A}_\phi^{\text{str}}$, $\mathcal{A}_\phi^{\text{symb}}$ and $\mathcal{A}_\phi^{\text{chain}}$ can be constructed in 2EXPTIME in the size of ϕ. Thus, the automaton $\mathcal{A}_\phi$ can be constructed in 2EXPTIME. Now, checking non-emptiness of a parity tree automaton is decidable and the upper bound stated in [20, Corollary 8.22 (1)] implies that the single-sided realizability problem for CLTL over $(\mathbb{Z}, <, =)$ is in 2EXPTIME. Now, the realizability problem for LTL is 2EXPTIME-complete [23] and hence, the single-sided realizability problem for CLTL over $(\mathbb{Z}, <, =)$ must also be 2EXPTIME-complete. ◀

6 Discussion and Future Work

We have seen in this paper that the CLTL realizability problem is decidable over domains satisfying completion property and that the single-sided CLTL realizability problem is decidable over integers with linear order and equality. But both these problems have a high complexity (both are 2EXPTIME-complete). It would be interesting to see if there are expressive fragments of CLTL with lower complexity, like the fragments of LTL studied in [22], which work on practical examples.

We believe that single-sided CLTL games over the domain of natural numbers $(\mathbb{N}, <, =)$ are also decidable. In [13], the authors extend the automata-characterization for the satisfiability problem for CLTL over the integer domain to the domain of natural numbers. A similar extension of the tree-automata characterization for the single-sided games over integers to one for single-sided games over the naturals seems possible, although the details need to be worked out.

Despite the decidability result that we have for the single-sided CLTL games over integers, the language of the tree automaton that we construct in this paper is an approximation of the set of all winning strategy trees. We do not have a machine-theoretic representation for winning strategies yet, and this is an interesting direction for future exploration.

References

1 Parosh Aziz Abdulla, Ahmed Bouajjani, and Julien d'Orso. Deciding monotonic games. In *International Workshop on Computer Science Logic*, pages 1–14. Springer, 2003.

2 Parosh Aziz Abdulla, Richard Mayr, Arnaud Sangnier, and Jeremy Sproston. Solving parity games on integer vectors. In *International Conference on Concurrency Theory*, pages 106–120. Springer, 2013.

3 Rajeev Alur and Thomas A Henzinger. A really temporal logic. *Journal of the ACM (JACM)*, 41(1):181–203, 1994.

4 Philippe Balbiani and Condotta Jean-François. Computational complexity of propositional linear temporal logics based on qualitative spatial or temporal reasoning. In *International Workshop on Frontiers of Combining Systems*, pages 162–176. Springer, 2002.

5 Béatrice Bérard, Benedikt Bollig, Mathieu Lehaut, and Nathalie Sznajder. Parameterized synthesis for fragments of first-order logic over data words. In *FoSSaCS*, pages 97–118, 2020.

6 Marcello M Bersani, Domenico Bianculli, Schahram Dustdar, Alessio Gambi, Carlo Ghezzi, and Srđan Krstić. Towards the formalization of properties of cloud-based elastic systems. In *proceedings of the 6th international workshop on principles of engineering service-oriented and cloud systems*, pages 38–47, 2014.

7 Ashwin Bhaskar and M. Praveen. Realizability problem for constraint ltl, 2022. `doi:10.48550/ARXIV.2207.06708`.

8 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic (TOCL)*, 12(4):1–26, 2011.

9 Cristian S Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, 51(2):STOC17–152, 2020.

10 Alonzo Church. Logic, arithmetic, and automata. *Journal of Symbolic Logic*, 29(4), 1964.

11 Stephane Demri, Deepak D'Souza, and Régis Gascon. Temporal logics of repeating values. *Journal of Logic and Computation*, 22, October 2012. `doi:10.1093/logcom/exr013`.

12 Stéphane Demri and Ranko Lazić. Ltl with the freeze quantifier and register automata. *ACM Trans. Comput. Logic*, 10(3), April 2009. `doi:10.1145/1507244.1507246`.

13 Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint ltl. *Information and Computation*, 205(3):380–415, 2007. `doi:10.1016/j.ic.2006.09.006`.

14 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over linearly ordered data domains. *arXiv preprint arXiv:2004.12141*, 2020.

15 Erich Gradel and Wolfgang Thomas. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

16 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

17 Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.

18 Orna Kupferman, Nir Piterman, and Moshe Vardi. From liveness to promptness. *Formal Methods in System Design*, 34, April 2009. `doi:10.1007/s10703-009-0067-z`.

19 René Mazala. *Infinite Games*, pages 23–38. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. `doi:10.1007/3-540-36387-4_2`.

20 Frank Nießner. Nondeterministic tree automata. In *Automata Logics, and Infinite games*, pages 135–152. Springer, 2002.

21 Nir Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3, 2007.

22 Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

23 Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In *International Colloquium on Automata, Languages, and Programming*, pages 652–671. Springer, 1989.

24 M Praveen, Diego Figueira, and Stephane Demri. Reasoning about data repetitions with counter systems. *Logical Methods in Computer Science*, 12, 2016.

25 M Praveen, Anirban Majumdar, and Diego Figueira. Playing with repetitions in data words using energy games. *Logical Methods in Computer Science*, 16, 2020.

26 Alexander Rabinovich. Decidable extensions of church's problem. In *International Workshop on Computer Science Logic*, pages 424–439. Springer, 2009.

27 Jean-François Raskin. An introduction to hybrid automata. In *Handbook of networked and embedded control systems*, pages 491–517. Springer, 2005.

28 Pierre-Alain Reynier, Emmanuel Filiot, and Léo Exibard. Synthesis of data word transducers. *Logical Methods in Computer Science*, 17, 2021.

29 A Prasad Sistla, Moshe Y Vardi, and Pierre Wolper. The complementation problem for büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2-3):217–237, 1987.

30 Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331. IEEE Computer Society, 1986.

A Details of Section 4

Proof of Lemma 9. ($\Rightarrow$) Suppose `system` has a winning strategy st in the CLTL game. We show that `system` has a winning strategy in the parity game. Plays in the parity game are of the form $(\bot, q_I)(\bot, q_I, pf_1)(f_1, q_1)(f_1, q_1, pf_2)(f_2, q_2)\cdots(f_i, q_i, pf_{i+1})(f_{i+1}, q_{i+1})\cdots$, where (f_i, pf_{i+1}) and (pf_{i+1}, f_{i+i}) are one-step compatible for all i. For any such play π, let $\pi \restriction i$ be $(\bot, q_I)(\bot, q_I, pf_1)(f_1, q_1)(f_1, q_1, pf_2)(f_2, q_2)\cdots(f_i, q_i, pf_{i+1})$. Let $\Pi = \{\pi \restriction i \mid \pi \text{ is a play in the parity game}, i \geq 0\}$. We will show the existence of a function $st_p \colon \Pi \to V_e \times EM \times SM$ satisfying some properties. Such a function can be used as a strategy by `system` in the parity game: for a play $\pi \restriction i$, `system`'s response (f_{i+1}, q_{i+1}) is given by st_p, i.e., $st_p(\pi \restriction i) = ((f_{i+1}, q_{i+1}), em_{i+1}, sm_{i+1})$. For such plays that `system` plays according st_p, let frames$(\pi \restriction i)$ be the symbolic model $f_1 f_2 \cdots f_{i+1}$ and let maps$(\pi \restriction i)$ be the concrete model $(em_1 \oplus sm_1)(em_2 \oplus sm_2)\cdots(em_{i+1} \oplus sm_{i+1})$.

We will show that there is a function st_p such that for all plays π that `system` plays according to st_p and all $i \geq 0$, maps$(\pi \restriction i)$ is a concrete model resulting from `system` playing the CLTL game according to st and frames$(\pi \restriction i) = \mu(\text{maps}(\pi \restriction i))$. We will define such a function st_p by induction on i. We assume this has been done for i and show how to extend to $i + 1$. We have $\pi \restriction (i + 1) = (\bot, q_I)(\bot, q_I, pf_1)(f_1, q_1)(f_1, q_1, pf_2)(f_2, q_2)\cdots(f_i, q_i, pf_{i+1})(f_{i+1}, q_{i+1})$ $(f_{i+1}, q_{i+1}, pf_{i+2})$. By induction hypothesis, $f_1 f_2 \cdots f_{i+1} = \mu(\text{maps}(\pi \restriction i))$. Since the constraint system satisfies the completion property and (f_{i+1}, pf_{i+2}) is one-step compatible, by Proposition 7, there is a mapping $em \colon EV \to D$ such that the symbolic model induced by maps$(\pi \restriction i) \cdot em$ is $f_1 f_2 \cdots f_{i+1} \cdot pf_{i+2}$. Let $sm \colon SV \to D = st(\text{maps}(\pi \restriction i) \cdot em)$ be `system`'s response in the CLTL game according to st. Let f_{i+2} be the frame such that $f_1 f_2 \cdots f_{i+1} f_{i+2} = \mu(\text{maps}(\pi \restriction i) \cdot (em \oplus sm))$. Set $st_p(\pi \restriction (i+1))$ to be $((f_{i+2}, q_{i+2}), em, sm)$, where q_{i+2} is the state A_ϕ reaches after reading f_{i+2} in state q_{i+1}. Now, maps$(\pi \restriction (i+1))$ is a concrete model resulting from `system` playing the CLTL game according to st and frames$(\pi \restriction (i+1)) = \mu(\text{maps}(\pi \restriction (i+1)))$, as required for the inductive construction.

Let π be any infinite play in the parity game that `system` plays according to st_p. Then maps(π) is a concrete model resulting from `system` playing the CLTL game according to st and frames$(\pi) = \mu(\text{maps}(\pi))$. Since st is a winning strategy for `system`, maps$(\pi), 0 \models \phi$. We infer from Lemma 5 that frames(maps$(\pi)), k \models_s \phi$. Hence, the sequence of states $q_I, q_1, q_2, \ldots$ contained in the sequence of vertices that are visited in π satisfy the parity condition of A_ϕ. Hence, π itself satisfies the parity condition and hence `system` wins π. Hence, st_p is a winning strategy for `system` in the parity game.

($\Leftarrow$) Suppose st_p is a positional strategy for `system` in the parity game. We will show that `system` has a winning strategy st in the CLTL game. We will define st by induction on the number of rounds played. For the base case, suppose `environment` starts by choosing a mapping $em_1 \colon EV \to D$. In the parity game, let `environment` go to the vertex $(\bot, q_I, pf_1)$ in the first round, where pf_1 is the 1-partial frame associated with em_1. Let $(f_1, q_1) = st_p((\bot, q_I, pf_1))$ be `system`'s response according to st_p. Since (pf_1, f_1) is one-step compatible and the constraint system satisfies the completion property, by Proposition 7, em_1 can be extended to a mapping $em_1 \oplus sm_1 \colon V \to D$ such that f_1 is the frame associated with $em_1 \oplus sm_1$. Set $st(em_1)$ to be sm_1. After i rounds of the CLTL game, suppose $(em_1 \oplus sm_1) \cdots (em_i \oplus sm_i)$ is the resulting concrete model and let $(\bot, q_I)(\bot, q_I, pf_1)(f_1, q_1) \cdots (f_i, q_i)$ be the corresponding play in the parity game. Suppose `environment` chooses em_{i+1} in the next round. Let $pf_{i+1}, f_{i+1}, q_{i+1}, sm_{i+1}$ be obtained similarly as in the base case. Set $st((em_1 \oplus sm_1) \cdots (em_i \oplus sm_i) \cdot em_{i+1})$ to be sm_{i+1}.

Suppose $(em_1 \oplus sm_1)(em_2 \oplus sm_2) \cdots$ is an infinite play in the CLTL game that `system` plays according to st. There is a play $(\bot, q_I)(\bot, q_I, pf_1)(f_1, q_1)(f_1, q_1, pf_2)(f_2, q_2) \cdots$ in the parity game that is winning for `system`. This satisfies the parity condition, hence A_ϕ accepts the symbolic model $f_1 f_2 \cdots$. The symbolic model $f_1 f_2 \cdots$ is the one associated with $(em_1 \oplus sm_1)(em_2 \oplus sm_2) \cdots$ by construction of st, so Lemma 5 implies that $(em_1 \oplus sm_1)(em_2 \oplus sm_2) \cdots, 0 \models \phi$. Hence, st is a winning strategy for `system` in the CLTL game. ◀

B Details of Section 5

Proof of Lemma 16. ($\Rightarrow$) Suppose T has a labeling function L such that (T, L) is a winning strategy tree. Since for every infinite path π, $T(\pi) = \mu(L(\pi))$, L should respect the relation $\sqsubset_T^+$, i.e., if $(\eta, x) \sqsubset_T^+ (\eta', y)$, then $L(\eta)(x) < L(\eta')(y)$. Hence, any chain between (η, x) and (η', y) cannot be longer than $L(\eta')(y) - L(\eta)(x)$.

($\Leftarrow$) Suppose T has bounded chains. We construct a labeling function L such that (T, L) is a winning strategy tree. At every node η, we choose mappings for future-blind variables V^b such that the gap function associated with $L(\eta) \upharpoonright V^b$ is $gp_{T(\eta)}$. These choices can be done independently for every node. For look-ahead variables, we construct L for every node by induction on depth of the node such that for any node variables $(\eta, x), (\eta', y)$ such that $(\eta, x) \sqsubset_T^+ (\eta', y)$ and $L(\eta), L(\eta')$ have been constructed, $L(\eta')(y) - L(\eta)(x)$ is at least as large as the length of the longest chain between (η, x) and (η', y). For the base case $\eta = \epsilon$, let $L(\eta)$ be the trivial mapping on the empty domain.

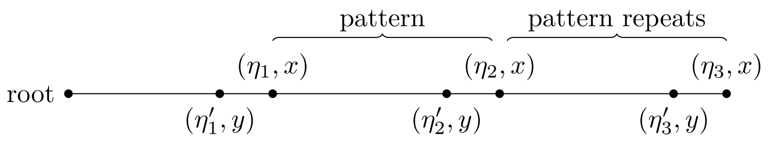
For the induction step, consider a node η. Let $(\eta, x_0), (\eta, x_1), \ldots$ be the node variables from η and let $(\eta_1, y_1), (\eta_2, y_2), \ldots$ be the node variables from all the ancestors of η. Arrange them in ascending order according to $\sqsubset_T^*$. In this arrangement, suppose $(\eta_i, y_i)(\eta, x_j)(\eta, x_{j+1}) \cdots (\eta, x_l)(\eta_{i+1}, y_{i+1})$ is a contiguous sequence of node variables from η surrounded by ancestor node variables (η_i, y_i) and (η_{i+1}, y_{i+1}). Set $L(\eta)(x_j)$ to be the sum of $L(\eta_i)(y_i)$ and the length of the longest chain between (η_i, y_i) and (η, x_j). Set $L(\eta)(x_{j+1})$ to be the sum of $L(\eta)(x_j)$ and the length of the longest chain between (η, x_j) and (η, x_{j+1}). Continue this way till (η, x_l). The value set for $L(\eta)(x_l)$ will be less than $L(\eta_{i+1})(y_{i+1})$ minus the length of the longest chain between $L(\eta)(x_l)$ and (η_{i+1}, y_{i+1}), since by induction hypothesis, $L(\eta_{i+1})(y_{i+1}) - L(\eta_i)(y_i)$ is large enough to accommodate the longest chain between (η_i, y_i) and (η_{i+1}, y_{i+1}) (note that any chain between (η, x_j) and (η, x_{j+1}) can be concatenated with any chain between (η, x_{j+1}) and (η, x_{j+2}) and so on to form a chain between (η_i, y_i) and (η_{i+1}, y_{i+1})). This way, all contiguous sequence of node variables from η can be mapped satisfactorily. This completes the induction step and hence the proof. ◀

Proof of Lemma 18. ($\Leftarrow$) We consider the first case; the other case is similar. Since $(\eta_i, x) \sqsubseteq_T (\eta'_i, y) \sqsubset^*_T (\eta'_{i-1}, y) \sqsubset^*_T \cdots \sqsubset^*_T (\eta'_1, y)$ for all $i \geq 1$, we have $(\eta_i, x) \sqsubset^*_T (\eta'_1, y)$. Hence, $(\eta_1, x) \sqsubset^+_T (\eta_2, x) \sqsubset^+_T \cdots \sqsubset^+_T (\eta_i, x) \sqsubset^*_T (\eta'_1, y)$ for all $i \geq 1$, demonstrating that there are chains of unbounded lengths between (η_1, x) and (η'_1, y).

($\Rightarrow$) We show the existence of a short segment that can be repeated arbitrarily many times to get the required infinite path. We show that there are node variables along a path satisfying the following conditions:

1. $$\begin{tikzcd} (\eta_1, x) \arrow[r, phantom, "\sqsubset^+_T"] \arrow[d, phantom, "\sqsubseteq_T"] & (\eta_2, x) \arrow[d, phantom, "\sqsubseteq_T"] \\ (\eta'_1, y) \arrow[r, phantom, "\sqsupset^*_T"] & (\eta'_2, y) \end{tikzcd} \quad \text{or} \quad \begin{tikzcd} (\eta_1, x) \arrow[r, phantom, "\sqsupset^+_T"] \arrow[d, phantom, "\sqsupseteq_T"] & (\eta_2, x) \arrow[d, phantom, "\sqsupseteq_T"] \\ (\eta'_1, y) \arrow[r, phantom, "\sqsubset^*_T"] & (\eta'_2, y) \end{tikzcd},$$
2. the nodes are arranged as $\eta'_1, \eta_1, \eta'_2, \eta_2$ in ascending order of depth, $|\eta'_1| > k$,
3. η_1, η_2 are isomorphic, η'_1, η'_2 are isomorphic and $|\eta_1| - |\eta'_1| = |\eta_2| - |\eta'_2| \leq k$.

The node variables mentioned above are as shown below.

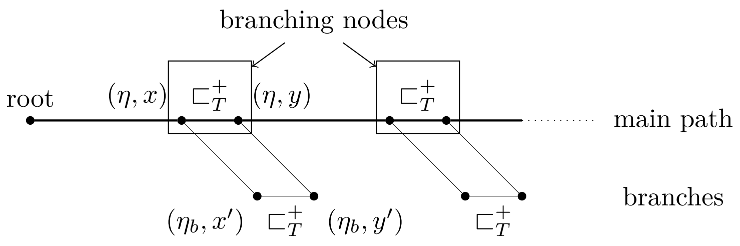


We first prove that the existence of such nodes is sufficient. Since η_1, η_2 are isomorphic, for any sequence of frames starting from η_1, the same sequence also starts from η_2. Hence there is a descendant η_3 of η_2 such that η_2, η_3 are isomorphic and $\hat{T}(\eta_1, \eta_2) = \hat{T}(\eta_2, \eta_3)$. The nodes $\eta'_1, \eta_1, \eta'_2, \eta_2$ occur within the influence of (η_1, η_2) and the nodes $\eta'_2, \eta_2, \eta'_3, \eta_3$ occur within the influence of (η_2, η_3). In the first case in the first condition above, $(\eta_1, x) \sqsubset^+_T (\eta_2, x) \sqsubseteq_T (\eta'_2, y) \sqsubset^*_T (\eta'_1, y)$ and Proposition 17 implies that $(\eta_2, x) \sqsubset^+_T (\eta_3, x) \sqsubseteq_T (\eta'_3, y) \sqsubset^*_T (\eta'_2, y)$. This pattern can be repeated arbitrarily many times, proving that there are node variables as stated in the first case of the lemma. The other case is similar.

Now we will show the existence of the short segment as claimed above. Since T is regular, the number of non-isomorphic subtrees of T is finite, say κ. Let $N = \kappa^2 |V^a|^2$. We will show subsequently that there is a chain of the form $(\eta, x_1) \sqsubset^+_T (\eta_1, y_1) \sqsubset^+_T (\eta_2, y_2) \sqsubset^+_T \cdots \sqsubset^+_T (\eta_{N+2}, y_{N+2}) \sqsubset^*_T (\eta', x_2)$ or $(\eta, x_1) \sqsupset^+_T (\eta_1, y_1) \sqsupset^+_T (\eta_2, y_2) \sqsupset^+_T \cdots \sqsupset^+_T (\eta_{N+2}, y_{N+2}) \sqsupset^*_T (\eta', x_2)$, where η_1 is a descendant of both η and η' of depth at least $(k+1)$ more than both η and η' and η_{i+1} is a descendant of η_i of depth at least $(k+1)$ more than η_i for all $i \in [1, N+1]$ (we call such chains straight segments). We will only consider the first case here; the other case is similar. Now $(\eta_{N+2}, y_{N+2}) \sqsubset^*_T (\eta', x_2)$ and η_{N+2} is a deep descendant of η' with $\eta_1, \ldots, \eta_{N+1}$ (which are themselves at least $(k+1)$ positions apart from each other) in between. Recall that $\sqsubset^*_T$ is the transitive closure of $\sqsubseteq_T$ and $\sqsubseteq_T$ holds only between node variables that are at most k positions apart. Hence, there must be intermediate node variables between $(\eta_{N+2}, y_{N+2}), (\eta', x_2)$ so that $(\eta_{N+2}, y_{N+2}) \sqsubset^*_T (\eta', x_2)$. For every $i \in [1, N+1]$, there must be some intermediate node variable (η'_i, y'_i) such that η'_i is an ancestor of η_i, $|\eta_i| - |\eta'_i| \leq k$ and $(\eta_{N+2}, y_{N+2}) \sqsubset^*_T (\eta'_i, y'_i) \sqsubset^*_T (\eta', x_2)$. Since $|\eta_i| - |\eta'_i| \leq k$, either $(\eta_i, y_i) \sqsubseteq_T (\eta'_i, y'_i)$ or $(\eta'_i, y'_i) \sqsubseteq_T (\eta_i, y_i)$ (the frame $T(\eta_i)$ spans η'_i also; hence the frame imposes an order between the node variables). If $(\eta'_i, y'_i) \sqsubseteq_T (\eta_i, y_i)$, then $(\eta_i, y_i) \sqsubset^+_T (\eta_{N+2}, y_{N+2}) \sqsubset^*_T (\eta'_i, y'_i) \sqsubseteq_T (\eta_i, y_i)$ implies that $(\eta_i, y_i) \sqsubset^+_T (\eta_i, y_i)$, contradicting the fact that $\sqsubset^+_T$ is irreflexive. Hence, $(\eta_i, y_i) \sqsubseteq_T (\eta'_i, y'_i)$. Consider the sequence $(\eta_1, y_1), (\eta'_1, y'_1), (\eta_2, y_2), (\eta'_2, y'_2), \ldots, (\eta_{N+1}, y_{N+1}), (\eta'_{N+1}, y'_{N+1})$. Since $N = \kappa^2 |V^a|^2$, there are i, j such that η_i (resp. η'_i) is isomorphic to η_j (resp. η'_j), $y_i = y_j$ and $y'_i = y'_j$. The node variables $(\eta_i, y_i), (\eta_j, y_i), (\eta'_i, y'_i), (\eta'_j, y'_i)$ satisfy the conditions required for $(\eta_1, x), (\eta_2, x), (\eta'_1, y), (\eta'_2, y)$ respectively in our claim about the existence of a short segment.

Next we will show that there are chains that go arbitrarily deep in a single branch. Suppose there are chains of unbounded lengths between (η_1, x_1) and (η_2, x_2). All such chains must pass through the least common ancestor (say η_a) of η_1, η_2. For some variable x_a, there must be chains of unbounded lengths between either (η_1, x_1) and (η_a, x_a) or between (η_a, x_a) and (η_2, x_2). Say there are unbounded chains between (η_1, x_1) and (η_a, x_a); the other case is similar. There is only one path between η_1 and η_a, so there must be chains of unbounded lengths that go beyond this path and come back. There must be node variables $(\eta_1, y_1), (\eta_1, y_2)$ or $(\eta_a, y_1), (\eta_a, y_2)$ such that there are chains of unbounded lengths between them. We will consider $(\eta_1, y_1), (\eta_1, y_2)$; the other case is similar. For the chains of unbounded lengths starting from (η_1, y_1) and ending at (η_2, y_2), let η be the highest node (nearest to the root) visited. There must be $(\eta, z_1), (\eta, z_2)$ such that there are chains of unbounded lengths between them that only visit descendants of η. If there is a bound (say B) on how deep the chains go below η and come back, the number of nodes that can be visited is bounded by the number of node variables that occur in the subtree of height B rooted at η (a node can occur at most once in a chain; otherwise, it will contradict the fact that $\sqsubset_T^+$ is irreflexive). Hence, for any bound B, there are chains that go deeper than B and come back.

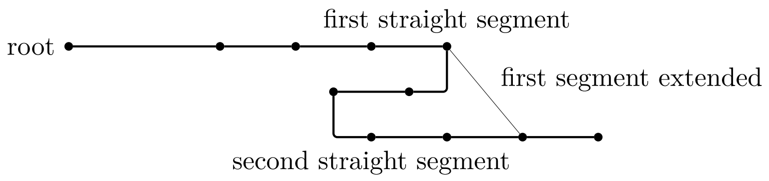
Next we prove that there is no bound on the number of node variables in a single path that belong to a chain. For this, first suppose that there is a node η and a chain goes down one child of η starting from (η, x), comes back to η via (η, y) and goes down another child. Then we have $(\eta, x) \sqsubset_T^+ (\eta, y)$ or $(\eta, y) \sqsubset_T^+ (\eta, x)$ (see the illustration below; if $(\eta, x) \sqsubset_T^* (\eta_b, x') \sqsubset_T^+ (\eta_b, y') \sqsubset_T^* (\eta, y)$ in the branch, we have $(\eta, x) \sqsubset_T^+ (\eta, y)$ in the main path by transitivity). Hence, every such node contributes a node variable in a chain.



So if there is no bound on the number of such branching nodes along a path, then there is no bound on the number of node variables in a single path that belong to a chain, as required. Suppose for the sake of contradiction that the number of such branching nodes along any path is bounded (by say B_1) and the number of node variables in a chain along any one path is also bounded (say by B_2). Then any chain is in a subtree with at most $|G|^{B_1}$ leaves (and hence at most as many paths) and at most B_2 node variables along any path, so the length of such chains is bounded. Hence, either the number of branching nodes along a path is unbounded or the number of node variables in a chain along a path is unbounded. Both of these imply that the number of node variables in a chain along a path is unbounded, as required.

A chain that goes deep down a path may make u-turns (first descend through descendants and then go to an ascendant or vice-versa) multiple times within the branch. We would like to prove that there is no bound on the length of chain segments that don't have u-turns (these are the straight segments that we need). Suppose for the sake of contradiction that there is a bound on the length of straight segments. Then there is no bound on the number of straight segments in a path, since we have already shown that the number of node variables in a chain along a path is unbounded. There can be only boundedly many distinct straight segments in a path of bounded depth, so the straight segments go deeper without any bound.

If there is a straight segment and another one occurs below the first one, the first straight segment can be extended by appending node variables of the second one, as can be seen in the illustration below.



This contradicts the hypothesis that length of straight segments is bounded. This shows that there are unboundedly long straight segments, completing the proof. ◀

B.1 Construction of $\mathcal{A}_\phi$

The automaton $\mathcal{A}_\phi^{\text{str}}$ has set of states $\mathcal{F}$. In state f, it can read the input label f and go to states $f_1, \ldots, f_{|G|}$ in its children, provided (f, f_i) is one-step compatible and (gp_i, f_i) is gap-compatible for all $i \in [1, |G|]$. All states are accepting in this Büchi automaton. This automaton just checks that every pair of consecutive frames along every branch of the tree is one-step compatible and gap-compatible and hence verifies that the tree accepted is a strategy tree. Now, the size of the set of states of $\mathcal{A}_\phi^{\text{str}}$ is $|\mathcal{F}|$, and the size of the transition set is $|\mathcal{F}| \times |\Sigma| \times |\mathcal{F}|^{|G|}$ where the input alphabet $\Sigma = \mathcal{F}$. Since, G is the set of all gap functions associated with mappings of the form $EV^b \to \mathbb{Z}$, by definition of G its range must be $\{0, \ldots, |EV^b|^2\}$ implying $|G| \leq |EV^b|^{(|EV^b|^2)}$. Also, from the definition of $\mathcal{F}$, we get $|\mathcal{F}| \leq 2^{(k.|V^a|)^2} \times (|V^b|^{|V^b|^2})^k$ where k is the X-length of ϕ. Thus, the size of $\mathcal{A}_\phi^{\text{str}}$ is double exponential in the size of ϕ.

The automaton $\mathcal{A}_\phi^{\text{symb}}$ checks that every path in the input tree is accepted by a Büchi automaton $\mathcal{B}_\phi^{\text{symb}}$, which ensures that the input sequence symbolically satisfies the formula ϕ. Given the Büchi automaton $\mathcal{B}_\phi^{\text{symb}}$, we first convert it to some deterministic parity automaton $\mathcal{C}_\phi^{\text{symb}}$ in exponential time in the size of $\mathcal{B}_\phi^{\text{symb}}$ and from that, it is easy to construct the parity tree automaton $\mathcal{A}_\phi^{\text{symb}}$ with the same size as $\mathcal{C}_\phi^{\text{symb}}$. The Büchi automaton $\mathcal{B}_\phi^{\text{symb}}$ needs to check symbolic satisfiability – whether an atomic formula is satisfied at a position can be decided by checking just the current frame, just like propositional LTL. Hence the standard Büchi automaton construction for LTL can be used to construct $\mathcal{B}_\phi^{\text{symb}}$ in EXPTIME [30]. Thus, the parity tree automaton $\mathcal{A}_\phi^{\text{symb}}$ can be constructed in 2EXPTIME in the size of ϕ.

Next, we describe the construction of the parity tree automaton $\mathcal{A}_\phi^{\text{chain}}$. It needs to check that there are no infinite forward or backward chains in any of the paths. For this we will first construct a Büchi word automaton that accepts all words not having an infinite forward or backward chain, convert it into a deterministic parity automaton $\mathcal{C}_\phi^{\text{chain}}$ and then as before, construct $\mathcal{A}_\phi^{\text{chain}}$ with the same size as $\mathcal{C}_\phi^{\text{chain}}$. This Büchi word automaton can be constructed by complementing the Büchi automaton $\mathcal{B}^{\text{chain}}$ which accepts all words that contain an infinite forward chain or an infinite backward chain in EXPTIME in the size of $\mathcal{B}^{\text{chain}}$ [29]. The construction of such a Büchi automaton $\mathcal{B}^{\text{chain}}$ is already described in [13]. The size of $\mathcal{B}^{\text{chain}}$ (as described in [13]) is polynomial in the size of the CLTL formula ϕ and hence, the size of $\mathcal{A}_\phi^{\text{chain}}$ is double exponential in the size of ϕ.

Reasoning on Dynamic Transformations of Symbolic Heaps

Nicolas Peltier
Univ. Grenoble Alpes, CNRS, LIG, Grenoble, France

Abstract

Building on previous results concerning the decidability of the satisfiability and entailment problems for separation logic formulas with inductively defined predicates, we devise a proof procedure to reason on dynamic transformations of memory heaps. The initial state of the system is described by a separation logic formula of some particular form, its evolution is modeled by a finite transition system and the expected property is given as a linear temporal logic formula built over assertions in separation logic.

2012 ACM Subject Classification Theory of computation → Separation logic; Theory of computation → Automated reasoning; Theory of computation → Modal and temporal logics

Keywords and phrases Separation Logic, Symbolic Heaps, Linear Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.9

Funding This work has been partially funded by the the French National Research Agency (ANR-21-CE48-0011).

1 Introduction

Separation logic (SL) [14] is a dialect of bunched logic [10], that was introduced in verification to reason on programs manipulating dynamically allocated memory. The logic uses a particular connective $*$ to assert that two formulas hold on disjoint parts of the memory, which allows for more concise specifications. It supports *local reasoning*, in the sense that the properties of a program can be asserted and proven by referring only to the part of the memory that is affected by the program, and not to the global state of the system. The expressive power of the logic may be enhanced by using inductively defined predicates, which can be used to define recursive data structures of unbounded sizes, such as lists or trees. For instance, the following rules define a predicate $\mathtt{lseg}(x, y)$ denoting a non empty list segment from x to y: $\{\mathtt{lseg}(x, y) \Leftarrow x \mapsto (y), \quad \mathtt{lseg}(x, y) \Leftarrow \exists z.(x \mapsto (z) * \mathtt{lseg}(z, y))\}$. Informally, x, y, z denotes locations (i.e., memory addresses), $x \mapsto (y)$ states that location x is allocated and points to location y and the *separating conjunction* $x \mapsto (z) * \mathtt{lseg}(z, y)$ states that the heap contains a list segment $\mathtt{lseg}(z, y)$ together with an additional memory cell x that points to z (it implicitly entails that x is distinct from all the memory locations allocated in the list segment from z to y). These predicates may be hard coded, but they may also be defined by the user, to tackle custom data structures. For the fragment of separation logic called *symbolic heaps* (formally defined later), satisfiability is decidable [3], but entailment is undecidable in general (entailment cannot be reduced to satisfiability since the fragment does not include negations). However, a general class of decidable entailment problems is described in [7], based on restrictions on the form of the inductive rules that define the semantics of the inductive predicates. More recently, it was shown that the entailment problem is 2-EXPTIME complete [11, 4] for such inductive rules. Building on these results, we devise in the present work a proof procedure to reason on dynamic transformations of data structures specified by SL formulas with inductively defined predicates. More precisely, we consider entailments of the form $\phi \models^{\mathcal{S}}_{\mathcal{R}} \Phi$, where ϕ is an SL formula (more precisely a symbolic heap), $\mathcal{R}$ is a set of inductive rules, $\mathcal{S}$ is a transition system and Φ is a formula combining symbolic heaps with temporal connectives of linear temporal logic (LTL) [13]. Informally, such an entailment is valid if the formula Φ holds w.r.t. all the runs obtained by starting from a structure satisfying the formula ϕ and following the transition system $\mathcal{S}$. The symbolic heap ϕ describes the initial state of the system, $\mathcal{R}$ defines the semantics of the inductively defined predicate symbols, $\mathcal{S}$ describes how the system evolves along time and Φ gives the expected behavior of the system. The system $\mathcal{S}$ may affect the considered structure by changing the value of variables, by allocating or freeing memory locations, or by redirecting already allocated locations. For instance, we may check whether an entailment $\mathtt{lseg}(x, \mathtt{nil}) \models_{\mathcal{R}}^{\mathcal{S}} \boldsymbol{F}\, \mathtt{lseg}(x, x)$ holds, meaning that an initial list segment from x to `nil` is eventually transformed into a circular list, or that $\mathtt{lseg}(x, \mathtt{nil}) \models_{\mathcal{R}}^{\mathcal{S}} \boldsymbol{G}(q \Rightarrow \mathtt{lseg}(x, \mathtt{nil}))$ holds, meaning that each time the system reaches state q the heap contains a list from x to `nil`. We show that the entailment problem is undecidable in general, but decidable if the considered transition system satisfies some conditions, which, intuitively, prevent actions affecting the value of the variables to occur inside loops (the other actions are not restricted). The proposed decision procedure is modular, and relies on a combination of the algorithm described in [12, 11] for checking the satisfiability of separation logic formulas with usual model checking and model construction procedures for LTL.

Related work

Dynamic transformations are usually tackled in SL using Hoare logic, with pre and post-conditions defined with the help of separating implications (see, e.g., [1]). Separating implication is not used in our approach due to the difficulty of reasoning automatically with this connective, especially in connection with inductive definitions (however, the so-called *context predicates* introduced in Section 7 can be viewed as a restricted form of separating implication). The combination of SL with temporal connectives is rather natural and has been considered in [2]. In [8, 6], temporal extensions of the related bunched logic are considered. Our approach departs from this work because the fragment of separation logic that we consider is very different: while the logic in [2] is based on quantifier-free separation logic formulas (with arbitrary combinations of boolean and separating connectives), we focus on symbolic heaps, i.e., on separating conjunctions of inductively defined atoms (with existential quantification). Thus on one hand our basic assertion language is more restricted because we strongly restricts the nesting of separation connectives, but on the other hand the addition of inductively defined predicates greatly increases the expressive power of the language and allows one to tackle richer data structures. In particular we emphasize that – without temporal connectives – entailment is 2-EXPTIME complete for the fragment that we consider, whereas satisfiability is PSPACE-complete for that considered in [2].

2 Separation Logic

We define the syntax and semantics of a fragment of separating logic called *symbolic heaps* and we recall the conditions on the inductive rules that ensure that the entailment problem is decidable. Most definitions are standard, see [14, 7] for additional explanations and examples.

▶ **Definition 1** (Symbolic Heaps). *Let $\mathcal{V}$ be a countably infinite set of* variables*. Let $\mathcal{P}$ be a finite set of* predicate symbols*. Each symbol p in $\mathcal{P}$ is associated with a unique natural number called the* arity *of p. Let κ be a fixed natural number, denoting the number of record fields. An* equational atom *is an expression of the form $x \simeq y$ or $x \not\simeq y$, where $x, y \in \mathcal{V}$. A* points-to atom *is an expression of the form* $x \mapsto (y_1, \dots, y_\kappa)$ *with* $x, y_1, \dots, y_\kappa \in \mathcal{V}$*. A* predicate atom *is an expression of the form* $p(x_1, \dots, x_n)$ *with* $p \in \mathcal{P}$, $n = arity(p)$ *and* $x_1, \dots, x_n \in \mathcal{V}$*. A* spatial atom *is either a points-to atom or a predicate atom. An* atom *is either an equational atom or a spatial atom. The set of* symbolic heaps *is the set of expressions of the form:* $\exists x_1 \dots \exists x_n.(\alpha_1 * \dots * \alpha_m)$ *where* $x_1, \dots, x_n$ *are variables and* $\alpha_1, \dots, \alpha_m$ *are atoms (with possibly* $n = 0$ *and/or* $m = 0$*). The connective* $*$ *is called* separating conjunction*. An empty separating conjunction is denoted by* emp*. For every symbolic heap* ϕ*, we denote by* $fv(\phi)$ *the set of variables freely occurring in* ϕ*.*

For all vectors $\boldsymbol{x} = (x_1, \dots, x_n)$ and $\boldsymbol{y} = (y_1, \dots, y_n)$ of the same length, we denote by $\boldsymbol{x} \simeq \boldsymbol{y}$ the separating conjunction $x_1 \simeq y_1 * \dots * x_n \simeq y_n$. If ϕ is a symbolic heap, then $\exists \boldsymbol{x}.\phi$ denotes the symbolic heap $\exists x_1 \dots \exists x_n.\phi$. For every symbolic heap ϕ we denote by $v_{\mapsto}(\phi)$ the set of free variables x such that ϕ contains a points-to atom of the form $x \mapsto (\boldsymbol{y})$.

▶ **Definition 2** (Substitutions)**.** *A* substitution *is a function mapping every variable* x *to a variable. For every substitution* σ *and for every symbolic heap* ϕ*, we denote by* $\phi\sigma$ *the symbolic heap obtained from* ϕ *by replacing every free occurrence of a variable* x *by* $\sigma(x)$*. If* $x_1, \dots, x_n$ *are pairwise distinct variables, we denote by* $\{x_1 \leftarrow y_1, \dots, x_n \leftarrow y_n\}$ *the substitution* σ *such that* $\sigma(x_i) = y_i$ *for all* $i = 1, \dots, n$ *and* $\sigma(x) = x$ *if* $x \notin \{x_1, \dots, x_n\}$*.*

Symbolic heaps are interpreted in structures defined as follows.

▶ **Definition 3** (SL Structures)**.** *Let* $\mathcal{L}$ *be a countably infinite set of so-called* locations*. An (SL)* structure *is a pair* $(\mathfrak{s}, \mathfrak{h})$ *where:*

- $\mathfrak{s}$ *is a* store*, i.e., a function mapping every variable to a location.*
- $\mathfrak{h}$ *is a* heap*, i.e., a finite partial function mapping locations to* κ*-tuples of locations. We denote by* $dom(\mathfrak{h})$ *the finite domain of* $\mathfrak{h}$*, by* $|\mathfrak{h}|$ *the cardinality of* $dom(\mathfrak{h})$ *and by* $locs(\mathfrak{h})$ *the set:* $\{\ell_i \mid \ell_0 \in dom(\mathfrak{h}), \mathfrak{h}(\ell_0) = (\ell_1, \dots, \ell_\kappa), 0 \leq i \leq \kappa\}$*.*

A location $\ell \in dom(\mathfrak{h})$ *is* allocated *in* $\mathfrak{h}$*. A variable* x *such that* $\mathfrak{s}(x) \in dom(\mathfrak{h})$ *is* allocated *in* $(\mathfrak{s}, \mathfrak{h})$*.*

Intuitively, $\mathfrak{s}$ gives the values of the variables and $\mathfrak{h}$ denotes the dynamically allocated memory. A heap will often be denoted as a set of tuples $\mathfrak{h} = \{(\ell_0, \dots, \ell_\kappa) \mid \ell_0 \in dom(\mathfrak{h}), \mathfrak{h}(\ell_0) = (\ell_1, \dots, \ell_n)\}$. In particular, $\emptyset$ denotes the heap that allocates no location. Two heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are *disjoint* if $dom(\mathfrak{h}_1) \cap dom(\mathfrak{h}_2) = \emptyset$. In this case $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ denotes the union of $\mathfrak{h}_1$ and $\mathfrak{h}_2$ defined as follows: $dom(\mathfrak{h}_1 \uplus \mathfrak{h}_2) \overset{\text{def}}{=} dom(\mathfrak{h}_1) \cup dom(\mathfrak{h}_2)$, and $\mathfrak{h}(\ell) = \mathfrak{h}_i(\ell)$ for all $i = 1, 2$ and $\ell \in dom(\mathfrak{h}_i)$.

The semantics of the predicate symbols is defined by user-provided inductive rules:

▶ **Definition 4** (Inductive Rules)**.** *A* set of inductive definitions *(SID) is a set of rules of the form* $p(x_1, \dots, x_n) \Leftarrow \phi$ *such that* $p \in \mathcal{P}$, $n = arity(p)$, $x_1, \dots, x_n$ *are pairwise distinct variables, and* ϕ *is a symbolic heap with* $fv(\phi) \subseteq \{x_1, \dots, x_n\}$*.*

For every symbolic heap ϕ*, we write* $\phi \Leftarrow_{\mathcal{R}} \phi'$ *if* ϕ *is of the form* $\exists \boldsymbol{u}.(p(y_1, \dots, y_n) * \phi')$*,* $\mathcal{R}$ *contains a rule* $p(x_1, \dots, x_n) \Leftarrow \exists \boldsymbol{v}.\psi$ *(where* ψ *contains no quantifier) and* $\phi' = \exists \boldsymbol{u} \exists \boldsymbol{v}.(\psi\{x_i \leftarrow y_i \mid i = 1, \dots, n\} * \phi')$*. We assume by* α*-renaming that the vector* $\boldsymbol{v}$ *contains no variable in* $\boldsymbol{u}$*,* $fv(\phi)$ *or* $(x_1, \dots, x_n)$*. As usual* $\Leftarrow_{\mathcal{R}}^*$ *is the reflexive and transitive closure of* $\Leftarrow_{\mathcal{R}}$*.*

The satisfiability relation is defined inductively as follows. We emphasize that equational atoms are valid only if the heap is empty; this convention allows us to simplify notations (it avoids having to use both the separating conjunction and the standard one).

▶ **Definition 5** (Satisfiability). *We write* $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ *if one of the following conditions holds:*

- $\mathfrak{h} = \emptyset$ *and either* $(\phi = (x \simeq y)$ *and* $\mathfrak{s}(x) = \mathfrak{s}(y))$*, or* $(\phi = (x \not\simeq y)$ *and* $\mathfrak{s}(x) \neq \mathfrak{s}(y))$*.*
- $\phi = x \mapsto (y_1, \dots, y_\kappa)$ *and* $\mathfrak{h} = \{(\mathfrak{s}(x), \mathfrak{s}(y_1), \dots, \mathfrak{s}(y_\kappa))\}$*.*
- $\phi = \phi_1 * \phi_2$ *and there exist disjoint heaps* $\mathfrak{h}_1$ *and* $\mathfrak{h}_2$ *such that* $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ *and* $(\mathfrak{s}, \mathfrak{h}_i) \models_{\mathcal{R}} \phi_i$*, for all* $i = 1, 2$*.*
- $\phi = p(x_1, \dots, x_n)$ *with* $p \in \mathcal{P}$*,* $p(x_1, \dots, x_n) \Leftarrow^*_{\mathcal{R}} \psi$*,* ψ *contains no predicate symbols and* $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \psi$*.*
- $\phi = \exists x.\psi$*, and there exists a store* $\mathfrak{s}'$ *coinciding with* $\mathfrak{s}$ *on all variables distinct from* x *such that* $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} \psi$*.*

An $\mathcal{R}$-model *of* ϕ *is a structure* $(\mathfrak{s}, \mathfrak{h})$ *such that* $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$*. If* ϕ, ϕ' *are symbolic heaps, we write* $\phi \models_{\mathcal{R}} \phi'$ *if the entailment* $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi \implies (\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi'$ *for all SL structures* $(\mathfrak{s}, \mathfrak{h})$*, and* $\phi \equiv_{\mathcal{R}} \psi$ *if* $\phi \models_{\mathcal{R}} \psi$ *and* $\psi \models_{\mathcal{R}} \phi$*.*

Restricting Inductive Definitions

While the entailment problem is undecidable in general for symbolic heaps with inductively defined predicates, a very general decidable class is identified in [7]. This fragment is defined by restricting the form of the inductive rules, which must satisfy three conditions, recalled below (we use the slightly more general version of establishment given in [11]).

▶ **Definition 6** (Progress, Connectedness and Establishment (PCE)). *A rule* $p(x_1, \dots, x_n) \leftarrow \exists \boldsymbol{y}.\phi$ *(where* ϕ *contains no quantifier) is:*

- progressing *if* ϕ *is of the form* $x_1 \mapsto (z_1, \dots, z_\kappa) * \phi'$*, where* ϕ' *contains no points-to atom (i.e., the rule allocates exactly one location* x_1*);*
- connected *if, moreover, every predicate atom in* ϕ' *is of the form* $q(z, \boldsymbol{v})$ *with* $z \in \{z_1, \dots, z_\kappa\}$ *(i.e., the locations allocated by the called predicates are successors of* x_1*).*

A SID $\mathcal{R}$ *is progressing (resp. connected) if all the rules in* $\mathcal{R}$ *are progressing (resp. connected). It is is* established *if for every atom* $p(x_1, \dots, x_n)$ *and for every formula* ϕ *containing no predicate symbol, if* $p(x_1, \dots, x_n) \Leftarrow^*_{\mathcal{R}} \phi$ *and* x *is existentially quantified in* ϕ *then* ϕ *contains atoms* $y_i \simeq y_{i+1}$ *(for* $i = 0, \dots, n$*, with* $n \geq 0$*) such that* $x = y_{n+1}$ *and either* ϕ *contains a points-to atom of the form* $y_0 \mapsto (\boldsymbol{z})$ *or* $y_0 \in \{x_1, \dots, x_n\}$ *(i.e., every existentially quantified variable either is equal to a free variable or is eventually allocated).*

▶ **Example 7.** The following set, defining a list segment ending at an arbitrary location, is progressing and connected, but *not* established:

$$\{\texttt{lseg}'(x) \Leftarrow \exists y.x \mapsto (y), \quad \texttt{lseg}'(x) \Leftarrow \exists z.(x \mapsto (z) * \texttt{lseg}'(z))\}$$

In the remainder of the paper we assume that a set of inductive rules $\mathcal{R}$ is given, satisfying the PCE conditions. This is the case for the rules given in the Introduction for the predicate `lseg`. We now introduce a notion of heap constraints, which combine positive and negative assertions denoted by symbolic heaps, with constraints specifying that some variables are unallocated:

▶ **Definition 8** (Heap Constraint). *A* heap constraint *is a triple* (S^+, S^-, X)*, where* S^+ *and* S^- *are sets of symbolic heaps,* $S^+ \neq \emptyset$ *and* $X \subseteq \mathcal{V}$*. We write* $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} (S^+, S^-, X)$ *if for all* $\phi \in S^+$*:* $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$*; for all* $\phi \in S^-$*:* $(\mathfrak{s}, \mathfrak{h}) \not\models_{\mathcal{R}} \phi$*; and for all* $x \in X$*:* $\mathfrak{s}(x) \notin dom(\mathfrak{h})$*.*

The decidability of the satisfiability problem for such constraints follows from [12, 11]:

▶ **Lemma 9.** *There exists an algorithm that, given a heap constraint* (S^+, S^-, X) *and a progressing, connected and established set of rules* $\mathcal{R}$*, checks whether there exists an SL structure* $(\mathfrak{s}, \mathfrak{h})$ *such that* $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} (S^+, S^-, X)$*.*

3 Actions Operating on SL Structures

We define the basic actions that can occur in transition systems. The set of actions includes tests, affectations, redirections of allocated locations, as well as allocations and desallocations.

▶ **Definition 10** (Actions). *Let $\mathcal{V}^\star$ be a finite set of variables. A* term *is either an element of $\mathcal{V}^\star$ or an expression of the form $x.i$ where $x \in \mathcal{V}^\star$ and $i \in \{1, \dots, \kappa\}$. A* condition *is a boolean combination of* atomic conditions, *that are expressions of the form $t \approx s$ where t, s are terms. An* action *is an expression of one of the following forms:* `pass` *(null action); $t := s$, where t and s are terms (affectation or redirection);* `alloc`(x) *or* `free`(x), *where $x \in \mathcal{V}^\star$ (allocation and desallocation); or* `test`(γ), *where γ is a condition (test).*

The semantics of conditions is defined below.

▶ **Definition 11** (Semantics of Conditions). *For every structure $(\mathfrak{s}, \mathfrak{h})$ and for every term t we write $t \rhd_{(\mathfrak{s},\mathfrak{h})} \ell$ (t* evaluates to *ℓ) if either $t \in \mathcal{V}^\star$ and $\mathfrak{s}(t) = \ell$, or $t = x.i$ with $x \in \mathcal{V}^\star$, $\mathfrak{s}(x) \in dom(\mathfrak{h})$, $\mathfrak{h}(\mathfrak{s}(x)) = (\ell_1, \dots, \ell_\kappa)$ and $\ell = \ell_i$. We write $(\mathfrak{s}, \mathfrak{h}) \models t \approx s$ if there exists $\ell \in \mathcal{L}$ such that $t \rhd_{(\mathfrak{s},\mathfrak{h})} \ell$ and $s \rhd_{(\mathfrak{s},\mathfrak{h})} \ell$. The relation $\models$ is extended to every boolean combination of atomic conditions inductively as usual.*

Observe that the semantics of $x \approx y$ is different from that of $x \simeq y$, which requires that the heap be empty. Furthermore, $(\mathfrak{s}, \mathfrak{h}) \models x.i \approx x.i$ (for $i = 1, \dots, \kappa$) holds iff x is allocated. We thus denote by $\mathtt{A}(x)$ (for "x is allocated") the formula $x.1 \approx x.1$. The semantics of actions is rather natural, and formally defined below (to make allocations deterministic we assume that the variable allocated by `alloc`(x) points to itself).

▶ **Definition 12** (Semantics of Actions). *For every SL structure $(\mathfrak{s}, \mathfrak{h})$ and action a, we denote by $(\mathfrak{s}, \mathfrak{h})[a]$ the result of the application of the action a on $(\mathfrak{s}, \mathfrak{h})$ defined as follows:*

- *If $a =$ `pass` then $(\mathfrak{s}, \mathfrak{h})[a] \overset{def}{=} (\mathfrak{s}, \mathfrak{h})$.*
- *If $a = (x := s)$ with $x \in \mathcal{V}^\star$ and $s \rhd_{(\mathfrak{s},\mathfrak{h})} \ell$ then $(\mathfrak{s}, \mathfrak{h})[a] \overset{def}{=} (\mathfrak{s}', \mathfrak{h})$, where $\mathfrak{s}'(x) = \ell$ and $\mathfrak{s}'$ coincides with $\mathfrak{s}$ on all variables distinct from x.*
- *If $a = (x.i := s)$ with $x \in \mathcal{V}^\star$, $\mathfrak{s}(x) \in dom(\mathfrak{h})$, $\mathfrak{h}(\mathfrak{s}(x)) = (\ell_1, \dots, \ell_n)$ and $s \rhd_{(\mathfrak{s},\mathfrak{h})} \ell$ then $(\mathfrak{s}, \mathfrak{h})[a] \overset{def}{=} (\mathfrak{s}, \mathfrak{h}')$, where $dom(\mathfrak{h}') = dom(\mathfrak{h})$, $\mathfrak{h}'(\mathfrak{s}(x)) = (\ell_1, \dots, \ell_{i-1}, \ell, \ell_{i+1}, \dots, \ell_\kappa)$, and $\mathfrak{h}'$ coincides with $\mathfrak{h}$ on all locations distinct from $\mathfrak{s}(x)$.*
- *If $a =$ `free`(x), $\mathfrak{s}(x) \in dom(\mathfrak{h})$ then $(\mathfrak{s}, \mathfrak{h})[a] \overset{def}{=} (\mathfrak{s}, \mathfrak{h}')$ where $dom(\mathfrak{h}') = dom(\mathfrak{h}) \setminus \{\mathfrak{s}(x)\}$ and $\mathfrak{h}'$ coincides with $\mathfrak{h}$ on all locations distinct from $\mathfrak{s}(x)$.*
- *If $a =$ `alloc`(x), $\mathfrak{s}(x) \notin dom(\mathfrak{h})$ then $(\mathfrak{s}, \mathfrak{h})[a] \overset{def}{=} (\mathfrak{s}, \mathfrak{h}')$ where $dom(\mathfrak{h}') = dom(\mathfrak{h}) \cup \{\mathfrak{s}(x)\}$, $\mathfrak{h}(\mathfrak{s}(x)) = (\mathfrak{s}(x), \dots, \mathfrak{s}(x))$ and $\mathfrak{h}'$ coincides with $\mathfrak{h}$ on all locations distinct from $\mathfrak{s}(x)$.*
- *If $a =$ `test`(γ) and $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \gamma$ then $(\mathfrak{s}, \mathfrak{h})[a] \overset{def}{=} (\mathfrak{s}, \mathfrak{h})$.*

Otherwise $(\mathfrak{s}, \mathfrak{h})[a]$ is undefined.

▶ **Proposition 13.** *For all structures $(\mathfrak{s}, \mathfrak{h})$ and actions a, if $(\mathfrak{s}', \mathfrak{h}') = (\mathfrak{s}, \mathfrak{h})[a]$ then $\mathfrak{s}'(\mathcal{V}^\star) \cup locs(\mathfrak{h}') \subseteq \mathfrak{s}(\mathcal{V}^\star) \cup locs(\mathfrak{h})$.*

Proof. By an immediate case analysis on the set of actions. ◀

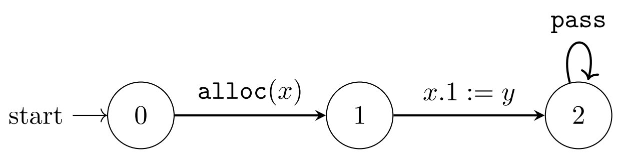
4 Transition Systems

Transition systems are finite state automata where the transitions are labeled by actions:

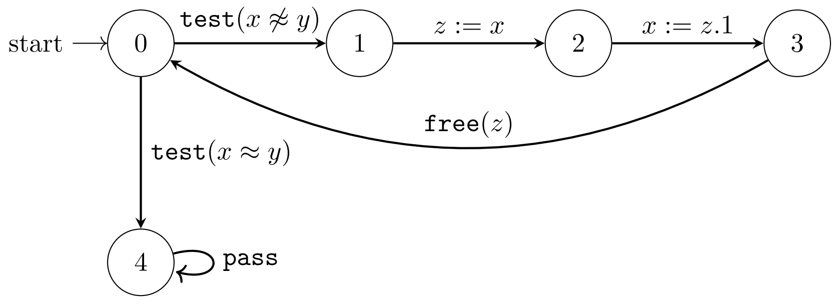
▶ **Definition 14** (Transition Systems). *Let $\mathbb{S}$ be a countably infinite set of* states. *A* transition system *is a triple $\mathcal{S} = (Q, R, q_I)$ where Q is a finite subset of $\mathbb{S}$, R is a finite set of* transition rules *of the form (q, a, q') where $q, q' \in Q$ and a is an action, and $q_I \in Q$ is the initial state. A* run *in $\mathcal{S}$* from *a structure $(\mathfrak{s}, \mathfrak{h})$ is an infinite sequence of tuples $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)_{i\in\mathbb{N}}$ such that $q_0 = q_I$, $(\mathfrak{s}_0, \mathfrak{h}_0) = (\mathfrak{s}, \mathfrak{h})$, and for every $i \in \mathbb{N}$: $(q_i, a_i, q_{i+1}) \in R$ and $(\mathfrak{s}_{i+1}, \mathfrak{h}_{i+1}) = (\mathfrak{s}_i, \mathfrak{h}_i)[a_i]$. We denote by $\succeq_{\mathcal{S}}$ the smallest transitive relation such that $(q, a, q') \in R \implies q \succeq_{\mathcal{S}} q'$. We write $q \sim_{\mathcal{S}} q'$ iff $q \succeq_{\mathcal{S}} q'$ and $q' \succeq_{\mathcal{S}} q$ and $q \succ_{\mathcal{S}} q'$ if $q \succeq_{\mathcal{S}} q'$ and $q' \not\succeq_{\mathcal{S}} q$.*

Note that the above definition entails that $(\mathfrak{s}_i, \mathfrak{h}_i)[a_i]$ must be defined, for all $i \in \mathbb{N}$. For simplicity we assume that all runs are infinite (finite runs may be encoded if needed by adding a final state q_F with a transition $(q_F, \texttt{pass}, q_F)$).

▶ **Example 15.** The following transition system adds an element x to a list starting at y:



▶ **Example 16.** The following transition system desallocates a list segment from x to y.



5 Temporal Formulas

We now define temporal formulas built over a set of assertions containing symbolic heaps, states, actions and conditions, using the usual set of LTL connectives:

▶ **Definition 17** (Syntax of LTL Formulas). *The set $\mathbb{A}_{\mathcal{S}}$ of* LTL atoms *contains all symbolic heaps ϕ with $fv(\phi) \subseteq \mathcal{V}^\star$, all atomic conditions, all actions and all states in $\mathbb{S}$. The set of* LTL formulas *is the least set containing $\mathbb{A}_{\mathcal{S}}$ and such that for all LTL formulas Φ, Ψ: $\neg\Phi$, $\Phi \vee \Psi$, $\boldsymbol{X}\,\Phi$, $\Phi\,\boldsymbol{U}\,\Psi$ are LTL formulas.*

The additional connectives $\wedge$, $\boldsymbol{F}$, $\boldsymbol{R}$ etc. are defined as usual. The semantics of LTL formulas is recalled below. Note that LTL atoms are interpreted arbitrarily at this point.

▶ **Definition 18** (Semantics of LTL Formulas). *An* LTL interpretation *$\mathcal{I}$ is a mapping from $\mathbb{A}_{\mathcal{S}} \times \mathbb{N}$ to $\{true, false\}$. For any LTL formula Φ, we write $\mathcal{I} \models \Phi$ if $(\mathcal{I}, 0) \models \Phi$, and $(\mathcal{I}, i) \models \Phi$ iff one of the following conditions holds:*

- *$\Phi \in \mathbb{A}_{\mathcal{S}}$ and $\mathcal{I}(\Phi, i) = true$, or $\Phi = \neg\Psi$ and $(\mathcal{I}, i) \not\models \Psi$;*
- *$\Phi = \Phi_1 \vee \Phi_2$ and $(\mathcal{I}, i) \models \Phi_j$, for some $j = 1, 2$; or $\Phi = \boldsymbol{X}\,\Psi$ and $(\mathcal{I}, i+1) \models \Psi$;*
- *$\Phi = \Phi_1 \boldsymbol{U} \Phi_2$ and there exists $j \geq i$ such that $(\mathcal{I}, j) \models \Phi_2$ and $(\mathcal{I}, k) \models \Phi_1$ for all $k \in \{i, \dots, j-1\}$.*

In the following, we will assume that all the considered LTL interpretations are *ultimately periodic* (so that they admit a finite representation) i.e., that there exist natural numbers k, l such that, for every $i \geq k$ and for every atom $\alpha \in \mathbb{A}_{\mathcal{S}}$: $\mathcal{I}(\alpha, i) = \mathcal{I}(\alpha, i+l)$. It is well-known that every satisfiable LTL formula admits an ultimately periodic model. Definition 19 relates the semantics of LTL atoms to SL structures and transition systems.

▶ **Definition 19** (Compatibility). *Let $\mathcal{S} = (Q, R, q_I)$ be a transition system. An LTL interpretation $\mathcal{I}$ is* compatible *with a run $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)_{i\in\mathbb{N}}$ in $\mathcal{S}$ w.r.t. a formula Φ iff the following conditions holds, for all $i \in \mathbb{N}$:*
- *For every symbolic heap or condition ϕ occurring in Φ, $\mathcal{I}(\phi, i) = true \iff (\mathfrak{s}_i, \mathfrak{h}_i) \models_{\mathcal{R}} \phi$.*
- *For all actions a, $\mathcal{I}(a, i) = true \iff a = a_i$.*
- *For all states $q \in Q$, $\mathcal{I}(q, i) = true \iff q_i = q$.*

An LTL interpretation $\mathcal{I}$ is compatible *with an SL structure $(\mathfrak{s}, \mathfrak{h})$ and a transition system $\mathcal{S} = (Q, R, q_I)$, w.r.t. a formula Φ if it is compatible with some run $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)_{i\in\mathbb{N}}$ in $\mathcal{S}$.*

We are now in the position to define the satisfiability relation that relates SL structures to LTL formulas, w.r.t. a given transition system.

▶ **Definition 20** (Entailment). *Let $\mathcal{S}$ be a transition system. For every structure $(\mathfrak{s}, \mathfrak{h})$ and LTL formula Φ, we write $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}}^{\mathcal{S}} \Phi$ iff $\mathcal{I} \models \Phi$ holds for every LTL interpretation compatible with $(\mathfrak{s}, \mathfrak{h})$ and $\mathcal{S}$ w.r.t. Φ. We write $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}}^{\mathcal{S}/(q_i,a_i)_{i\in\mathbb{N}}} \Phi$ if there exists a run $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)_{i\in\mathbb{N}}$ in $\mathcal{S}$ with $\mathfrak{s}_0 = \mathfrak{s}$ and $\mathfrak{h}_0 = \mathfrak{h}$, and an LTL interpretation $\mathcal{I}$ that is compatible with $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)_{i\in\mathbb{N}}$ such that $\mathcal{I} \models \Phi$.*

For every symbolic heap ϕ, we write $\phi \models_{\mathcal{R}}^{\mathcal{S}} \Phi$ if the entailment $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi \implies (\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}}^{\mathcal{S}} \Phi$ holds for all structures $(\mathfrak{s}, \mathfrak{h})$.

▶ **Example 21.** If $\mathcal{S}$ is the transition system of Example 15, then the entailments $\mathtt{lseg}(y, z) \models_{\mathcal{R}}^{\mathcal{S}} \boldsymbol{F}\,\mathtt{lseg}(x, z)$ and $\mathtt{lseg}(y, z) \models_{\mathcal{R}}^{\mathcal{S}} \boldsymbol{X}\,\boldsymbol{X}\,\boldsymbol{G}\,\mathtt{lseg}(x, z)$ are valid. Note that the structures in which x is initially allocated are not considered for testing the entailment.

If $\mathcal{S}$ now denotes the transition system of Example 16, then the entailment $\mathtt{lseg}(x, y) \models_{\mathcal{R}}^{\mathcal{S}} \boldsymbol{F}\,\mathit{emp}$ is *not* valid (because the initial list segment may be cyclic).

It is easy to see that model checking is decidable:

▶ **Lemma 22.** *The problem of checking whether $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}}^{\mathcal{S}/(q_i,a_i)_{i\in\mathbb{N}}} \Phi$ is decidable (if the sequence $(q_i, a_i)_{i\in\mathbb{N}}$ is ultimately periodic).*

Proof. Since $\mathfrak{s}$, $\mathfrak{h}$, q_i and a_i are given, the run (if it exists) $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)_{i\in\mathbb{N}}$ such that $\mathfrak{s}_0 = \mathfrak{s}$ and $\mathfrak{h}_0 = \mathfrak{h}$, and the compatible LTL interpretation $\mathcal{I}$ are easy to compute, using Definition 11. Using Proposition 13, we get $\mathfrak{s}_i(\mathcal{V}^\star) \cup \mathit{locs}(\mathfrak{h}_i) \subseteq \mathfrak{s}(\mathcal{V}^\star) \cup \mathit{locs}(\mathfrak{h})$ for all $i \in \mathbb{N}$, thus the set of structures $\{(\mathfrak{s}_i, \mathfrak{h}_i) \mid i \in \mathbb{N}\}$ is necessarily finite. Thus, the interpretation $\mathcal{I}$ is ultimately periodic and the test $\mathcal{I} \models \Phi$ can be performed using well-known algorithms for LTL. ◀

However, the entailment problem is undecidable in general:

▶ **Theorem 23.** *The problem of checking whether $\phi \models_{\mathcal{R}}^{\mathcal{S}} \Phi$ is undecidable (even if $\mathcal{R}$ is progressing, connected and established).*

Proof (Sketch). Turing machines (TM) may be simulated by transition systems: the elements of the alphabet are denoted by pairwise distinct free variables, a tape $(x_1, \dots, x_n)$ is encoded as a heap (denoting a doubly linked list): $\{(\ell_i, \ell'_i, \ell_{i-1}, \ell_{i+1}) \mid i = 1, \dots, n\}$ with $\ell'_i = \mathfrak{s}(x_i)$, and the position of the head is denoted by a variable x. Moves are encoded by actions $x := x.2$ (left move) or $x := x.3$ (right move). Tests are performed by actions of the form $\mathtt{test}(x.1 \approx y)$, where y is the variable associated with the considered symbol. The action $x.1 := y$ writes y at the current position in the tape. Note that if the initial heap does not contain enough allocated locations then the transition system may be "stuck" (because a right move cannot be applied, hence no run will exist). However, the following rules define a predicate p that allocates a tape of arbitrary size filled with a symbol u (which may be instantiated by a blank denoted by some free variable b). The variables y and z denote the start and the end of the tape, respectively: $\{p(x,y,z,u) \Leftarrow x \mapsto (u,y,z), \quad p(x,y,z,u) \Leftarrow \exists x'.(x \mapsto (u,y,x') * p(x',x,z,u))\}$. It is easy to check that the non termination of the considered TM (on an empty tape) can be checked by testing whether the entailment $p(x,y,z,b) \models^{\mathcal{S}}_{\mathcal{R}} \boldsymbol{G}(\neg \bigvee_{q \in Q_F} q)$ holds, where Q_F is the set of final states (note however that the entailment $p(x,y,z,b) \models^{\mathcal{S}}_{\mathcal{R}} \boldsymbol{F}(\bigvee_{q \in Q_F} q)$ does *not* encode termination, as it may have counter models in which $p(x,y,z,b)$ does not allocate enough memory cells to execute the TM). ◀

To overcome this issue we require that no action of the form $x := t$ occurs inside a loop:

▶ **Definition 24** (Oriented Transition System). *A transition system $\mathcal{S} = (Q, R, q_I)$ is* oriented *if for every transition (q, a, q') in R, if a is of the form $x := t$ then $q \succ_{\mathcal{S}} q'$.*

The transition system of Example 15 is oriented, but not that of Example 16.

6 Symbolic Execution of Actions

We now show how to execute actions symbolically on SL formulas. We first define an LTL formula encoding the conditions ensuring that an action can be performed:

▶ **Definition 25** (Precondition). *For all actions a, $pre(a)$ is defined as follows (with $x, y \in \mathcal{V}^\star$): $pre(\mathtt{alloc}(x)) \stackrel{def}{=} \neg \mathtt{A}(x)$, $pre(\mathtt{free}(x)) \stackrel{def}{=} \mathtt{A}(x)$, $pre(x.i := y) \stackrel{def}{=} \mathtt{A}(x)$, $pre(x := y.i) \stackrel{def}{=} \mathtt{A}(y)$, $pre(x.i := y.j) \stackrel{def}{=} \mathtt{A}(x) \wedge \mathtt{A}(y)$, $pre(\mathtt{test}(\gamma)) \stackrel{def}{=} \gamma$ and $pre(a) \stackrel{def}{=} \top$ otherwise.*

▶ **Proposition 26.** *For every action a and for every structure $(\mathfrak{s}, \mathfrak{h})$, $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} pre(a)$ iff $(\mathfrak{s}, \mathfrak{h})[a]$ is defined.*

Proof. Immediate. ◀

Given a symbolic heap ϕ and action a, it is sometimes possible to compute the strongest postcondition of ϕ w.r.t. to a, which describes the state of the memory after action a is performed on a structure satisfying ϕ:

▶ **Definition 27** (Strongest Postcondition). *For every symbolic heap ϕ and for every action a, we define a formula $spc(\phi, a)$ (strongest postcondition of α w.r.t. a) as follows (where x' denotes a fresh variable).*

- $spc(\phi, \mathtt{pass}) \stackrel{def}{=} \phi$.
- $spc(\exists \boldsymbol{y}.\phi, \mathtt{alloc}(x)) \stackrel{def}{=} \exists \boldsymbol{y}.(x \mapsto (x, \dots, x) * \phi)$.
- $spc(\exists \boldsymbol{y}.(x \mapsto (y_1, \dots, y_\kappa) * \phi), \mathtt{free}(x)) \stackrel{def}{=} \exists \boldsymbol{y}.\phi$.
- $spc(\phi, x := y) \stackrel{def}{=} \exists x'.(\phi\{x \leftarrow x'\} * x \simeq y)$ *(if $x, y \in \mathcal{V}^\star$).*
- $spc(\exists \boldsymbol{u}.(\phi * y \mapsto (y_1, \dots, y_\kappa)), x := y.i) \stackrel{def}{=} \exists \boldsymbol{u} \exists x'.((\phi * y \mapsto (y_1, \dots, y_\kappa))\{x \leftarrow x'\} * x \simeq y_i)$.

- *spc*$(\exists \boldsymbol{u}.(x \mapsto (x_1,\dots,x_\kappa) * \phi), x.i := z) \overset{def}{=} \exists \boldsymbol{u}.(x \mapsto (x_1,\dots,x_{i-1},z,x_{i+1},\dots,x_\kappa) * \phi)$ *(if* $z \in \mathcal{V}^\star$*).*
- *spc*$(\exists \boldsymbol{u}.(x \mapsto (x_1,\dots,x_\kappa) * \phi), x.i := x.j) \overset{def}{=} \exists \boldsymbol{u}.(x \mapsto (x_1,\dots,x_{i-1},x_j,x_{i+1},\dots,x_\kappa) * \phi)$.
- *spc*$(\exists \boldsymbol{u}.(x \mapsto (x_1,\dots,x_\kappa) * y \mapsto (y_1,\dots,y_\kappa) * \phi), x.i := y.j) \overset{def}{=} \exists \boldsymbol{y}.(x \mapsto (x_1,\dots,x_{i-1},y_j, x_{i+1},\dots,x_\kappa) * y \mapsto (y_1,\dots,y_\kappa) * \phi)$ *(if* $x \neq y$*).*
- *Otherwise spc*(ϕ, a) *is undefined.*

In all cases, ϕ *may be emp.*

▶ **Example 28.** For instance, we have:

$$\begin{array}{lcl} spc(x \mapsto (y,z), x.1 := x) & = & x \mapsto (x,z) \\ spc(x \mapsto (y,z), x := y) & = & \exists x'.(x' \mapsto (y,z) * x \simeq y) \\ spc(x \mapsto (y,z), \mathtt{free}(x)) & = & emp \end{array}$$

But both $spc(x \mapsto (y,z), y.1 := x)$ and $spc(\mathtt{lseg}(x,y), x.1 := y)$ are undefined.

▶ **Lemma 29.** *Let* ϕ *be a symbolic heap and let* a *be an action. If* $spc(\phi,a)$ *is defined then for every structure* $(\mathfrak{s},\mathfrak{h})$ *such that* $(\mathfrak{s},\mathfrak{h})[a]$ *is defined, we have* $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \phi \implies (\mathfrak{s},\mathfrak{h})[a] \models_{\mathcal{R}} spc(\phi,a)$*.*

Proof. By inspection of the different actions, using Definition 11. ◀

Similarly, it is possible in some cases to define the weakest precondition of a symbolic heap w.r.t. an action, asserting conditions that guarantee that the given formula is satisfied after the action is performed:

▶ **Definition 30** (Weakest Precondition). *For every symbolic heap* ϕ *and for every action* a*, the formula* $wpc(\phi,a)$ *is defined as follows (where* x' *denotes a fresh variable).*

- *wpc*$(\phi, \mathtt{pass}) \overset{def}{=} \phi$.
- *wpc*$(\exists \boldsymbol{x}.(\phi * x \mapsto (y_1,\dots,y_\kappa)), \mathtt{alloc}(x)) \overset{def}{=} \exists \boldsymbol{x}.(\phi * y_1 \simeq x * \dots * y_\kappa \simeq x)$.
- *wpc*$(\exists \boldsymbol{x}.\phi, \mathtt{free}(x)) \overset{def}{=} \exists \boldsymbol{x} \exists y_1 \dots \exists y_\kappa.(\phi * x \mapsto (y_1,\dots,y_\kappa))$.
- *wpc*$(\phi, x := y) \overset{def}{=} \phi\{x \leftarrow y\}$ *(if* $x,y \in \mathcal{V}^\star$*).*
- *wpc*$(\exists \boldsymbol{x}.(\phi * x \mapsto (x_1,\dots,x_\kappa)), x.i := y) \overset{def}{=} \exists \boldsymbol{x} \exists x'.(\phi * x \mapsto (x_1,\dots,x_{i-1},x',x_{i+1},\dots,x_\kappa) * x_i \simeq y)$ *(if* $y \in \mathcal{V}^\star$*).*
- *wpc*$(\exists \boldsymbol{x}.(\phi * x \mapsto (x_1,\dots,x_\kappa) * y \mapsto (y_1,\dots,y_\kappa)), x.i := y.j) \overset{def}{=} \exists \boldsymbol{x} \exists x'.(\phi * x \mapsto (x_1,\dots, x_{i-1},x',x_{i+1},\dots,x_\kappa) * y \mapsto (y_1,\dots,y_\kappa) * x_i \simeq y_j)$.
- *wpc*$(\exists \boldsymbol{x}.(\phi * x \mapsto (x_1,\dots,x_\kappa)), x.i := x.j) \overset{def}{=} \exists \boldsymbol{x} \exists x'.(\phi * x \mapsto (x_1,\dots,x_{i-1},x',x_{i+1},\dots,x_\kappa) * x_i \simeq x_j)$.
- *wpc*$(\exists \boldsymbol{x}.(\phi * x \mapsto (x_1,\dots,x_\kappa)), y := x.i) \overset{def}{=} \exists \boldsymbol{x}.((\phi * x \mapsto (x_1,\dots,x_\kappa))\{y \leftarrow x_i\})$ *if* $x \neq y$*. The case where* $x = y$ *is handled by encoding the action* $x := x.i$ *as the sequence* $z := x; x := z.i$*, where* z *is a special variable in* $\mathcal{V}^\star$ *not occurring in the considered transition system.*
- *Otherwise, wpc*(α, a) *is undefined.*

▶ **Example 31.** For instance, we have:

$$\begin{array}{lcl} wpc(x \mapsto (y,z), x.1 := x) & = & \exists x'.(x \mapsto (x',z) * y \simeq x) \\ wpc(x \mapsto (y,z), x := y) & = & y \mapsto (y,z) \\ wpc(x \mapsto (y,z), \mathtt{alloc}(x)) & = & y \simeq x * z \simeq x \end{array}$$

Both $wpc(x \mapsto (y,z), \mathtt{free}(y))$ and $wpc(\mathtt{lseg}(x,y), x.1 := y)$ are undefined.

▶ **Lemma 32.** *Let ϕ be a symbolic heap and let a be an action. If $wpc(\phi, a)$ is defined then for every structure $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h})[a]$ is defined, we have $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} wpc(\phi, a) \iff (\mathfrak{s}, \mathfrak{h})[a] \models_{\mathcal{R}} \phi$.*

Proof. By inspection of the different cases. ◀

Intuitively, the weakest pre-conditions will be used to propagate towards the initial time all the constraints occurring along the run, while strongest post-conditions will be used to ensure that, at any time, the shape of the heap can be described as a symbolic heap, so that all the conditions that hold along the run can be embedded in a heap constraint.

7 Context Predicates

As shown in the previous section, post and preconditions cannot be defined for all symbolic heaps. Indeed, in some cases, the conditions can be computed only if the consider formula contains some specific points-to atom(s) $x \mapsto (\dots)$, where x is some variable involved in the action (for instance for actions $x.i := y$). In this section we devise an algorithm that, given a symbolic heap ϕ and a variable x, returns a disjunction of symbolic heaps equivalent to ϕ (on structures that allocate x), and such that all symbolic heaps contain a points-to atom of the form $x \mapsto (\boldsymbol{y})$. The latter condition will enable the computation of post and preconditions. To this aim, we consider so-called *context predicates* (adapted from [5]). For every pair of predicates p, q with $arity(p) = n$ and $arity(q) = m$, we define a predicate $(q \multimap p)$ of arity $n + m$ in such a way that $(q \multimap p)(x_1, \dots, x_n, y_1, \dots, y_m)$ is satisfied by all (non empty) structures that will satisfy $p(x_1, \dots, x_n)$ after a disjoint heap satisfying $q(y_1, \dots, y_m)$ is added to the current heap. Intuitively, the rules of $(q \multimap p)$ are defined exactly as those of p, except that exactly one call to $q(y_1, \dots, y_m)$ is removed. More formally, for each rule $p(u_1, \dots, u_n) \Leftarrow \exists \boldsymbol{w}.(u_1 \mapsto (\boldsymbol{y}) * p'(\boldsymbol{z}) * \psi)$ in $\mathcal{R}$ we introduce two rules:

$$(q \multimap p)(u_1, \dots, u_n, v_1, \dots, v_m) \Leftarrow \exists \boldsymbol{w}.(u_1 \mapsto (\boldsymbol{y}) * (q \multimap p')(\boldsymbol{z}, v_1, \dots, v_m) * \psi)$$

$$(q \multimap p)(u_1, \dots, u_n, v_1, \dots, v_m) \Leftarrow \exists \boldsymbol{w}.(u_1 \mapsto (\boldsymbol{y}) * \boldsymbol{z} \simeq (v_1, \dots, v_m) * \psi) \quad \text{if } q = p'$$

It is easy to check that these rules fulfill the conditions of Definition 6. Note that the $\multimap$ operation may be nested, e.g., one may consider predicates such as $(\texttt{lseg} \multimap (\texttt{lseg} \multimap \texttt{lseg}))$. Thus $\mathcal{R}$ is actually infinite, and the rules must be computed on demand.

▶ **Example 33.** For instance $(\texttt{lseg} \multimap \texttt{lseg})$ is defined by the rules:

$$(\texttt{lseg} \multimap \texttt{lseg})(x, y, u, v) \Leftarrow \exists z.(x \mapsto (z) * z \simeq u * v \simeq y)$$

and

$$(\texttt{lseg} \multimap \texttt{lseg})(x, y, u, v) \Leftarrow \exists z.x \mapsto (z) * (\texttt{lseg} \multimap \texttt{lseg})(x, z, u, v)$$

The proposed transformation algorithm relies on the use of these context predicates. The idea is that, by Definition 6, a variable x is allocated in a structure validating a predicate atom ϕ iff the corresponding unfolding of ϕ contains a predicate atom of the form $q(z_1, \dots, z_m)$, for some $q \in \mathcal{P}$, where z_1 has the same value as x. Using context predicates it is possible to transform the formula in a way that this atom occurs explicitly in it, since a predicate atom $p(\boldsymbol{y})$ calling $q(\boldsymbol{z})$ is equivalent to $q(\boldsymbol{z}) * (q \multimap p)(\boldsymbol{z}, \boldsymbol{y})$. Then, it suffices to unfold this atom once to get a points-to atom of the form $x \mapsto (\dots)$. More formally:

▶ **Definition 34** (Computation of $\langle \phi \rangle_x$). *Let ϕ be a symbolic heap and let $x \in \mathcal{V}^\star$. The set $\langle \phi \rangle_x$ is defined as follows:*

1. *If $x \in v_{\mapsto}(\phi)$ then $\langle \phi \rangle_x \stackrel{def}{=} \{\phi\}$.*
2. *Otherwise, $\langle \phi \rangle_x$ is the set of formulas that are of one of the following forms:*
   a. *$\exists \boldsymbol{u}.(x \simeq x' * x \mapsto (\boldsymbol{y}) * \psi)$ where ϕ is of the form $\exists \boldsymbol{u}.(x' \mapsto (\boldsymbol{y}) * \psi)$.*
   b. *$\exists \boldsymbol{u} \exists \boldsymbol{v}.(x \simeq x' * \psi' * \psi)$ where ϕ is of the form $\exists \boldsymbol{u}.(p(x', \boldsymbol{y}) * \psi)$ and $p(x, \boldsymbol{y}) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.\psi'$.*
   c. *$\exists \boldsymbol{u} \exists \boldsymbol{v} \exists z_1 \dots \exists z_m.((q \multimap p)(\boldsymbol{y}, z_1, \dots, z_m) * z_1 \simeq x * \psi' * \psi)$ where ϕ is of the form $\exists \boldsymbol{u}.(p(\boldsymbol{y}) * \psi)$, $q \in \mathcal{P}$, $m = arity(q)$, $z_1, \dots, z_m$ are pairwise distinct fresh variables and $q(x, z_2, \dots, z_m) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.\psi'$.*

Item 1 corresponds to the trivial case where ϕ already contains an atom $x \mapsto (\dots)$. Item 2a corresponds to the case where ϕ contains an atom $x' \mapsto (\dots)$ where $x \simeq x'$ holds. Item 2b handles the case where ϕ contains an atom $p(x', \boldsymbol{y})$ that (immediately) allocates x (by the progress condition this happens iff $x \simeq x'$ holds). Finally, Item 2c tackles the general case, where ϕ contains an atom $p(\boldsymbol{y})$ which (eventually) calls an atom $q(z_1, z_2, \dots, z_m)$ that allocates x. For instance $\langle \mathtt{lseg}(x, y) \rangle_z$ contains the symbolic heaps: $\exists u.(z \mapsto (u) * \mathtt{lseg}(u, y) * x \simeq z)$ and $\exists u, v, w.(\mathtt{lseg} \multimap \mathtt{lseg})(x, y, u, v) * z \mapsto (w) * \mathtt{lseg}(w, v) * u \simeq z)$. Note that both formulas contain a points-to atom of the form $z \mapsto (\dots)$. The following lemmata state that $\langle \phi \rangle_x$ fulfills all the expected properties.

▶ **Lemma 35.** *Let ϕ be a symbolic heap and let $x \in \mathcal{V}^\star$. For every formula $\phi' \in \langle \phi \rangle_x$, $x \in v_{\mapsto}(\phi')$. Thus if $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi'$ then $\mathfrak{s}(x) \in dom(\mathfrak{h})$.*

Proof. Let $\phi' \in \langle \phi \rangle_x$. If $x \in v_{\mapsto}(\phi)$ then $\langle \phi \rangle_x = \{\phi\}$ thus $\phi' = \phi$ and $x \in v_{\mapsto}(\phi')$. In all other cases in Definition 34, either ϕ' contains a points-to atom $x \mapsto (\boldsymbol{y})$, or ϕ' contains a formula ψ' such that there exists an atom α of root x (α is either $p(x, \boldsymbol{y})$ or $q(x, z_2, \dots, z_m)$) such that $\alpha \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.\psi'$. By the progress condition necessarily $x \in v_{\mapsto}(\psi')$, so that $x \in v_{\mapsto}(\phi')$. The second part of the lemma follows immediately from the definition of the semantics. ◀

▶ **Lemma 36.** *Let ϕ be a symbolic heap and let $x \in \mathcal{V}^\star$. For every formula $\psi \in \langle \phi \rangle_x$ and for all SL structures $(\mathfrak{s}, \mathfrak{h})$: $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \psi \implies (\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$.*

▶ **Lemma 37.** *Let ϕ be a symbolic heap and let $x \in \mathcal{V}^\star$. For every SL structure $(\mathfrak{s}, \mathfrak{h})$ such that $\mathfrak{s}(x) \in dom(\mathfrak{h})$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$, we have $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \psi$, for some $\psi \in \langle \phi \rangle_x$.*

8 Axioms

Building on the previous results, we define LTL axioms ensuring that an LTL interpretation is compatible with some SL structure, for a given transition system $\mathcal{S} = (Q, R, q_I)$. The axioms are obtained by embedding all the previous definitions and properties in LTL (a, ϕ, γ and x range over the set of actions, symbolic heaps, conditions and variables in $\mathcal{V}^\star$, respectively and t, s are terms).

1. $\boldsymbol{G}(x.i \approx s \Rightarrow \mathtt{A}(x))$ for all $i \in \{1, \dots, \kappa\}$.
2. $\boldsymbol{G}(a \Rightarrow (\psi \Rightarrow \boldsymbol{X}\, spc(\psi, a)))$ (if $spc(\psi, a)$ is defined).
3. $\boldsymbol{G}(a \Rightarrow (wpc(\psi, a) \Leftrightarrow \boldsymbol{X}\, \psi))$ (if $wpc(\psi, a)$ is defined).
4. $\boldsymbol{G}(\mathtt{A}(x) \Rightarrow (\psi \Leftrightarrow \bigvee_{\xi \in \langle \psi \rangle_x} \xi)) \wedge \boldsymbol{G}(\psi \Rightarrow \bigwedge_{y \in v_{\mapsto}(\psi)} \mathtt{A}(y))$.
5. $\boldsymbol{G}(\exists \boldsymbol{u}.(x \mapsto (x_1, \dots, x_\kappa) * \psi) \Rightarrow (x.i \approx y \Leftrightarrow \exists \boldsymbol{u}.(x \mapsto (x_1, \dots, x_\kappa) * \psi * x_i \simeq y)))$, if $y \in \mathcal{V}^\star$.
6. $\boldsymbol{G}(\exists \boldsymbol{u}.(x \mapsto (x_1, \dots, x_\kappa) * y \mapsto (y_1, \dots, y_\kappa) * \psi) \Rightarrow (x.i \approx y.j \Leftrightarrow \exists \boldsymbol{u}.(x \mapsto (x_1, \dots, x_\kappa) * y \mapsto (y_1, \dots, y_\kappa) * \psi * x_i \simeq y_j)))$.
7. $\boldsymbol{G}((\exists \boldsymbol{u}.\psi) \Rightarrow (x \approx y \Leftrightarrow \exists \boldsymbol{u}.(\psi * x \simeq y)))$.

8. $\boldsymbol{G}((\texttt{pass} \vee t := s \vee \texttt{test}(\gamma)) \Rightarrow (\texttt{A}(x) \Leftrightarrow \boldsymbol{X}\, \texttt{A}(x)))$, where $t \neq x$.
9. $\boldsymbol{G}(\texttt{free}(x) \Rightarrow \bigwedge_{y \in \mathcal{V}^\star}((x \approx y \Rightarrow \boldsymbol{X}\, \neg\texttt{A}(y))) \wedge (x \not\approx y \Rightarrow (\texttt{A}(y) \Leftrightarrow \boldsymbol{X}\, \texttt{A}(y))))$.
10. $\boldsymbol{G}(\texttt{alloc}(x) \Rightarrow \bigwedge_{y \in \mathcal{V}^\star}((x \approx y \Rightarrow \boldsymbol{X}\, \texttt{A}(y))) \wedge (x \not\approx y \Rightarrow (\texttt{A}(y) \Leftrightarrow \boldsymbol{X}\, \texttt{A}(y))))$.
11. $\boldsymbol{G}(\neg x \vee \neg y)$, if $x \neq y$ and (either x, y are both actions, or $\{x, y\} \subseteq Q$).
12. $\boldsymbol{G}(q \Rightarrow \bigvee_{(q,a,q') \in R}(a \wedge \boldsymbol{X}\, q))$.
13. $\boldsymbol{G}(a \Rightarrow \mathit{pre}(a))$.
14. $\bigwedge_{\psi \in S^+} \phi \wedge \bigwedge_{x \in V} \neg\texttt{A}(x) \Rightarrow \bigvee_{\xi \in S^-} \phi$, if (S^+, S^-, X) is a unsatisfiable heap constraint. This formula is denoted by $\Gamma(S^+, S^-, X)$ in the following.

This set of axioms is infinite, as the set of symbolic heaps is infinite. To ensure termination, we need to further restrict the axioms. To this aim, we define (given a symbolic heap ϕ) two sets $\texttt{Fw}(\mathcal{S}, \phi)$ and $\texttt{Bw}(\mathcal{S}, \phi, \Phi)$, which, informally, contain triples (ψ, q, X), where ψ denotes a symbolic heap obtained by (forward or backward) propagation along the runs in $\mathcal{S}$ (starting from formulas occurring in the initial entailment), and q is the corresponding state. The set X contains variables that either occur in $v_{\mapsto}(\phi)$ or are known to be non allocated at state q (this information is essential for finiteness because it allows one to "block" some generation rules). The sets are defined inductively as follows:

- $(\phi, q_I, \emptyset) \in \texttt{Fw}(\mathcal{S}, \phi)$, and if $q \in Q$ and ψ occurs in Φ then $(\psi, q, \emptyset) \in \texttt{Bw}(\mathcal{S}, \phi, \Phi)$.
- If $(\psi, q, X) \in \texttt{Fw}(\mathcal{S}, \phi)$, $(q, a, q') \in R$ and $\phi' = \mathit{spc}(\psi, a)$ then $(\psi', q', X') \in \texttt{Fw}(\mathcal{S}, \phi)$, where $X' = X$ if a is not of the form $x := t$ with $x \in \mathcal{V}^\star$ and otherwise $X' = \emptyset$.
- If $(\psi, q', X) \in \texttt{Bw}(\mathcal{S}, \phi, \Phi)$, $(q, a, q') \in R$ and $\phi' = \mathit{wpc}(\psi, a)$ then $(\phi', q, X') \in \texttt{Bw}(\mathcal{S}, \phi, \Phi)$, where $X' = \emptyset$ if a is of the form $x := t$ with $x \in \mathcal{V}^\star$, and $X' = X$ otherwise.
- If $(\psi, q, X) \in \texttt{Fw}(\mathcal{S}, \phi)$ (resp. $(\psi, q, X) \in \texttt{Bw}(\mathcal{S}, \phi, \Phi)$) and $\xi \in \langle\psi\rangle_x$ with $x \in \mathcal{V}^\star \setminus X$ then $(\xi, q, X \cup \{x\}) \in \texttt{Fw}(\mathcal{S}, \phi)$ (resp. $(\xi, q, X \cup \{x\}) \in \texttt{Bw}(\mathcal{S}, \phi, \Phi)$).
- If $(\exists \boldsymbol{u}.\psi, q, X) \in \texttt{Fw}(\mathcal{S}, \phi)$ then $(\exists \boldsymbol{u}.(\psi \wedge x \simeq y), q, X) \in \texttt{Bw}(\mathcal{S}, \phi, \Phi)$, for all $x, y \in \mathit{fv}(\psi) \cup \mathcal{V}^\star$.

The sets $\texttt{Fw}(\mathcal{S}, \phi)$ and $\texttt{Bw}(\mathcal{S}, \phi, \Phi)$ are finite (up to some simplifications) if $\mathcal{S}$ is oriented (see Lemma 41 in Appendix D). We denote by $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi)$ the set of axioms satisfying the following conditions. For Axiom 3 we require that the considered symbolic heap ψ occurs in some triple in $\texttt{Bw}(\mathcal{S}, \phi, \Phi)$. For Axiom 14 all the symbolic heaps in S^+ and S^- must occur in $\texttt{Bw}(\mathcal{S}, \phi, \Phi)$. For Axiom 4, ψ must occur in either $\texttt{Fw}(\mathcal{S}, \phi)$ or $\texttt{Bw}(\mathcal{S}, \phi, \Phi)$. For 5, 6 and 7, the symbolic heap at the left-hand side of $\Rightarrow$ must occur in $\texttt{Fw}(\mathcal{S}, \phi)$ (which entails that the one occurring at the right-hand side occurs in $\texttt{Bw}(\mathcal{S}, \phi, \Phi)$). The following theorems relate the considered entailment problem with standard LTL satisfiability.

▶ **Theorem 38.** *Every LTL model $\mathcal{I}$ that is compatible with $(\mathfrak{s}, \mathfrak{h})$ and $\mathcal{S}$ w.r.t. all symbolic heaps occurring in $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi) \cup \{\phi, q_I, \Phi\}$ satisfies $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi) \cup \{\phi, q_I, \Phi\}$.*

Proof (Sketch). The soundness of Axioms 2 and 3 stems from Lemmata 29 and 32, respectively. The soundness of Axiom 13 stems from Proposition 26. Axioms 12 and 11 encode the semantics of actions and states, according to the transition system $\mathcal{S}$. The soundness of Axiom 4 is a consequence of Lemmata 36 and 37. The soundness of Axioms 14 follows from the semantics of heap constraints. The soundness of Axioms 8,9, 10 is a consequence of Definition 11. Finally, the soundness of Axioms 1, 5, 6 and 7 stems from the semantics of atomic conditions (Axioms 5, 6 and 7 embed conditions of the form $t \simeq s$ into symbolic heaps). ◀

▶ **Theorem 39.** *If $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi) \cup \{\phi, q_I, \Phi\}$ admits an LTL model $\mathcal{I}$ then there exists a structure $(\mathfrak{s}, \mathfrak{h})$ such that $\mathcal{I}$ is compatible with $(\mathfrak{s}, \mathfrak{h})$ and $\mathcal{S}$, w.r.t. ϕ and all symbolic heaps in Φ.*

9 Proof Procedure

Algorithm 1 Entailment Checking Algorithm.

```
Require: A progressing, connected and established SID $\mathcal{R}$, an oriented transition system $\mathcal{S}$,
Require: a symbolic heap $\phi$ and an LTL formula $\Phi$
  $\mathcal{A} \leftarrow \{\phi, q_I, \neg\Phi\}$
  while $\mathcal{A}$ admits an LTL interpretation $\mathcal{I}$ do
    $S^+ \leftarrow \{\phi \in \mathbb{A}_{\mathcal{S}} \mid \mathcal{I}(\phi, 0) = true,\ \phi$ is a symbolic heap$\}$
    $S^- \leftarrow \{\phi \in \mathbb{A}_{\mathcal{S}} \mid \mathcal{I}(\phi, 0) = false,\ \phi$ is a symbolic heap$\}$
    $X \leftarrow \{x \in \mathcal{V}^\star \mid \mathcal{I}(\phi, 0) \not\models \mathtt{A}(x)\ \}$
    if $(S^+, S^-, X)$ is unsatisfiable {This test is decidable by Lemma 9} then
      $\mathcal{A} \leftarrow \mathcal{A} \cup \Gamma(S^+, S^-, X)$
    else
      Let $(\mathfrak{s}, \mathfrak{h})$ be an $\mathcal{R}$-model of $(S^+, S^-, X)$
      if $r_{\mathcal{I}}$ is defined and $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}}^{\mathcal{S}/r_{\mathcal{I}}} \neg\Phi$ {the test is decidable by Lemma 22} then
        Return $(\mathfrak{s}, \mathfrak{h})$
      else
        Let $\Psi$ be a formula in $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi)$ s.t. $(\mathfrak{s}, \mathfrak{h}) \not\models_{\mathcal{R}}^{\mathcal{S}} \Psi$ {$\Psi$ exists by Theorem 39}
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{\Psi\}$
      end if
    end if
  end while
  Return $\top$
```

Even if $\mathcal{S}$ is oriented, the set $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi)$ is exponential w.r.t. the size of $\mathcal{R}$, ϕ and $\mathcal{S}$, and only a small part of this set will be relevant, hence computing all axioms explicitly is not practical. Algorithm 1 computes these axioms on demand, in the spirit of the well-known *DPLL*($\mathcal{T}$) procedure (see, e.g., [9]) by calling external tools to solve LTL and SL satisfiability problems. The idea is to construct an LTL interpretation and to refine it incrementally by adding relevant axioms until we get either a model that is compatible with some SL structure, or a set of axioms that is unsatisfiable (in LTL). For all LTL interpretations $\mathcal{I}$, $r_{\mathcal{I}}$ is the sequence $(q_i, a_i)_{i \in \mathbb{N}}$ (if it exists) such that q_i is the unique state in Q (resp. the only action) with $\mathcal{I}(q_i, i) = true$ (resp. $\mathcal{I}(a_i, i) = true$).

▶ **Theorem 40.** *If Algorithm 1 returns $\top$ then the entailment $\phi \models_{\mathcal{R}}^{\mathcal{S}} \Phi$ holds. If it returns an SL structure $(\mathfrak{s}, \mathfrak{h})$ then $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ and $(\mathfrak{s}, \mathfrak{h}) \not\models_{\mathcal{R}}^{\mathcal{S}} \Phi$. Moreover, if $\mathcal{S}$ is oriented then the algorithm always terminates.*

Proof. Termination is immediate (if $\mathcal{S}$ is oriented) since at each iteration one new formula from $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi)$ is added in $\mathcal{A}$ and the set $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi)$ is finite (as $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$ and $\mathtt{Fw}(\mathcal{S}, \phi)$ are both finite). If $\top$ is returned then by definition of the algorithm $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi) \cup \{q_I, \phi, \neg\Phi\}$ is unsatisfiable thus the entailment $\phi \models_{\mathcal{R}}^{\mathcal{S}} \Phi$ is valid by Theorem 38. If the algorithm returns a structure $(\mathfrak{s}, \mathfrak{h})$ then by definition $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}}^{\mathcal{S}/(q_i, a_i)_{i \in \mathbb{N}}} \neg\Phi$ for some sequence $(q_i, a_i)_{i \in \mathbb{N}}$, thus there is a run $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)_{i \in \mathbb{N}}$ and a compatible LTL interpretation $\mathcal{I}$ such that $\mathcal{I} \not\models \Phi$. ◀

10 Discussion

A natural issue is to determine whether Algorithm 1 is complete for refutation (when $\mathcal{S}$ is not oriented), i.e., whether it always returns a counter model if the entailment is not valid (by Theorem 23 it cannot be complete for validity). Another natural continuation is to extend the expressive power of the logic by considering more complex temporal connectives (to allow for quantification over paths). It would also be interesting to extend the language in order to handle more complex (possibly non deterministic) actions. For instance, it should be noticed that actions in our framework cannot create new locations (as evidenced by Proposition 13). This is important, because, otherwise, since universal quantification is not allowed, the corresponding pre/post-conditions could not be expressed in the language. This entails that C-like allocations for instance are not built-in: they must be performed by handling a stack of available locations, allocated in the symbolic heap describing the initial state of the system by an atom such as $\mathtt{lseg}(x, y)$ (an instruction such as $\mathtt{malloc}(z)$ can be simulated by two actions $z := x$ and $x := x.1$). The complexity of the entailment problem for oriented systems also deserves to be precisely identified (it is 2-EXPTIME hard by [4]).

References

1 Callum Bannister, Peter Höfner, and Gerwin Klein. Backwards and forwards with separation logic. In Jeremy Avigad and Assia Mahboubi, editors, *ITP 2018, FloC 2018, Oxford, UK, July 9–12, 2018, Proceedings*, volume 10895 of *LNCS*, pages 68–87. Springer, 2018.

2 Rémi Brochenin, Stéphane Demri, and Étienne Lozes. Reasoning about sequences of memory states. In Sergei N. Artëmov and Anil Nerode, editors, *LFCS 2007, New York, NY, USA, June 4–7, 2007, Proceedings*, volume 4514 of *LNCS*, pages 100–114. Springer, 2007.

3 James Brotherston, Carsten Fuhs, Juan Antonio Navarro Pérez, and Nikos Gorogiannis. A decision procedure for satisfiability in separation logic with inductive predicates. In Thomas A. Henzinger and Dale Miller, editors, *CSL-LICS '14, Vienna, Austria, July 14–18, 2014*, pages 25:1–25:10. ACM, 2014.

4 Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Entailment checking in separation logic with inductive definitions is 2-exptime hard. In *LPAR 2020, Alicante, Spain, May 22–27, 2020*, volume 73 of *EPiC Series in Computing*, pages 191–211. EasyChair, 2020.

5 Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Decidable entailments in separation logic with inductive definitions: Beyond establishment. In *CSL 2021*, EPiC Series in Computing. EasyChair, 2021.

6 Didier Galmiche and Daniel Méry. Labelled tableaux for linear time bunched implication logic. In *ASL 2022 (Workshop on Advancing Separation Logic)*, 2022.

7 Radu Iosif, Adam Rogalewicz, and Jiri Simacek. The tree width of separation logic with recursive definitions. In *Proc. of CADE-24*, volume 7898 of *LNCS*, 2013.

8 Norihiro Kamide. Temporal BI: proof system, semantics and translations. *Theor. Comput. Sci.*, 492:40–69, 2013.

9 Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.

10 Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bull. Symb. Log.*, 5(2):215–244, 1999.

11 Jens Pagel, Christoph Matheja, and Florian Zuleger. Complete entailment checking for separation logic with inductive definitions, 2020. `arXiv:2002.01202`.

12 Jens Pagel and Florian Zuleger. Beyond symbolic heaps: Deciding separation logic with inductive definitions. In *LPAR-23*, volume 73 of *EPiC Series in Computing*, pages 390–408. EasyChair, 2020.

13 Amir Pnueli. The temporal logic of programs. In *FOCS, Providence, Rhode Island, USA*, pages 46–57. IEEE Computer Society, 1977.
14 J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS'02*, 2002.

A Proof of Lemma 9

We use the algorithm developed in [5] to test the validity of entailments between SL formulas (if the considered SID is progressing, connected and established), combined with the technique devised in [12] to cope with conjunctions (see also [11]). Let $X = \{x_1, \dots, x_n\}$, where the order on the x_i is arbitrary. For every $i = 1, \dots, n$, we denote by Ψ_i the formula: $(\bigvee_{j=1}^{i-1} x_i \simeq x_j) \vee x_i \mapsto (x_i, \dots, x_i)$. Let $\Psi = \Psi_1 * \dots * \Psi_n$. By definition, if $(\mathfrak{s}, \mathfrak{h}') \models_{\mathcal{R}} \Psi$ then $dom(\mathfrak{h}') = \{\mathfrak{s}(x) \mid x \in X\}$, hence, for every structure $(\mathfrak{s}, \mathfrak{h})$, there is at most one heap $\mathfrak{h}' \subseteq \mathfrak{h}$ with $(\mathfrak{s}, \mathfrak{h}') \models_{\mathcal{R}} \Psi$. Moreover, for all stores $\mathfrak{s}$, we have $(\mathfrak{s}, \mathfrak{h}_{\mathfrak{s}}) \models_{\mathcal{R}} \Psi$, where $\mathfrak{h}_{\mathfrak{s}}$ denotes the heap: $\{(\ell, \dots, \ell) \mid \exists x \in X \text{ s.t. } \ell = \mathfrak{s}(x)\}$. Let $\Phi = \bigwedge_{\phi \in S^+} (\Psi * \phi) \wedge \neg(\bigvee_{\phi \in S^-} (\Psi * \phi))$. Note that since S^+ is not empty, Φ is a guarded formula (as defined in [12, Fig. 1]), except that it contains existential quantifiers (the fact that S^+ is non empty is essential, as otherwise the negation would not be guarded). The satisfiability of Φ can be tested by combining the techniques devised in [12] and [5]. The idea is to compute an abstraction of the possible models of Φ bottom-up. Points-to atoms, inductive predicates, separating conjunctions and existential quantifications can be handled as explained in [5], whereas conjunctions and guarded negations are handled as it is done in [12]. We prove that (S^+, S^-, X) is satisfiable iff Φ is satisfiable:

- Assume that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} (S^+, S^-, X)$. Then $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ for all $\phi \in S^+$, $(\mathfrak{s}, \mathfrak{h}) \not\models_{\mathcal{R}} \phi$ for all $\phi \in S^-$, and $\mathfrak{s}(x) \notin dom(\mathfrak{h})$ for all $x \in X$. Then $\mathfrak{h}_{\mathfrak{s}}$ and $\mathfrak{h}$ are disjoint, thus we get $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_{\mathfrak{s}}) \models_{\mathcal{R}} \phi * \Psi$ for all $\phi \in S^+$. If $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_{\mathfrak{s}}) \models_{\mathcal{R}} \phi * \Psi$ for some $\phi \in S^-$ then since $\mathfrak{h}_{\mathfrak{s}}$ is the unique heap such that $(\mathfrak{s}, \mathfrak{h}_{\mathfrak{s}}) \models_{\mathcal{R}} \Psi$, we deduce that we must have $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$, which contradicts our assumption. Thus $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_{\mathfrak{s}}) \models_{\mathcal{R}} \Phi$.
- Assume that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \Phi$. Then, we get $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi * \Psi$, for all $\phi \in S^+$. Since $\mathfrak{h}_{\mathfrak{s}}$ is the unique heap such that $(\mathfrak{s}, \mathfrak{h}_{\mathfrak{s}}) \models_{\mathcal{R}} \Psi$, this entails that $(\mathfrak{s}, \mathfrak{h}') \models_{\mathcal{R}} \phi$, with $\mathfrak{h}' = \mathfrak{h} \setminus \mathfrak{h}_{\mathfrak{s}}$. Since $dom(\mathfrak{h}_{\mathfrak{s}}) = \{\mathfrak{s}(x) \mid x \in X\}$ we get $\mathfrak{s}(x) \notin dom(\mathfrak{h}')$, for all $x \in X$. If $(\mathfrak{s}, \mathfrak{h}') \models_{\mathcal{R}} \phi$, for some $\phi \in S^-$ then we deduce $(\mathfrak{s}, \mathfrak{h}' \uplus \mathfrak{h}_{\mathfrak{s}}) \models_{\mathcal{R}} \phi * \Psi$, which contradicts our hypothesis. Thus $(\mathfrak{s}, \mathfrak{h}') \models_{\mathcal{R}} (S^+, S^-, X)$.

B Proof of Lemma 36

Assume that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \psi$. We show, by induction on $|\mathfrak{h}|$, that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$. If $x \in v_{\mapsto}(\phi)$ then $\langle \phi \rangle_x = \{\phi\}$ thus $\phi = \psi$ and the proof is immediate. Otherwise, we distinguish the following cases, following Definition 34:

- $\phi' = \exists \boldsymbol{u}.(x \simeq x' * x \mapsto (\boldsymbol{y}) * \psi)$ and $\phi = \exists \boldsymbol{u}.(x' \mapsto (\boldsymbol{y}) * \psi)$. It is clear that $\phi' \models_{\mathcal{R}} \phi$.
- $\phi' = \exists \boldsymbol{u} \exists \boldsymbol{v}.(x \simeq x' * \psi' * \psi)$, and $\phi = \exists \boldsymbol{u}.(p(x', \boldsymbol{y}) * \psi)$ with $p(x, \boldsymbol{y}) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.\psi'$. In this case, we get $\phi' \models_{\mathcal{R}} \exists \boldsymbol{u}.(x \simeq x' * p(x, \boldsymbol{y}) * \psi)$, thus $\phi' \models_{\mathcal{R}} \exists \boldsymbol{u}.(p(x', \boldsymbol{y}) * \psi) = \phi$.
- $\phi' = \exists \boldsymbol{u} \exists \boldsymbol{v} \exists z_1 \dots \exists z_m.((q \multimap p)(\boldsymbol{y}, z_1, \dots, z_m) * z_1 \simeq x * \psi' * \psi)$ and $\phi = \exists \boldsymbol{u}.(p(\boldsymbol{y}) * \psi)$, with $q(x, z_2, \dots, z_m) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.\psi'$. Then we get $\phi' \models_{\mathcal{R}} \exists \boldsymbol{u} \exists z_1 \dots \exists z_m.((q \multimap p)(\boldsymbol{y}, z_1, \dots, z_m) * z_1 \simeq x * q(x, z_2, \dots, z_m) * \psi)$, and by definition of the rules associated with the predicate $(q \multimap p)$, one of the following conditions holds (with $\boldsymbol{y} = (y_1, \dots, y_n)$):
  - $\phi' \models_{\mathcal{R}} \exists \boldsymbol{u} \exists z_1 \dots \exists z_m \exists \boldsymbol{w} \exists \boldsymbol{v}.(y_1 \mapsto (\boldsymbol{y}') * (q \multimap p')(\boldsymbol{z}', z_1, \dots, z_m) * \psi'' * z_1 \simeq x * \psi' * \psi)$ with $p(\boldsymbol{y}) \Leftarrow_{\mathcal{R}} y_1 \mapsto (\boldsymbol{y}') * p'(\boldsymbol{z}') * \psi''$. This entails that there exists a store $\mathfrak{s}'$ coinciding with $\mathfrak{s}$ with all the variables not occurring in $\boldsymbol{u}, z_1, \dots, z_m, \boldsymbol{w}, \boldsymbol{v}$ and disjoint

heaps $\mathfrak{h}', \mathfrak{h}''$ such that $\mathfrak{h} = \mathfrak{h}' \uplus \mathfrak{h}''$, $(\mathfrak{s}', \mathfrak{h}') \models_{\mathcal{R}} y_1 \mapsto (\boldsymbol{y}') * \psi''$ and $(\mathfrak{s}', \mathfrak{h}'') \models_{\mathcal{R}} (q \multimap p')(\boldsymbol{z}', z_1, \dots, z_m) * z_1 \simeq x * \psi' * \psi$. This entails that $\mathfrak{h}' \neq \emptyset$ thus $|\mathfrak{h}| > |\mathfrak{h}''|$. By definition, $(q \multimap p')(\boldsymbol{z}', z_1, \dots, z_m) * z_1 \simeq x * \psi' * \psi \in \langle p'(\boldsymbol{z}') * \psi \rangle_x$, hence by the induction hypothesis we get $(\mathfrak{s}', \mathfrak{h}'') \models_{\mathcal{R}} p'(\boldsymbol{z}') * \psi$, so that $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} y_1 \mapsto (\boldsymbol{y}') * \psi'' * p'(\boldsymbol{z}') * \psi$, hence $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{u}.(p(\boldsymbol{y}) * \psi) = \phi$.

- $\phi' \models_{\mathcal{R}} \exists \boldsymbol{u} \exists z_1 \dots \exists z_m \exists \boldsymbol{v} \exists \boldsymbol{w}.(y_1 \mapsto (\boldsymbol{y}') * \boldsymbol{z}' \simeq (z_1, \dots, z_m) * \psi'' * z_1 \simeq x * \psi' * \psi)$ with $p(\boldsymbol{y}) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{w}.(y_1 \mapsto (\boldsymbol{y}') * q(\boldsymbol{z}') * \psi'')$. Since $q(x, z_2, \dots, z_m) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.\psi'$, we deduce that $\phi' \models_{\mathcal{R}} \exists \boldsymbol{u} \exists z_1 \dots \exists z_m \exists \boldsymbol{w}.(y_1 \mapsto (\boldsymbol{y}') * \boldsymbol{z}' \simeq (z_1, \dots, z_m) * \psi'' * z_1 \simeq x * q(x, z_2, \dots, z_m) * \psi)$, thus $\phi' \models_{\mathcal{R}} \exists \boldsymbol{u} \exists \boldsymbol{w}.(y_1 \mapsto (\boldsymbol{y}') * \psi'' * q(\boldsymbol{z}') * \psi)$, hence $\phi' \models_{\mathcal{R}} \exists \boldsymbol{u}.(p(\boldsymbol{y}) * \psi) = \phi$.

C Proof of Lemma 37

Assume that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ and that $\mathfrak{s}(x) \in dom(\mathfrak{h})$. We show, by induction on $|\mathfrak{h}|$, that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \psi$, for some $\psi \in \langle \phi \rangle_x$. The symbolic heap ϕ is necessarily of the form $\exists \boldsymbol{u}.(\phi_1 * \dots * \phi_k)$, where the $\phi_1, \dots, \phi_k$ are atoms. We assume by α-renaming that x does not occur in $\boldsymbol{u}$. By definition of the semantics of SL, there exists a store $\mathfrak{s}'$ (coinciding with $\mathfrak{s}$ on all variables not occurring in $\boldsymbol{u}$) and disjoint heaps $\mathfrak{h}_i$ such that $(\mathfrak{s}', \mathfrak{h}_i) \models_{\mathcal{R}} \phi_i$ (for all $i = 1, \dots, n$) and $\mathfrak{h} = \mathfrak{h}_1 \uplus \dots \uplus \mathfrak{h}_k$. Since $\mathfrak{s}(x) \in dom(\mathfrak{h})$, necessarily $\mathfrak{s}(x) \in dom(\mathfrak{h}_i)$ for some $i = 1, \dots, k$, say $i = 1$. Let $\phi' = \phi_2 * \dots * \phi_k$. We distinguish several cases.

- Assume that ϕ_1 is a points-to atom $x' \mapsto (\boldsymbol{y})$. If $x = x'$, then $x \in v_{\mapsto}(\phi)$, so that $\langle \phi \rangle_x = \{\phi\}$, hence the proof is immediate. Otherwise, since (by definition of the semantics of SL) $dom(\mathfrak{h}_1) = \{\mathfrak{s}'(x')\}$ and $\mathfrak{s}(x) \in \mathfrak{h}_1$ we must have $\mathfrak{s}(x') = \mathfrak{s}(x) = \mathfrak{s}'(x)$. By Definition 34 (2a), $\langle \phi \rangle_x$ contains a formula of the form $\exists \boldsymbol{u}.(x \mapsto (\boldsymbol{y}) * x \simeq x' * \phi')$. We have $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} x \mapsto (\boldsymbol{y}) * x \simeq x' * \phi'$, so that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{u}.(x \mapsto (\boldsymbol{y}) * x \simeq x' * \phi')$.
- Assume that ϕ_1 is a predicate atom $p(x', \boldsymbol{y})$ and that $\mathfrak{s}'(x') = \mathfrak{s}(x)$. Then we get $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} p(x, \boldsymbol{y}) * x \simeq x' * \phi'$, so that $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{v}.(\psi' * x \simeq x' * \phi')$ with $p(x, \boldsymbol{y}) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.\psi'$. Thus $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{u} \exists \boldsymbol{v}.(\psi' * x \simeq x' * \phi')$, and by Def. 34 (2b), this formula is in $\langle \phi \rangle_x$.
- Finally, assume that ϕ_1 is a predicate atom $p(x', \boldsymbol{y})$ and that $\mathfrak{s}'(y_1) \neq \mathfrak{s}(x)$. Necessarily, $p(x', \boldsymbol{y}) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.(x' \mapsto (\boldsymbol{y}') * \psi)$ with $(\mathfrak{s}'', \mathfrak{h}_1) \models_{\mathcal{R}} x' \mapsto (\boldsymbol{y}') * \psi$, for some store $\mathfrak{s}''$ coinciding with $\mathfrak{s}'$ on all variables not occurring in $\boldsymbol{v}$. Since $\mathfrak{s}'(y_1) \neq \mathfrak{s}(x)$ and $\mathfrak{s}(x) \in dom(\mathfrak{h}_1)$, ψ must be of the form $p'(\boldsymbol{z}) * \psi'$ (with possibly $\psi' = emp$) and there exist disjoint heaps $\mathfrak{h}', \mathfrak{h}''$ such that $\mathfrak{h}_1 = \mathfrak{h}' \uplus \mathfrak{h}''$, $(\mathfrak{s}'', \mathfrak{h}') \models_{\mathcal{R}} x' \mapsto (\boldsymbol{y}') * \psi'$ and $(\mathfrak{s}'', \mathfrak{h}'') \models_{\mathcal{R}} p'(\boldsymbol{z})$. This entails that $\mathfrak{h}' \neq \emptyset$, thus $|\mathfrak{h}''| < |\mathfrak{h}|$, and by the induction hypothesis, we deduce that $\langle p'(\boldsymbol{z}) \rangle_x$ contains a formula ψ'' such that $(\mathfrak{s}'', \mathfrak{h}'') \models_{\mathcal{R}} \psi''$. We get $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{u} \exists \boldsymbol{v}.(x' \mapsto (\boldsymbol{y}') * \psi'' * \psi' * \phi')$. By Definition 34, ψ'' is of one of the following forms:

  2b $\psi'' = \exists \boldsymbol{w}.(\xi * z_1 \simeq x)$, with $\boldsymbol{z} = (z_1, \dots, z_m)$ and $p'(x, z_2, \dots, z_m) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{w}.\xi$. Then we get $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{u} \exists \boldsymbol{v} \exists \boldsymbol{w}.(x' \mapsto (\boldsymbol{y}') * \xi * z_1 \simeq x * \psi' * \phi')$. By definition of the rules defining $(p' \multimap p)$, we have: $(p' \multimap p)(x', \boldsymbol{y}, \boldsymbol{z})) \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.(x' \mapsto (\boldsymbol{y}') * \psi' * \boldsymbol{z} \simeq \boldsymbol{z})$, so that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{u} \exists \boldsymbol{w}.((p' \multimap p)(x', \boldsymbol{y}, \boldsymbol{z})) * \xi * z_1 \simeq x * \phi')$, hence $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{u} \exists \boldsymbol{w} \exists \boldsymbol{z}'.((p' \multimap p)(x', \boldsymbol{y}, \boldsymbol{z}')) * \xi * z_1' \simeq x * \phi')$, where $\boldsymbol{z}' = (z_1', \dots, z_m')$ is a vector of fresh pairwise distinct variables. By Definition 34 (2c), the latter formula occurs in $\langle \phi \rangle_x$.

  2c $\psi'' = \exists z_1', \dots, z_m'.((q \multimap p')(\boldsymbol{z}, z_1', \dots, z_m') * z_1' \simeq x * \xi)$, with $q(x, z_2', \dots, z_m') \Leftarrow_{\mathcal{R}} \xi$. We get $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{u} \exists \boldsymbol{v} \exists z_1', \dots, z_m'.(x' \mapsto (\boldsymbol{y}') * (q \multimap p')(\boldsymbol{z}, z_1', \dots, z_m') * z_1' \simeq x * \xi * \psi' * \phi')$. By definition of the rules defining $(q \multimap p)$, we have $(q \multimap p)(x', \boldsymbol{y}, z_1', \dots, z_m') \Leftarrow_{\mathcal{R}} \exists \boldsymbol{v}.(x' \mapsto (\boldsymbol{y}') * (q \multimap p')(\boldsymbol{z}, z_1', \dots, z_m') * \psi')$, thus $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \boldsymbol{u} \exists z_1', \dots, z_m'.((q \multimap p)(x', \boldsymbol{y}, z_1', \dots, z_m') * z_1' \simeq x * \xi * \phi')$. By Definition 34 (2c), this formula is in $\langle \phi \rangle_x$.

D Finiteness of $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$ and $\mathtt{Fw}(\mathcal{S}, \phi)$

We write $\phi \rightarrow_s \psi$ if ψ is obtained from ϕ by using one of the above simplification rules.

$$\mathtt{C}_{\simeq} : \exists \boldsymbol{u}.(x \not\simeq x * \xi) \rightarrow \bot \qquad \mathtt{E}_{\not\simeq} : \exists \boldsymbol{u} \exists x.(x \not\simeq x_1 * \ldots x \not\simeq x_n * \xi) \rightarrow \exists \boldsymbol{u}.\xi \text{ if } x \not\in fv(\xi) \cup \{x_1, \ldots, x_n\}$$

$$\mathtt{C}_{*} : \exists \boldsymbol{u}.(x \mapsto (\boldsymbol{y}) * x \mapsto (\boldsymbol{z}) * \xi) \rightarrow \bot \qquad \mathtt{E}_{\simeq} : \exists \boldsymbol{u} \exists x.(x \simeq y * \xi) \rightarrow \exists \boldsymbol{u}.\xi\{x \leftarrow y\}$$

It is easy to verify that $\rightarrow_s$ is well-founded, and that $\phi \rightarrow_s \psi \implies \phi \equiv_{\mathcal{R}} \psi$.

▶ **Lemma 41.** *If $\mathcal{S}$ is oriented then the sets $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$ and $\mathtt{Fw}(\mathcal{S}, \phi)$ are finite (up to associativity and commutativity of $*$, α-renaming and equivalence w.r.t. $\rightarrow_s$).*

Proof. We assume that all symbolic heaps are in normal form w.r.t. $\rightarrow_s$. Let $\mathcal{S} = (Q, R, q_I)$. We give the proof for $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, the set $\mathtt{Fw}(\mathcal{S}, \phi)$ can be handled in a similar way (the only difference is that one must consider the order $\preceq_{\mathcal{S}}$ instead of $\succeq_{\mathcal{S}}$, and that $\mathtt{Fw}(\mathcal{S}, \phi)$ does not depend on $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, while $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$ depends on $\mathtt{Fw}(\mathcal{S}, \phi)$). If $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$ is infinite then by definition (assuming that $\mathtt{Fw}(\mathcal{S}, \phi)$ is finite) by Köning's lemma there must exist an infinite sequence of pairwise distinct triples (ϕ_i, q_i, X_i) $(i \in \mathbb{N})$ such that $X_0 = \emptyset$ and for every $i \in \mathbb{N}$, one of the following conditions holds:

- there exists an action a_i such that $(q_{i+1}, a_i, q_i) \in R$ and $\phi_{i+1} = wpc(\phi_i, a_i)$, where $X_{i+1} = X_i \setminus \{x_i\}$ if a_i is of the form $x_i := t_i$ with $x_i \in \mathcal{V}^\star$, and $X_{i+1} = \emptyset$ otherwise, or;
- $\phi_{i+1} = \psi_i$ with $\psi_i \in \langle \phi_i \rangle_{x_i}$ for some variable $x \in \mathcal{V}^\star \setminus X_i$, $q_{i+1} = q_i$ and $X_{i+1} = X_i \cup \{x_i\}$.

In both cases we have $q_{i+1} \succeq_{\mathcal{S}} q_i$, by definition of $\succeq_{\mathcal{S}}$ (see Definition 14). Since the set of states Q is finite, necessarily there exists a natural number k such that, $q_{i+1} \not\succ_{\mathcal{S}} q_i$ holds for all $i \geq k$. Since by hypothesis $\mathcal{S}$ is oriented, this entails that R contains no transition of the form $(q_{i+1}, x_i := t_i, q_i)$ with $x_i \in \mathcal{V}^\star$ and $i \geq k$. Consequently, we must have $X_{i+1} \supseteq X_i$, for all $i \geq k$. By definition of $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, $X_i \subseteq \mathcal{V}^\star$ for all $i \in \mathbb{N}$, and since $\mathcal{V}^\star$ is finite we deduce that there exists $l \in \mathbb{N}$ such that $l \geq k$ and $X_i = X_{i+1}$ for all $i \geq l$. By definition of $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, this entails that ϕ_{i+1} must be of form $wpc(\phi_i, a_i)$, for all $i \geq l$, and a_i is not of the form $x_i := t_i$. Note that this implies that all the predicates symbols occurring in ϕ_i occur in ϕ_l (since all the predicates in $wpc(\phi_i, a_i)$ must occur in ϕ_i). For all $i \geq l$, we denote by n_i the number of atoms in ϕ_i that are not equational and not of the form $x \mapsto (\boldsymbol{y})$ with $x \in \mathcal{V}^\star$. By inspection of the different cases in Definition 30 (taking into account the fact that a_i is not of the form $x_i := t_i$), it is easy to check that $n_{i+1} = n_i$ holds for all $i \geq l$. Indeed, the only case in which $wpc(\phi_i, a_i)$ contains an atom that does not occur in ϕ_i is when this atom is either an equation or a points-to atom with a left-hand side in $\mathcal{V}^\star$ (furthermore, the simplification rules in $\rightarrow_s$ cannot add new atoms in the formula). By irreducibility w.r.t. the rule $\mathtt{C}_*$, this entails that the number of spatial atoms in ϕ_i (for $i \geq l$) is at most $card(\mathcal{V}^\star) + n_l$. Assume that ϕ_i contains an existential variable x that does not occur in a spatial atom. By irreducibility w.r.t. the rule $\mathtt{E}_{\simeq}$, x cannot occur in an equation. By irreducibility w.r.t. $\mathtt{C}_{\simeq}$, it cannot occur in a disequation $x \not\simeq x$. Thus the only atoms in which x occurs are of the form $x \not\simeq x_i$, with $x_i \neq x$, and the rule $\mathtt{E}_{\not\simeq}$ applies, which contradicts the fact that ϕ_i is in normal form w.r.t. $\rightarrow_s$. Consequently, all the existential variables in ϕ_i occur in a spatial atom. Since the number of such atoms is bounded, necessarily the number of existential variables is bounded. As both the set of free variables $\mathcal{V}^\star$ and the set of predicate symbols in ϕ_i is finite, this entails that there exist finitely many symbolic heaps ϕ_i (with $i \geq l$), which contradicts our assumption. ◀

E Proof of Theorem 39

We construct a run $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)_{i\in\mathbb{N}}$, and a corresponding sequence of triples $(\phi'_i, q_i, X_i)_{i\in\mathbb{N}}$ by induction on i, with $(\mathfrak{s}_0, \mathfrak{h}_0) \models_{\mathcal{R}} \phi$. We simultaneously establish the following inductive invariant:

a The equivalence $\mathcal{I}(\xi, i) = \mathit{true} \iff (\mathfrak{s}_i, \mathfrak{h}_i) \models_{\mathcal{R}} \xi$ holds for all atomic conditions ξ, and also for all symbolic heaps ξ such that there exists $q \in Q$ and $X \subseteq \mathcal{V}^\star$ with $(\xi, q, X) \in \mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, for all $x \in X \setminus v_{\mapsto}(\xi)$.

b For all $q \in Q$ and for all actions a, $\mathcal{I}(q, i) = \mathit{true}$ iff $q = q_i$ and $\mathcal{I}(a, i) = \mathit{true}$ iff $a = a_i$.

c $(\phi'_i, q_i, X_i) \in \mathtt{Fw}(\mathcal{S}, \phi)$ with $\mathcal{I}(\phi'_i, i) = \mathit{true}$ and $\forall x \in X_i \setminus v_{\mapsto}(\phi'_i) : \mathfrak{s}_i(x) \notin \mathit{dom}(\mathfrak{h}) \wedge (\mathfrak{s}_i, \mathfrak{h}_i) \not\models \mathtt{A}(x)$.

Note that the invariant entails in particular that $\mathcal{I}$ is compatible with $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)_{i\in\mathbb{N}}$, w.r.t. all symbolic heaps occurring in Φ. Indeed, by definition of $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, $(\psi, q, \emptyset) \in \mathtt{Bw}(\mathcal{S}, \phi, \Phi)$ for all symbolic heaps occurring in Φ and for all states q.

- **Base case $(i = 0)$.** Let $q_0 \overset{\text{def}}{=} q_I$, $\phi'_0 \overset{\text{def}}{=} \phi$ and $X_0 \overset{\text{def}}{=} \emptyset$. Let S^+ (resp. S^-) be the set of symbolic heaps ψ occurring in $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi)$, $\{\phi\}$ or Φ such that $\mathcal{I}(\psi, 0) = \mathit{true}$ (resp. $\mathcal{I}(\psi, 0) = \mathit{false}$). By hypothesis, $\Phi \in S^+$, thus $S^+ \neq \emptyset$. Let X be the set of variables x such that $\mathcal{I}(\mathtt{A}(x), 0) = \mathit{false}$. By definition, $\mathcal{I} \not\models \Gamma(S^+, S^-, X)$, thus, by Axiom 14, (S^+, S^-, X) cannot be unsatisfiable, and there exists a structure $(\mathfrak{s}_0, \mathfrak{h}_0)$ such that $(\mathfrak{s}_0, \mathfrak{h}_0) \models_{\mathcal{R}} (S^+, S^-, X)$. By construction, $\mathcal{I}(\xi, 0) = \mathit{true} \iff (\mathfrak{s}_0, \mathfrak{h}_0) \models_{\mathcal{R}} \xi$ holds for all symbolic heaps ξ occurring in $\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi)$, $\{\phi\}$ or Φ, and in particular, $(\mathfrak{s}_0, \mathfrak{h}_0) \models_{\mathcal{R}} \phi$. Still by construction, $\mathcal{I}(\mathtt{A}(x), 0) = \mathit{false} \implies (\mathfrak{s}_0, \mathfrak{h}_0) \not\models_{\mathcal{R}} \mathtt{A}(x)$. Conversely, if $\mathcal{I}(\mathtt{A}(x), 0) = \mathit{true}$, then by Axiom 4, necessarily $(\mathcal{I}, 0) \models \xi$, for some $\xi \in \langle\phi\rangle_x$, thus $(\mathfrak{s}_0, \mathfrak{h}_0) \models_{\mathcal{R}} \xi$ and by Lemma 35, we get $(\mathfrak{s}_0, \mathfrak{h}_0) \models_{\mathcal{R}} \mathtt{A}(x)$. Thus Property a holds for all symbolic heaps and for all conditions of the form $\mathtt{A}(x)$.
  By hypothesis we have $(\mathcal{I}, 0) \models q_I$, and, by Axiom 11, $(\mathcal{I}, 0) \models \neg q$, for all states $q \neq q_I$. By Axioms 12 and 11, there exists a unique action a_0 such that $(\mathcal{I}, 0) \models a_0$. Thus Property b holds.
  By definition of $\mathtt{Fw}(\mathcal{S}, \phi)$ we have $(\phi, q_I, \emptyset) \in \mathtt{Fw}(\mathcal{S}, \phi)$ thus Property c holds.
  It only remains to prove that Property a holds for all atomic conditions (other than those of the form $\mathtt{A}(x)$). Consider any atomic condition α, and assume that $\mathcal{I}(\alpha, 0) = \mathit{true}$ (the case whether $\mathcal{I}(\alpha, 0) = \mathit{false}$ is handled in a similar way). We show that $(\mathfrak{s}_0, \mathfrak{h}_0) \models \alpha$. Assume that α is of the form $x \approx y$, with $x, y \in \mathcal{V}^\star$. By definition ϕ'_0 is of the form $\exists \boldsymbol{u}.\phi'$, for some symbolic heap ϕ' containing no quantifier. By definition of $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$ we have $(\exists \boldsymbol{u}.(\phi' * x \simeq y), q_0, X_0) \in \mathtt{Bw}(\mathcal{S}, \phi, \Phi)$. By Axiom 7, since $\mathcal{I}(\phi'_0, 0) = \mathit{true}$, we get $\mathcal{I}(\exists \boldsymbol{u}.(\phi' * x \simeq y), 0) = \mathit{true}$, thus $(\mathfrak{s}_0, \mathfrak{h}_0) \models_{\mathcal{R}} \exists \boldsymbol{u}.(\phi' * x \simeq y)$ (by Property a, which has already been established for symbolic heaps). Thus $\mathfrak{s}_0(x) = \mathfrak{s}_0(y)$ and therefore $(\mathfrak{s}_0, \mathfrak{h}_0) \models_{\mathcal{R}} x \approx y$. Assume that α is of the form $x.i \approx y$, with $x, y \in \mathcal{V}^\star$. By Axiom 1 we must have $\mathcal{I}(\mathtt{A}(x), 0) = \mathit{true}$, hence by Axiom 4 we deduce that $\mathcal{I}(\psi, 0) = \mathit{true}$, for some $\psi \in \langle\phi'_0\rangle_x$ (note that, by definition of $\mathtt{Fw}(\mathcal{S}, \phi)$, we have $(\psi, q_0, X_0 \cup \{x\}) \in \mathtt{Fw}(\mathcal{S}, \phi)$). By Lemma 35, ψ is of the form $\exists \boldsymbol{v}.(x \mapsto (x_1, \dots, x_\kappa) * \psi')$. We have $(\exists \boldsymbol{v}.(x \mapsto (x_1, \dots, x_\kappa) * \psi' * x_i \simeq y), q_0, X_0 \cup \{x\}) \in \mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, thus by Axiom 5 we deduce that $\mathcal{I}(\exists \boldsymbol{v}.(x \mapsto (x_1, \dots, x_\kappa) * \psi' * x_i \simeq y), 0) = \mathit{true}$, so that $(\mathfrak{s}_0, \mathfrak{h}_0) \models_{\mathcal{R}} x.i \approx y$. The proof is similar if α is of the form $x.i \approx y.j$ (using Axiom 6).
- **Inductive case.** Assume that $(q_i, \mathfrak{s}_i, \mathfrak{h}_i, a_i)$ has been constructed and that the invariant above holds for all $i \leq k$. As $\mathcal{I}(a_k, k) = \mathit{true}$, by Axiom 13, we have $(\mathcal{I}, k) \models \mathit{pre}(a_k)$, hence $(\mathfrak{s}_k, \mathfrak{h}_k) \models_{\mathcal{R}} \mathit{pre}(a_k)$ (by Property a). By Proposition 26, we deduce that $(\mathfrak{s}_k, \mathfrak{h}_k)[a_k]$

is defined. Let $(\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1}) = (\mathfrak{s}_k, \mathfrak{h}_k)[a_k]$. By Axioms 12 and 11, there exist a unique action a_{k+1} and state q_{k+1} such that $\mathcal{I}(a_{k+1}, k+1) = \mathcal{I}(q_{k+1}, k+1) = \mathit{true}$, with $(q_k, a_k, q_{k+1}) \in R$.

We show that Property a is satisfied for $k+1$. We first observe that, if a_k is not of the form $x := s$, then, using Axioms 8, 9 and 10, it is easy to check that $\mathcal{I}(\mathtt{A}(x), k+1)$ holds iff $\mathfrak{s}_{k+1}(x) \in \mathit{dom}(\mathfrak{h}_{k+1})$, i.e., that Property a holds if $\xi = \mathtt{A}(x)$. Indeed, if a_k is of the form $\mathtt{free}(x)$ (resp. $\mathtt{alloc}(x)$) then we have by Axiom 9 (resp. Axiom 10), $\mathcal{I}(\mathtt{A}(y), k+1) = \mathit{false}$ (resp. $\mathcal{I}(\mathtt{A}(y), k+1) = \mathit{true}$) if $\mathcal{I}(x \simeq y, k) = \mathit{true}$ and $\mathcal{I}(\mathtt{A}(y), k+1) = \mathcal{I}(\mathtt{A}(y), k)$ otherwise. Furthermore, by Axiom 8, $\mathcal{I}(\mathtt{A}(y), k+1) = \mathcal{I}(\mathtt{A}(y), k)$ holds for all $y \in \mathcal{V}^\star$ if a_k is not of the above forms.

Consider a triple $(\psi, q, X) \in \mathtt{Bw}(\mathcal{S}, \phi, \Phi)$ such that for all $x \in X \setminus v_{\mapsto}(\psi)$, $\mathfrak{s}_{k+1}(x) \notin \mathit{dom}(\mathfrak{h}_{k+1})$ (†). If a_k contains a term $x.i$ where $x \notin v_{\mapsto}(\psi)$, then (by definition of $(\mathfrak{s}_k, \mathfrak{h}_k)[a_k]$) $\mathfrak{s}_{k+1}(x) \in \mathit{dom}(\mathfrak{h}_{k+1})$, so that $x \notin X$. Thus, by definition of $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, $(\xi, q, X \cup \{x\}) \in \mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, for all $\xi \in \langle\psi\rangle_x$. Note that (since actions of the form $x := x.i$ are forbidden) we must have $\mathcal{I}(\mathtt{A}(x), k+1) \Leftrightarrow \mathfrak{s}_{k+1}(x) \in \mathit{dom}(\mathfrak{h}_{k+1})$, hence $\mathcal{I}(\mathtt{A}(x), k+1) = \mathit{true}$ and by Axiom 4, necessarily $(\mathcal{I}, k+1) \models \psi \Leftrightarrow \bigvee_{\xi \in \langle\psi\rangle_x} \xi$. By Lemma 35, $x \in v_{\mapsto}(\xi)$, for all $\xi \in \langle\psi\rangle_x$. By repeating this process (if needed) on any other variable y such that that the condition above holds (in case a_k contains another occurrence of a term $y.j$), we eventually obtain a set of symbolic heaps S such that $(\mathcal{I}, k+1) \models \psi \Leftrightarrow \bigvee_{\xi \in S} \xi$, for all $\xi \in S$, $\mathit{wpc}(\xi, a_k)$ is defined, and there exists X' such that $(\xi, q_k, X') \in \mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, with $X' = X \cup Y$, for some set of variables $Y \subseteq v_{\mapsto}(\xi)$. This entails (by definition of $\mathtt{Bw}(\mathcal{S}, \phi, \Phi)$) that, for all $\xi \in S$, $(\mathit{wpc}(\xi, a_k), q_{k+1}, X'') \in \mathtt{Bw}(\mathcal{S}, \phi, \Phi)$, for some X'' that is either empty (if a_k is of the form $x := t$ with $x \in \mathcal{V}^\star$) or identical to X' (otherwise). Furthermore, we have $(\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1}) \models_{\mathcal{R}} \psi \iff \exists \xi \in S$ s.t. $(\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1}) \models_{\mathcal{R}} \xi$, by Lemmata 36 and 37. By Property a in the inductive invariant (at rank k) the equivalence $\mathcal{I}(\mathit{wpc}(\xi, a_k), k) = \mathit{true} \iff (\mathfrak{s}_k, \mathfrak{h}_k) \models_{\mathcal{R}} \mathit{wpc}(\xi, a_k)$ holds for all $\xi \in S$. Indeed, we have $\forall x \in X'' \setminus v_{\mapsto}(\mathit{wpc}(\xi, a_k))$, $\mathfrak{s}_k(x) \notin \mathit{dom}(\mathfrak{h}_k)$: if the condition is not fulfilled, then a_k cannot be of the form $x := t$ (otherwise $X'' = \emptyset$) and either a_k is of the form $\mathtt{free}(x)$ and x must occur in $v_{\mapsto}(\mathit{wpc}(\xi, a_k))$, by definition of $\mathit{wpc}(\xi, a_k)$; or x must be allocated in $(\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1})$, and then (by †, since $X'' \subseteq X \cup Y \subseteq X \cup v_{\mapsto}(\xi)$) $x \in v_{\mapsto}(\psi) \subseteq v_{\mapsto}(\xi)$, so that $x \in v_{\mapsto}(\mathit{wpc}(\xi, a_k))$ by definition of $\mathit{wpc}(\xi, a_k)$ (as $a_k \neq \mathtt{alloc}(x)$, since $\mathfrak{s}_k(x) \in \mathit{dom}(\mathfrak{h}_k)$). By Lemma 32 we get $\mathcal{I}(\mathit{wpc}(\xi, a_k), k) = \mathit{true} \iff (\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1}) \models_{\mathcal{R}} \xi$, and by Axiom 3, this yields: $\mathcal{I}(\xi, k+1) = \mathit{true} \iff (\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1}) \models_{\mathcal{R}} \xi$, so that $\mathcal{I}(\psi, k+1) = \mathit{true} \iff (\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1}) \models_{\mathcal{R}} \psi$.

We now show that $\mathtt{Fw}(\mathcal{S}, \phi)$ contains a tuple $(\phi'_{k+1}, q_{k+1}, X_{k+1})$ such that $\mathcal{I}(\phi'_{k+1}, k+1) = \mathit{true}$. Let Y be the set of variables y such that a_k contains a term of the form $y.i$ (for some $i \in \mathbb{N}$) and $y \notin v_{\mapsto}(\phi'_k)$. By applying the function $\langle\rangle_x$ on all variables in Y, we get a set S of symbolic heaps such that $(\mathcal{I}, k) \models \phi'_k \Leftrightarrow \bigvee_{\xi \in S} \xi$. Furthermore, for all variables $y \in Y$, we have $\mathfrak{s}(y) \in \mathit{dom}(\mathfrak{h}_k)$ (since $(\mathfrak{s}_k, \mathfrak{h}_k)[a'_k]$ is defined), thus $y \notin X_k$. By definition of $\mathtt{Fw}(\mathcal{S}, \phi)$, we deduce that for all $\xi \in S$, $(\xi, q_k, X_k \cup Y) \in \mathtt{Fw}(\mathcal{S}, \phi)$. Moreover, by Lemma 35, we have $Y \subseteq v_{\mapsto}(\xi)$, and $\mathit{spc}(\xi, a_k)$ is defined for all $\xi \in S$. Then we get by Axiom 2, $(\mathcal{I}, k+1) \models \mathit{spc}(\xi, a_k)$, for some $\xi \in S$. We define: $\phi'_{k+1} \overset{\text{def}}{=} \mathit{spc}(\xi, a_k)$. By definition of $\mathtt{Fw}(\mathcal{S}, \phi)$, $(\mathit{spc}(\xi, a_k), q_{k+1}, X_{k+1}) \in \mathtt{Fw}(\mathcal{S}, \phi)$ for some set X_{k+1}. Let $x \in x \in X_{k+1} \setminus v_{\mapsto}(\phi'_{k+1})$. Assume that $\mathfrak{s}_{k+1}(x) \in \mathit{dom}(\mathfrak{h})$. By definition of $\mathtt{Fw}(\mathcal{S}, \phi)$, a_k cannot be of the form $z := t$, where $z \in \mathcal{V}^\star$ (otherwise X_{k+1} would be empty). Thus $X_{k+1} = X_k \cup Y$. Since $x \in \mathit{dom}(\mathfrak{s}_{k+1})$, we have $a_k \neq \mathtt{free}(x)$. Since $x \notin v_{\mapsto}(\phi'_{k+1})$, we have $a_k \neq \mathtt{alloc}(x)$, by definition of $\mathit{spc}(\xi, a_k)$. Thus a_k is of the form $t := s$ where t is not a variable, and we must have $x \in \mathit{dom}(\mathfrak{h}_k)$, and $v_{\mapsto}(\phi'_{k+1}) = v_{\mapsto}(\xi) \supseteq Y$. This entails that $x \in X_k$, which contradicts Property c at rank k.

Finally, using the symbolic heap ϕ'_{k+1}, the equivalence $\mathcal{I}(\alpha, k+1) = \mathit{true} \iff (\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1}) \models_{\mathcal{R}} \alpha$ can be established for all atomic conditions α exactly as for the base case. The case where $\alpha = \mathtt{A}(x)$ and a_k is of the form $x := t$ is handled by noting that we have both $(\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1}) \models \mathtt{A}(x) \Leftrightarrow \bigvee_{\xi \in \langle \phi'_{k+1} \rangle_x} \xi$ (since $(\mathfrak{s}_{k+1}, \mathfrak{h}_{k+1}) \models \phi'_{k+1}$, using Lemmata 35, 36 and 37) and $(\mathcal{I}, k+1) \models \mathtt{A}(x) \Leftrightarrow \bigvee_{\xi \in \langle \phi'_{k+1} \rangle_x} \xi$ (by Axiom 4).

Gabbay Separation for the Duration Calculus

Dimitar P. Guelev
Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Sofia, Bulgaria

Abstract

Gabbay's separation theorem about linear temporal logic with past has proved to be one of the most useful theoretical results in temporal logic. In particular it enables a concise proof of Kamp's seminal expressive completeness theorem for LTL. In 2000, Alexander Rabinovich established an expressive completeness result for a subset of the Duration Calculus (DC), a real-time interval temporal logic. DC is based on the *chop* binary modality, which restricts access to subintervals of the reference time interval, and is therefore regarded as *introspective*. The considered subset of DC is known as the $\lceil P \rceil$-subset in the literature. Neighbourhood Logic (NL), a system closely related to DC, is based on the *neighbourhood* modalities, also written $\langle A \rangle$ and $\langle \bar{A} \rangle$ in the notation stemming from Allen's system of interval relations. These modalities are *expanding* as they allow writing future and past formulas to impose conditions outside the reference interval. This setting makes temporal separation relevant: is expressive power ultimately affected, if past constructs are not allowed in the scope of future ones, or vice versa? In this paper we establish an analogue of Gabbay's separation theorem for the $\lceil P \rceil$-subset of the extension of DC by the neighbourhood modalities, and the $\lceil P \rceil$-subset of the extension of DC by the neighbourhood modalities and *chop*-based analogue of *Kleene star*. We show that the result applies if the *weak chop inverses*, a pair *binary* expanding modalities, are given the role of the neighbourhood modalities, by virtue of the inter-expressibility between them and the neighbourhood modalities in the presence of *chop*.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Modal and temporal logics; Theory of computation → Logic and verification

Keywords and phrases Gabbay separation, Neighbourhood Logic, Duration Calculus, expanding modalities

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.10

Introduction

Separation for Linear Temporal Logic (LTL, cf., e.g., [28]) was established by Dov Gabbay in [14]. Separation is about expressing temporal properties without making reference to the past in the scope of future constructs and vice versa. Gabbay proved that such a restriction does not affect the ultimate expressive power of past LTL, by a syntactically defined translation from arbitrary formulas to ones that are *separated*, i.e., satisfy the restriction. The applications of this theorem are numerous and important on their own right. They include a concise proof of Kamp's seminal expressive completeness result for LTL (see, e.g., [13]), the elimination of the past modalities from LTL, which simplifies the study of extensions of LTL, c.f., e.g., [10], Fisher's clausal normal form for past LTL [12], other normal forms [19, 15], etc. In this paper we establish an analogue of Gabbay's separation theorem for the extension of a subset of the Duration Calculus (DC) with a pair of expanding modalities known as the *neighbourhood modalities*, with and without the *chop*-based analogue of Kleene star, which is also called *iteration* in DC.

The Duration Calculus (DC, [32, 30]) is an extension of real time *Interval Temporal Logic* (ITL), which was first proposed by Moszkowski for discrete time [24, 25, 11]. DC is a real-time interval-based predicate logic for the modeling of hybrid systems. Unlike time points, time intervals, the possible worlds in DC, have an internal structure of subintervals.

This justifies calling modalities like *chop introspective* for their providing access to these subintervals only. Modalities for reaching outside the reference interval are called *expanding*. Several sets of such modalities have been proposed in the literature.

In this paper we prove a separation theorem for the $\lceil P \rceil$-subset of DC with the expanding *neighbourhood modalities* $\diamond_l$ and $\diamond_r$ added to DC's *chop* and *iteration*. The system based on $\diamond_l$ and $\diamond_r$ only, which are also written $\langle A \rangle$ and $\langle \overline{A} \rangle$ after Allen's interval relations [3], is called Neighbourhood Logic (NL, [4]), whereas we target DC with $\diamond_l$ and $\diamond_r$. Our theorem holds with *iteration* included too. We write DC-NL (DC-NL*) for DC with $\diamond_l$ and $\diamond_r$ (and *iteration*). In separated formulas, $\diamond_d$ cannot appear in the scope of other modalities, except $\diamond_d$, $d = l, r$. $\diamond_r$-free formulas are regarded as *past*, and $\diamond_l$-free formulas are *future*. The *strict* forms of past (future) formulas are defined by further restricting *chop* and *iteration* to occur only in the scope of a $\diamond_l$ ($\diamond_r$). DC is a predicate logic. We prove that formulas in each of $\lceil P \rceil$-subsets of DC-NL and DC-NL* have *separated* equivalents in their respective subsets. These subsets are compatible with the system of DC from Rabinovich's expressive completeness result [29]. We also show that the *weak chop inverses*, which are *binary* expanding modalities, are expressible using $\diamond_l$ and $\diamond_r$ in the considered subset. Their use in the *Mean-value Calculus*, another system from the DC family, was studied in [26]. $\diamond_l$ and $\diamond_r$ are definable using the weak chop inverses. Consequently, our separation theorem applies to the extensions of DC and DC* by the weak chop inverses too.

The technique of our proofs builds on our finds from [16] which led to establishing separation for discrete time ITL.

Structure of the paper. Section 1 gives preliminaries on DC and DC*, the neighbourhood modalities, the weak chop inverses, and a supplementary result on quantification over state in DC. In Section 2 we state our separation theorem for the $\lceil P \rceil$-subsets of DC-NL and DC-NL* and give a simple example application. Section 3 is dedicated to the proof. The transformations for separating DC-NL and DC-NL* formulas are given in Sections 3.2 and 3.3, respectively, and use a lemma which is given in the preceding Section 3.1. Section 4 is about the expressibility of the weak chop inverses in the $\lceil P \rceil$-subsets of DC-NL and DC-NL*, using the lemma from Section 3.1 too. We conclude by pointing to some related work and making some comments on the relevance of the result.

1 Preliminaries

An in-depth presentation of DC and its extensions can be found in [30]. The syntax of the $\lceil P \rceil$-subset of DC is built starting from a set V of *state variables*. It includes *state expressions* S and *formulas* A. Let P stand for a *state variable*. The BNFs are:

$$S ::= \mathbf{0} \mid P \mid S \Rightarrow S \qquad\qquad A ::= \bot \mid \lceil\rceil \mid \lceil S \rceil \mid A \Rightarrow A \mid A; A$$

Semantics. Given a set of state variables V, the type of *valuations* I is $V \times \mathbb{R} \to \{0, 1\}$. Valuations I are required to have *finite variability*:

For any $P \in V$ and any bounded interval $[a, b] \subset \mathbb{R}$ there exists a finite sequence $t_0 = a < t_1 < \ldots < t_n = b$ such that $\lambda t.I(P, t)$ is constant in (t_{i-1}, t_i), $i = 1, \ldots, n$.

The *value* $I_t(S)$ *of state expression* S *at time* $t \in \mathbb{R}$ is defined by the clauses:

$$I_t(\mathbf{0}) \mathrel{\hat{=}} 0, \qquad I_t(P) \mathrel{\hat{=}} I(P, t), \qquad I_t(S_1 \Rightarrow S_2) \mathrel{\hat{=}} \max\{I_t(S_2), 1 - I_t(S_1)\}.$$

Satisfaction has the form $I, [a, b] \models A$, where $[a, b] \subset \mathbb{R}$. The defining clauses are:

$$\begin{array}{lll}
I, [a, b] \not\models \bot, & I, [a, b] \models \lceil\rceil & \text{iff} \quad a = b, \\
I, [a, b] \models \lceil S \rceil & \text{iff} & a < b \text{ and } I_t(S) = 1 \text{ for all but finitely many } t \in [a, b], \\
I, [a, b] \models A \Rightarrow B & \text{iff} & I, [a, b] \models B \text{ or } I, [a, b] \not\models A, \\
I, [a, b] \models A; B & \text{iff} & I, [a, m] \models A \text{ and } I, [m, b] \models B \text{ for some } m \in [a, b].
\end{array}$$

The connectives $\neg$, $\wedge$, $\vee$ and $\Leftrightarrow$ are defined as usual in both state expressions and formulas. Furthermore $\mathbf{1} \mathrel{\hat{=}} \mathbf{0} \Rightarrow \mathbf{0}$ and $\top \mathrel{\hat{=}} \bot \Rightarrow \bot$. A formula A is *valid* in DC, written $\models A$, if $I, [a, b] \models A$ for all I and all intervals $[a, b]$. In this paper we consider the extension of the $\lceil P \rceil$-subset of DC by the *neighbourhood modalities* $\Diamond_d$, $d \in \{l, r\}$. The defining clauses for their semantics are as follows:

$$\begin{array}{lll}
I, [a, b] \models \Diamond_l A & \text{iff} & I, [a', a] \models A \text{ for some } a' \leq a, \\
I, [a, b] \models \Diamond_r A & \text{iff} & I, [b, b'] \models A \text{ for some } b' \geq b.
\end{array}$$

The universal duals $\Box_d$ of $\Diamond_d$ are defined by putting $\Box_d A \mathrel{\hat{=}} \neg\Diamond_d\neg A$, $d \in \{l, r\}$. *Chop* $A; B$ is written $A^\frown B$ in much of the literature. We write DC-NL for the extension of DC by $\Diamond_l$ and $\Diamond_r$. We also consider DC-NL*, the extension of DC-NL by *iteration*, the *chop*-based form of Kleene star, included. The defining clause for this operator is

$$\begin{array}{lll}
I, [a, b] \models A^* & \text{iff} & a = b \text{ or there exist a finite sequence } m_0 = a < m_2 < \cdots < m_n = b \\
& & \text{such that } I, [m_{i-1}, m_i] \models A \text{ for } i = 1, \ldots, n.
\end{array}$$

Iteration is interdefinable with *positive iteration* $A^+ \mathrel{\hat{=}} A; (A^*)$, which we assume to be the derived one of the two: $\models A^* \Leftrightarrow \lceil\rceil \vee A^+$.

Predicate DC and NL include a (defined) flexible constant ℓ for the length $b - a$ of reference interval $[a, b]$. Using ℓ, *chop* can be defined in NL:

$$A; B \ \mathrel{\hat{=}}\ \exists x \exists y (x + y = \ell \wedge \Diamond_l \Diamond_r (A \wedge \ell = x) \wedge \Diamond_r \Diamond_l (B \wedge \ell = y)).$$

This definition is not available in NL's $\lceil P \rceil$-subset. Therefore we discern the $\lceil P \rceil$-subsets of NL and DC-NL.

Quantification over state in DC. Given a state variable P, $I, [a, b] \models \exists P\, A$ iff $I', [a, b] \models A$ for some I' such that $I'(Q, t) = I(Q, t)$ and all $Q \in V \setminus \{P\}$, $t \in \mathbb{R}$. Quantification over state is expressible in the $\lceil P \rceil$-subset of DC*:

▶ **Theorem 1.** *For every $\lceil P \rceil$-formula A in* DC* *and every state variable P there exists a (quantifier-free) $\lceil P \rceil$-formula B in* DC* *such that $\models B \Leftrightarrow \exists P\, A$.*

Mind that B is not guaranteed to be *iteration*-free, even in case A is.

This theorem follows from a correspondence between stutter-invariant regular languages and the $\lceil P \rceil$-subset that led to the decidability of the $\lceil P \rceil$-subset in [31]. It is not our contrubution, but the transformations from its proof supplement those from our other proofs.

Notation. In this paper write ε, possibly with subscripts, to denote *optional* occurrences of the negation sign $\neg$, e.g, ε_Q below. We write $[A/B]C$ to denote the result of simultaneously replacing all the occurrences of B by A in C, e.g., $[\mathbf{0}/P]S$ below.

Proof of Theorem 1. Following [31], A translates into a regular expression over the alphabet

$$\Sigma \mathrel{\widehat{=}} \{ \bigwedge_{Q \text{ is a state variable in } A} \varepsilon_Q Q : \varepsilon_Q \text{ is either } \neg \text{ or nothing}\}\,. \tag{1}$$

The translation clauses are as follows:

$$\begin{array}{lll} t(\bot) \mathrel{\widehat{=}} \emptyset & t(\lceil S \rceil) \mathrel{\widehat{=}} (\{\sigma \in \Sigma : \models \sigma \Rightarrow S\})^+ & t(A;B) \mathrel{\widehat{=}} t(A);t(B) \\ t(\lceil\rceil) \mathrel{\widehat{=}} \epsilon \text{ (the empty string)} & t(A \Rightarrow B) \mathrel{\widehat{=}} t(B) \cup \Sigma^* \setminus t(A) & t(A^*) \mathrel{\widehat{=}} t(A)^* \end{array}$$

Up to equivalence, t can be inverted. Regular expressions admit complementation- and $\cap$-free equivalents; hence these operations can be omitted in the converse translation $\bar{t}$:

$$\begin{array}{llll} \bar{t}(\emptyset) \mathrel{\widehat{=}} \bot & \bar{t}(a) \mathrel{\widehat{=}} \lceil a \rceil \text{ for } a \in \Sigma & \bar{t}(R_1 \cup R_2) \mathrel{\widehat{=}} \bar{t}(R_1) \vee \bar{t}(R_2) & \bar{t}(R^*) \mathrel{\widehat{=}} \bar{t}(R)^* \\ \bar{t}(\varepsilon) \mathrel{\widehat{=}} \lceil\rceil & \bar{t}(\Sigma^*) \mathrel{\widehat{=}} \lceil\rceil \vee \lceil \mathbf{1} \rceil & \bar{t}(R_1;R_2) \mathrel{\widehat{=}} \bar{t}(R_1);\bar{t}(R_2) & \end{array}$$

Given a regular expression $R = t(A)$, $\bar{t}(R')$ is equivalent to A for any R' that defines the same language as R. Applying $\bar{t}$ to a complementation- and $\cap$-free equivalent R' to $t(A)$ produces an equivalent to A with $\vee$ as the only propositional connective, except possibly inside state expressions. Given this, $\exists\, P$ can be eliminated from formulas of the form $\bar{t}(R')$:

$$\begin{array}{lll} \models \exists\, P \,\bot \Leftrightarrow \bot & \models \exists\, P \lceil S \rceil \Leftrightarrow \lceil [\mathbf{0}/P]S \vee [\mathbf{1}/P]S \rceil^+ & \models \exists\, P\,(A_1;A_2) \Leftrightarrow \exists\, P\, A_1; \exists P\, A_2 \\ \models \exists\, P \lceil\rceil \Leftrightarrow \lceil\rceil & \models \exists\, P\,(A_1 \vee A_2) \Leftrightarrow \exists\, P\, A_1 \vee \exists\, P\, A_2 & \models \exists\, P\, A^* \Leftrightarrow (\exists\, P\, A)^*. \end{array}$$

The equivalence $\exists\, P \lceil S \rceil$ above hinges on the finite variability of $I_t(P)$. ◀

The weak chop inverses A/B and $A\backslash B$, cf., e.g., [26], are defined by the clauses:

$$\begin{array}{lll} I,[a,b] \models A/B & \text{iff} & \text{for all } r \geq b, \text{ if } I,[b,r] \models B \text{ then } I,[a,r] \models A. \\ I,[a,b] \models A\backslash B & \text{iff} & \text{for all } l \leq a, \text{ if } I,[l,a] \models B \text{ then } I,[l,b] \models A. \end{array}$$

$\Diamond_l A$ and $\Diamond_r A$ can be defined as $\neg(\bot\backslash A)$ and $\neg(\bot/A)$, respectively. In Section 4 we show how A/B and $A\backslash B$ can be expressed using $\Diamond_l$ and $\Diamond_r$ too for $\lceil P \rceil$-formulas A and B, but with the expressing formulas built in a more complex way.

Separation as Known for LTL. We relate the setting and statement of Gabbay's separation theorem about past LTL as our work builds in the example of this theorem. Let p stand for an atomic proposition. Discrete time LTL formulas with past have the syntax:

$$A ::= \bot \mid p \mid A \Rightarrow A \mid \bigcirc A \mid A\,\mathcal{U}\,A \mid \ominus A \mid A\,\mathcal{S}\,A$$

$\ominus$ and $\mathcal{S}$ are the past mirror operators of $\bigcirc$ and $\mathcal{U}$. $\ominus$- and $\mathcal{S}$-free formulas are called *future* formulas, and $\bigcirc$- and $\mathcal{U}$-free formulas are called *past*. Formulas of the form $\bigcirc F$ where F is future are called *strictly future*. In [14], Dov Gabbay demonstrated that any formula in LTL with past is equivalent to a Boolean combination of past and strictly future formulas for flows of time which are either finite or infinite, in either the future or the past, or both.

Modal heights $h_{\Diamond_l}(.)$, $h_{\Diamond_r}(.)$ and $h_{*}(.)$ of formulas wrt the neighbourhood modalities and *iteration* appear in our inductive reasoning below. In general, $h(A)$ denotes the length of the longest chain of A's subformulas, including possibly A, with the main connective being the specified modality wrt the (transitive closure of) the subformula relation.

2 The Separation Theorem

In this section we formulate the main contribution of the paper, Theorems 2 and 3, which are separation theorems for the $\lceil P \rceil$-subsets of DC-NL and DC-NL*, and use Theorem 2 to demonstrate the expressibility of an interval-based version of the "past-forgetting" operator from [18] as a simple example application.

We call DC-NL (DC-NL*) formula F (non-strictly) *future* if it has the syntax

$$F ::= C \mid \neg F \mid F \vee F \mid \Diamond_r F$$

where C stands for a DC (DC*) formula, where *chop* (and *iteration*) are the only modalities. Non-strictly *past* formulas are defined similarly, with $\Diamond_l$ instead of $\Diamond_r$. A *separated formula* is a Boolean combination of past and future formulas.

Following the example of LTL, we call Boolean combinations of $\Diamond_l$-, resp. $\Diamond_r$-formulas with non-strict past, resp. future operands *strictly past*, resp. *strictly future* formulas. Such formulas can impose no conditions on the reference interval; they only refer to the adjacent past and future intervals along the timeline. These adjacent intervals still include the respective endpoints of the reference interval. However the $\lceil P \rceil$ construct cannot tell apart interpretations I of the state variables such that $\lambda t.I(P,t)$ varies only at finitely many time points t. Unlike that, in discrete time an extra step away from the present time using $\ominus$, resp., $\bigcirc$ is necessary to prevent a formula from imposing conditions on the reference time point or the reference interval's respective endpoint. This shared time point causes strictly past and strictly future formulas to be defined differently in discrete time ITL. *Separated* formulas can also be defined as Boolean combinations of strictly past formulas, strictly future formulas and *introspective*, i.e., just DC (DC*), formulas, where the only modalities are *chop* (and *iteration*), that are known as *introspective* too.

▶ **Theorem 2.** *Let A be a $\lceil P \rceil$-formula in* DC-NL *(*DC-NL**). Then there exists a separated $\lceil P \rceil$-formula A' in* DC-NL *(*DC-NL**) such that $\models A \Leftrightarrow A'$.*

In Section 4 we demonstrate the inter-expressibility between $(./.)$ and $(.\backslash.)$, and $\Diamond_l$ and $\Diamond_r$, respectively. This implies that Theorem 2 holds for the weak chop inverses instead of the respective $\Diamond_d$, $d \in \{l, r\}$ wrt a corresponding notion of separated formula too:

▶ **Theorem 3.** *Let A be a $\lceil P \rceil$-formula in the extension of* DC *(*DC**) by $(./.)$ and $(.\backslash.)$. Then there exists a separated $\lceil P \rceil$-formula A' in* DC *(*DC**) with $(./.)$ and $(.\backslash.)$ such that $\models A \Leftrightarrow A'$.*

An Example Application: Expressing the N operator. The temporal operator N ("now") was proposed for past LTL in [18], see also [17], as a means for "preventing access" into the past beyond the time of applying N. Assuming $\sigma \mathrel{\hat{=}} \sigma^0\sigma^1\ldots$ to be a sequence of states

$$\sigma, i \models_{\mathrm{LTL}} \mathsf{N}A \text{ iff } \sigma^i\sigma^{i+1}\ldots, 0 \models_{\mathrm{LTL}} A\ .$$

If an arbitrary closed interval $D \subseteq \mathbb{R}$, and not only the whole of $\mathbb{R}$, is allowed to be the time domain, N can be defined for (real-time) DC-NL too. With such time domains, the endpoints of "all time" can be identified, because, e.g., $D, I, [a,b] \models \Box_l \lceil\rceil$ iff $a = \min D$. (Since the $\lceil P \rceil$-subset of DC-NL is merely *topological*, as opposed to *metric*, it cannot distinguish *open* time domains from $\mathbb{R}$.) We can define N on intervals by putting:

$$D, I, [a,b] \models \mathsf{N}_l A \text{ iff } \{x \in D : x \geq a\}, I, [a,b] \models A$$
$$D, I, [a,b] \models \mathsf{N}_r A \text{ iff } \{x \in D : x \leq b\}, I, [a,b] \models A$$

Theorem 2 entails that N_l and N_r are expressible in DC-NL:

▶ **Proposition 4.** DC-NL + $\mathsf{N}_l, \mathsf{N}_r$ *has the same expressive power as* DC-NL.

Proof. Let A' be a separated equivalent of A. Then $\models \mathsf{N}_d A \Leftrightarrow [\Diamond_d(B \wedge \lceil\rceil)/\Diamond_d B : B \in \mathsf{Subf}(A')]A'$, $d \in \{l, r\}$, where $\mathsf{Subf}(F)$ stands for the set of the subformulas of F, including F. ◀

3 The Proof of Separation for DC-NL and DCNL*

In this section we propose a set of valid equivalences which, if appropriately used as transformation rules starting from some arbitrary given formula from the $\lceil P\rceil$-subset of DC-NL*, lead to a separated formula in DC-NL*. If the given formula is *iteration*-free, i.e., in DC-NL, then so is the separated equivalent. This amounts to proving Theorem 2.

Our key observation is that formulas which are supposed to be evaluated at intervals that extend some given interval into either the future or the past have equivalents which consist of subformulas to be evaluated at the given interval and subformulas to be evaluated at intervals which are adjacent to it, these two subintervals being appropriately referenced using *chop* as parts of the enveloping interval. In our proof of separation, this observation is refered to as a lemma that states the possibility to express any introspective formula as a case distinction of *chop*-formulas with the LHS (RHS) operands of chop forming a full system. The lemma can be seen as a generalization of the *guarded normal form*, which is ubiquitous in process logics, with the full systems of guards describing a primitive opening move replaced by full systems of interval-based temporal conditions to be satisfied at whatever prefixes (suffixes) of the reference runs necessary. Later on we use the lemma in expressing $(./.)$ $((.\backslash.))$ in terms of $\Diamond_r$ $(\Diamond_l)$ too.

3.1 The Key Lemma

A finite set of formulas $A_1, \ldots, A_n$ is a *full system*, if $\models \bigvee\limits_{k=1}^{n} A_k$ and, given $1 \le k_1 < k_2 \le n$, $\models \neg(A_{k_1} \wedge A_{k_2})$.

▶ **Lemma 5.** *Let A be a $\lceil P\rceil$-formula in* DC *(*DC**). Then there exists an $n < \omega$ and some* DC *(*DC**) $\lceil P\rceil$-formulas A_k, A'_k, $k = 1, \ldots, n$, such that $A_1, \ldots, A_n$ form a full system and*

$$\models A \Leftrightarrow \bigvee_{k=1}^{n} A_k; A'_k \text{ and } \models A \Leftrightarrow \bigwedge_{k=1}^{n} \neg(A_k; \neg A'_k). \tag{2}$$

Furthermore, $h_*(A_k) \le h_*(A)$ and $h_*(A'_k) \le h_*(A)$.

Informally, this means that, $I, [a, b] \models A$ iff whenever $m \in [a, b]$ and $I, [a, m] \models A_k$, $I, [m, b] \models A'_k$ holds. Furthermore, for every $m \in [a, b]$ there is a unique k such that $I, [a, m] \models A_k$. Interestingly, the construct $\neg(F; \neg G)$ used in the second equivalence (2) is regarded as a form of *temporal implication*, written $F \mapsto G$, in ITL [23, 5]. This construct is akin to *suffix implication* [2], see also [1]. It requires the suffix of an interval to satisfy B, if the complementing prefix satisfies A. Much like $\Rightarrow$'s being the right adjoint of $\wedge$, $\mapsto$ is the right adjoint of *chop*:

$$\models (A \mapsto (B \mapsto C)) \Leftrightarrow ((A; B) \mapsto C) \ .$$

In this paper we stick to the notation in terms of *chop* for both $\mapsto$ and its mirror $\neg(\neg G; F)$.

Proof of Lemma 5. Induction on the construction of A. For $\bot$, $\lceil\rceil$ and $\lceil P\rceil$, we have:

$$\models \bot \;\Leftrightarrow\; (\top;\bot) \qquad \models \lceil\rceil \;\Leftrightarrow\; (\lceil\rceil;\lceil\rceil) \vee (\neg\lceil\rceil;\bot)$$
$$\models \lceil P\rceil \;\Leftrightarrow\; (\lceil P\rceil;(\lceil P\rceil \vee \lceil\rceil)) \vee (\lceil\rceil;\lceil P\rceil) \vee (\neg(\lceil\rceil \vee \lceil P\rceil);\bot)$$

Let $B_1,\dots,B_n$, $B'_1,\dots,B'_n$ satisfy (2) for B and $C_1,\dots,C_m$, $C'_1,\dots,C'_m$ satisfy (2) for C. Then:

$$\models B\,op\,C \;\Leftrightarrow\; \bigvee_{k=1}^{n}\bigvee_{l=1}^{m}(B_k \wedge C_l);(B'_k\,op\,C'_l),\;\; op \in \{\Rightarrow,\vee,\wedge,\Leftrightarrow\}$$
$$\models B;C \;\Leftrightarrow\; \bigvee_{k=1}^{n}\bigvee_{X\subseteq\{1,\dots,m\}}\left(B_k \wedge \bigwedge_{l\in X}(B;C_l) \wedge \bigwedge_{l\notin X}\neg(B;C_l)\right);\left((B'_k;C) \vee \bigvee_{l\in X} C'_l\right)$$

For the equivalence about *iteration*, let $C_1,\dots,C_m$, and $C'_1,\dots,C'_m$ satisfy (2) for $C \mathrel{\hat{=}} B \vee \lceil\rceil$. Then $B^* \Leftrightarrow C^*$, and:

$$\models B^* \;\Leftrightarrow\; \bigvee_{X\subseteq\{1,\dots,m\}}\left(\bigwedge_{l\in X}(B^*;C_l) \wedge \bigwedge_{l\notin X}\neg(B^*;C_l)\right);\left(\bigvee_{l\in X}(C'_l;B^*)\right) \qquad (3)$$

The equivalences on the right in (2) are written similarly. The RHSs of these equivalences have the form required in the lemma. Using these equivalences as transformation rules bottom up, an arbitrary A can be given that form.

A direct check is sufficient for establishing (2) about $\bot$, $\lceil\rceil$ and $\lceil P\rceil$. The case of $B\,op\,C$, esp. $op = \Rightarrow$, admits the proof that works for the *Guarded Normal Form* in [6].

For the equivalence on the left in (2) about $B;C$, ($\Rightarrow$): let $I,[a,b] \models B;C$, $m \in [a,b]$, and $I,[a,m] \models B$ and $I,[m,b] \models C$. Let $t \in [a,b]$. If $t \in [a,m]$, then $I,[a,t] \models B_k$ for some unique k. If $t \in [m,b]$, then a unique $X \subseteq \{1,\dots,m\}$ exists such that $I,[a,t] \models B;C_l$ holds iff $l \in X$. The conjunctions of $B_k \wedge \bigwedge_{l\in X}(B;C_l) \wedge \bigwedge_{l\notin X}\neg(B;C_l)$, $k = 1,\dots,n$, $X \subseteq \{1,\dots,m\}$ form a full system because so do both the B_ks, and the conjunctions $\bigwedge_{l\in X}(B;C_l) \wedge \bigwedge_{l\notin X}\neg(B;C_l)$, $X \subseteq \{1,\dots,m\}$. Since $I,[a,m] \models B$ and $I,[m,b] \models C$, for an $[a,t]$ satisfying the member of this full system for any given k and X, we can conclude that $I,[t,b] \models (B'_k;C) \vee \bigvee_{l\in X} C'_l$ from the assumptions on the B'_ks and the C'_ls. For the converse implication ($\Leftarrow$), let $[a,b]$ be an arbitrary interval, $t \in [a,b]$, and let $I,[a,t] \models B_k \wedge \bigwedge_{l\in X}(B;C_l) \wedge \bigwedge_{l\notin X}\neg(B;C_l)$, which is bound to be true for some unique pair k,X. Then, $I,[t,b] \models B'_k;C$ implies $I,[a,b] \models B_k;B'_k;C$, and $I,[t,b] \models C'_l$ implies $I,[a,b] \models B;C_l;C'_l$ for any $l \in X$. In both cases $I,[a,b] \models B;C$ follows because $\models B_k;B'_k \Rightarrow B$ and $\models C_l;C'_l \Rightarrow C$. The LHS equivalence (2) about B^* is established similarly, with the use of C facilitating a uniform handling of the case of B^* holding trivially at 0-length intervals. The RHS equivalences (2) follow from the LHS ones by the assumption that the A_ks form a full system.

Observe that the equivalence (3) about $A = B^*$ satisfies $h_*(A_k) \le h_*(A)$ and $h_*(A'_k) \le h_*(A)$. The non-increase of $h_*(.)$ also holds for the rest of the equivalences, which, despite not featuring *iteration* explicitly, may become used for transforming formulas with *iteration*. Hence, $h_*(A_k) \le h_*(A)$ and $h_*(A'_k) \le h_*(A)$ for all A. ◀

The time mirror image of Lemma 5 holds too, with the time mirror of (2) reading

$$\models A \Leftrightarrow \bigvee_{k=1}^{n} A'_k;A_k \text{ and } \models A \Leftrightarrow \bigwedge_{k=1}^{n}\neg(\neg A'_k;A_k).$$

The proof is no different because all the modalities are symmetrical wrt the direction of time. For this reason, in the sequel we omit “mirror” statements and their proofs.

On the complexity of the transformations from Lemma 5. Interestingly, a peak (exponential) blowup in the transformations from Lemma 5's proof occurs in the clause for *chop* and not the clause for $\neg$, the typical source of such blowups. However, a closer look at the inductive assumptions shows that the pairwise inconsistency achieved at the cost of using $A_k \wedge \bigwedge\limits_{l\in X}(A;B_l) \wedge \bigwedge\limits_{l\notin X}\neg(A;B_l)$ for all $k \in \{1,\dots,m\}$ and the 2^n different $X \subseteq \{1,\dots,m\}$ in the required full system is instrumental for the correctness of the clause about the binary Boolean connectives, where negation is obtained for $op = \Rightarrow$ and $B = \bot$. Hence this blowup can be linked to the alternation of $\neg$ and monotone operators such as *chop* that is common in proofs of the non-elementariness of the blowup upon reaching normal forms.

Lemma 5 admits an automata-theoretic proof, along the lines of the proof of Theorem 1. We have sketched such a proof for discrete time ITL in [16]. That proof leads to different A_k and A'_k satisfying (2) for the same A, and allows a non-elementary upper bound on the length of these formulas to be established using the size of a deterministic FSM recognizing A. Unlike the automata-based proof, the equivalences of this proof suggest transformations that are valuable for their compositionality and their validity in DC in general, and not just for the $\lceil P\rceil$-subset. Furthermore, the proof given here facilitates establishing that *-height is not increased upon moving to the RHSs of (2).

3.2 Separating the Neighbourhood Modalities in DC-NL

In this section we prove Theorem 2 by showing how occurrences of $\Diamond_d$ can be taken out of the scope of *chop* and $\Diamond_{\overline{d}}$, $d \in \{l,r\}$, $\overline{l} \mathrel{\hat=} r$, $\overline{r} \mathrel{\hat=} l$. The transformations that we propose are supposed to be applied bottom up, on formulas with *chop* or $\Diamond_d$, $d \in \{l,r\}$, as the main connective, assuming that the operands of are already separated. If the main connective is $\Diamond_d$, then we need to target only the $\Diamond_{\overline{d}}$-subformulas in $\Diamond_d$'s operand, possibly at the cost of introducing some $\Diamond_{\overline{d}}$-subformulas in the scope of *chop*, to be subsequently extracted from there too.

To show that the above transformations combine into a terminating procedure which produces a separated formula, for DC-NL, we reason by induction on the $\Diamond_d$-height of the relevant formulas. In the case of DC-NL*, which is the topic of Section 3.2, we also keep track of *-height. It is not increased upon applying Lemma 5, nor by the transformations for separating formulas with $\Diamond_l$, $\Diamond_r$ or *chop* as the main connective. The effect on *-height of eliminating some quantification over state which appears at an intermediate stage of the transformations by an application of Theorem 1 on *-height is irrelevant because it involves only introspective, i.e., DC*, formulas. In most cases, we give detail only on the extracting of $\Diamond_r$-subformulas, because of the time symmetry.

Separating $\Diamond_d$-formulas. Let $d = l$; the case of $d = r$ is its mirror. Since

$$\models \Diamond_l(A_1 \vee A_2) \Leftrightarrow \Diamond_l A_1 \vee \Diamond_l A_2 \ , \tag{4}$$

the availability of DNF for A of $\Diamond_l A$ makes it sufficient to consider the case of A of the form $P \wedge \bigwedge\limits_{k=1}^{n} \varepsilon_k \Diamond_r F_k$ where P is (non-strictly) past and $F_1,\dots,F_n$ are future. Observe that

$$\models \Diamond_l\left(P \wedge \bigwedge_{k=1}^{n} \varepsilon_k \Diamond_r F_k\right) \Leftrightarrow \Diamond_l P \wedge \bigwedge_{k=1}^{n}((\lceil\rceil \wedge \varepsilon_k \Diamond_r F_k);\top) \ . \tag{5}$$

Using (4) and (5) does not increase $\Diamond_l$-height and implies that separating $\Diamond_l A$ reduces to separating $((\lceil\rceil \wedge \varepsilon\Diamond_r F_k);\top)$, which are *chop*-formulas. Here follow the transformations for doing this.

Separating *chop*-formulas. We need to consider only *chop* applied to conjunctions of introspective formulas and possibly negated past $\Diamond_l$-formulas or future $\Diamond_r$-formulas because

$$\models (L_1 \vee L_2); R \Leftrightarrow (L_1; R) \vee (L_2; R) \text{ and } \models L; (R_1 \vee R_2) \Leftrightarrow (L; R_1) \vee (L; R_2)$$

Past $\Diamond_l$-formulas (future $\Diamond_r$-formulas) can be extracted from the left (right) operand of *chop* using that

$$\models (L \wedge \varepsilon\Diamond_l P); R \Leftrightarrow (L; R) \wedge \varepsilon\Diamond_l P \text{ and } \models L; (R \wedge \varepsilon\Diamond_r F) \Leftrightarrow (L; R) \wedge \varepsilon\Diamond_r F. \tag{6}$$

Much like (4), this does not affect $\Diamond_d$-height. It remains to consider $(L \wedge \bigwedge\limits_{k=1}^{n} \varepsilon_k \Diamond_r F_k); R$, which, by virtue of the time symmetry, will explain separating $L; (R \wedge \bigwedge\limits_{k=1}^{n} \varepsilon_k \Diamond_l P_k)$ too.

The transformations of formulas of the form $(L \wedge \varepsilon\Diamond_r F); R$ below are about the designated $\varepsilon\Diamond_r F$ only, and are supposed to be used repeatedly, if L has more conjuncts of this form. By (4), F can be assumed to be a conjunction $C \wedge G$ where C is introspective and G is strictly future. Let C_k, C'_k, $k = 1, \dots, n$, satisfy Lemma 5 for C. We do the cases of $(L \wedge \Diamond_r F); R$ and $(L \wedge \neg\Diamond_r F); R$ separately.

$(L \wedge \Diamond_r F); R$: Observe that

$$\models (L \wedge \Diamond_r(C \wedge G)); R \Leftrightarrow (L; (R \wedge ((C \wedge G); \top))) \vee \bigvee_{k=1}^{n} (L; (R \wedge C_k)) \wedge \Diamond_r(C'_k \wedge G)$$

and further process the RHS of $\Leftrightarrow$ in it. The two disjuncts on the RHS above correspond to F being satisfied at an interval which is shorter, or the same length, or longer than the one which presumably satisfies R. Since C_k and C'_k are introspective, the newly introduced formulas $\Diamond_r(C'_k \wedge G)$ on the RHS of $\Leftrightarrow$ are separated. G can be extracted from the scope of *chop* in $L; (R \wedge ((C \wedge G); \top))$ too, because $h_{\Diamond_r}(G) < h_{\Diamond_r}((L \wedge \Diamond_r F); R)$.

$(L \wedge \neg\Diamond_r F); R$: Satisfying $(L \wedge \neg\Diamond_r(C \wedge G)); R$ requires $\neg(C \wedge G)$ to hold at all the intervals which start at the right end of the one where L presumably holds. Therefore we can use that

$$\models (L \wedge \neg\Diamond_r(C \wedge G)); R \Leftrightarrow \bigvee_{k=1}^{n} (L; (R \wedge C_k \wedge \neg((C \wedge G); \top))) \wedge \neg\Diamond_r(C'_k \wedge G).$$

Again, G must be extracted from the scope of *chop* in the newly introduced $L; (R \wedge C_k \wedge \neg((C \wedge G); \top))$ on the RHS of the equivalence. This can be accomplished because $h_{\Diamond_r}(G) < h_{\Diamond_r}((L \wedge \neg\Diamond_r F); R)$.

The transformations above are sufficient for establishing Theorem 2 about DC-NL. By Lemma 5, these transformations do not cause *-height to increase. This is relevant in separating formulas in DC-NL*, which is explained next.

3.3 Separating *iteration* formulas

To extract $\Diamond_l$ and $\Diamond_r$ from the scope of *iteration*, we use the inter-expressibility between *iteration* and quantification over state, and the expressibility of quantification over state in the $\lceil P \rceil$-subset of DC* (Theorem 1). Consider B^* where B is a separated formula. Without loss of generality, B can be assumed to be $\bigvee\limits_{s=1}^{t} B_s$ where

$$B_s \mathrel{\hat{=}} H_s \wedge \bigwedge_{i=1}^{u} \varepsilon^p_{s,i} \Diamond_l P_i \wedge \bigwedge_{j=1}^{v} \varepsilon^f_{s,j} \Diamond_r F_j,$$

H_s, $s = 1, \ldots, t$ are introspective, P_i, $i = 1, \ldots, u$, are past formulas, and F_j, $j = 1, \ldots, v$, are future formulas. Furthermore, P_i, $i = 1, \ldots, u$, (F_j, $j = 1, \ldots, v$) can be assumed to be conjunctions of introspective and strictly past (strictly future) formulas by (4) and its mirror equivalence.

Let T, S^p_i, $i = 1, \ldots, u$, and S^f_j, $j = 1, \ldots, v$, be fresh state variables. Then

$$\models B^* \Leftrightarrow \exists T \exists S^p_1 \ldots S^p_u \exists S^f_1 \ldots S^f_v \left((\lceil T \rceil ; \lceil \neg T \rceil) \wedge \bigvee_{s=1}^{t} \left(B_s \wedge \bigwedge_{i=1}^{u} \lceil \varepsilon^p_{s,i} S^p_i \rceil \wedge \bigwedge_{j=1}^{v} \lceil \varepsilon^f_{s,j} S^f_j \rceil \right) \right)^* ,$$

This equivalence states that an interval $[a, b]$ such that $I, [a, b] \models B^*$ can be partitioned into subintervals $[m_0, m_1], \ldots, [m_{d-1}, m_e]$ so that each subinterval satisfies B_s for some $s \in \{1, \ldots, t\}$, and an assignment of T, S^p_1, …, S^p_u and S^f_1, …, S^f_v can be chosen so that, for $d = 1, \ldots, e$, $[m_{d-1}, m_d]$ is a maximal $\lceil T \rceil ; \lceil \neg T \rceil$-interval, and for some $s \in \{1, \ldots, t\}$ such that $I, [m_{d-1}, m_d] \models B_s$, $I, [m_{d-1}, m_d] \models \lceil \varepsilon^p_{s,i} S^p_i \rceil$ iff $I, [m_{d-1}, m_d] \models \varepsilon^p_{s,i} \Diamond_l P_i$, $i = 1, \ldots, u$, and $I, [m_{d-1}, m_d] \models \lceil \varepsilon^f_{s,j} S^f_j \rceil$ iff $I, [m_{d-1}, m_d] \models \varepsilon^f_{s,j} \Diamond_r F_j$, $j = 1, \ldots, v$.

Now observe that $I, [m_{d-1}, m_d] \models B_s$ would follow, if $I, [m_{d-1}, m_d] \models H_s$, and, for some appropriate $a' \leq m_{d-1}$, $I, [a', m_{d-1}] \models \varepsilon^p_{s,i} P_i$, $i = 1, \ldots, u$, and, for some appropriate $b' \geq m_d$, $I, [m_k, b'] \models \varepsilon^f_{s,j} F_j$, $i = 1, \ldots, v$. Here *appropriate* stands for *all* $b' \geq m_d$ ($a' \leq m_{d-1}$), if $\varepsilon^f_{s,j}$ ($\varepsilon^p_{s,i}$) is $\neg$; otherwise it stands for *some* $b' \geq m_d$ ($a' \leq m_{d-1}$). Furthermore, the m_d such that $I, [m_d, b'] \models \varepsilon^f_{s,j} F_j$ is required for all (some) $b' \geq m_d$ can be identified by the condition that $\neg T \wedge \varepsilon^f_{s,j} S^f_j$ holds in a left neighbourhood of m_d and T holds in a right neighbourhood of m_d, for $d = 1, \ldots, e - 1$. For $d = e$, $m_d = b$, and, unless $a = b$, $\neg T \wedge \varepsilon^f_{s,j} S^f_j$ holds in a left neighbourhood of m_d. The mirror conditions allow identifying the m_{d-1} for which $I, [a', m_{d-1}] \models \varepsilon^p_{s,i} P_i$ is required, for either some or all $a' \leq m_{d-1}$, depending on $\varepsilon^p_{s,i}$, $d = 1, \ldots, e$, with m_0 similarly handled separately.

Given the possibility to identify the relevant m_d as observed, $I, [m_d, b'] \models \varepsilon^f_{s,j} F_j$ for the required $b' \geq m_d$ can be expressed as $I, [a, b] \models \varphi_j$ where

$$\varphi_j \mathrel{\hat{=}} \left(\begin{array}{l} (\top ; \lceil S^f_j \rceil) \Rightarrow \Diamond_r F_j \wedge \neg((\top ; \lceil S^f_j \wedge \neg T \rceil) ; ((\lceil T \rceil ; \top) \wedge \neg((\Diamond_r F_j \wedge \lceil\rceil) ; \top))) \wedge \\ (\top ; \lceil \neg S^f_j \rceil) \Rightarrow \neg \Diamond_r F_j \wedge \neg((\top ; \lceil \neg S^f_j \wedge \neg T \rceil) ; ((\lceil T \rceil ; \top) \wedge ((\Diamond_r F_j \wedge \lceil\rceil) ; \top))) \end{array} \right) . \quad (7)$$

The time mirrors of φ_j can be used to enforce $I, [a', m_{d-1}] \models \varepsilon^p_{s,i} P_i$ for the required $a' \leq m_{d-1}$, $i = 1, \ldots, u$. Let these formulas be π_i, $i = 1, \ldots, u$. Then B^* is equivalent to

$$\exists T \exists S^p_1 \ldots \exists S^p_u \exists S^f_1 \ldots \exists S^f_v \left(\begin{array}{l} \left((\lceil T \rceil ; \lceil \neg T \rceil) \wedge \bigvee\limits_{s=1}^{t} H_s \wedge \bigwedge\limits_{i=1}^{u} \lceil \varepsilon^p_{s,i} S^p_i \rceil \wedge \bigwedge\limits_{j=1}^{v} \lceil \varepsilon^f_{s,j} S^f_j \rceil \right)^* \\ \wedge \bigwedge\limits_{i=1}^{u} \pi_i \wedge \bigwedge\limits_{j=1}^{v} \varphi_j \end{array} \right) . \quad (8)$$

$\Diamond_r F_j$ occurs in the left operand of *chop* in φ_j. As mentioned above, by the mirror equivalence of (4), F_j can be assumed to be the conjunction of some introspective C_j and some strictly future G_j. Let $C_{j,k}$ and $C'_{j,k}$, $k = 1, \ldots, n$, satisfy Lemma 5 for C_j. Then

$$\models ((\Diamond_r F_j \wedge \lceil\rceil) ; \top) \Leftrightarrow ((C_j \wedge G_j) ; \top) \vee \bigwedge_{k=1}^{n} C_{j,k} \Rightarrow \Diamond_r (C'_{j,k} \wedge G_j). \quad (9)$$

Since $h_{\Diamond_r}(G_j) < h_{\Diamond_r}(B)$ and $h_*(G_j) < h_*(B)$, G_j can be extracted from the left operand of *chop* in the RHS of (9). This produces a (non-strictly) future formula which is equivalent to $((\Diamond_r F_j \wedge \lceil\rceil) ; \top)$. After replacing $((\Diamond_r F_j \wedge \lceil\rceil) ; \top)$ by this future formula in (7), the $\Diamond_r$-subformulas of this future formula and the formulas $\Diamond_r (C'_{j,k} \wedge G_j)$ can be further extracted from the right operand of *chop* in (7) using the right equivalence of (6). This leads to a future equivalent of φ_j, by which we replace φ_j in (8), $j = 1, \dots, v$. We use the time mirror of (9) and the left equivalence of (6) to similarly replace π_i, $i = 1, \dots, u$, by some appropriate past equivalents. This leads to a separated formula as the operand of $\exists T \exists S_1^p \dots \exists S_u^p \exists S_1^f \dots \exists S_v^f$ in (8).

In order to obtain a separated equivalent to B^*, we need to eliminate this quantifier prefix. To this end, observe that the $\Diamond_l$- and $\Diamond_r$-subformulas which appear in the separated equivalents of π_i, $i = 1, \dots, u$, and φ_j, $j = 1, \dots, v$, have no occurrences of $T, S_1^p, \dots, S_u^p, S_1^f, \dots, S_v^f$, and are linked with the remaining introspective subformulas in the scope of $\exists T \exists S_1^p \dots \exists S_u^p \exists S_1^f \dots \exists S_v^f$, which may have such occurrences, by Boolean connectives only. Hence the $\Diamond_l$- and $\Diamond_r$-subformulas can be extracted using the De Morgan laws and

$$\models \exists S\,(X \vee Y) \Leftrightarrow \exists S\, X \vee \exists S\, Y, \text{ and, for } S\text{-free } X, \models \exists S(X \wedge Y) \Leftrightarrow X \wedge \exists S\, Y,$$

Then the quantifier prefix can be eliminated by Theorem 1, which is about introspective formulas only. Hence Theorem 2 holds about DC-NL* too.

4 Expressing the Weak Chop Inverses by the Neighbourhood Modalities and Separation for the Weak Chop Inverses

In this section we prove that the weak chop inverses are expressible in DC-NL, which means that separation applies to DC with these expanding modalities instead of $\Diamond_l$ and $\Diamond_r$ too.

Suppose that A_1, A_2, B are separated formulas in DC-NL (DC-NL*). Then the availability of conjunctive normal forms and the validity of the equivalences

$$(A_1 \wedge A_2)/B \Leftrightarrow A_1/B \wedge A_2/B$$

entails that we need to consider only formulas A/B where A is a disjunction of introspective formulas, strictly future formulas and strictly past formulas. Strictly past disjuncts P in the left operand of $(./.)$ can be extracted using the validity of

$$(A \vee P)/B \Leftrightarrow P \vee A/B.$$

The following proposition shows how to express A/B in case A is a disjunction of introspective and possibly negated $\Diamond_r$-formulas.

▶ **Proposition 6.** *Let A be a $\lceil P \rceil$-formula in* DC *(*DC**) and A_k, A'_k, $k = 1, \dots, n$ satisfy Lemma 5 for A. Let B be a $\lceil P \rceil$-formula in* DC-NL**. Let F be a strictly future formula. Then*

$$\models (A \vee F)/B \Leftrightarrow \bigvee_{k=1}^{n} A_k \wedge \Box_r(B \Rightarrow (A'_k \vee F)) \ . \tag{10}$$

Proof. ($\Rightarrow$): Let $I, [a, b]$ satisfy the RHS of (10). Consider an arbitrary $r \geq b$ such that $I, [b, r] \models B$. Then $I, [a, r] \models A \vee F$. There is a (unique) $k \in \{1, \dots, n\}$ such that $I, [a, b] \models A_k$. Hence $I, [b, r] \models A'_k \vee F$ follows from $I, [a, r] \models A \vee F$ and $\models A \Rightarrow \neg(A_k; \neg A'_k)$, which follows from Lemma 5. The ($\Leftarrow$) direction is trivial to check and we omit it. ◀

The formula for A/B in terms of $\Diamond_l$ and $\Diamond_r$ in the RHS of (10) can be further separated to extract past subformulas of B from the scope of $\Box_r$ as in DC-NL (DC-NL*). The above argument shows that $(./.)$-formulas whose operands are in the $\lceil P \rceil$-subset of DC-NL (DC-NL*)

have equivalents in the $\lceil P \rceil$-subset of DC-NL (DC-NL*) themselves. Observe that, in the presence of *chop*, it takes only $\Diamond_r$ to eliminate (./.). Similarly, (.\.), which is about looking to the left of reference interval, can be eliminated using only *chop* and $\Diamond_l$. As mentioned in the Preliminaries section, expressing $\Diamond_l$ and $\Diamond_r$ by means of (.\.) and (./.) is straightforward. This concludes our reduction of the $\lceil P \rceil$-subset of DC-NL (DC-NL*) with the weak chop inverses to the $\lceil P \rceil$-subset of DC-NL (DC-NL*), and entails that separation applies to that system too as stated in Theorem 3.

Concluding Remarks

In this paper we have shown how separation after Gabbay applies to the $\lceil P \rceil$-subsets of DC-NL and DC-NL*, the extensions of DC by the neighbourhood modalities. These subsets correspond to the subset of DC whose expressive completeness was demonstrated in [29].

The $\lceil S \rceil$-construct, which is definitive for the $\lceil P \rceil$-subsets of DC-NL and DC-NL*, has a considerable similarity with the *homogeneity principle* which is known from studies on neighbourhood logics of discrete time. That principle was proposed in [22, 20] and was adopted in a number of more recent works such as [7, 8, 9]. Unlike the *locality principle* from Moszkowski's (standard) discrete time ITL, where the satisfaction of an atomic proposition p is determined by the labeling of the initial state of the reference interval, homogeneity means that atomic proposition p must label all the states in the reference interval for p to hold at that interval as a formula. The two variants are ultimately interdefinable, but facilitate applications in a slightly different way. Homogeneity can be compared with DC's $\lceil P \rceil$ because $\lceil P \rceil$ means that P is supposed to hold "almost everywhere" in the reference interval. The main difference is that varying valuations at zero-length interval is negligible in real-time NL and DC, whereas the labeling of the only point in such intervals can be referred to in discrete time. This leads to different notions of strictly past and strictly future formulas. It is known that past expanding modalities increase the ultimate expressive power of discrete time ITL [21], and not just its succinctness, the latter being the case in past LTL. This adds to the relevance of algorithmic methods for interval-based expanding modalities in general.

Providing a separation theorem to the $\lceil P \rceil$-subset of DC-NL improves our understanding of the logic and may facilitate further results. One obvious avenue of future study would be to consider interval-based variants of the applications of separation that are known about point-based past LTL. In particular, one rather straightforward application would be to simplify the theoretical considerations that are needed for the study of extensions, especially branching time ones such as [27], by making it sufficient to consider future-only formulas, while still enjoying the succinctness contributed by the availability of past operators.

References

1 IEEE 1364-2005 - IEEE Standard for Verilog Hardware Description Language, 2005. URL: `https://ieeexplore.ieee.org/document/1620780`.

2 IEEE 1850-2010 - IEEE Standard for Property Specification Language (PSL), 2010. URL: `https://standards.ieee.org/standard/1850-2010.html`.

3 James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983. `doi:10.1145/182.358434`.

4 Rana Barua, Suman Roy, and Zhou Chaochen. Completeness of neighbourhood logic. In Christoph Meinel and Sophie Tison, editors, *STACS 99, Proceedings*, volume 1563 of *LNCS*, pages 521–530. Springer, 1999. `doi:10.1007/3-540-49116-3_49`.

5 Marc Boulé and Zeljko Zilic. Psl and sva assertion languages. In *Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-Fabrication Debugging and On-Line Monitoring*, pages 55–82. Springer Netherlands, Dordrecht, 2008. `doi:10.1007/978-1-4020-8586-4_4`.

6 Howard Bowman and Simon Thompson. A Decision Procedure and Complete Axiomatisation of Finite Interval Temporal Logic with Projection. *Journal of Logic and Computation*, 13(2):195–239, 2003.

7 Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, and Pietro Sala. Interval vs. point temporal logic model checking: an expressiveness comparison. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *FSTTCS 2016*, volume 65 of *LIPIcs*, pages 26:1–26:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.FSTTCS.2016.26`.

8 Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, and Pietro Sala. Interval vs. point temporal logic model checking: An expressiveness comparison. *ACM Trans. Comput. Log.*, 20(1):4:1–4:31, 2019. `doi:10.1145/3281028`.

9 Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, and Pietro Sala. Which fragments of the interval temporal logic HS are tractable in model checking? *Theor. Comput. Sci.*, 764:125–144, 2019. `doi:10.1016/j.tcs.2018.04.011`.

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

11 Antonio Cau, Ben Moszkowski, and Hussein Zedan. ITL web pages. URL: `http://www.antonio-cau.co.uk/ITL/`.

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

13 Dov Gabbay, Ian Hodkinson, and Mark Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects. Volume I.* Oxford University Press, 1994.

14 Dov M. Gabbay. Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In *Proceedings of the Colloquium of Temporal Logic in Specification*, volume 398 of *LNCS*, pages 67–89. Springer, 1989.

15 Dimitar P. Guelev. A syntactical proof of the canonical reactivity form for past linear temporal logic. *J. Log. Comput.*, 18(4):615–623, 2008. `doi:10.1093/logcom/exn002`.

16 Dimitar P. Guelev and Ben Moszkowski. A separation theorem for discrete-time interval temporal logic. *Journal of Applied Non-Classical Logics*, 32(1):28–54, 2022. `doi:10.1080/11663081.2022.2050135`.

17 François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Temporal logic with forgettable past. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 383–392. IEEE Computer Society, 2002. `doi:10.1109/LICS.2002.1029846`.

18 François Laroussinie and Philippe Schnoebelen. A hierarchy of temporal logics with past (extended abstract). In Patrice Enjalbert, Ernst W. Mayr, and Klaus W. Wagner, editors, *STACS 94, Proceedings*, volume 775 of *LNCS*, pages 47–58. Springer, 1994. `doi:10.1007/3-540-57785-8_130`.

19 Zohar Manna and Amir Pnueli. A Hierarchy of Temporal Properties. In *9th Symposium on Principles of Distributed Computing*, pages 377–408. ACM Press, 1990.

20 Alberto Molinari, Angelo Montanari, Aniello Murano, Giuseppe Perelli, and Adriano Peron. Checking interval properties of computations. *Acta Informatica*, 53(6-8):587–619, 2016. `doi:10.1007/s00236-015-0250-1`.

21 Dario Della Monica, Angelo Montanari, and Pietro Sala. The importance of the past in interval temporal logics: The case of propositional neighborhood logic. In Alexander Artikis, Robert Craven, Nihan Kesim Cicekli, Babak Sadighi, and Kostas Stathis, editors, *Logic Programs, Norms and Action - Essays in Honor of Marek J. Sergot on the Occasion of His 60th Birthday*, volume 7360 of *LNCS*, pages 79–102. Springer, 2012. `doi:10.1007/978-3-642-29414-3_6`.

22 Angelo Montanari, Aniello Murano, Giuseppe Perelli, and Adriano Peron. Checking interval properties of computations. In Amedeo Cesta, Carlo Combi, and François Laroussinie, editors, *Proceedings of TIME 2014*, pages 59–68. IEEE Computer Society, 2014. `doi:10.1109/TIME.2014.24`.

23 Ben Moszkowski. *Reasoning about Digital Circuits*. Ph.D. thesis, Department of Computer Science, Stanford University, 1983. URL: `http://www.antonio-cau.co.uk/ITL/publications/reports/thesis-ben.pdf`.

24 Ben Moszkowski. Temporal Logic For Multilevel Reasoning About Hardware. *IEEE Computer*, 18(2):10–19, 1985.

25 Ben Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, 1986. URL: `http://www.antonio-cau.co.uk/ITL/publications/reports/tempura-book.pdf`.

26 Paritosh K. Pandya. Some extensions to propositional mean-value caculus: Expressiveness and decidability. In Hans Kleine Büning, editor, *CSL '95, Selected Papers*, volume 1092 of *LNCS*, pages 434–451. Springer, 1995. `doi:10.1007/3-540-61377-3_52`.

27 Paritosh K. Pandya. Model checking CTL*[DC]. In Tiziana Margaria and Wang Yi, editors, *TACAS 2001, Proceedings*, volume 2031 of *LNCS*, pages 559–573. Springer, 2001. `doi:10.1007/3-540-45319-9_38`.

28 Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Symposium Foundations of Computer Science*, pages 46–57. IEEE, 1977.

29 Alexander Moshe Rabinovich. Expressive completeness of duration calculus. *Inf. Comput.*, 156(1-2):320–344, 2000. `doi:10.1006/inco.1999.2816`.

30 Zhou Chaochen and Michael R. Hansen. *Duration Calculus. A Formal Approach to Real-Time Systems*. Springer, 2004.

31 Zhou Chaochen, Michael R. Hansen, and Peter Sestoft. Decidability and undecidability results for duration calculus. In Patrice Enjalbert, Alain Finkel, and Klaus W. Wagner, editors, *STACS 93, Proceedings*, volume 665 of *LNCS*, pages 58–68. Springer, 1993. `doi:10.1007/3-540-56503-5_8`.

32 Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1991.

A Quantitative Extension of Interval Temporal Logic over Infinite Words

Laura Bozzelli
University of Napoli "Federico II", Italy

Adriano Peron
University of Napoli "Federico II", Italy

Abstract

Model checking (MC) for Halpern and Shoham's interval temporal logic HS has been recently investigated in a systematic way, and it is known to be decidable under three distinct semantics (state-based, trace-based and tree-based semantics), all of them assuming *homogeneity* in the propositional valuation. Here, we focus on the *trace-based semantics*, where the main semantic entities are the infinite execution paths (traces) of the given Kripke structure. We introduce a quantitative extension of HS over traces, called *Difference* HS (DHS), allowing one to express timing constraints on the difference among interval lengths (*durations*). We show that MC and satisfiability of full DHS are in general undecidable, so, we investigate the decidability border for these problems by considering natural syntactical fragments of DHS. In particular, we identify a maximal decidable fragment DHS_{simple} of DHS proving in addition that the considered problems for this fragment are at least 2EXPSPACE-hard. Moreover, by exploiting new results on linear-time hybrid logics, we show that for an equally expressive fragment of DHS_{simple}, the problems are EXPSPACE-complete. Finally, we provide a characterization of HS over traces by means of the one-variable fragment of a novel hybrid logic.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Interval temporal logic, Homogeneity Assumption, Quantitative Constraints, Model checking, Decision Procedures, Complexity issues, Linear-time Hybrid Logics

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.11

Related Version *Full Version*: `https://arxiv.org/abs/2206.13920` [12]

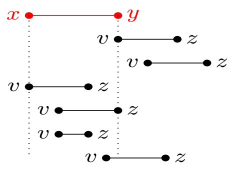
1 Introduction

Interval Temporal Logics (ITLs, see [17, 29, 34]) provide an alternative setting for reasoning about time with respect to the more popular *Point-based* Temporal Logics (PTLs) whose most known representatives are the linear-time temporal logic LTL [30] and the branching-time temporal logics CTL and CTL* [15]. ITLs assume intervals, instead of points, as their primitive temporal entities allowing one to specify temporal properties that involve, e.g., actions with duration, accomplishments, and temporal aggregations, which are inherently "interval-based", and thus cannot be naturally expressed by PTLs. The most prominent example of ITLs is *Halpern and Shoham's modal logic of time intervals* (HS) [17] featuring modalities for any Allen's relation [1]. The satisfiability problem for HS turns out to be highly undecidable for all interesting (classes of) linear orders [17] both for the full logic and most of its fragments [13, 22, 26].

Model checking (MC) of (finite) Kripke structures against HS has been investigated in recent papers [23, 24, 25, 27, 5, 6, 7, 28, 3, 8] which provide more encouraging results. In the model checking setting, each finite path of a Kripke structure is an *interval* having a labelling derived from the labelling of the component states: a proposition letter holds over an interval if and only if it holds over each component state (*homogeneity assumption* [31]). Most of the results have been obtained by adopting the so-called *state-based semantics* [27]:

intervals/paths are "forgetful" of the history leading to their starting state, and time branches both in the future and in the past. In this setting, MC of full HS is decidable: the problem is at least EXPSPACE-hard [4], while the only known upper bound is non-elementary [27]. The known complexity bounds for full HS coincide with those for the linear-time fragment BE of HS which features modalities $\langle B\rangle$ and $\langle E\rangle$ for prefixes and suffixes. Whether or not model checking for BE can be solved elementarily is a difficult open question. On the other hand, in the state-based setting, the exact complexity of MC for many meaningful (linear-time or branching-time) syntactic fragments of HS, which ranges from **coNP** to $\mathbf{P}^{\mathbf{NP}}$, PSPACE, and beyond, has been determined in a series of papers [5, 7, 9, 11, 8].
The expressiveness of HS with the state-based semantics has been studied in [6], together with other two decidable variants: the *computation-tree-based semantics* and the *traces-based* one. For the first variant, past is linear: each interval may have several possible future, but only a unique past. Moreover, past is finite and cumulative, and is never forgotten. The trace-based approach instead relies on a linear-time setting, where the infinite paths (traces) of the given Kripke structure are the main semantic entities. It is known that the computation-tree-based variant of HS is expressively equivalent to finitary CTL^* (the variant of CTL^* with quantification over finite paths), while the trace-based variant is equivalent to LTL [6]. The state-based variant is more expressive than the computation-tree-based variant and expressively incomparable with both LTL and CTL^* [6].
In this paper, we introduce a quantitative extension of the interval temporal logic HS under the *trace-based semantics*, called *Difference* HS (DHS). The extension is obtained by means of equality and inequality constraints on the temporal modalities which allow to specify integer bounds on the difference between the durations (lengths) of the current interval and the interval selected by the modality. The logic DHS can also encode in a succinct way constraints on the duration of the current interval. Thus, the considered framework non-trivially generalizes well-known discrete-time quantitative extensions of standard LTL, such as Metric Temporal Logic (MTL) [20], where one can essentially express integer bounds on the duration of the interval having as endpoints the current position and the one selected by the temporal modality.
We prove that MC and satisfiability of full DHS are in general undecidable. Thus, we investigate the decidability border of these problems by considering the syntactical fragments of DHS obtained by restricting the set of allowed constrained modalities. In particular, we prove that for the syntactical fragment, namely DHS_{simple}, whose constrained temporal modalities are associated with the Allen's relations subsuming the subset relation or its inverse, the problems are decidable, though at least 2EXPSPACE-hard. On the other hand, we show that any constrained modality not supported by DHS_{simple} is inherently problematic, since the addition to HS of such a modality leads to undecidability. These results are a little surprising since it is well-known that under the adopted strict semantics admitting singleton intervals, all temporal modalities in HS can be expressed in terms of the ones associated with the Allen's relations subsuming the subset relation or its inverse. Additionally, we identify an expressively complete fragment of DHS_{simple} for which satisfiability and model checking are shown to be EXPSPACE-complete. The upper bound in EXPSPACE is obtained by an elegant automaton-theoretic approach which exploits as a preliminary step a linear-time translation of the fragment of DHS_{simple} into a quantitative extension of the one-variable fragment of *linear-time hybrid logic* HL [16, 32, 2]. Finally, we provide a characterization of HS over traces in terms of a novel hybrid logic, namely SHL_1, which lies between the one-variable and the two-variable fragment of HL. We prove that there are linear-time translations from HS formulas into equivalent formulas of SHL_1, and vice versa.

Table 1 Allen's relations and corresponding HS modalities.

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

2 Preliminaries

We fix the following notation. Let $\mathbb{Z}$ be the set of integers, $\mathbb{N}$ the set of natural numbers, and $\mathbb{N}_+ \overset{\text{def}}{=} \mathbb{N} \setminus \{0\}$. For a finite or infinite word w over some alphabet, $|w|$ denotes the length of w ($|w| = \infty$ if w is infinite) and for all $0 \leq i < |w|$, $w(i)$ is the $(i+1)$-th letter of w.

We fix a finite set $\mathcal{AP}$ of atomic propositions. A *trace* is an infinite word over $2^{\mathcal{AP}}$.

Let $\mathfrak{F}$ and $\mathfrak{F}'$ be two logics interpreted over traces. For a formula $\varphi \in \mathfrak{F}$, $\mathcal{L}(\varphi)$ denotes the set of traces satisfying φ. Given $\varphi \in \mathfrak{F}$ and $\varphi' \in \mathfrak{F}'$, φ and φ' are *equivalent* if $\mathcal{L}(\varphi) = \mathcal{L}(\varphi')$. The satisfiability problem for $\mathfrak{F}$ is checking for a given formula $\varphi \in \mathfrak{F}$, whether $\mathcal{L}(\varphi) \neq \emptyset$.

Kripke Structures. A *(finite) Kripke structure* over $\mathcal{AP}$ is a tuple $\mathcal{K} = (\mathcal{AP}, S, E, Lab, s_0)$, where S is a finite set of states, $E \subseteq S \times S$ is a left-total transition relation, $Lab : S \to 2^{\mathcal{AP}}$ is a labelling function assigning to each state s the set of propositions that hold over it, and $s_0 \in S$ is the initial state. An infinite path π of $\mathcal{K}$ is an infinite word over S such that $\pi(0) = s_0$ and $(\pi(i), \pi(i+1)) \in E$ for all $i \geq 0$. A finite path of $\mathcal{K}$ is a non-empty infix of some infinite path of $\mathcal{K}$. An infinite path π induces the trace given by $Lab(\pi(0))Lab(\pi(1))\ldots$. We denote by $\mathcal{L}(\mathcal{K})$ the set of traces associated with the infinite paths of $\mathcal{K}$. For a logic $\mathfrak{F}$ interpreted over traces, the *model checking* (MC) *problem against* $\mathfrak{F}$ is checking for a given Kripke structure $\mathcal{K}$ and a formula $\varphi \in \mathfrak{F}$, whether $\mathcal{L}(\mathcal{K}) \subseteq \mathcal{L}(\varphi)$.

2.1 Allen's relations and Interval Temporal Logic HS

An interval algebra to reason about intervals and their relative orders was proposed by Allen [1], while a systematic logical study of interval representation and reasoning was done a few years later by Halpern and Shoham, who introduced the interval temporal logic HS featuring one modality for each Allen relation [17].

Let $\mathbb{U} = (U, <)$ be a linear order over the nonempty set $U \neq \emptyset$, and $\leq$ be the reflexive closure of $<$. Given two elements $x, y \in U$ such that $x \leq y$, we denote by $[x, y]$ the (non-empty closed) *interval* over U given by the set of elements $z \in U$ such that $x \leq z$ and $z \leq y$. We denote the set of all intervals over $\mathbb{U}$ by $\mathbb{I}(\mathbb{U})$. Table 1 gives a graphical representation of the Allen's relations $\mathcal{R}_A$, $\mathcal{R}_L$, $\mathcal{R}_B$, $\mathcal{R}_E$, $\mathcal{R}_D$, and $\mathcal{R}_O$ for the given linear order together with the corresponding HS (existential) modalities. For each $X \in \{A, L, B, E, D, O\}$, the Allen's relation $\mathcal{R}_{\overline{X}}$ is defined as the inverse of relation $\mathcal{R}_X$, i.e. $[x, y]\,\mathcal{R}_{\overline{X}}\,[v, z]$ if $[v, z]\mathcal{R}_X[x, y]$.

HS formulas φ over $\mathcal{AP}$ are defined as follows:

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

where $p \in \mathcal{AP}$ and $\langle X \rangle$ is the existential temporal modality for the (non-trivial) Allen's relation $\mathcal{R}_X$, where $X \in \{A, L, B, E, D, O, \overline{A}, \overline{L}, \overline{B}, \overline{E}, \overline{D}, \overline{O}\}$. The size $|\varphi|$ of a formula φ is the number of distinct subformulas of φ. We also exploit the standard logical connectives $\vee$

and $\rightarrow$ as abbreviations, and for any temporal modality $\langle X \rangle$, the dual universal modality $[X]$ defined as: $[X]\,\psi \overset{\text{def}}{=} \neg\langle X \rangle \neg\psi$. Given any subset of Allen's relations $\{\mathcal{R}_{X_1}, .., \mathcal{R}_{X_n}\}$, we denote by $\mathsf{X_1 \cdots X_n}$ the HS fragment featuring temporal modalities for $\mathcal{R}_{X_1}, .., \mathcal{R}_{X_n}$ only.

The logic HS is interpreted on *interval structures* $\mathcal{S} = (\mathcal{AP}, \mathbb{U}, \mathit{Lab})$, which are linear orders $\mathbb{U}$ equipped with a labelling function $\mathit{Lab} : \mathbb{I}(\mathbb{U}) \rightarrow 2^{\mathcal{AP}}$ assigning to each interval the set of propositions that hold over it. Given an HS formula φ and an interval $I \in \mathbb{I}(\mathbb{U})$, the satisfaction relation $I \models_{\mathcal{S}} \varphi$, meaning that φ holds at the interval I of $\mathcal{S}$, is inductively defined as follows (we omit the semantics of the Boolean connectives which is standard):

$$\begin{array}{ll} I \models_{\mathcal{S}} p & \Leftrightarrow p \in \mathit{Lab}(I) \\ I \models_{\mathcal{S}} \langle X \rangle\, \varphi & \Leftrightarrow \text{there is an interval } J \in \mathbb{I}(\mathbb{U}) \text{ such that } I\,\mathcal{R}_X\,J \text{ and } J \models_{\mathcal{S}} \varphi \end{array}$$

It is worth noting that we assume the *non-strict semantics of HS*, which admits intervals consisting of a single point. Under such an assumption, all HS-temporal modalities can be expressed in terms of $\langle B \rangle, \langle E \rangle, \langle \overline{B} \rangle$, and $\langle \overline{E} \rangle$ [34]. As an example, $\langle D \rangle\,\varphi$ can be expressed in terms of $\langle B \rangle$ and $\langle E \rangle$ as $\langle B \rangle\,\langle E \rangle\,\varphi$, while $\langle A \rangle\,\varphi$ can be expressed in terms of $\langle E \rangle$ and $\langle \overline{B} \rangle$ as $([E]\,\neg\top \wedge (\varphi \vee \langle \overline{B} \rangle\,\varphi)) \vee \langle E \rangle([E]\,\neg\top \wedge (\varphi \vee \langle \overline{B} \rangle\,\varphi))$.

Interpretation of HS over traces. In this paper, we focus on interval structures $\mathcal{S} = (\mathcal{AP}, (\mathbb{N}, <), \mathit{Lab})$ over the standard linear order on $\mathbb{N}$ ($\mathbb{N}$-interval structures for short) satisfying the *homogeneity principle*: a proposition holds over an interval if and only if it holds over all its subintervals. Formally, $\mathcal{S}$ is *homogeneous* if for every interval $[i, j]$ over $\mathbb{N}$ and every $p \in \mathcal{AP}$, it holds that $p \in \mathit{Lab}([i, j])$ if and only if $p \in \mathit{Lab}([h, h])$ for every $h \in [i, j]$. Note that homogeneous $\mathbb{N}$-interval structures over $\mathcal{AP}$ correspond to traces where, intuitively, each interval is mapped to an infix of the trace. Formally, each trace w induces the homogeneous $\mathbb{N}$-interval structure $\mathcal{S}(w)$ whose labeling function Lab_w is defined as follows: for all $i, j \in \mathbb{N}$ with $i \leq j$ and $p \in \mathcal{AP}$, $p \in \mathit{Lab}_w([i, j])$ if and only if $p \in w(h)$ for all $h \in [i, j]$. This mapping from traces to homogeneous $\mathbb{N}$-interval structures over $\mathcal{AP}$ is evidently a bijection. For a trace w, an interval I over $\mathbb{N}$, and an HS formula φ, we write $I \models_w \varphi$ to mean that $I \models_{\mathcal{S}(w)} \varphi$. The trace w satisfies φ, written $w \models \varphi$, if $[0, 0] \models_w \varphi$. For an interval $I = [i, j]$ over $\mathbb{N}$, we denote by $|I|$ the length of I, given by $j - i + 1$.

It is known that HS over traces has the same expressiveness as standard LTL [6], where the latter is expressively complete for standard first-order logic FO over traces [19]. In particular, the fragment AB of HS is sufficient for capturing full LTL [6]: given an LTL formula, one can construct in linear-time an equivalent AB formula [6]. Note that when interpreted on infinite words w, modality $\langle B \rangle$ allows to select proper non-empty prefixes of the current infix subword of w, while modality $\langle A \rangle$ allows to select subwords whose first position coincides with the last position of the current interval. For each $k \geq 1$, we denote by len_k the B formula capturing the intervals of length k: $\mathsf{len}_k \overset{\text{def}}{=} (\underbrace{\langle B \rangle \ldots \langle B \rangle}_{k-1 \text{ times}} \top) \wedge (\underbrace{[B] \ldots [B]}_{k \text{ times}} \neg\top)$.

3 Difference Interval Temporal Logic

In this section, we introduce a quantitative extension of the logic HS under the trace-based semantics, we call *Difference HS* (DHS for short). The extension is obtained by means of equality and inequality constraints on the temporal modalities of HS which allow to compare the difference between the length of the interval selected by the temporal modality and the length of the current interval with an integer constant.

The set of DHS formulas φ over $\mathcal{AP}$ is inductively defined as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle \mathrm{X} \rangle \varphi \mid \langle \mathrm{X} \rangle_{\Delta \sim c} \varphi$$

where $p \in \mathcal{AP}$, $\sim \in \{<, \leq, =, >, \geq\}$, $c \in \mathbb{Z}$, and $\langle \mathrm{X} \rangle_{\Delta \sim c}$ is the existential *constrained* temporal modality for the Allen's relation $\mathcal{R}_X$ where $X \in \{A, L, B, E, D, O, \overline{A}, \overline{L}, \overline{B}, \overline{E}, \overline{D}, \overline{O}\}$. We exploit the symbol Δ in $\langle \mathrm{X} \rangle_{\Delta \sim c}$ to emphasize that the constraint $\sim c$ refer to the difference between the lengths of two intervals, the one selected by the modality $\langle \mathrm{X} \rangle$ and the current one. For any constrained modality $\langle \mathrm{X} \rangle_{\Delta \sim c}$, the dual universal modality $[\mathrm{X}]_{\Delta \sim c}$ is an abbreviation for $\neg \langle \mathrm{X} \rangle_{\Delta \sim c} \neg\varphi$. We assume that the constants c in the difference constraints are encoded in binary. Thus, the size $|\varphi|$ of a DHS formula φ is defined as the number of distinct subformulas of φ multiplied the number of bits for encoding the maximal constant occurring in φ. The semantics of the constrained modalities is as follows:

- $I \models_w \langle \mathrm{X} \rangle_{\Delta \sim c} \varphi \Leftrightarrow$ for some interval J such that $I\, \mathcal{R}_X\, J$ and $|J| - |I| \sim c$, $J \models_w \varphi$.

Note that the constrained modalities of the form $\langle \mathrm{X} \rangle_{\Delta \prec c}$ where $\prec \in \{<, \leq\}$ are *upward-monotone* in the sense that for all constants c and c' such that $c' \geq c$, $I \models_w \langle \mathrm{X} \rangle_{\Delta \prec c} \varphi$ entails that $I \models_w \langle \mathrm{X} \rangle_{\Delta \prec c'} \varphi$. On the other hand, the constrained modalities of the form $\langle \mathrm{X} \rangle_{\Delta \succ c}$ where $\succ \in \{>, \geq\}$ are *downward-monotone*, i.e., for all constants c and c' such that $c' \leq c$, $I \models_w \langle \mathrm{X} \rangle_{\Delta \succ c} \varphi$ entails that $I \models_w \langle \mathrm{X} \rangle_{\Delta \succ c'} \varphi$. Thus, we say that a formula φ is *monotonic* if it does not use equality constraints $\Delta = c$ as subscripts of the temporal modalities.
We consider the following fragments of DHS and their monotonic versions:

- The fragment DHS_{simple} which disallows constrained modalities for the Allen's relations $\mathcal{R}_A$, $\mathcal{R}_L$, $\mathcal{R}_O$, and their inverses, and for any Allen's relation $\mathcal{R}_X$, the fragment DHS_X allowing constrained modalities for the Allen's relation $\mathcal{R}_X$ only.
- For any subset of Allen's relations $\{\mathcal{R}_{X_1}, .., \mathcal{R}_{X_n}\}$, the fragment $\mathsf{D}(\mathsf{X_1} \ldots \mathsf{X_n})$ featuring temporal modalities for $\mathcal{R}_{X_1}, .., \mathcal{R}_{X_n}$ only, and the common fragment of $\mathsf{D}(\mathsf{X_1} \ldots \mathsf{X_n})$ and DHS_{simple}, denoted by $\mathsf{D}_{simple}(\mathsf{X_1} \ldots \mathsf{X_n})$.

Expressiveness issues. As mentioned in Section 2, all the temporal modalities of HS can be expressed in terms of $\langle \mathrm{B} \rangle$, $\langle \mathrm{E} \rangle$, $\langle \overline{\mathrm{B}} \rangle$, and $\langle \overline{\mathrm{E}} \rangle$. In the considered quantitative setting, these interdefinability results cannot be generalized to the constrained versions of the temporal modalities. In particular, we will show in Section 5 that the fragment DHS_{simple} of DHS, featuring constrained modalities only for the Allen's relations $\mathcal{R}_B$, $\mathcal{R}_D$, $\mathcal{R}_E$, and their inverses, is not more expressive than HS. On the other hand, we will establish in Section 4 that the fragments DHS_X, where $X \in \{A, L, O, \overline{A}, \overline{L}, \overline{O}\}$, are highly undecidable.

Unlike HS, in DHS_{simple} we can succinctly express that an arbitrary HS property φ holds in the *maximal proper sub-intervals* of the current non-singleton interval by the formula $(\langle \mathrm{E} \rangle_{\Delta \geq -1} \varphi) \wedge (\langle \mathrm{B} \rangle_{\Delta \geq -1} \varphi)$. Moreover, we can succinctly encode constraints on the length of the current interval. For an integer $n > 0$, the DHS_{simple} formula $\langle \mathrm{B} \rangle_{\Delta \leq -n+1} \top$ (resp., $\neg \langle \mathrm{B} \rangle_{\Delta \leq -n} \top$) characterizes the intervals of length at least (resp., at most) n.

▶ **Example 1.** We consider the behaviour of a scheduler serving N processes which continuously request the use of a common resource. The behaviour of each process P_i, with $1 \leq i \leq N$, is represented by the Kripke structure $\mathcal{K}_{P_i}$, depicted in Figure 1, whose atomic propositions p_I^i, p_R^i, and p_U^i label the states where the process is idling, requests the resource, and uses the resource, respectively. The behaviour of the scheduler H is modeled by the Kripke structure $\mathcal{K}_H$ in Figure 1 whose propositions q_I, $q_{U_1}, \ldots q_{U_N}$ label the states where H is idling or assigns the resource to the i-th process (proposition q_{U_i}). The considered Kripke structure $\mathcal{K}_{Sched}$, depicted in Figure 1 for $N = 2$, is the Cartesian product of the Kripke

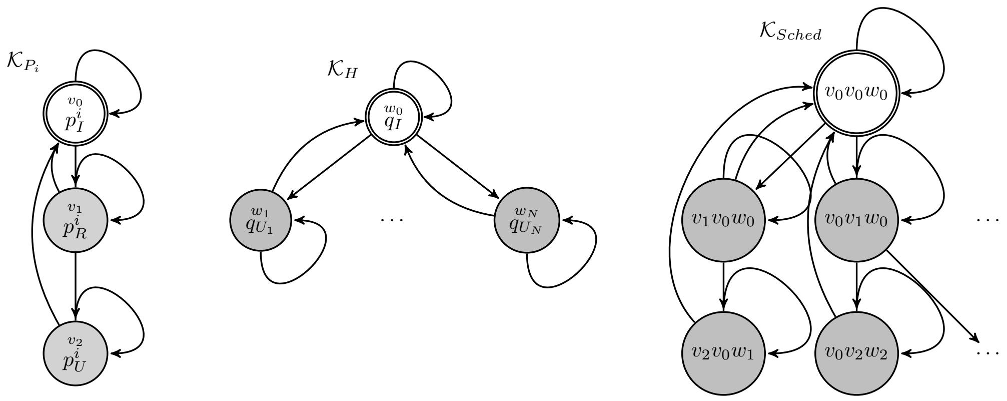


Figure 1 The Kripke structure $\mathcal{K}_{Sched}$ for two processes.

structures $\mathcal{K}_{P_1}, \ldots, \mathcal{K}_{P_N}, \mathcal{K}_H$ with the additional requirement that the scheduler is in state w_i *iff* the i-th process is in state v_2. The set of atomic propositions labelling each compound state is the union of the sets of propositions labelling the component states.

As an example of specification, we consider the requirement that the i-th process can unsuccessfully iterate a request (i.e., without finally having the resource granted) for an interval of at least m and at most M time units. This can be expressed in DHS_{simple} as:

$$[\mathrm{A}]\,[\mathrm{A}][(Max_{p_R^i} \wedge \langle \overline{\mathrm{B}} \rangle_{\Delta \leq 1} \langle \mathrm{E} \rangle\, p_I^i) \rightarrow (\langle \mathrm{B} \rangle_{\Delta \leq -m+1} \top \wedge \neg \langle \mathrm{B} \rangle_{\Delta \leq -M} \top)]$$

where for a proposition p, $Max_p \overset{\text{def}}{=} p \wedge (\neg \langle \overline{\mathrm{B}} \rangle\, p) \wedge (\neg \langle \overline{\mathrm{E}} \rangle\, p)$ captures the maximal length intervals where p homogeneously holds. Note that $\langle \overline{\mathrm{B}} \rangle_{\Delta \leq 1} \langle \mathrm{E} \rangle\, p_I^i$ ensures that the maximal homogeneous interval where p_R^i holds is followed by a p_I^i-state.

4 Undecidability of DHS

In this section, we establish that model checking and satisfiability for the novel logic DHS are highly undecidable even for the fragments DHS_X, where $X \in \{A, L, O, \overline{A}, \overline{L}, \overline{O}\}$.

▶ **Theorem 2.** *Model checking and satisfiability for the fragment DHS_X of DHS, where $X \in \{A, L, O, \overline{A}, \overline{L}, \overline{O}\}$, are Σ_1^1-hard even if the unique constant used in the constraints is 0, and in case $X \in \{A, O, \overline{A}, \overline{O}\}$ even if the unique exploited constraint is ≥ 0 (or, dually, ≤ 0).*

We prove Theorem 2 for the part concerning the satisfiability problem for the fragments DHS_A, DHS_L, and DHS_O (the parts for the model checking problem and for the fragments $\mathsf{DHS}_{\overline{A}}$, $\mathsf{DHS}_{\overline{L}}$, and $\mathsf{DHS}_{\overline{O}}$ being similar). We provide polynomial-time reductions from the *recurrence problem* of *non-deterministic Minsky 2-counter machines* [18]. Fix such a machine which is a tuple $M = (Q, \Delta, \delta_{init}, \delta_{rec})$, where Q is a finite set of (control) locations, $\Delta \subseteq Q \times L \times Q$ is a transition relation over the instruction set $L = \{\mathsf{inc}, \mathsf{dec}, \mathsf{if\_zero}\} \times \{1, 2\}$, and $\delta_{init} \in \Delta$ and $\delta_{rec} \in \Delta$ are two designated transitions, the initial and the recurrent one. For each counter $c \in \{1, 2\}$, let $Inc(c)$, $Dec(c)$, and $Zero(c)$ be the sets of transitions $\delta \in \Delta$ whose instruction is (inc, c), (dec, c), and $(\mathsf{if\_zero}, c)$, respectively.

An M-configuration is a pair (δ, ν) consisting of a transition $\delta \in \Delta$ and a counter valuation $\nu : \{1, 2\} \rightarrow \mathbb{N}$. A computation of M is an *infinite* sequence of configurations of the form $((q_0, (op_0, c_0), q_1), \nu_0), ((q_1, (op_1, c_1), q_2), \nu_1), \ldots$ such that for each $i \geq 0$: (i) $\nu_{i+1}(3 - c_i) = \nu_i(3 - c_i)$; (ii) $\nu_{i+1}(c_i) = \nu_i(c_i) + 1$ if $op_i = \mathsf{inc}$; (iii) $\nu_{i+1}(c_i) = \nu_i(c_i) - 1$

if op_i = dec; and (iv) $\nu_{i+1}(c_i) = \nu_i(c_i) = 0$ if op_i = if_zero. A *recurrent computation* is a computation starting at the initial configuration (δ_{init}, ν_0), where $\nu_0(c) = 0$ for each $c \in \{1, 2\}$, which visits the transition δ_{rec} infinitely often. The *recurrence problem* is to decide whether for the given machine M, there is a recurrent computation. This problem is known to be Σ^1_1-complete [18].

For each $X \in \{A, L, O\}$, we construct a DHS_X formula $\varphi_{M,X}$ such that M has a recurrent computation *iff* φ_M is satisfiable. The reduction for the fragment DHS_L, given in the following, is quite different from the ones for the fragments DHS_A and DHS_O, which are given in [12]. Indeed, while the quantitative versions of modalities $\langle \mathrm{A} \rangle$ and $\langle \mathrm{O} \rangle$ allow to impose quantitative constraints on adjacent encodings of M-configurations, this is not possible for the quantitative version of modality $\langle \mathrm{L} \rangle$ whose semantics is not "local", and for this modality, a different encoding of the computations of M is required.

We exploit some auxiliary DHS formulas. Let ψ be an arbitrary DHS formula. Formulas *left*(ψ) and *right*(ψ) assert that ψ holds at the singular intervals corresponding to the left and right endpoints, respectively, of the current interval.

$$\mathit{left}(\psi) \overset{\text{def}}{=} (\mathsf{len}_1 \wedge \psi) \vee \langle \mathrm{B} \rangle (\mathsf{len}_1 \wedge \psi) \qquad \mathit{right}(\psi) \overset{\text{def}}{=} \langle \mathrm{A} \rangle (\mathsf{len}_1 \wedge \psi)$$

For the current interval $[i, j]$, *right_next*(ψ) (resp., *left_next*(ψ)) asserts that ψ holds at the singleton interval $[j+1, j+1]$ (resp., $[i+1, i+1]$), while *Int*(ψ) requires that there is an internal position $i < h < j$ such that ψ holds at the singleton interval $[h, h]$.

$$\mathit{right\_next}(\psi) \overset{\text{def}}{=} \langle \mathrm{A} \rangle (\mathsf{len}_2 \wedge \langle \mathrm{A} \rangle (\mathsf{len}_1 \wedge \psi)) \qquad \mathit{left\_next}(\psi) \overset{\text{def}}{=} \mathit{left}(\mathit{right\_next}(\psi))$$

$$\mathit{Int}(\psi) \overset{\text{def}}{=} \langle \mathrm{B} \rangle (\neg \mathsf{len}_1 \wedge \mathit{right}(\psi))$$

Reduction from the recurrence problem for DHS_L. Some ideas in the proposed reduction for the logic DHS_L are taken from [14], where it is shown that model checking one-counter automata against LTL with registers in undecidable.

We first provide a characterization of the recurrent computations of M. Let $\xi = \delta_0, \delta_1, \ldots$ be an infinite sequence of M-transitions. We say that ξ satisfy the *consecution requirement* if (i) $\delta_0 = \delta_{init}$, (ii) for all $i \geq 0$, δ_i is of the form (q_i, op_i, q_{i+1}), and (iii) for infinitely many $j \geq 0$, it holds that $\delta_j = \delta_{rec}$. In order to characterize the sequences ξ for which there exists a corresponding computation of M, we associate a positive natural number (called *value*) to each transition δ_i along ξ. For each counter $c \in \{1, 2\}$, we require that the value associated to a transition δ_i of ξ which increments counter c is obtained by incrementing the natural number associated to the previous incrementation of counter c, if any, along ξ. A similar requirement is imposed on the transitions along ξ decrementing counter c except that the values associated to c-decrementations must not exceed the values associated to previous c-incrementations. Intuitively, this ensures that at each position $i \geq 0$ along ξ, the value of counter c is never negative. In order to simulate the zero-test, we require that for each transition δ_i associated to a zero-test for c, the previous values associated to c-incrementations correspond to previous values associated to c-decrementions.

Formally, a *flat configuration* is a pair (δ, n) consisting of a transition $\delta \in \Delta$ and a positive natural number $n > 0$ such that $n = 1$ if $\delta \in Zero(c)$ for some counter c. We say that n is the value of (δ, n). A *well-formed M-sequence* is an infinite sequence $\rho = (\delta_0, n_0), (\delta_1, n_1), \ldots$ of flat configurations satisfying the following requirements:

- The infinite sequence of transitions $\delta_0, \delta_1, \ldots$ satisfies the consecution requirement.
- *Increment progression* (resp., *Decrement progression*): for each counter $c \in \{1,2\}$, let $\xi = (\delta_{i_0}, n_{i_0}), (\delta_{i_1}, n_{i_1}), \ldots$ be the (possibly empty) ordered sub-sequence of the flat configurations in ρ associated with incrementation (resp., decrementation) of counter c. Then, $n_{i_0} = 1$ and $n_{i_h} = n_{i_{h-1}} + 1$ for all $0 < h < |\xi|$.
- *Increment domination*: for each $c \in \{1,2\}$ and $j \geq 0$ such that $\delta_j \in \mathit{Dec}(c)$, there is $0 \leq h < j$ such that $\delta_h \in \mathit{Inc}(c)$ and $n_h \geq n_j$.
- *Zero-test checking*: let $c \in \{1,2\}$ and $j \geq 0$ such that $\delta_j \in \mathit{Zero}(c)$ and there are $h < j$ such that δ_h is a c-incrementation or c-decrementation. Then, the greatest $h_{\max}$ of such h is associated to a c-decrementation and for each $h < h_{\max}$ such that δ_h is a c-incrementation, it holds that $n_h \leq n_{h_{\max}}$.

▶ **Lemma 3.** *There is a recurrent computation of M* iff *there is a well-formed M-sequence.*

Construction of the DHS_L formula $\varphi_{L,M}$. Let $\mathcal{AP} \overset{\text{def}}{=} \Delta \cup \{1, \#\}$. A flat configuration (δ, n) is encoded by the finite word $\{\delta\} \cdot \{1\}^n \cdot \{\#\}$. A well-formed M-sequence $\rho = (\delta_0, n_0), (\delta_1, n_1), \ldots$ is encoded by the trace obtained by concatenating the codes of the flat configurations visited by ρ starting from the first one.

We construct a DHS_L formula $\varphi_{L,M}$ characterizing the well-formed M-sequences.

$$\varphi_{L,M} \overset{\text{def}}{=} \varphi_{\mathsf{con}} \wedge \varphi_{\mathsf{inc}} \wedge \varphi_{\mathsf{dec}} \wedge \varphi_{\mathsf{if\_zero}} \wedge \varphi_{\mathsf{dom}}$$

The conjunct φ_{con} is a formula in the AB-fragment of DHS_L capturing the traces which are concatenations of codes of flat configurations and satisfy the consecution requirement. The construction of φ_{con} is an easy task and we omit the details here. The conjunct φ_{inc} (resp., φ_{dec}) ensures the increment (resp., decrement) progression requirement. We focus on the formula φ_{inc} (the definition of φ_{dec} being similar) which requires that (i) the value associated to the first c-incrementation, if any, is 1, and (ii) if a c-incrementation $\mathcal{I}$ with value n_1 is followed by a c-incrementation with value n_2, then $n_2 > n_1$ and there is also a c-incrementation following $\mathcal{I}$ with value $n_1 + 1$. The first requirement can be easily expressed by an AB formula. The second requirement is captured by the following DHS_L formula.

$$\bigwedge_{c \in \{1,2\}} \bigwedge_{\delta \in \mathit{Inc}(c)} [\mathrm{A}]\,[\mathrm{A}]\Big((\mathit{left}(\delta) \wedge \mathit{right}(\#) \wedge \neg \mathit{Int}(\#)) \rightarrow \Big[\neg \bigvee_{\delta' \in \mathit{Inc}(c)} \langle \mathrm{L} \rangle_{\Delta \leq 0}(\mathit{left}(\delta') \wedge \mathit{right}(\#))$$
$$\wedge \big(\bigvee_{\delta' \in \mathit{Inc}(c)} \langle \mathrm{A} \rangle\, \mathit{right}(\delta') \rightarrow \bigvee_{\delta' \in \mathit{Inc}(c)} \langle \mathrm{L} \rangle_{\Delta = 0}(\mathit{left}(\delta') \wedge \neg \mathit{Int}(\#) \wedge \mathit{right\_next}(\#))\big)\Big]\Big)$$

The conjunct $\varphi_{\mathsf{if\_zero}}$ expresses the zero-test checking requirement. It ensures that for each counter c, (i) there is no c-incrementation $\mathcal{I}$ s.t. the first c-operation following $\mathcal{I}$ is a zero-test, and (ii) there is no c-incrementation followed by a c-decrementation $\mathcal{D}$ with a smaller value such that the first c-operation following $\mathcal{D}$ is a zero-test. The first requirement can be easily expressed by an AB formula. The second requirement is captured in DHS_L as follows.

$$\neg \bigvee_{c \in \{1,2\}} \bigvee_{\delta_i \in \mathit{Inc}(c)} \bigvee_{\delta_d \in \mathit{Dec}(c)} \bigvee_{\delta_0 \in \mathit{Zero}(c)} \langle \mathrm{A} \rangle\, \langle \mathrm{A} \rangle \Big((\mathit{left}(\delta_i) \wedge \mathit{right}(\#) \wedge \neg \mathit{Int}(\#)) \wedge$$
$$\langle \mathrm{L} \rangle_{\Delta < 0}[\mathit{left}(\delta_d) \wedge \mathit{right}(\#) \wedge \langle \mathrm{A} \rangle(\mathit{right}(\delta_0) \wedge \bigwedge_{\delta \in \mathit{Inc}(c) \cup \mathit{Dec}(c) \cup \mathit{Zero}(c)} \neg \mathit{Int}(\delta))]\Big)$$

Finally, the conjunct φ_{dom} characterizes the increment domination requirement. One can easily check that the following conditions capture increment domination.

- If there is some c-decrementation, then there is some c-incrementation.
- c-incrementations have values greater than previous c-decrementations.
- If a c-incrementation $\mathcal{I}$ with value n is not followed by other c-incrementations, then each c-decrementation following $\mathcal{I}$ has a value smaller or equal to n.

We focus on the third requirement which can be expressed in DHS_L as follows (the specification of the first and second requirements are simpler):

$$\bigwedge_{c\in\{1,2\}} \bigwedge_{\delta_i\in Inc(c)} [\mathrm{A}]\,[\mathrm{A}]\Big(\big[\mathit{left}(\delta_i) \wedge \mathit{right}(\#) \wedge \neg \mathit{Int}(\#) \wedge [\mathrm{A}] \bigwedge_{\delta\in Inc(c)} \neg \mathit{right}(\delta)\big] \longrightarrow$$
$$\neg\,\langle \mathrm{L}\rangle_{\Delta>0} \bigvee_{\delta_d\in Dec(c)} [\mathit{left}(\delta_d) \wedge \mathit{right}(\#) \wedge \neg \mathit{Int}(\#)]\Big)$$

Note that the unique constant used in the constraints of $\varphi_{L,M}$ is 0. By construction, the DHS_L formula $\varphi_{L,M}$ captures the traces encoding the well-formed M-sequences. Thus, by Lemma 3, $\varphi_{L,M}$ is satisfiable iff M has a recurrent computation.

5 Decidable fragments of DHS

In this section, we show that model checking and satisfiability of DHS_{simple} are decidable though 2EXPSPACE-hard. Moreover, by exploiting new results on the *linear-time hybrid logic* HL [16, 32, 2], we show that for the fragment of DHS_{simple} given by monotonic $\mathsf{D}_{simple}(\mathsf{AB\overline{B}})$, the considered problems are exactly EXPSPACE-complete. Note that DHS_{simple} represents the maximal fragment of DHS which is not covered by the undecidability results of Section 4, while $\mathsf{D}_{simple}(\mathsf{AB\overline{B}})$ corresponds to the extension of $\mathsf{AB\overline{B}}$ with the constrained versions of the modalities for the Allen's relations $\mathcal{R}_B$ and $\mathcal{R}_{\overline{B}}$. Additionally, we provide a characterization of HS in terms of a novel hybrid logic which lies between the one-variable and the two-variable fragment of HL. We establish that there are linear time translations from HS formulas into equivalent formulas of the novel logic, and vice versa. This result is of independent interest since while for the one-variable fragment of HL, model checking and satisfiability are EXPSPACE-complete [32, 2], for the two-variable fragments of HL, these problems are already non-elementarily decidable [32, 2].

Constrained HL. HL [16, 32, 2] extends standard LTL + past by first-order concepts. Here, we consider a constrained version of HL (CHL) where the temporal modalities are equipped with timing constraints. Formally, CHL formulas φ over $\mathcal{AP}$ and a set X of (position) variables are defined by the following syntax:

$$\varphi \stackrel{\text{def}}{=} \top \mid p \mid x \mid \neg\,\varphi \mid \varphi \wedge \varphi \mid \mathsf{F}_{\sim c}\varphi \mid \mathsf{P}_{\sim c}\varphi \mid {\downarrow}x.\varphi$$

where $p \in \mathcal{AP}$, $x \in X$, $\sim\in \{<, \leq, =, >, \geq\}$, $c \in \mathbb{Z}$, $\mathsf{F}_{\sim c}$ is the *constrained strict eventually* modality and $\mathsf{P}_{\sim c}$ is its past counterpart, and ${\downarrow}x$ is the *downarrow binder* operator which assigns the variable name x to the current position. A formula is *monotonic* if it does not use equality constraints $= c$. We also exploit the constrained modalities $\mathsf{G}_{\sim c}$ (*always*) and $\mathsf{H}_{\sim c}$ (*past always*) as abbreviations for $\neg\mathsf{F}_{\sim c}\neg\varphi$ and $\neg\mathsf{P}_{\sim c}\neg\varphi$, respectively. The standard strict eventually (resp., always) modality F (resp., G) corresponds to $\mathsf{F}_{>0}$ (resp., $\mathsf{G}_{>0}$), and its past counterpart P (resp., H) corresponds to $\mathsf{P}_{>0}$ (resp., $\mathsf{H}_{>0}$). The logic HL [16, 32, 2] corresponds to the CHL fragment using only the temporal modalities F and P. We denote by CHL_1 and CHL_2 (resp., HL_1 and HL_2) the one-variable and two-variable fragments of CHL (resp., HL). A CHL sentence is a formula where each variable x is not free (i.e., occurs in the scope of modality $\downarrow x$). The size $|\varphi|$ of a CHL formula φ is the number of distinct subformulas of φ multiplied the number of bits for encoding the maximal constant occurring in φ.

CHL formulas φ are interpreted over traces w. For a position $i \geq 0$ and a *valuation* g assigning to each variable a position, the satisfaction relation $(w, i, g) \models \varphi$ is defined as follows (we omit the semantics of propositions and Boolean connectives which is standard):

$$\begin{array}{ll}
(w,i,g) \models x & \Leftrightarrow i = g(x) \\
(w,i,g) \models \mathsf{F}_{\sim c}\,\varphi & \Leftrightarrow \text{there is } j > i \text{ such that } j - i \sim c \text{ and } (w,j,g) \models \varphi \\
(w,i,g) \models \mathsf{P}_{\sim c}\,\varphi & \Leftrightarrow \text{there is } j < i \text{ such that } i - j \sim c \text{ and } (w,j,g) \models \varphi \\
(w,i,g) \models \downarrow x.\varphi & \Leftrightarrow (w,i,g[x \mapsto i]) \models \varphi
\end{array}$$

where $g[x \mapsto i](x) = i$ and $g[x \mapsto i](y) = g(y)$ for $y \neq x$. We write $(w, i) \models \varphi$ to mean that $(w, i, g_0) \models \varphi$, where g_0 maps each variable to position 0, and $w \models \varphi$ to mean that $(w, 0) \models \varphi$.

From $\mathsf{D}_{simple}(\mathsf{AB\overline{B}})$ to CHL_1. We show that (monotonic) $\mathsf{D}_{simple}(\mathsf{AB\overline{B}})$ formulas can be translated in linear time into equivalent (monotonic) CHL_1 sentences. For a constraint $\sim c$, we write $(\sim c)^{-1}$ for $\sim' -c$, where $\sim'$ is the inverse of $\sim$. For example, $<$ is the inverse of $>$, while $\leq$ is the inverse of $\geq$.

▶ **Proposition 4.** *Given a (monotonic) $\mathsf{D}_{simple}(\mathsf{AB\overline{B}})$ formula φ, one can construct in linear-time an equivalent (monotonic) CHL_1 sentence.*

Proof. Fix a variable x. In the translation, x and the current position refer to the left endpoint and right endpoint of the current interval in $\mathbb{N}$, respectively. We can assume that the modalities for the Allen's relations $\mathcal{R}_B$ and $\mathcal{R}_{\overline{B}}$ occur only in a constrained form (for example, $\langle \mathrm{B} \rangle$ corresponds to $\langle \mathrm{B} \rangle_{<0}$). Formally, the translation $f : \mathsf{D}_{simple}(\mathsf{AB\overline{B}}) \mapsto \mathsf{CHL}_1$ is homomorphic w.r.t. the Boolean connectives (i.e., preserves the Boolean connectives) and is inductively defined as follows:

$$\begin{array}{rclrcl}
f(p) & \overset{\text{def}}{=} & p \wedge \neg\mathsf{P}(\neg p \wedge (x \vee \mathsf{P}x)) & f(\langle \mathrm{A} \rangle\,\varphi) & \overset{\text{def}}{=} & \downarrow x.\,(f(\varphi) \vee \mathsf{F}f(\varphi)) \\
f(\langle \mathrm{B} \rangle_{\sim c}\,\varphi) & \overset{\text{def}}{=} & \mathsf{P}_{(\sim c)^{-1}}(f(\varphi) \wedge (x \vee \mathsf{P}x)) & f(\langle \overline{\mathrm{B}} \rangle_{\sim c}\,\varphi) & \overset{\text{def}}{=} & \mathsf{F}_{\sim c} f(\varphi)
\end{array}$$

By a straightforward induction on φ, we obtain that given a trace w, an interval $[i, j]$, a valuation g such that $g(x) = i$, it holds that $[i, j] \models_w \varphi$ if and only if $(w, j, g) \models f(\varphi)$. The desired CHL_1 sentence φ' equivalent to φ is then defined as follows: $\varphi' \overset{\text{def}}{=} \downarrow x.\, f(\varphi)$. ◀

In Section 5.1, we show that model checking and satisfiability of monotonic CHL_1 are Expspace-complete. By [10], for the logic AB over traces, the considered problems are already Expspace-hard. Thus, by Proposition 4 we obtain the following result.

▶ **Theorem 5.** *MC and satisfiability of monotonic $\mathsf{D}_{simple}(\mathsf{AB\overline{B}})$ are* Expspace*-complete.*

Decidability of DHS_{simple}. We first introduce a variant of CHL, we call *swap CHL* (SCHL). SCHL formulas φ are defined as follows: $\varphi \overset{\text{def}}{=} \top \mid p \mid x \mid \neg\,\varphi \mid \varphi \wedge \varphi \mid \mathsf{F}_{\sim c}\,\varphi \mid \mathsf{P}_{\sim c}\,\varphi \mid \mathsf{swap}_x.\varphi$. The novel modality swap_x simultaneously assigns to x the value of the current position and updates the current position to the value previously referenced by x. Formally, its semantics is defined as follows: $(w, i, g) \models \mathsf{swap}_x.\varphi \Leftrightarrow (w, g(x), g[x \mapsto i]) \models \varphi$.

We are interested in the one-variable fragment SCHL_1 of SCHL, and in the unconstrained version SHL_1 of SCHL_1 where the unique temporal modalities are F and P. From a succinctness point of view, the fragment SCHL_1 lies between CHL_1 and CHL_2.

▶ **Proposition 6.** *Given a CHL$_1$ (resp., HL$_1$) sentence, one can construct in linear time an equivalent SCHL$_1$ (resp., SHL$_1$) sentence. Moreover, given a SCHL$_1$ (resp., SHL$_1$) sentence, one can construct in linear time an equivalent CHL$_2$ (resp., HL$_2$) sentence.*

Proof. The translation function $f : \mathsf{CHL}_1 \mapsto \mathsf{SCHL}_1$ from CHL$_1$ formulas to SCHL$_1$ formulas is homomorphic w.r.t. proposition, variables, Boolean connectives and temporal modalities. Moreover, for a CHL$_1$ formula φ using variable x, $f(\downarrow x. \varphi)$ is defined as $\mathsf{swap}_x. \mathsf{FP}(x \wedge f(\varphi))$.

For the second part of Proposition 6, let φ be a SCHL$_1$ formula using variable x, and let x_1 and x_2 be two distinct variables. For each $h = 1, 2$, we define a CHL$_2$ formula $F(\varphi, x_h)$ using only variables x_1 and x_2 and such that only x_h can occur free in $F(\varphi, x_h)$. The mapping F is homomorphic w.r.t. propositions, Boolean connectives and temporal modalities, and is defined as follows for variable x and the swap modality: $F(x, x_h) \overset{\text{def}}{=} x_h$ and $F(\mathsf{swap}_x. \varphi, x_h) \overset{\text{def}}{=} \downarrow x_{3-h}. \mathsf{FP}(x_h \wedge F(\varphi, x_{3-h}))$. By a straightforward induction on the structure of the SCHL$_1$ formula φ, given a trace w, $h = 1, 2$, two positions $i, j \geq 0$, a valuation g s.t. $g(x) = j$, and a valuation g' s.t. $g'(x_h) = j$, it holds that $(w, i, g) \models \varphi$ iff $(w, i, g') \models F(\varphi, x_h)$. Hence, if φ is a SCHL$_1$ sentence, then $\downarrow x_h. F(\varphi, x_h)$ is a CHL$_2$ sentence equivalent to φ. ◀

We show that DHS$_{simple}$ formulas can be converted in exponential time into equivalent SCHL$_1$ sentences. Moreover, the logic HS over traces exactly corresponds to SHL$_1$, i.e., there are linear-time translations from HS formulas into equivalent SHL$_1$ sentences, and vice versa.

▶ **Proposition 7.**

1. *Given a DHS$_{simple}$ formula φ, one can construct in singly exponential time an equivalent SCHL$_1$ sentence ψ. Moreover, if φ is a D(BE$\overline{\text{BE}}$) formula, then ψ can be constructed in linear time, and $\psi \in$ SHL$_1$ if $\varphi \in$ HS.*
2. *Given a SHL$_1$ sentence φ, one can construct in linear time an equivalent HS formula ψ.*

Proof. We focus on the proof of statement 1 in Proposition 7. A proof of statement 2 can be found in [12]. First note that DHS$_{simple}$ corresponds to D(BED$\overline{\text{BDE}}$) since the unconstrained modalities for the Allen's relations $\mathcal{R}_A$, $\mathcal{R}_L$, and $\mathcal{R}_O$ and their inverses can be expressed in linear time into BE$\overline{\text{BE}}$. Moreover, the constrained versions of the modalities $\langle \text{D} \rangle$ and $\langle \overline{\text{D}} \rangle$ can be easily expressed in D(BE$\overline{\text{BE}}$) though with a singly exponential blow-up. For example, for $n > 0$, $\langle \overline{\text{D}} \rangle_{\geq n} \varphi$ is equivalent to $\bigvee\limits_{n_1 \geq 0, n_2 \geq 0: n_1 + n_2 = n} \langle \overline{\text{B}} \rangle_{\geq n_1} \langle \overline{\text{E}} \rangle_{\geq n_2} \varphi$.

Thus, it suffices to show that a D(BE$\overline{\text{BE}}$) formula can be converted in linear time into an equivalent SCHL$_1$ sentence. Let x be a position variable. We use the expression $x < cur$ to indicate that the position referenced by variable x is smaller than the current position. The meaning of the expression $x > cur$ (resp., $x = cur$) is similar. Given a D(BE$\overline{\text{BE}}$) formula φ and $\tau \in \{x < cur, x > cur, x = cur\}$, we inductively define an SCHL$_1$ formula $f(\varphi, \tau)$ using variable x. Intuitively, the position referenced by x and the current position represent the endpoints of the interval on which φ is currently evaluated. We can assume that φ contains only constrained temporal modalities. Indeed, the modalities in BE$\overline{\text{BE}}$ can be trivially converted into equivalent constrained versions. The mapping f is homomorphic w.r.t. the Boolean connectives and is inductively defined as follows:

- $f(p, x < cur) \overset{\text{def}}{=} p \wedge \neg \mathsf{P}(\neg p \wedge (x \vee \mathsf{P}x))$.
- $f(\langle \text{B} \rangle_{\Delta \sim c} \varphi, x < cur) \overset{\text{def}}{=} \mathsf{P}_{(\sim c)^{-1}}[(f(\varphi, x = cur) \wedge x) \vee (f(\varphi, x < cur) \wedge \mathsf{P}x)]$.
- $f(\langle \overline{\text{B}} \rangle_{\Delta \sim c} \varphi, x < cur) \overset{\text{def}}{=} \mathsf{F}_{\sim c} f(\varphi, x < cur)$.
- $f(\langle \text{E} \rangle_{\Delta \sim c} \varphi, x < cur) \overset{\text{def}}{=} \mathsf{swap}_x. \mathsf{F}_{(\sim c)^{-1}}[(f(\varphi, x = cur) \wedge x) \vee (f(\varphi, x > cur) \wedge \mathsf{F}x)]$.

- $f(\langle\overline{\mathrm{E}}\rangle_{\Delta\sim c}\,\varphi, x < cur) \overset{\text{def}}{=} \mathsf{swap}_x.\, \mathsf{P}_{\sim c}\, f(\varphi, x > cur)$.
- $f(p, x > cur) \overset{\text{def}}{=} p \wedge \neg\mathsf{F}(\neg p \wedge (x \vee \mathsf{F}x))$.
- $f(\langle\mathrm{B}\rangle_{\Delta\sim c}\,\varphi, x > cur) \overset{\text{def}}{=} \mathsf{swap}_x.\, \mathsf{P}_{(\sim c)^{-1}}[(f(\varphi, x = cur) \wedge x) \vee (f(\varphi, x < cur) \wedge \mathsf{P}x)]$.
- $f(\langle\overline{\mathrm{B}}\rangle_{\Delta\sim c}\,\varphi, x > cur) \overset{\text{def}}{=} \mathsf{swap}_x.\, \mathsf{F}_{\sim c}\, f(\varphi, x < cur)$.
- $f(\langle\mathrm{E}\rangle_{\Delta\sim c}\,\varphi, x > cur) \overset{\text{def}}{=} \mathsf{F}_{(\sim c)^{-1}}[(f(\varphi, x = cur) \wedge x) \vee (f(\varphi, x > cur) \wedge \mathsf{F}x)]$.
- $f(\langle\overline{\mathrm{E}}\rangle_{\Delta\sim c}\,\varphi, x > cur) \overset{\text{def}}{=} \mathsf{P}_{\sim c}\, f(\varphi, x > cur)$.
- $f(p, x = cur) \overset{\text{def}}{=} p$.
- $f(\langle\mathrm{X}\rangle_{\Delta\sim c}\,\varphi, x = cur) \overset{\text{def}}{=} \neg\top$ for each $X \in \{B, E\}$.
- $f(\langle\overline{\mathrm{B}}\rangle_{\Delta\sim c}\,\varphi, x = cur) \overset{\text{def}}{=} \mathsf{F}_{\sim c}\, f(\varphi, x < cur)$.
- $f(\langle\overline{\mathrm{E}}\rangle_{\Delta\sim c}\,\varphi, x = cur) \overset{\text{def}}{=} \mathsf{P}_{\sim c}\, f(\varphi, x > cur)$.

By a straightforward induction on the structure of φ, we obtain that given a trace w, a position $j \geq 0$, and a valuation g such that $g(x) = j$, the following holds:

- $[j, j] \models_w \varphi$ iff $(w, j, g) \models f(\varphi, x = cur)$.
- For each $i > j$, $[j, i] \models_w \varphi$ iff $(w, i, g) \models f(\varphi, x < cur)$.
- For each $i < j$, $[i, j] \models_w \varphi$ iff $(w, i, g) \models f(\varphi, x > cur)$.

It follows that the SCHL_1 sentence $\mathsf{swap}_x.\, f(\varphi, x = cur)$ is equivalent to φ. Note that the previous sentence is in SHL_1 if $\varphi \in \mathsf{HS}$. ◀

By [32, 2], model checking and satisfiability of CHL_2 are decidable though with a non-elementary complexity. We can show that for the logic DHS_{simple}, the considered problems are at least 2EXPSPACE-hard even for the fragment given by monotonic $\mathsf{D}_{simple}(\mathsf{ABE})$ (for a proof, see [12]). Moreover, note that CHL formulas can be trivially translated into equivalent formulas of first-order logic FO over traces. Thus, by the first-order expressiveness completeness of the fragment AB of HS [6] (under the considered trace-based semantics), and Propositions 6–7, we obtain the following result.

▶ **Theorem 8.** *Model checking and satisfiability of DHS_{simple} are decidable and 2EXPSPACE-hard even for the fragment given by monotonic $\mathsf{D}_{simple}(\mathsf{ABE})$. Moreover, DHS_{simple}, monotonic $\mathsf{D}_{simple}(\mathsf{AB\overline{B}})$, and HS have the same expressiveness.*

5.1 EXPSPACE-completeness of monotonic CHL_1

In this section, we describe an asymptotically optimal automata-theoretic approach to solve satisfiability and model checking of monotonic CHL_1 (MCHL_1 for short), which is based on a direct translation of MCHL_1 sentences into *two-way alternating finite-state word automata* (2AWA) equipped with standard generalized Büchi acceptance conditions.

A MCHL_1 formula is in *monotonic normal form* (*MNF*) if negation is applied only to atomic propositions and variables, and the constrained temporal modalities are of the form $\mathsf{O}_{\leq c}$ with $c \geq 1$ and $\mathsf{O} \in \{\mathsf{F}, \mathsf{G}, \mathsf{P}, \mathsf{H}\}$. A MCHL_1 formula φ can be easily converted in linear-time into an equivalent MCHL_1 formula in *MNF* φ_M, called the *MNF* of φ (for details, see [12]). The *dual* $\widetilde{\varphi_M}$ *of* φ_M is the *MNF* of $\neg\varphi_M$.

Characterization of the satisfaction relation. We fix a monotonic CHL_1 formula φ with variable x, where x may occur free. W.l.o.g. we assume that φ is in *MNF*, and $\mathcal{AP}$ is the set of atomic propositions occurring in φ. First, we give an operational characterization of the satisfaction relation $w \models \varphi$ which non-trivially generalizes the classical notion of Hintikka-sequence of LTL. Essentially, for each trace w and valuation g of variable x, we associate to w and g infinite sequences $\rho = A_0, A_1, \ldots$ of sets, where for each $i \geq 0$, A_i is an *atom* and intuitively describes a maximal set of subformulas of φ which can hold at position i along w w.r.t. the valuation g. As for LTL, the notion of atom syntactically captures the semantics of Boolean connectives. The fixpoint characterization of the unconstrained temporal modalities and the semantics of the constrained temporal modalities are locally captured by requiring that consecutive pairs A_i, A_{i+1} along the sequence ρ satisfy certain syntactical constraints. Finally, the sequence ρ has to satisfy additional non-local conditions reflecting the liveness requirements ψ in the eventually subformulas $\mathsf{F}\psi$ of φ, and the semantics of the binder modality $\downarrow x$. Now, we give the technical details.

A formula ψ is a *first-level subformula* of φ if there is an occurrence of ψ in φ which is not in the scope of modality $\downarrow x$. The *closure* $cl(\varphi)$ of φ is the smallest set containing (i) x, $\top$, the propositions in $\mathcal{AP}$, formula $\mathsf{P}_{\leq 1}\top$, and (ii) all the first-level subformulas ψ of φ together with $\mathsf{F}_{\leq 1}\psi$ and $\mathsf{P}_{\leq 1}\psi$, and (iii) the duals of the formulas in the points (i) and (ii). Note that $\varphi \in cl(\varphi)$ and $|cl(\varphi)| = O(|\varphi|)$. Moreover, the set $obl(\varphi)$ of φ-*obligations* is the set of pairs of the form $(\mathsf{O}_{\leq c}\psi, d)$ such that $\mathsf{O} \in \{\mathsf{F}, \mathsf{P}, \mathsf{G}, \mathsf{H}\}$, $\mathsf{O}_{\leq c}\psi \in cl(\varphi)$, $c > 1$, and $1 \leq d \leq c - 1$. Intuitively, the obligations are exploited for capturing in a succinct way the semantics of the constrained temporal modalities. In particular, an obligation of the form $(\mathsf{F}_{\leq c}\psi, d)$ asserted at a position i means that there is $j > i$ such that ψ holds at position j, and $i + d$ *is the smallest of such* j. Note that two distinct obligations associated to the same formula $\mathsf{F}_{\leq c}\psi$ cannot hold simultaneously at the same position. Dually, an obligation of the form $(\mathsf{G}_{\leq c}\psi, d)$ asserted at a position i means that there is $j > i$ such that ψ holds at all positions in $[i+1, j]$, and $i + d$ *is the greatest of such* j. The meaning of the obligations associated to the past constrained modalities is similar. Evidently, $|obl(\varphi)| = 2^{O(|\varphi|)}$. A φ-*atom* A is a subset of $cl(\varphi) \cup obl(\varphi)$ such that $\top \in A$ and the following holds, where $c > 1$ and $\mathsf{O} \in \{\mathsf{F}, \mathsf{P}, \mathsf{G}, \mathsf{H}\}$:

- for each $\psi \in cl(\varphi)$, $\psi \in A$ iff $\widetilde{\psi} \notin A$;
- for each $\psi_1 \wedge \psi_2 \in cl(\varphi)$, $\psi_1 \wedge \psi_2 \in A$ iff $\{\psi_1, \psi_2\} \subseteq A$;
- for each $\psi_1 \vee \psi_2 \in cl(\varphi)$, $\psi_1 \vee \psi_2 \in A$ iff $\{\psi_1, \psi_2\} \cap A \neq \emptyset$;
- for each $\mathsf{O}_{\leq c}\psi \in cl(\varphi)$, *there is at most one obligation* of the form $(\mathsf{O}_{\leq c}\psi, d)$ in A.

It is worth noting that the set $\mathsf{Atoms}(\varphi)$ of φ-atoms has a cardinality which is at most singly exponential in $|\varphi|$, i.e. $|\mathsf{Atoms}(\varphi)| = 2^{O(|\varphi|)}$. A φ-atom A is *initial* if A does not contain formulas of the form $\mathsf{P}\psi$ or $\mathsf{P}_{\leq c}\psi$ and obligations of the form $(\mathsf{O}_{\leq c}\psi, d)$ with $\mathsf{O} \in \{\mathsf{P}, \mathsf{H}\}$.

We now define the function Succ_φ which maps each atom $A \in \mathsf{Atoms}(\varphi)$ to a subset of $\mathsf{Atoms}(\varphi)$. Intuitively, if A is the atom associated with a position i of the given trace w, then $\mathsf{Succ}_\varphi(A)$ contains the set of atoms associable to the next position $i + 1$ (w.r.t. a given valuation of variable x). Formally, $A' \in \mathsf{Succ}_\varphi(A)$ iff A' is not initial and the following holds:

- *F-requirements*: for all $\mathsf{F}\psi \in cl(\varphi)$, $\mathsf{F}\psi \in A \Leftrightarrow \{\mathsf{F}\psi, \psi\} \cap A' \neq \emptyset$.
- *P-requirements*: for all $\mathsf{P}\psi \in cl(\varphi)$, $\mathsf{P}\psi \in A' \Leftrightarrow \{\mathsf{P}\psi, \psi\} \cap A \neq \emptyset$.
- *G-Requirements*: for all $\mathsf{G}\psi \in cl(\varphi)$, $\mathsf{G}\psi \in A \Leftrightarrow \{\mathsf{G}\psi, \psi\} \subseteq A'$.
- *H-requirements*: for all $\mathsf{H}\psi \in cl(\varphi)$, $\mathsf{H}\psi \in A' \Leftrightarrow \{\mathsf{H}\psi, \psi\} \subseteq A$.
- *$\mathsf{F}_{\leq c}$-requirements*: for all $\mathsf{F}_{\leq c}\psi \in cl(\varphi)$,
  - $\mathsf{F}_{\leq c}\psi \in A \Leftrightarrow$ *either* $\psi \in A'$ *or* $c > 1$ and $(\mathsf{F}_{\leq c}\psi, d) \in A'$ for some $1 \leq d < c$;
  - for each $1 \leq d < c$, $(\mathsf{F}_{\leq c}\psi, d) \in A \Leftrightarrow$ *either* $d = 1$ and $\psi \in A'$, *or* $d > 1$, $\psi \notin A'$, and $(\mathsf{F}_{\leq c}\psi, d - 1) \in A'$.

- *$P_{\leq c}$-requirements*: for all $\mathsf{P}_{\leq c}\psi \in cl(\varphi)$,
  - $\mathsf{P}_{\leq c}\psi \in A' \Leftrightarrow$ *either* $\psi \in A$ *or* $c > 1$ and $(\mathsf{P}_{\leq c}\psi, d) \in A$ for some $1 \leq d < c$;
  - for each $1 \leq d < c$, $(\mathsf{P}_{\leq c}\psi, d) \in A' \Leftrightarrow$ *either* $d = 1$ and $\psi \in A$, *or* $d > 1$, $\psi \notin A$, and $(\mathsf{P}_{\leq c}\psi, d-1) \in A$.
- *$G_{\leq c}$-requirements*: for all $\mathsf{G}_{\leq c}\psi \in cl(\varphi)$,
  - $\mathsf{G}_{\leq c}\psi \in A \Leftrightarrow \psi \in A'$ and, in case $c > 1$, either $\mathsf{G}_{\leq c}\psi \in A'$ or $(\mathsf{G}_{\leq c}\psi, c-1) \in A'$;
  - for each $1 \leq d < c$, $(\mathsf{G}_{\leq c}\psi, d) \in A \Leftrightarrow$ *either* $d = 1$, $\psi \in A'$, and $\mathsf{F}_{\leq 1}\psi \notin A'$, *or* $d > 1$, $\psi \in A'$, and $(\mathsf{G}_{\leq c}\psi, d-1) \in A'$.
- *$H_{\leq c}$-requirements*: for all $\mathsf{H}_{\leq c}\psi \in cl(\varphi)$,
  - $\mathsf{H}_{\leq c}\psi \in A' \Leftrightarrow \psi \in A$ and, in case $c > 1$, either $\mathsf{H}_{\leq c}\psi \in A$ or $(\mathsf{H}_{\leq c}\psi, c-1) \in A$;
  - for each $1 \leq d < c$, $(\mathsf{H}_{\leq c}\psi, d) \in A' \Leftrightarrow$ *either* $d = 1$, $\psi \in A$, and $\mathsf{P}_{\leq 1}\psi \notin A$, *or* $d > 1$, $\psi \in A$, and $(\mathsf{H}_{\leq c}\psi, d-1) \in A$.

Note that Succ_φ captures the semantics of the constrained modalities in accordance to the intended meaning of the associated obligations. Let w be a trace and $\ell \geq 0$. A *φ-sequence over the pointed trace* (w, ℓ) is an infinite sequence $\rho = A_0, A_1, \ldots$ of φ-atoms such that:

- A_0 is initial, $x \in A_\ell$, and $x \notin A_i$ for each $i \neq \ell$;
- for each $i \geq 0$, $A_i \cap \mathcal{AP} = w(i)$ (*propositional consistency*), and $A_{i+1} \in \mathsf{Succ}_\varphi(A_i)$;
- *Fairness*: for each $\mathsf{F}\psi \in cl(\varphi)$ and for infinitely many $i \geq 0$, either $\psi \in A_i$ or $\mathsf{F}\psi \notin A_i$.

The standard fairness requirement ensures that the requirements ψ in the first-level subformulas $\mathsf{F}\psi$ of φ are eventually satisfied. In order to capture the semantics of the modality $\downarrow x$, we now give the notion of *fulfilling φ-sequence* by induction on the nesting depth of $\downarrow x$. Formally, a φ-sequence $\rho = A_0, A_1, \ldots$ over (w, ℓ) is *fulfilling* if for all $i \geq 0$ and $\downarrow x.\, \psi \in A_i$, there is a fulfilling ψ-sequence $\rho' = A'_0, A'_1, \ldots$ over the pointed trace (w, i) such that $\psi \in A'_i$.

The notion of fulfilling φ-sequence over a pointed trace (w, ℓ) provides a characterization of the satisfaction relation $(w, i, g) \models \varphi$ with $g(x) = \ell$ (a proof is in [12]).

▶ **Theorem 9.** *Let ϕ be a MCHL$_1$ sentence in MNF. Then, $w \in \mathcal{L}(\phi)$ if and only if there exists a fulfilling ϕ-sequence $\rho = A_0, A_1, \ldots$ over $(w, 0)$ such that $\phi \in A_0$.*

Automata-theoretic approach for MCHL$_1$. By Theorem 9, given a MCHL$_1$ sentence φ in *MNF*, it is not a difficult task to construct in singly exponential time a generalized Büchi 2AWA $\mathcal{A}_\varphi$ accepting $\mathcal{L}(\varphi)$. Given an input trace w, $\mathcal{A}_\varphi$ guesses a φ-sequence $\rho = A_0, A_1, \ldots$ over $(w, 0)$ by simulating it in forward mode along the "main" path of the run-tree. At the i^{th}-node of such a path, $\mathcal{A}_\varphi$ keeps track in its state of the φ-atom A_i. Moreover, in order to check that ρ is fulfilling, for each binder formula $\downarrow x.\, \psi \in A_i$, $\mathcal{A}_\varphi$ recursively checks the existence of a fulfilling ψ-sequence $\rho' = A'_0, A'_1, \ldots$ over (w, i) by guessing the ψ-atom A'_i, with $\{x, \psi\} \subseteq A'_i$, and by activating two secondaries copies: the first one moves in backward mode by guessing the finite sequence $A'_{i-1}, \ldots, A'_0$, and the second one moves in forward mode by guessing the infinite sequence $A'_{i+1}, A'_{i+2}, \ldots$. Details about the construction of $\mathcal{A}_\varphi$ can be found in [12]. By [33, 21], generalized Büchi 2AWA can be converted on the fly and in singly exponential time into equivalent Büchi nondeterministic finite-state automata (Büchi NWA). Recall that non-emptiness of Büchi NWA is NLOGSPACE-complete, and the standard model checking algorithm consists in checking emptiness of the Büchi NWA given by the synchronous product of the given Kripke structure with the Büchi NWA associated with the negation of the given formula. Thus, we obtain algorithms for satisfiability and model-checking of monotonic CHL$_1$ which run in non-deterministic single exponential space. Therefore, since EXPSPACE = NEXPSPACE, and for the logic HL$_1$, the considered problems are already EXPSPACE-hard [32], we obtain the following result.

▶ **Corollary 10.** *Model checking and satisfiability of monotonic CHL$_1$ are* EXPSPACE*-complete.*

6 Conclusion

We have investigated decidability and complexity issues for satisfiability and model checking of a quantitative extension of HS, namely DHS, under the trace-based semantics. The novel logic provides constrained versions of the HS temporal modalities which can express bounds on the difference between the durations of the current interval and the interval selected by the modality. A different and natural choice would have been to consider constraints on the sum of the durations. In this setting, one can show that the logic HS extended with sum constraints is decidable under the trace-based semantics by means of an exponential-time translation of formulas into equivalent HL_2 sentences.

References

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

2 L. Bozzelli and R. Lanotte. Complexity and succinctness issues for linear-time hybrid logics. *Theor. Comput. Sci.*, 411(2):454–469, 2010. `doi:10.1016/j.tcs.2009.08.009`.

3 L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. Model checking interval temporal logics with regular expressions. *Information and Computation*, 272:104498, 2020. `doi:10.1016/j.ic.2019.104498`.

4 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval Temporal Logic Model Checking: the Border Between Good and Bad HS Fragments. In *Proc. 8th IJCAR*, LNAI 9706, pages 389–405. Springer, 2016. `doi:10.1007/978-3-319-40229-1_27`.

5 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Model checking for fragments of the interval temporal logic HS at the low levels of the polynomial time hierarchy. *Information and Computation*, 262(Part):241–264, 2018. `doi:10.1016/j.ic.2018.09.006`.

6 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval vs. Point Temporal Logic Model Checking: An Expressiveness Comparison. *ACM Trans. Comput. Log.*, 20(1):4:1–4:31, 2019. `doi:10.1145/3281028`.

7 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Which fragments of the interval temporal logic HS are tractable in model checking? *Theor. Comput. Sci.*, 764:125–144, 2019. `doi:10.1016/j.tcs.2018.04.011`.

8 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Satisfiability and model checking for the logic of sub-intervals under the homogeneity assumption. *Log. Methods Comput. Sci.*, 18(1), 2022. `doi:10.46298/lmcs-18(1:24)2022`.

9 L. Bozzelli, A. Montanari, and A. Peron. Complexity analysis of a unifying algorithm for model checking interval temporal logic. *Inf. Comput.*, 280:104640, 2021. `doi:10.1016/j.ic.2020.104640`.

10 L. Bozzelli, A. Montanari, A. Peron, and P. Sala. Adding the Relation Meets to the Temporal Logic of Prefixes and Infixes makes it EXPSPACE-Complete. In *Proc. 12th GandALF*, EPTCS 346, pages 179–194, 2021. `doi:10.4204/EPTCS.346.12`.

11 L. Bozzelli, A. Montanari, A. Peron, and P. Sala. Pspace-Completeness of the Temporal Logic of Sub-Intervals and Suffixes. In *Proc. 28th TIME*, LIPIcs 206, pages 9:1–9:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.TIME.2021.9`.

12 L. Bozzelli and A. Peron. A quantitative extension of interval temporal logic over infinite words. *CoRR*, abs/2206.13920, 2022. `arXiv:2206.13920`.

13 D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. The dark side of interval temporal logic: marking the undecidability border. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):41–83, 2014. `doi:10.1007/s10472-013-9376-4`.

14 S. Demri, R. Lazic, and A. Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci.*, 411(22-24):2298–2316, 2010. `doi:10.1016/j.tcs.2010.02.021`.

15 E. A. Emerson and J. Y. Halpern. “Sometimes” and “not never” revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986. doi:10.1145/4904.4999.

16 M. Franceschet, M. de Rijke, and B.H. Schlingloff. Hybrid Logics on Linear Structures: Expressivity and Complexity. In *Proc. 10th TIME-ICTL*, pages 166–173. IEEE Computer Society, 2003. doi:10.1109/TIME.2003.1214893.

17 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.

18 D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *J. ACM*, 33(1):224–248, 1986. doi:10.1145/4904.4993.

19 J.A.W. Kamp. *Tense logic and the theory of linear order*. University of California, Los Angeles, 1968.

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

21 O. Kupferman, N. Piterman, and M.Y. Vardi. Extended temporal logic revisited. In *Proc. 12th CONCUR*, LNCS 2154, pages 519–535. Springer, 2001. doi:10.1007/3-540-44685-0_35.

22 K. Lodaya. Sharpening the undecidability of interval temporal logic. In *Proc. 6th ASIAN*, LNCS 1961, pages 290–298. Springer, 2000. doi:10.1007/3-540-44464-5_21.

23 A. Lomuscio and J. Michaliszyn. An epistemic Halpern-Shoham logic. In *Proc. 23rd IJCAI*, pages 1010–1016. IJCAI/AAAI, 2013.

24 A. Lomuscio and J. Michaliszyn. Decidability of model checking multi-agent systems against a class of EHS specifications. In *Proc. 21st ECAI*, pages 543–548. IOS Press, 2014. doi:10.3233/978-1-61499-419-0-543.

25 A. Lomuscio and J. Michaliszyn. Model checking multi-agent systems against epistemic HS specifications with regular expressions. In *Proc. 15th KR*, pages 298–308. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12823.

26 J. Marcinkowski and J. Michaliszyn. The undecidability of the logic of subintervals. *Fundamenta Informaticae*, 131(2):217–240, 2014. doi:10.3233/FI-2014-1011.

27 A. Molinari, A. Montanari, A. Murano, G. Perelli, and A. Peron. Checking interval properties of computations. *Acta Informatica*, 53(6-8):587–619, 2016. doi:10.1007/s00236-015-0250-1.

28 A. Molinari, A. Montanari, and A. Peron. Model checking for fragments of Halpern and Shoham’s interval temporal logic based on track representatives. *Information and Computation*, 259(3):412–443, 2018. doi:10.1016/j.ic.2017.08.011.

29 B. Moszkowski. *Reasoning About Digital Circuits*. PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, 1983.

30 A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

31 P. Roeper. Intervals and tenses. *Journal of Philosophical Logic*, 9:451–469, 1980.

32 T. Schwentick and V. Weber. Bounded-Variable Fragments of Hybrid Logics. In *Proc. 24th STACS*, volume 4393 of *LNCS 4393*, pages 561–572. Springer, 2007. doi:10.1007/978-3-540-70918-3_48.

33 M.Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th ICALP*, LNCS 1443, pages 628–641. Springer, 1998. doi:10.1007/BFb0055090.

34 Y. Venema. Expressiveness and Completeness of an Interval Tense Logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990. doi:10.1305/ndjfl/1093635589.

A Neuro-Symbolic Approach for Real-World Event Recognition from Weak Supervision

Gianluca Apriceno
University of Trento, Italy
Fondazione Bruno Kessler, Trento, Italy

Andrea Passerini
University of Trento, Italy

Luciano Serafini
Fondazione Bruno Kessler, Trento, Italy

Abstract

Events are structured entities involving different components (e.g, the participants, their roles etc.) and their relations. Structured events are typically defined in terms of (a subset of) simpler, atomic events and a set of temporal relation between them. Temporal Event Detection (TED) is the task of detecting structured and atomic events within data streams, most often text or video sequences, and has numerous applications, from video surveillance to sports analytics. Existing deep learning approaches solve TED task by implicitly learning the temporal correlations among events from data. As consequence, these approaches often fail in ensuring a consistent prediction in terms of the relationship between structured and atomic events. On the other hand, neuro-symbolic approaches have shown their capability to constrain the output of the neural networks to be consistent with respect to the background knowledge of the domain. In this paper, we propose a neuro-symbolic approach for TED in a real world scenario involving sports activities. We show how by incorporating simple knowledge involving the relative order of atomic events and constraints on their duration, the approach substantially outperforms a fully neural solution in terms of recognition accuracy, when little or even no supervision is available on the atomic events.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning; Computing methodologies → Activity recognition and understanding

Keywords and phrases structured events, temporal event detection, neuro-symbolic integration

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.12

Supplementary Material *Software (Source Code)*: `https://github.com/Gianlu94/A-neuro-symbolic-approach-for-real-world-event-recognition-from-weak-supervision`

Funding This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

1 Introduction

Events are structured entities that involve multiple components, like the participants, their role, the type and the atomic events composing it. For example, in athletics, the event *high jump* involves one person (the athlete), performing the atomic events *run*, *jump* and *fall* in sequence. One of the main challenging tasks is temporal event detection (TED) that consists in detecting events within data stream, like text and video. Continuing the example, it consists in identifying the class of the atomic events and the interval of time where they occurred. Many sub-symbolic approaches, mostly based on neural networks, have been proposed for event recognition [1, 25]. One of the main drawbacks of these kind of approaches is the amount of training data. Indeed, having a large training set is fundamental for an appropriate and effective training of the model. Furthermore, "rich" annotations at different levels are required in order to solve the task (e.g., frame-by-frame annotation of atomic events). Large amounts of deeply annotated data are hard to collect. Additionally, errors in the annotations (e.g. in case of crowdsourced ones) may introduce noise in the model and compromise its accuracy. More importantly, purely neural approaches cannot guarantee consistency of the predictions with the domain knowledge, in terms of the relationship between the structured event and the atomic events that compose it. Neuro-symbolic approaches [11] have recently gained increasing popularity as a means to make the best of both worlds, by combining the effectiveness in low-level processing of deep learning technology with the ability of symbolic approaches to express complex domain knowledge. Popular frameworks including DeepProbLog [16], DeepStochLog [24], Logic Tensor Networks [6], LYRICS [17] and NeurASP [28] have been proposed and applied to solve different structured tasks, like Semantic Image Interpretation [8]. In the context of event recognition, the DeepProbLog framework has proved effective in recognizing both structured and simple events as well as generalizing to unseen outcomes [3, 23]. However, these results have been obtained on artificial scenarios, and the framework has serious issues of scalability when the complexity of the setting increases [9].

In this paper, we present a neuro-symbolic approach for structured event recognition in sport videos. The task is out of reach of popular neuro-symbolic frameworks like DeepProbLog, because of the computational complexity given by number of frames in a video combined with the temporal constraints of the background knowledge. We tackle the problem by combining neural predictions on individual frames with a mixed integer linear programming formulation enforcing satisfaction of (soft) temporal constraints from the background knowledge and similarity with the neural outputs. The approach is fully differentiable and end-to-end trainable.

Our experimental evaluation shows how the neuro-symbolic approach provides substantially more accurate predictions with respect to a fully neural solution, with the additional feature of guaranteeing that predictions satisfy existing hard constraints. Quite remarkably, the approach is capable of predicting the sequence of atomic events that constitute a structured event even without having any supervision on them, by simply leveraging background knowledge in terms of duration constraints to guide the learning of the underlying neural network.

The rest of the paper is structured as follows: Section 2 briefly reviews the state of the art approaches that have been proposed for event recognition; Section 3 formally defines the problem; Section 4 describes our proposed approach; Section 5 presents the experimental setting; Section 6 reports the experimental results. Finally, Section 7 draws some concluding remarks and discusses directions for future work.

2 State of the art

Event recognition has always attracted researchers coming from different fields, like Computer Vision and NLP. The particular attention towards event recognition may be motivated by its multiple data stream nature and by its impact in people's daily life. Looking at the literature, approaches to event recognition can be classified into three categories: sub-symbolic, symbolic and neuro-symbolic. Sub-symbolic approaches (mostly based on neural networks) moved from manually crafted features to automatic features learning (see [1] and [25] for a survey). These approaches require a huge amount of training data with a rich annotation at different levels (e.g, the type and temporal location of the event). These data are hard to collect and it is difficult to ensure high-quality annotations for large amounts of example, so that high levels of annotation errors can affect the accuracy of the networks being trained. Furthermore, the trained model is a black box model that cannot explain its decisions and is not guaranteed to be consistent with existing background knowledge. An attempt to make sub-symbolic approaches more interpretable is represented by Concept Bottleneck Models [15] where the activation of (a subset of) human-specified concepts is used to explain the model's final decision. However, works like [15] focus on atemporal domains (e.g. image). In [12], authors propose a novel approach that addresses concepts explanation in videos, but they are not able to capture spatial and temporal relationships between concepts. On the other hand, symbolic approaches like [5], are explainable and knowledge-consistent, but are not robust in the presence of noise. Therefore, symbolic approaches dealing with uncertainty have been proposed [2]. In [20, 4], authors recognize higher events by combining evidence of simple events with domain knowledge using the probabilistic logic programming framework ProbLog [18] . In this case, knowledge on low level events is assumed to be given. Recently, neuro-symbolic approaches have started to be applied in the context of event recognition. In works like [13, 14, 27, 22, 10], pre-trained neural networks are used to extract lower events and then passed to a symbolic layer that encodes the knowledge of the domain in the form of logic rules. In [26], authors propose an end-to-end model where a neural network is also used to learn to simulate the symbolic layer. One of the drawbacks is that the neural network has to be re-trained in case of even minimal changes/updates of the existing knowledge. The DeepProbLog neuro-symbolic framework [16] has been employed in a couple of recent studies [3, 23] to perform structured event recognition in videos and audio streams respectively. By using DeepProbLog, any change/update to knowledge can be easily integrated. Both [3] and [23] perform event recognition in artificial scenarios. This may be motivated by the scalability issue of DeepProbLog, that is particularly acute when considering tasks involving time. Indeed, the authors of the NESTER framework [9] showed how an optimization modulo theory [19] reasoning layer refining neural network predictions is substantially more efficient than a solution based on DeepProbLog on a toy handwritten equation recognition task. The approach however assumes complete supervision on the neural network outputs at training time, and does not address temporal event detection. Our solution adapts this idea to address structured and atomic event recognition on real-world video streams. In this case, the reasoning layer is also used to provide supervision when limited/no labels for the events are available.

3 Problem Definition

Our problem can be summarized as follows: Given a data sequence $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{l}$ of real-value tensors $\boldsymbol{x}_i$ and some background knowledge $\mathcal{K}$ about the relationship between structured and atomic events, we are interested in providing a description of the atomic and structured events that are happening during the sequence. Let us now specify all the details of the problem. To represent background knowledge about structured and atomic events we use a variation of the event calculus based on First Order Logic. Let $\mathcal{L}$ be a first order language that contains a set of constants $\mathcal{E}$ for event types, which is partitioned in two disjoint sets of constants $\mathcal{A}$ and $\mathcal{S}$ for atomic and structured events, respectively. $\mathcal{L}$ contains also the set of constants $\mathbb{N}$ of natural numbers which are used to denote time points. The set of predicates of $\mathcal{L}$ includes the equality predicate, the order relation defined as usual on time points, and the ternary predicate $\mathit{happens}(e, t_1, t_2)$, where e is a term for an event type and t_1, and t_2 are terms for time points. The atom $\mathit{happens}(e, t_1, t_2)$ represents the proposition that an event of type e starts at time t_1 and terminates at time t_2 (not included). In addition to the usual axioms for the equality and order relation we have the axiom

$$\forall xyz(\mathit{happens}(x, y, z) \rightarrow y < z)$$

As far as the semantics of $\mathcal{L}$, we consider the set of Herbrand Interpretations of $\mathcal{L}$, i.e., all the subsets of the Herbrand Base $\mathcal{H}$ defined as

$$\mathcal{H} = \{happens(e, t_1, t_2) \mid e \in \mathcal{E}, t_1 < t_2 \in \mathbb{N}\}$$

Now we are ready to provide a more precise formulation of our problem: Given a knowledge base $\mathcal{K}$ in the language $\mathcal{L}$ that expresses general knowledge about the event types in $\mathcal{E}$, and a data sequence $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{l}$, we have to find an herbrand interpretation $\mathcal{I}$, such that $\mathcal{I} \models \mathcal{K}$, and such that $happens(e, t_1, t_2) \in \mathcal{I}$ if and only if the data sub-sequence $\boldsymbol{X}_{t_1:t_2} = \{\boldsymbol{x}_i\}_{i=t_1}^{t_2-1}$ shows that an event of type e is happening. Intuitively, $\mathcal{I}$ describes the type and the class of the events happening in $\boldsymbol{X}$ and when they happened.

▶ **Example 1.** Let $\boldsymbol{X}$ be a video where a person is doing a high jump (structured event) in the interval $[1, 31]$. The background knowledge contains the fact that a high jump can be decomposed into a sequence of three atomic events: run, jump and fall, which can be expressed by the following formula:

$$\begin{aligned}\forall b_{hj} e_{ij} (happens(highjump, b_{hj}, e_{hj}) \leftrightarrow \exists\, b_r, e_r, b_j, e_j, b_f, e_f(&\\ happens(run, b_r, e_r) \wedge happens(jump, b_j, e_j) \wedge happens(fall, b_f, e_f) \wedge&\\ b_r = b_{hj} \wedge e_r = b_j \wedge e_j = b_f \wedge e_f = e_{hj}))&\end{aligned}$$

Two examples of interpretations that satisfy the above constraints are:

$$\begin{aligned}\mathcal{I}_1 &= \{happens(highjump, 1, 31), happens(run, 1, 21), happens(jump, 21, 25),\\ &\quad happens(fall, 25, 31)\}\\ \mathcal{I}_2 &= \{happens(highjump, 1, 31), happens(run, 1, 23), happens(jump, 23, 28),\\ &\quad happens(fall, 28, 31)\}\\ &\quad\vdots\end{aligned}$$

Since we may have more than one interpretation that satisfy $\mathcal{K}$, we define a cost function $c : I \to R$ and select the interpretation I_c^* with the minimum cost:

$$I_c^* = \underset{I_c \models \mathcal{K}}{\operatorname{argmin}}\, c(I_c)$$

In order to find I_c^*, we define a neuro-symbolic approach that combines low-level neural processing with high level reasoning in terms of background knowledge on the events.
The kind of supervision we provide to train a neuro-symbolic model in order to find I_c^* is:

$$\left\{\boldsymbol{X}^{(i)}, G_a^{(i)}\right\}_{i=1}^{n}$$

where $G_a^{(i)}$ is a set of ground atoms which are true in $\boldsymbol{X}^{(i)}$. Supervision is always assumed to be partial, including the case in which supervision is limited to structured events, and atomic events need to be learned in a fully unsupervised way. See experiments for the details.

4 Proposed Approach

Our objective consists in finding an interpretation I that has to explain what happened in $\boldsymbol{X}$ both in terms of structured and atomic events. In order to achieve it, we have to recognize the classes of structured and atomic events happening in $\boldsymbol{X}$ and the (interval of)

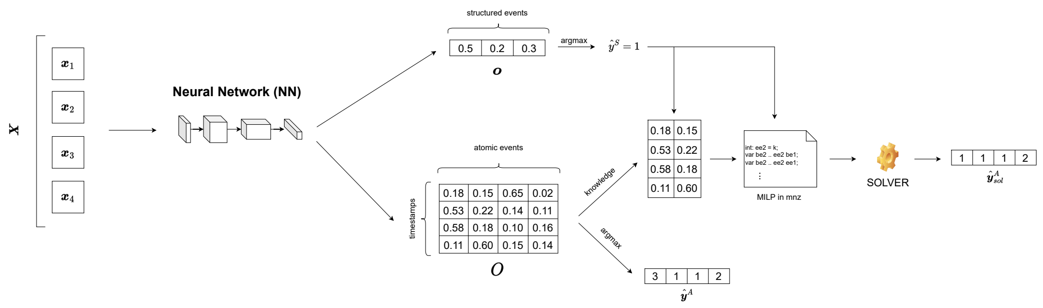


Figure 1 Inference steps of our neuro-symbolic approach on a data sequence of length 4, with 3 structured events and 4 atomic events. The structured event of class 1, which is the one predicted by the NN, is defined in terms of the sequence of atomic events 1 and 2, respectively.

time where they occurred. To achieve this objective, we use an end-to-end differentiable neuro-symbolic approach that combines low level processing of a neural network with a logic layer that leverages knowledge about structured and atomic events. An overall overview of our neuro-symbolic approach is depicted in Figure 1. As can be seen by looking at the figure, the first step consists in passing $\boldsymbol{X}$ to a neural network NN. NN may be any kind of network (e.g., Convolutional, RNN and LSTM) and has two different heads, one for structured and one for atomic events. The head for the structured events returns as output a vector $\boldsymbol{o}$ where $o_i \in [0,1]$, with $i = 1,\ldots,k$ (assuming k is the number of structured events), is the probability of the $i-th$ structured event. On the other hand, the head for the atomic events returns a matrix $O \in [0,1]^{l \times n}$ where entry $O[i,j]$, with $i = 1,\ldots,l$ and $j = 1,\ldots,n$ (assuming l and n are the length of $\boldsymbol{X}$ and the number of atomic events, respectively), represents the probability that event j happens at timestamp i. The predicted structured event for a video $\boldsymbol{X}$ is the one maximizing the probability of the corresponding output head of NN, i.e.:

$$\hat{y}^{\mathcal{S}} = \underset{i=1,\ldots,k}{\text{argmax}}\ o_i$$

In principle, the sequence of atomic events could be predicted in a similar fashion by maximizing for each frame the probability of the atomic event head of NN at that frame, i.e.:

$$\hat{y}^{\mathcal{A}}_i = \underset{1 \leq j \leq l}{\text{argmax}}\ O[i,j] \tag{1}$$

Indeed, this is how our fully-neural baseline works. However, the vector $\hat{\boldsymbol{y}}^{\mathcal{A}}$ of atomic event predictions for the frames of a video $\boldsymbol{X}$ may contain inconsistencies (e.g., predicting the atomic event *jump* as part of a *javelinthrow* structured event, predicting *fall* before *jump* within a *highjump*, or even predicting a *jump* that is much longer than the *run* that precedes it). Our neuro-symbolic architecture prevents these inconsistencies by combining neural network predictions with hard and soft constraints provided by the domain knowledge. The domain knowledge we exploit is quite simple, and provides hard constraints determining the sequence of atomic events that constitute a structured event, and soft constraints about minimal and maximal duration of each atomic event and relative duration between atomic events making up a structured event. Table 1 reports an example of the hard constraints that we consider for the *javelinthrow* structured event. Similar constraints are generated for the other structured events. Given the structured event $\hat{y}^{\mathcal{S}}$ predicted by NN, the corresponding sequence of atomic events is computed by solving a MILP problem encoding the (hard

Table 1 Example of hard constraints for the *javelinthrow* structured event, divided into generic constraints that hold for any structured event, and those specific of the *javelinthrow* event.

Hard Constraints	
Generic Constraints (assuming k atomic events)	
$e_i > b_i \quad \forall\, i$	Events should end after they began
$b_1 = 1 \wedge e_k = l$	Sequence of atomic events should span the whole clip
$e_i = b_{i+1} - 1 \quad \forall\, i \in 0 \ldots l-1$	No gap among consecutive events
Specific Constraints (for the *javelinthrow* structured event)	
$a_1 = run \wedge a_2 = throw$	*javelinthrow* is a *run* followed by a *throw*
$d_1 > d_2$	*run* should take longer than *throw*

and soft) constraints combined with a scoring function measuring the compatibility of the sequence of atomic events with the NN outputs O. The MILP problem for a structured event (we have a separate problem for each possible structured event) is defined as follows:

$$\begin{aligned} \underset{V}{\text{minimize}} \quad & -f(V,O) + \sum_{j=1}^{m_s} \xi_t c_j(V) \\ \text{subject to} \quad & h_i(V) \quad \forall\, i = 1, \ldots, m_h \end{aligned} \tag{2}$$

Here V is a sequence of triplets (a, b, e), where $a \in \mathcal{A}$ is an atomic event, $b, e \in \mathbb{N}$ are the starting and ending frames of the event respectively. The number of atomic events is determined by the structured event being modelled. The scoring function $f(V, O)$ computes the compatibility of V with O as follows:

$$f(V,O) = \sum_{(a,b,e)\in V} \left(\sum_{i=b}^{e} O[i,a] - \sum_{j=1}^{b-1} O[j,a] - \sum_{j=e+1}^{l} O[j,a] \right)$$

It basically computes the sum of the probabilities of each atomic event in the range in which it is predicted, and subtracts its probability outside of this range (l is the overall length of the video clip).

The soft constraints $c_j(V)$ encode duration ranges for atomic events or combination of atomic events. For instance, the constraint that the sum of the durations of *run* and *jump* should be within the sum of the maximal and minimal durations respectively is formalized as follows:

$$\min(|d_1 + d_2 - max_{run} - max_{jump}|, |d_1 + d_2 - min_{run} - min_{jump}|)$$

where:

$$d_1 = e_1 - b_1 + 1,\ d_2 = e_2 - b_2 + 1$$
$$a_1 = run,\ a_2 = jump$$

The hard constraints $h_i(V)$ encode temporal relations between atomic events and with respect to the structured event. See Table 1 for examples. Intuitively, the solutions of the MILP problem for the predicted structured event $\hat{y}^{\mathcal{S}}$ where only hard contraints $h_i(V)$ are

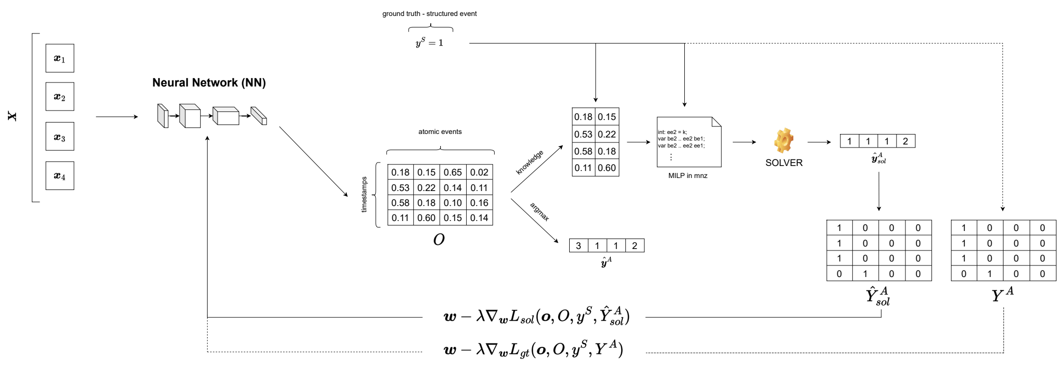


Figure 2 Training of our neuro-symbolic approach.

considered, provide a set of candidate interpretations for $\boldsymbol{X}$. By including the objective $f(V, O)$ and the soft constraints $c_j(V)$ we obtain the interpretation with the minimum cost for $\boldsymbol{X}$ (i.e, Y_c^*). The label vector $\hat{y}^{\mathcal{A}}_{sol}$ in Figure 1, which is the neuro-symbolic counterpart of $\hat{y}^{\mathcal{A}}$ in Eq. 1, is obtained by "unrolling" the optimal V into an atomic label for each frame in its predicted range.

▶ **Example 2.** Let $\boldsymbol{X}$ a video of length 20 where a person is performing a structured event, but we do not know which kind of structured event. Now, suppose we have, in addition to the structured event highjump of example 1, the structured event javelin throw and that the background knowledge contains the fact that a javelin throw can be decomposed into a sequence of two atomic events: run and jump, which can be expressed by the following formula:

$$\begin{aligned}&\forall b_{jt} e_{jt} (happens(javelinthrow, b_{jt}, e_{jt}) \leftrightarrow \exists\, b_r, e_r, b_t, e_t, (\\&\qquad happens(run, b_r, e_r) \wedge happens(throw, b_t, e_t) \wedge\\&\qquad b_r = b_{jt} \wedge e_r = b_t \wedge e_t = e_{hj}))\end{aligned}$$

Also, suppose that the head of NN for structure events predicted $\hat{y}^{\mathcal{S}} = javelinthrow$ for $\boldsymbol{X}$. If we solve the MILP for the javelin throw where only hard constraints are considered (for example the ones in table 1), we have that some of the candidate interpretations will be:

$$\begin{aligned}\mathcal{I}_1 &= \{happens(javelinthrow, 1, 21), happens(run, 1, 13), happens(throw, 13, 21)\}\\\mathcal{I}_2 &= \{happens(javelinthrow, 1, 21), happens(run, 1, 17), happens(throw, 17, 21)\}\\&\ \ \vdots\end{aligned}$$

By considering and solving the whole MILP, we obtain I_c^*. Continuing the example, if we have a soft constraint which penalizes interpretations having short duration for *run*, we have that $I_c^* = I_2$, that corresponds to the solution $V = [(run, 1, 17), (jump, 17, 21)]$ of Problem 2.

Figure 2 shows the training process of the architecture. As we assume that the ground-truth $y^{\mathcal{S}}$ of $\boldsymbol{X}$ is available at training time, the head for the structured events is not used anymore to predict $\hat{y}^{\mathcal{S}}$, but the ground-truth itself is used instead. Furthermore, if the ground-truth for the atomic events ($Y^{\mathcal{A}} \in R^{l \times n}$) is also available, we use it to train the head of the atomic events. If this information is not available, we use pseudo-labels generated from the architecture. The generation of such pseudo-labels consists of an inference step

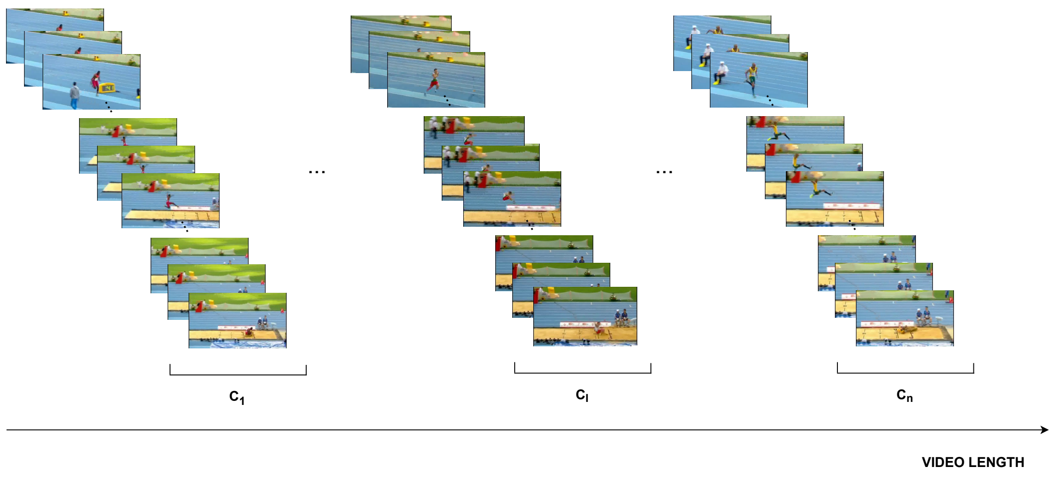


Figure 3 Extraction of clips of structured events from an untrimmed video.

in the currently trained architecture as per Figure 1, with the only difference that the structured event is given by the ground-truth $y^{\mathcal{S}}$ rather than the NN output. The atomic event prediction vector $\hat{\boldsymbol{y}}^{\mathcal{A}}_{sol}$ is turned into a binary label matrix $\hat{Y}^{\mathcal{A}}_{sol} \in \{0,1\}^{l \times n}$ by one-hot encoding atomic labels (i.e., $\hat{Y}^{\mathcal{A}}_{sol}[i,j]$ is set to 1 if $j = \hat{y}^{\mathcal{A}}_{sol}[i]$ and 0 otherwise). Then, we define two losses:

$$L_{gt}(\boldsymbol{o}, O, y^{\mathcal{S}}, Y^{\mathcal{A}}) = L(\boldsymbol{o}, y^{\mathcal{S}}) + L(O, Y^{\mathcal{A}}) \tag{3}$$

$$L_{sol}(\boldsymbol{o}, O, y^{\mathcal{S}}, \hat{Y}^{\mathcal{A}}_{sol}) = L(\boldsymbol{o}, y^{\mathcal{S}}) + L(O, \hat{Y}^{\mathcal{A}}_{sol}) \tag{4}$$

Where Loss 3 refers to the case where both ground-truth (structured and atomic) are available, while Loss 4 refers to the case where the ground-truth for structured events is available and the ground-truth for the atomic events is not, and, then, we use the pseudo labels. In order to train NN to recognize both kind of the events, we minimize, depending on the case, one of the aforementioned loss and use gradient descent to update its weights.

5 Experimental setting

Our experimental setting has the aim to show if our proposed neuro-symbolic approach leads to an advantage in the recognition of both structured and atomic events with respect to a fully neural approach, when both approaches are trained with weak and limited amount of supervision in terms of events. In details, we want to see how the knowledge is able to compensate in the case when no or few and potentially noisy labels for events are available. To achieve this objective, we first need a dataset of structured and atomic events. We build such dataset from the Multi-THUMOS untrimmed video dataset [29]. Since in [29], there is no explicit distinction between structured and atomic events, we define it and splits the events according. In particular, we consider as structured events those events that can be decomposed as a sequence of other (atomic) events, and cut each video into clips corresponding to structured events (Figure 3).

For each structured event, we do not consider all the clips, but we remove those clips where some of the atomic events defining the structured event are not present (e.g, replay). The structured and atomic events we consider are shown in appendix B. Then, after building the dataset, we define the scenario. The scenario consists of clips of different length where, in each clip, a person is performing one (and only one) structured event among the ones reported in appendix B. The learning setting we consider to evaluate the fully neural approach and our proposed neuro-symbolic approach consists in having full supervision at level of structured events and limited and potentially noisy (e.g., overlapping between atomic events) supervision in terms of atomic events. The kind of supervision we provide is as follows:

$$\{happens(highjump, 1, 50),\ happens(run, 1, 31),\ happens(jump, 31, 45),\ happens(fall, 45, 50)\}$$

$$\{happens(hammerthrow, 1, 30),\ happens(windup, 1, 15),\ happens(spin, 10, 25),\ happens(release, 25, 30)\}$$

$$\{happens(javelinthrow, 1, 30)\}$$

The first video is an example of noiseless labeling with full supervision on both structured and atomic events. The second video is fully supervised too, but atomic supervision is noisy, as there is an overlap between the *windup* and *spin* atomic events (this type of overlapping labeling is not rare in the dataset). The third video is a case where supervision is only provided at the structured event level, and there is no supervision on atomic events.

In this setting, we are interested in observing how the prediction in terms of atomic and structured events change when increasing the availability of data for atomic events. Furthermore, in the case of the neuro-symbolic approach, we are interested in seeing how complementing the supervision coming from the dataset with the supervision coming from the knowledge affects the predictions of the overall model. A noteworthy case is the one where no direct supervision at level of atomic events is provided at all, and then the model is completely trained with the supervision coming from the knowledge. The underlying model we use for both approaches is the one described in [21] where features extracted from a pre-trained two-stream I3D [7] are given as input in order to predict a matrix of events (more details about the model can be found in Appendix C). Differently from [21], we distinguish between structured and atomic events and consider two separate heads, as discussed in the previous section. About the training, we train the model for 20 epochs with learning rate of 1e−3 and weight decay of 1e−6, using Adam as optimizer. We also created a validation set of 10% of the training data in order to select the best model.

6 Results

In this section, we show and compare the results of the fully neural approach with respect to our proposed neuro-symbolic approach on the task described in Section 5. Figure 4 reports average F_1 scores of structured event prediction over 5 runs, for an increasing amount of supervision in terms of atomic events (from 0 to 100%). Note that in all cases supervision in terms of structured events is always provided. The green curve indicates the fully neural baseline, while the red curve indicates our neuro-symbolic approach. Results indicate that unsurprisingly, when there is full supervision on the structured event the addition of knowledge does not help in its identification.

Figure 5 reports average F_1 scores for the prediction of atomic events, again for a growing amount of supervision at the atomic level. The difference between the fully neural and the neuro-symbolic approach is striking. Substantial improvements of the neuro-symbolic approach can be observed for almost all atomic events. Only for the atomic events *run* and *jump*, we can see that the fully neural approach is really close to our approach. This is probably due to the temporal duration of these events, that is substantially higher than that

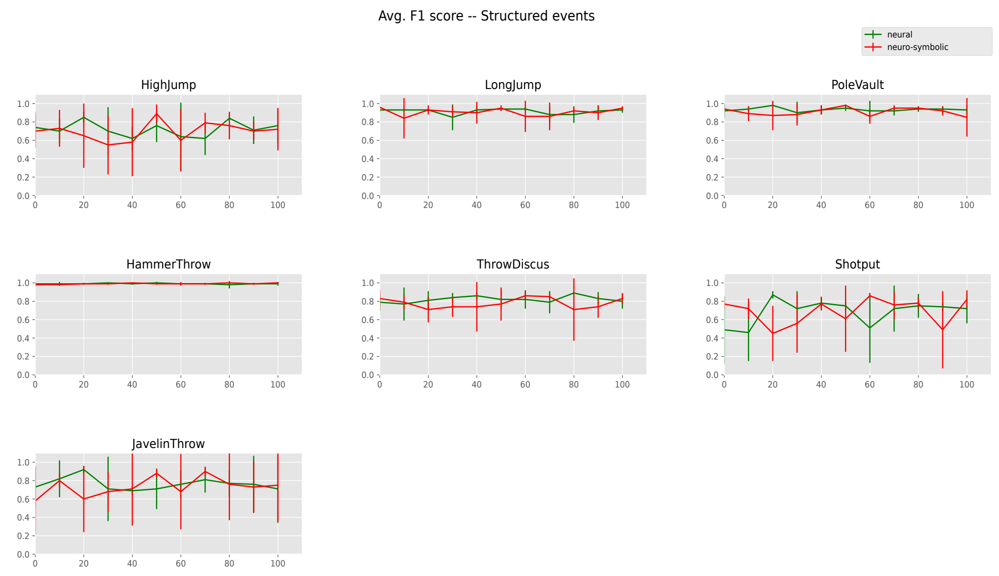


Figure 4 F_1 scores on structured events averaged over 5 runs for the fully neural (green) and the neuro-symbolic (red) approaches, for increasing amount of supervision on atomic events.

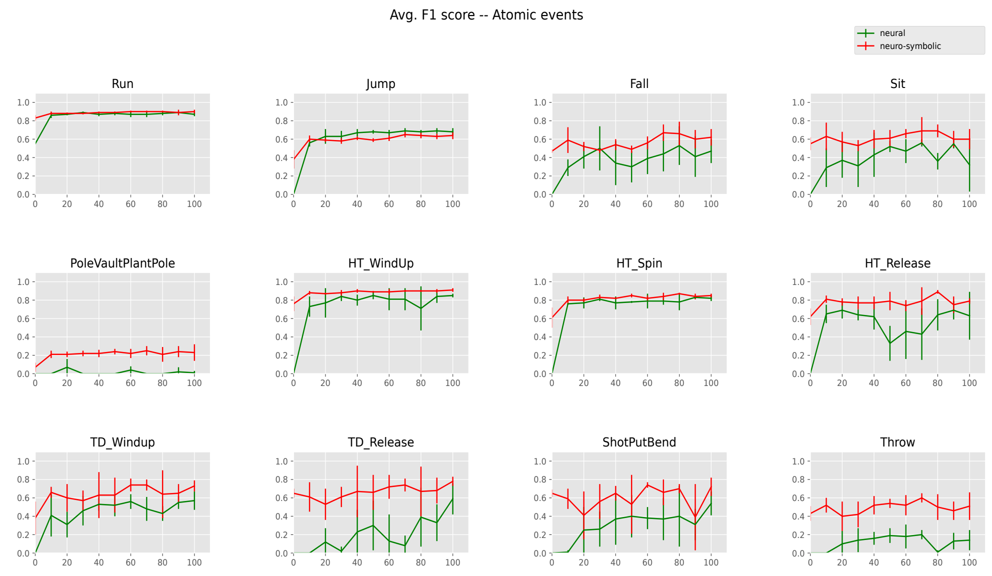


Figure 5 F_1 scores on atomic events averaged over 5 runs for the fully neural (green) and the neuro-symbolic (red) approaches, for increasing amount of supervision on atomic events.

of the others. This implies that a reasonable number of frames labelled as *run* and *jump* will be available for the neural network even with a small fraction of labelled videos. On the other hand, atomic events like *release* and *throw* have a performance improvement that goes up as 60% and 50% respectively.

As stated in Section 5, a particularly significant case is the one where no direct supervision at all is provided at the level of atomic events. This corresponds to the leftmost point in the figures. The F_1 score of the fully neural approach is close to zero for almost all atomic events, corresponding to random guessing. On the other hand, the F_1 scores of the neuro-symbolic approach are often comparable to those of a fully supervised setting, showing how knowledge can be exploited to completely bypass the need for human supervision at the frame level, with major implications in terms of applicability and training costs.

Figure 6 shows some representative examples of the labeling provided by the two approaches highlighting the differences in prediction consistency between the two, both in terms of atomic events being detected and relative duration. Note that results are achieved with 100% supervision on atomic events, and highlight the importance of knowledge in guaranteeing the consistency of predictions. The Figure shows prediction for all frames in a video for three videos, a *highjump*, a *hammerthrow* and a *longjump* respectively. For each video, we compare ground truth atomic labels (yellow) with fully neural predictions (green) and neuro-symbolic ones (green). In the *highjump* case (top), the fully neural approach completely misses the last event and mispredicts it as part of the *jump* event. On the other hand, the neuro-symbolic approach correctly detects the last event a *fall*, and has a better estimate of the duration of each event. In the *hammerthrow* case (middle), the fully neural approach detects a *run* event, that cannot be part of a *hammerthrow*, and misses the *release* event. Again, the neuro-symbolic approach provides quite accurate estimates of the duration of each event, despite the short duration of *release* with respect to *windup* and *spin*. Finally, in the *longjump* case (bottom), the neural approach correctly identifies the initial *run*, but breaks the rest of the video into a sequence of short *jump*, *fall*, *sit* and even *run* events which is completely inconsistent, while the neuro-symbolic approach again accurately recovers both sequence and duration of the atomic events.

7 Conclusion and Future Work

In this work, we have proposed a neuro-symbolic approach for (structured and atomic) event recognition where knowledge about the events and their temporal relations is exploited both at training and inference time. We have instantiated our approach on a real-world scenario consisting of clips of sports events. Our experimental evaluation showed how our neuro-symbolic solution achieves substantial improvements over a fully neural baseline in terms of recognition of the atomic events that constitute a structured event. The approach is capable of learning to detect atomic events even with no supervision at all on them during training, by simply combining supervision on structured events, low-level neural processing and knowledge. While these results are promising, there are several directions which are left open for future research. A major direction consists in increasing the complexity of the scenarios being considered, by dealing with structured events involving multiple actors and complex relationships between the events, without making the underlying reasoning problem prohibitively expensive, something other neuro-symbolic frameworks currently struggle with.

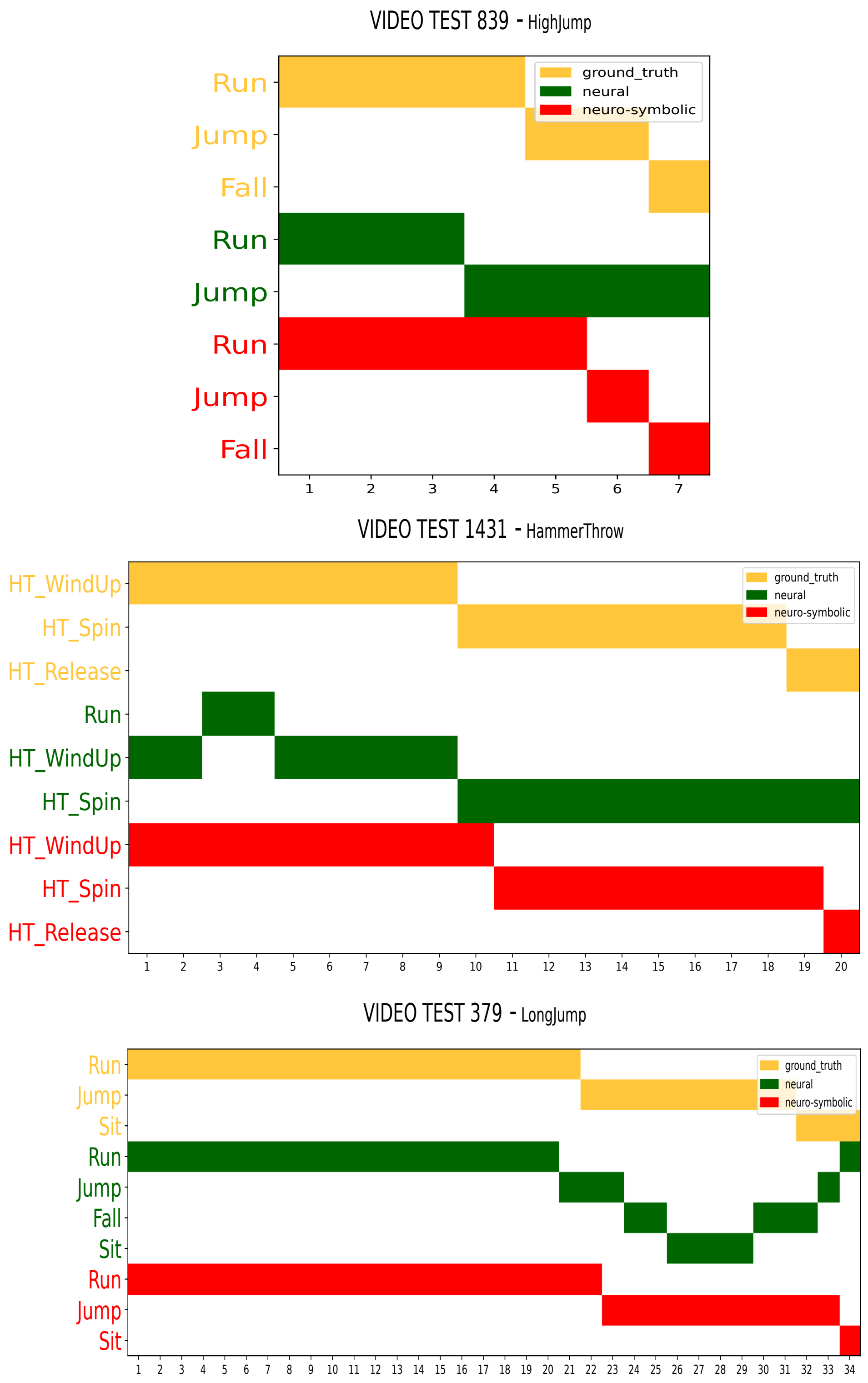


Figure 6 Prediction of the sequence of atomic events for three clips representing a *highjump* (top), a *hammerthrow* (middle) and a *longjump* (bottom) respectively. Ground truth is in yellow, while the neural and neuro-symbolic predictions are in green and red respectively. Both models were trained with 100 % supervision on the atomic events. Clips were selected to show examples of inconsistencies in neural predictions.

References

1 Kashif Ahmad Ahmad and Nicola Conci. How Deep Features Have Improved Event Recognition in Multimedia: A Survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 15(2):39:1–39:27, 2019. doi:10.1145/3306240.

2 Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. Probabilistic Complex Event Recognition: A Survey. *ACM Computing Surveys*, 50(5):71:1–71:31, 2017. doi:10.1145/3117809.

3 Gianluca Apriceno, Andrea Passerini, and Luciano Serafini. A neuro-symbolic approach to structured event recognition. In Carlo Combi, Johann Eder, and Mark Reynolds, editors, *28th International Symposium on Temporal Representation and Reasoning, TIME 2021, September 27-29, 2021, Klagenfurt, Austria*, volume 206 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.TIME.2021.11.

4 Alexander Artikis, Evangelos Makris, and Georgios Paliouras. A probabilistic interval-based event calculus for activity recognition. *Annals of Mathematics and Artificial Intelligence*, 89(1-2):29–52, 2021. doi:10.1007/s10472-019-09664-4.

5 Alexander Artikis, Anastasios Skarlatidis, François Portet, and Georgios Paliouras. Logic-based event recognition. *The Knowledge Engineering Review*, 27(4):469–506, 2012. doi:10.1017/S0269888912000264.

6 Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artif. Intell.*, 303:103649, 2022. doi:10.1016/j.artint.2021.103649.

7 João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4724–4733. IEEE Computer Society, 2017. doi:10.1109/CVPR.2017.502.

8 Ivan Donadello, Luciano Serafini, and Artur S. d'Avila Garcez. Logic tensor networks for semantic image interpretation. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1596–1602. ijcai.org, 2017. doi:10.24963/ijcai.2017/221.

9 Paolo Dragone, Stefano Teso, and Andrea Passerini. Neuro-symbolic constraint programming for structured prediction. In Artur S. d'Avila Garcez and Ernesto Jiménez-Ruiz, editors, *Proceedings of the 15th International Workshop on Neural-Symbolic Learning and Reasoning as part of the 1st International Joint Conference on Learning & Reasoning (IJCLR 2021), Virtual conference, October 25-27, 2021*, volume 2986 of *CEUR Workshop Proceedings*, pages 6–14. CEUR-WS.org, 2021. URL: http://ceur-ws.org/Vol-2986/paper2.pdf.

10 José Roldán Gómez, Juan Boubeta-Puig, José Luis Martínez, and Guadalupe Ortiz. Integrating complex event processing and machine learning: An intelligent architecture for detecting iot security attacks. *Expert Syst. Appl.*, 149:113251, 2020. doi:10.1016/j.eswa.2020.113251.

11 Pascal Hitzler and Md Kamruzzaman Sarker. *Neuro-Symbolic Artificial Intelligence: The State of the Art*. IOS Press, 2022.

12 Jeya Vikranth Jeyakumar, Luke Dickens, Luis Garcia, Yu-Hsi Cheng, Diego Ramirez-Echavarria, Joseph Noor, Alessandra Russo, Lance M. Kaplan, Erik Blasch, and Mani B. Srivastava. Automatic concept extraction for concept bottleneck-based video classification. *CoRR*, abs/2206.10129, 2022. doi:10.48550/arXiv.2206.10129.

13 Abdullah Khan, Loris Bozzato, Luciano Serafini, and Beatrice Lazzerini. Visual Reasoning on Complex Events in Soccer Videos Using Answer Set Programming. In Diego Calvanese and Luca Iocchi, editors, *GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence, Bozen/Bolzano, Italy, 17-19 September 2019*, volume 65, pages 42–53. EasyChair, 2019. doi:10.29007/pjd4.

14 Abdullah Khan, Luciano Serafini, Loris Bozzato, and Beatrice Lazzerini. Event Detection from Video Using Answer Set Programing. In Alberto Casagrande and Eugenio G. Omodeo, editors, *Proceedings of the 34th Italian Conference on Computational Logic, Trieste, Italy, June 19-21, 2019*, volume 2396 of *CEUR Workshop Proceedings*, pages 48–58. CEUR-WS.org, 2019. URL: http://ceur-ws.org/Vol-2396/paper25.pdf.

15 Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5338–5348. PMLR, 2020. URL: `http://proceedings.mlr.press/v119/koh20a.html`.

16 Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc D. Raedt. DeepProbLog: Neural Probabilistic Logic Programming. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, volume 31, pages 3753–3763, 2018. URL: `https://proceedings.neurips.cc/paper/2018/hash/dc5d637ed5e62c36ecb73b654b05ba2a-Abstract.html`.

17 Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. LYRICS: A General Interface Layer to Integrate Logic Inference and deep Learning. In Ulf Brefeld, Élisa Fromont, Andreas Hotho, Arno J. Knobbe, Marloes H. Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases – European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II*, volume 11907 of *Lecture Notes in Computer Science*, pages 283–298, 2019. `doi:10.1007/978-3-030-46147-8_17`.

18 Luc D. Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2462–2467, 2007. URL: `http://ijcai.org/Proceedings/07/Papers/396.pdf`.

19 Roberto Sebastiani and Silvia Tomasi. Optimization Modulo Theories with Linear Rational Costs. *ACM Transactions on Computational Logics*, 16(2), 2015.

20 Anastasios Skarlatidis, Alexander Artikis, Jason Filippou, and Georgios Paliouras. A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming*, 15(2):213–245, 2015. `doi:10.1017/S1471068413000690`.

21 Praveen Tirupattur, Kevin Duarte, Yogesh Singh Rawat, and Mubarak Shah. Modeling multi-label action dependencies for temporal action localization. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 1460–1470. Computer Vision Foundation / IEEE, 2021. URL: `https://openaccess.thecvf.com/content/CVPR2021/html/Tirupattur_Modeling_Multi-Label_Action_Dependencies_for_Temporal_Action_Localization_CVPR_2021_paper.html`, `doi:10.1109/CVPR46437.2021.00151`.

22 Marc Roig Vilamala, Liam Hiley, Yulia Hicks, Alun D. Preece, and Federico Cerutti. A pilot study on detecting violence in videos fusing proxy models. In *22th International Conference on Information Fusion, FUSION 2019, Ottawa, ON, Canada, July 2-5, 2019*, pages 1–8. IEEE, 2019. URL: `https://ieeexplore.ieee.org/document/9011329`.

23 Marc Roig Vilamala, Tianwei Xing, Harrison Taylor, Luis Garcia, Mani B. Srivastava, Lance M. Kaplan, Alun D. Preece, Angelika Kimmig, and Federico Cerutti. Using deepproblog to perform complex event processing on an audio stream. *CoRR*, abs/2110.08090, 2021. `arXiv:2110.08090`.

24 Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. *CoRR*, abs/2106.12574, 2021. `arXiv:2106.12574`.

25 Wei Xiang and Band Wan. A Survey of Event Extraction From Text. *IEEE Access*, 7:173111–173137, 2019. `doi:10.1109/ACCESS.2019.2956831`.

26 Tianwei Xing, Luis Garcia, Marc Roig Vilamala, Federico Cerutti, Lance M. Kaplan, Alun D. Preece, and Mani B. Srivastava. Neuroplex: learning to detect complex events in sensor networks through knowledge injection. In Jin Nakazawa and Polly Huang, editors, *SenSys '20: The 18th ACM Conference on Embedded Networked Sensor Systems, Virtual Event, Japan, November 16-19, 2020*, pages 489–502. ACM, 2020. `doi:10.1145/3384419.3431158`.

27 Tianwei Xing, Marc R. Vilamala, Luis Garcia, Federico Cerutti, Lance Kaplan, Alun Preece, and Mani Srivastava. DeepCEP: Deep Complex Event Processing Using Distributed Multimodal Information. In *IEEE International Conference on Smart Computing, SMARTCOMP 2019, Washington, DC, USA, June 12-15, 2019*, pages 87–92. IEEE, 2019. doi:10.1109/SMARTCOMP.2019.00034.

28 Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing Neural Networks into Answer Set Programming. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1755–1762. ijcai.org, 2020. doi:10.24963/ijcai.2020/243.

29 Serena Yeung, Olga Russakovsky, Ning Jin, Mykhaylo Andriluka, Greg Mori, and Li Fei-Fei. Every moment counts: Dense detailed labeling of actions in complex videos. *Int. J. Comput. Vis.*, 126(2-4):375–389, 2018. doi:10.1007/s11263-017-1013-y.

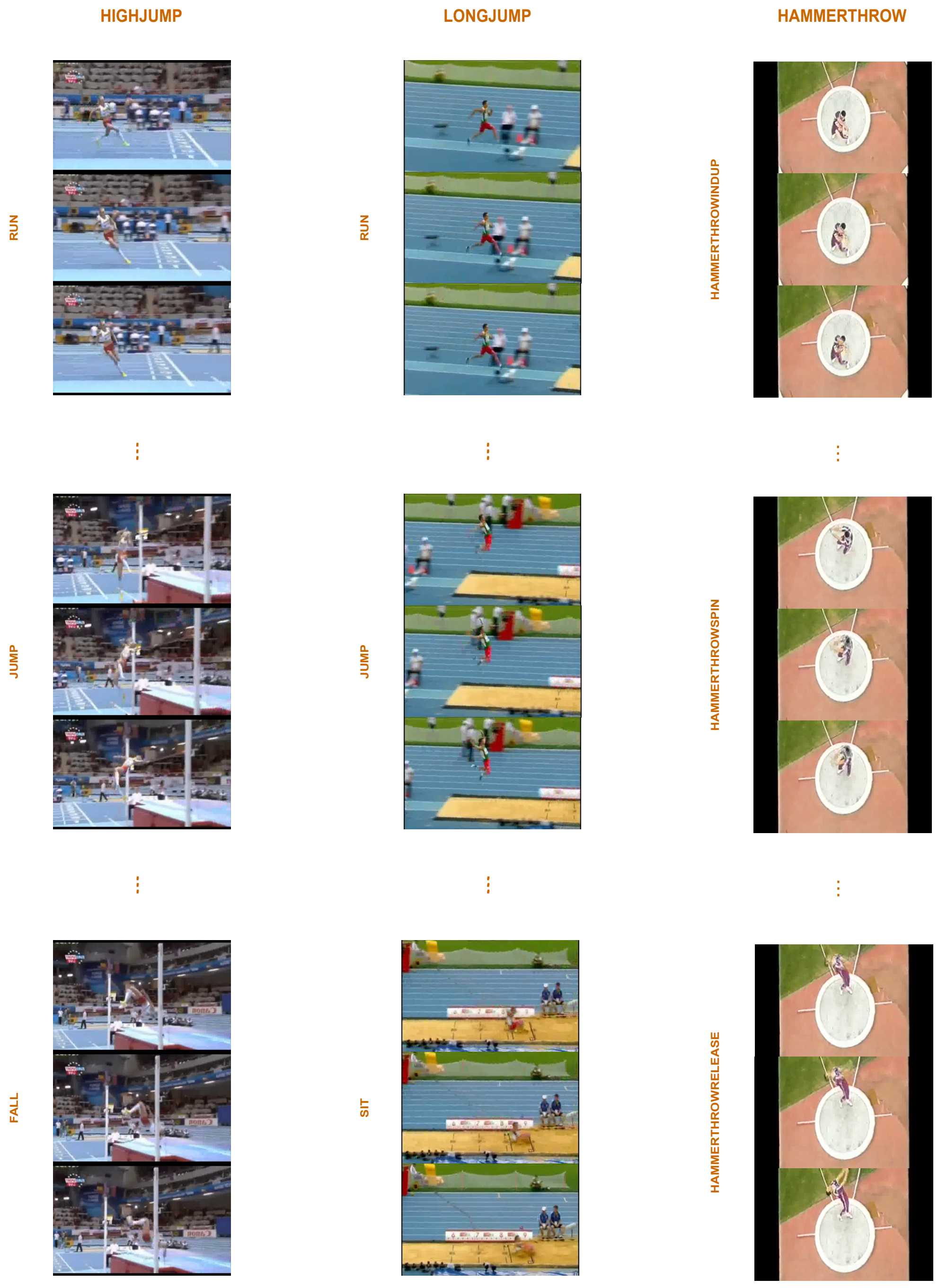
A MILP for high jump

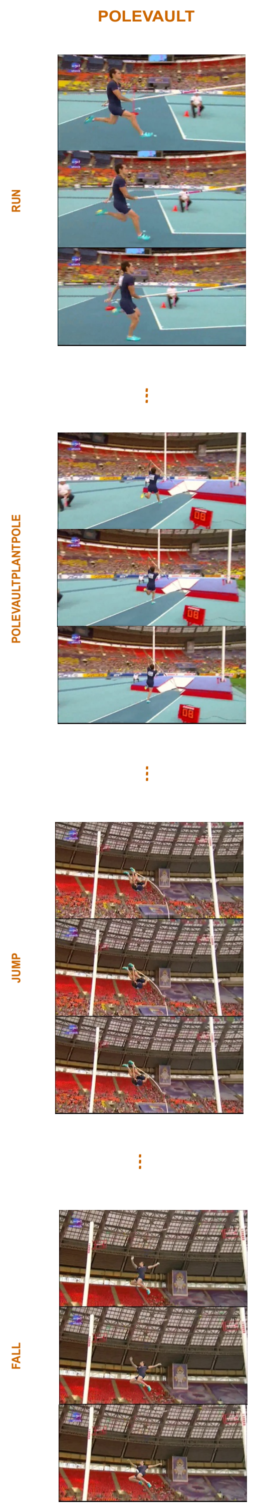
Listing 1 Encoding high jump as a MILP using MiniZinc

```
% HJ = High Jump
% R  = Run
% J  = Jump
% F  = Fall

int: bHJ;
int: eHJ;

int: minSum_R_J = 3;
int: maxSum_R_J = 48;
int: minSum_R_F = 4;
int: maxSum_R_F = 57;
int: minSum_J_F = 3;
int: maxSum_J_F = 33;

int: target_R_J = maxSum_R_J - minSum_R_J + 1;
int: target_R_F = maxSum_R_F - minSum_R_F + 1;
int: target_J_F = maxSum_J_F - minSum_J_F + 1;

var bHJ .. eHJ: bR;
var bHJ .. eHJ: eR;
var bHJ .. eHJ: bJ;
var bHJ .. eHJ: eJ;
var bHJ .. eHJ: bF;
var bHJ .. eHJ: eF;

var int: lenR = eR - bR + 1;
var int: lenJ = eJ - bJ + 1;
var int: lenF = eF - bF + 1;


constraint eR > bR /\ eJ > bJ /\ eF > bF;
constraint bR == bHJ /\ eR == (bJ-1) /\ eJ == (bF-1) /\ eF == eHJ;
constraint lenR >= (lenJ + lenF) /\ lenJ < (lenR + lenF) /\ lenF < (lenR + lenJ);


var int: cost_comp_run_pos = - sum (t in bR..eR) (ae_predictions[1,t]);
var int: cost_comp_run_neg = sum (t in (eR+1)..eHJ) (ae_predictions[1,t]);
var int: cost_comp_jump_pos = - sum (t in bJ..eJ) (ae_predictions[2,t]);
var int: cost_comp_jump_neg_1 = sum (t in bHJ..(bJ-1)) (ae_predictions[2,t]);
var int: cost_comp_jump_neg_2 = sum (t in (eJ+1)..eHJ) (ae_predictions[2,t]);
var int: cost_comp_fall_pos = - sum (t in bF..eF) (ae_predictions[3,t]);
var int: cost_comp_fall_neg = sum (t in bHJ..(bF-1)) (ae_predictions[3,t]);

var int: cost = (
  cost_comp_run_pos + cost_comp_run_neg
  + cost_comp_jump_pos + cost_comp_jump_neg_1 + cost_comp_jump_neg_2
  + cost_comp_fall_pos + cost_comp_fall_neg
  + 1000 * abs(target_R_J - (lenR + lenJ))
  + 1000 * abs(target_R_F - (lenR + lenF))
  + 1000 * abs(target_J_F - (lenJ + lenF))
);

solve minimize cost;
```

In lines 6-7, the constants representing the begin and the end of the clip are declared. These are going to be filled at running time and will change depending on the length of the clip processed. The blocks of lines 9-14 and 16-18 contains, the minimum/maximum sum of the length of the intervals of two atomic events and the target length in which the sum of the two intervals has to lie in. The declaration of the optimizer decision variables is contained in line 20-25. These variables are going to be set by Gurobi at the end of the optimization. In lines 27-29, the variables representing the length of the interval of each atomic events are defined. Lines 32-34 represent hard constraints that Gurobi has to satisfy. In details, line 32 states that the end of each atomic event has to be greater than their corresponding begin, while lines 33 and 34 defines respectively temporal relations among events and algebraic constraints among the length of the intervals of atomic events. In addition to the satisfaction of those constraints, Gurobi has to minimize a cost function (lines 45-62). The cost function can be split in two parts. The former (lines 46-48) is composed by the sum of components defined in lines 37-43 where we want to maximize (i.e. minimize) the sum of probability where the solver states the atomic events are happening (pos), and minimize (i.e. maximize) the sum of probability where the atomic events are not happening (neg). The latter (lines 49-51) refers to soft constraints where the solver has to set the the sum of the length of two atomic events' intervals to be as closed as possible to their target length.

B Subset of structured and atomic events considered


HIGHJUMP
LONGJUMP
HAMMERTHROW
RUN
RUN
HAMMERTHROWINDUP
JUMP
JUMP
HAMMERTHROWSPIN
FALL
SIT
HAMMERTHROWRELEASE


POLEVAULT
RUN
POLEVAULTPLANTPOLE
JUMP
FALL

C Temporal action localization model

The input of the model consists in a series of feature vectors $\boldsymbol{f}$ extracted by a pre-trained two-stream I3D [7], where each $f_i \in \boldsymbol{f}$ corresponds to 8 frames (or 0.33 seconds) and contains global information at both frame and video-clip level. A non linear transformation is applied on these features in order to obtain class level features $(C \times T \times H)$ with C representing the number of classes, T the number of timestamps and H the dimension of embedding space. Then, the class-level features are refined using L attention-based Multi-Label Action Dependency (MLAD) layers. These layers are composed by two disjoint branches which adopt a self-attention operation to model the relationships between actions that happened within the same timestamp (referred as Co-occurence Dependency branch) and actions happening at different timestamps (referred as Temporal Dependency branch). As a result, a refined set of features is returned by each branch, respectively. At the end, a linear combination of the two branches' features is applied (through a learnt value $\alpha \in [0, 1]$) and the result is passed to C individual classification layers which outputs class probabilities for each timestamp $(T \times C)$.

Neural-Symbolic Temporal Decision Trees for Multivariate Time Series Classification

Giovanni Pagliarini
Department of Mathematics and Computer Science, University of Ferrara, Italy
Department of Mathematics, Physics, and Computer Science, University of Parma, Italy

Simone Scaboro
Department of Mathematics, Physics, and Computer Science, University of Udine, Italy

Giuseppe Serra
Department of Mathematics, Physics, and Computer Science, University of Udine, Italy

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

Ionel Eduard Stan
Department of Mathematics and Computer Science, University of Ferrara, Italy
Department of Mathematics, Physics, and Computer Science, University of Parma, Italy

Abstract

Multivariate time series classification is a widely known problem, and its applications are ubiquitous. Due to their strong generalization capability, neural networks have been proven to be very powerful for the task, but their applicability is often limited by their intrinsic black-box nature. Recently, temporal decision trees have been shown to be a serious alternative to neural networks for the same task in terms of classification performances, while attaining higher levels of transparency and interpretability. In this work, we propose an initial approach to neural-symbolic temporal decision trees, that is, an hybrid method that leverages on both the ability of neural networks of capturing temporal patterns and the flexibility of temporal decision trees of taking decisions on intervals based on (possibly, externally computed) temporal features. While based on a proof-of-concept implementation, in our experiments on public datasets, neural-symbolic temporal decision trees show promising results.

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

Keywords and phrases Machine learning, neural-symbolic, temporal logic, hybrid temporal decision trees

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.13

Supplementary Material *Software (ModalDecisionTrees.jl)*: `https://zenodo.org/record/7040420`

Funding The authors acknowledge the partial support by the Italian INDAM GNCS project *Natural Language Elaboration and Temporal Logic for Text Formalization.*

1 Introduction

A *multivariate time series* is a collection of time-stamped tuples, each composed by the value of several attributes. Time series describe a variety of situations, and *classification* of time series is an active area of research across many scientific disciplines: air quality control and prediction in climate science, prices and rates of inflation in economics, infectious diseases trends and spreading patterns in medicine, pronunciation of word signs in linguistics, sensor recordings of systems in aerospace engineering, among many others [29].

As it is true for any other classification problem, the classification of multivariate time series too can be approached by means of both *symbolic* and *functional* (or *parametric*) machine learning, which are two fundamental pillars of modern machine learning; in short, one may say that functional learning is the process of learning a *function* that represents the theory of the underlying problem (functional methods range from simple *regression* techniques to the modern *neural network* models, in their several variants), while symbolic learning is the process of learning a *logical description* that represents that theory (typical symbolic methods are *rule-based classifiers* and *decision trees*). Whether one or the other approach should be preferred raised a long-standing debate among experts, whose roots lie in the fact that functional methods tend to be more accurate than symbolic ones, as they are capable to better generalize the problem at hand, while symbolic methods are able to extract models that can be explained, interpreted, and integrated with human expert knowledge. The higher *interpretability* degree of symbolic approaches over functional ones, both for political (consider, e.g., the General Data Protection Regulation (GDPR) that highlights the need for interpretable/explainable automatic decision processes) and technical reasons, are sometimes used as arguments for preferring a symbolic approach over a functional one. As suggested in [9, 10, 24] (among others), however, in order to solve the functional/symbolic duality, one can think of an *hybrid* approach: hybrid systems combine the strengths of both symbolic and functional methods, with the aim of guaranteeing highi degrees of interpretability of the learned models, while retaining high enough statistical performances.

On the one side, we shall consider the native, symbolic time series classification method proposed by Sciavicco and Stan [31]. The *temporal decision tree* prediction model, as it is called, is an extension and generalization the decision tree paradigm. Temporal decision trees work as the classical ones, but decisions are taken on *intervals* of time series, instead of the series as a whole; therefore, there is no need of an initial feature extraction phase, but, on the contrary, features are extracted dynamically, following the standard greedy approach. Temporal decision trees have already been shown to be competitive for time series classification. On the other side, we shall take, as a representative example, a *time-preserving autoencoder* neural network model [37, 39], which is characterized by being able to play both the role of time series classifier and the one of feature extractor, essentially without any modification. As it turns out, in the pursue of an hybrid decision tree model there are at least three independent parameters which can be combined, namely, the possibility of using a network for an initial screening of the dataset, to be later dealt with in a more precise way by one of several potentially different trees (*root hybridization*), the possibility of querying an external network as a feature extractor in order to take split decisions in a single tree (*split hybridization*), and the possibility of consulting one of several potentially different networks at the leaves of a decision tree before deciding the class (*leaf hybridization*). These techniques give rise to eight possible hybrid models, the simplest one of which is just a decision tree. In this paper, we propose to increase the generalization capacity of temporal decision trees by leveraging the power of a pre-trained autoencoder to partition instances in a decision node (split hybridization). To assess the value of our proposal, we perform several experiments on three public datasets. In particular, we consider the problem of establishing if the neural-symbolic approach is, in fact, beneficial, when compared with the pure neural network-based one as well as the pure temporal symbolic one. Therefore, our experiments are not designed to achieve the highest absolute performances (such as in Bagnall et al.'s [3] work, from which our datasets are taken), which is often the results of a very intensive hyperparametrization, complex feature extraction, and model stacking/bagging; instead, they are structured in such a way as to highlight the advantages of the hybrid approach over its constituents.

The paper is organized as follows: in Section 2 we review the most important contributions in the area of hybridization decision trees with neural networks; in Section 3 we present neural-symbolic temporal decision trees; then, in Section 4 we benchmark the proposed methodology based on a proof-of-concept implementation against public datasets, before concluding.

2 Related Work

Decision trees and neural networks are well-known alternatives for pattern recognition, and their strengths and weaknesses have been studied for more than three decades [2, 35]. Notoriously, decision trees favor the interpretability of their decisions which, due to their symbolic nature, represent coarse concepts in numeric domains, whereas neural networks are more difficult to interpret (they are often referred to as *black-box models*), but have a better generalization capacity. Let us focus on the recent literature concerning the hybridization of these two models.

From decision trees to neural networks. First, we mention Sethi [32], Brent [6] and Ivanova and Kubat [15], who proposed a mapping algorithm of a decision tree into a 2-hidden-layers neural network, whose topology is inferred by the structure of the decision tree; then the weights can be retrained by error-backpropagation to increase generalization. Ivanova and Kubat's Tree-Based Neural Network has been further investigated by Setiono and Leow [33] by compressing and removing redundant units and connections in the resulting network, a method called Pruning-Based Neural Network. Radial-basis function networks can also be initialized by decision trees, where each region in the instance space discovered by the decision tree is then turned into a neuron, each of which representing a basis function, in the resulting network [19].

From neural networks to decision trees. Craven and Shavlik [7] observed that decision tree induction algorithms are limited by the fact that splits are performed over fewer and fewer instances recursively. The decision tree inducer algorithm that they have proposed alleviates such issue by querying the oracle (i.e., the trained neural network) which generates new instances aiming at performing better statistically solid m-to-n Boolean splits [38] (i.e., m out of n conditions must be satisfied) that are greedily evaluated using gain ratio [28]. Dancey et al. [8] proposed a similar method where the splits are univariate. Krishnan et al. [18], inspired by Craven and Shavlik's work, extracted a decision tree from inputs generated by the neural network instead of doing it directly from the data, that is, they also query the oracle. Unlike Craven and Shavlik's method, Schmitz et al. [30] proposed an approach that can be applied to inputs and outputs that are both discrete or continuous with a novel attribute significance analysis to perform splits. Zhou and Jiang [42], instead, proposed to extract a C4.5 decision tree from the input instances merged with new randomly generated instances from an ensemble of neural networks. Such an approach an be seen as a loop that uses a genetic algorithm to generate *prototypes* (i.e., representative input instances for which the neural network give a desired output classification) to train the decision tree; at this stage, the decision tree is tested on an independent test set and if the performances are acceptable then the procedure stops; otherwise, other prototypes are generated by the GA and the cycle continues.

Hybrid neural-symbolic models. Li et al. [21] proposed a top-down adaptive neural tree for hierarchical classification that can add and delete nodes incrementally while inducing the tree structure. To increase the generalization of CART-like decision trees [5], Guo and Gelfand's [12] method trained a small 1-hidden-layer perceptron with one output at each node of the decision tree to learn a non-linear, multivariate feature $f(\cdot)$ that splits the subset of the training instances at that node based on the test $f(\cdot) < 0$ (since the output's activation function is the *tanh* function) and showed that their method performed better

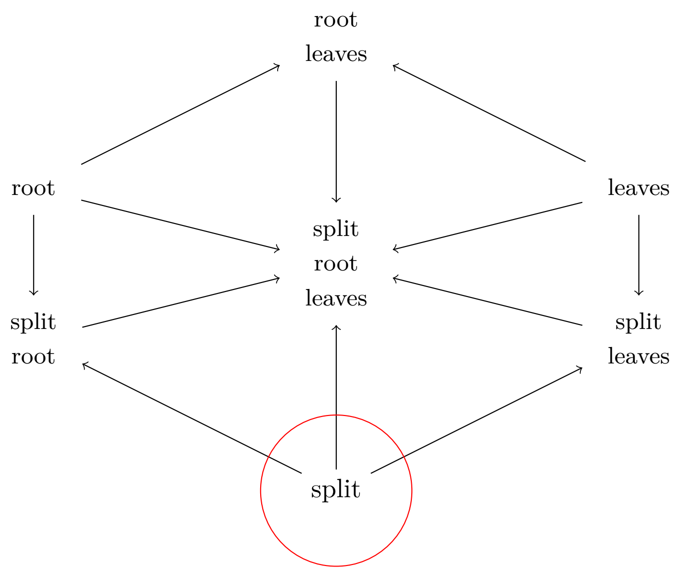


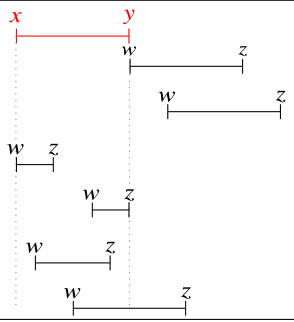
Figure 1 Different types of hybridization, partially ordered by the residual level of interpretability. In this paper, we focus our attention on temporal decision trees hybridized at the split level only.

than CART in terms of accuracy and better than a larger multi-layered neural network trained with backpropagation in terms of training time. A similar approach can be found in Setiono and Liu's [34] work for generating oblique decision trees. Zhou and Chen [41] proposed a methodology to induce a hybrid decision tree where, first, at the internal nodes of the decision tree the splits are done over unordered attributes only, if any, to perform qualitative analysis and, then, at the leaf nodes a virtual feed-forward neural network is embedded to perform quantitative analysis over the ordered attributes only, if any. More recently, Micheloni et al. [23] proposed a novel neural tree by using two innovations, namely perceptron substitution and pattern removal, to produce k-ary balanced trees ($k \geq 2$). In the field of computer vision, tree-structured neural networks have been proposed, among others, by Srivastava and Salakhutdinov [36] and Hinton et al. [14] to transfer knowledge, by Kontschieder et al. [17] where a decision tree has been introduced after a fully connected layer as part of the convolutional neural network, by Murthy et al. [26] where each decision node of the decision tree is a convolutional neural network, by Murdock et al. [25] where several layers are fused into the decision nodes of the decision tree, by Alaniz et al. [1] where the structure of the decision tree is encoded in the memory of a recurrent neural network jointly learned by two models acting as agents through message communication, and by Wan et al. [40] where the final layer of the network is replaced by a decision tree.

In an attempt at taxonomizing the existing work, one could argue that there are at least three independent parameters which can be combined, namely the possibility of using a network for an initial screening of the dataset, to be later dealt with in a more precise way by one of several potentially different trees (*root hybridization*), the possibility of querying an external network as a feature extractor in order to take split decisions in a single tree (*split hybridization*), and the possibility of consulting one of several potentially different networks at the leaves of a decision tree before deciding the class (*leaf hybridization*). As a result, there are at least eight types of hybridization (the simplest one of which is just a decision tree), which can be, in a sense, partially ordered by their residual level of interpretability, as

Table 1 Allen's interval relations and their representation.

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



in Fig. 1. When the architecture of the underlying neural networks are comparable, different hybridization types become comparable as well. In our initial exploration, we focus on the hybridization at the split level only.

3 Neural-Symbolic Temporal Decision Trees

A single *multivariate time series* has n (temporal) attributes $A_1, \ldots, A_n$ evolving through a time axis, whose values are time-stamped by N integers (the *length* of the series), that is, its underlying domain of interest is $(\{0, 1, \ldots, N-1\}, <)$. A *temporal labelled dataset* is a set of m multivariate time series, each labelled with a *class*. The multivariate time series *classification problem* is the problem of automatically extract a classifier from a temporal labelled dataset. Existing multivariate time series classification methods can be divided into *feature-based* (see, e.g., [20]), *instance-based* (see, e.g., [3]) and *native* ones (see, e.g., [11]).

Designed as a native time series classification method, *temporal decision trees* have been recently proposed [22, 31]. Let $\mathcal{T} = \{T_1, \ldots, T_m\}$ be a temporal dataset of m instances, where each is a multivariate time series described by n attributes $\{A_1, \ldots, A_n\}$. Given $T \in \mathcal{T}$ and a time point t, we denote by $A(t)$ the value of A at the point t, and by $dom(A)$ the domain of A. Now, let f a *dynamic feature* of the variable A (e.g., the average value of A over an interval); in its simplest form, f is a *scalar descriptor* for A within any non-point interval of the series.

The key idea of interval temporal decision trees is that decisions are taken over *strict* intervals, that is, intervals of the type $[x, y]$ with $x < y$, and for time series whose length is N, there are $N \cdot (N-1)/2$ such intervals. A temporal decision tree starts off by looking at the whole dataset from the point of view of the first temporal value, and searches through all possible non-point intervals, and, for each one of them, it computes a predetermined set of dynamic features; then, it searches through all possible interval-interval relations, and it establishes which other interval, and which other dynamic feature over that interval, is most informative in the considered sub-dataset. In this way, it applies the same abstract approach of the classical static decision tree up until a dataset is small enough, or pure enough, so that a stopping criteria can be applied and a leaf can be created. For each possible feature f, let $dom(f(A))$ denote the set of possible values that f takes over A throughout $\mathcal{T}$. The temporal dataset $\mathcal{T}$ entails a propositional alphabet $\mathcal{AP}$ defined as follows:

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

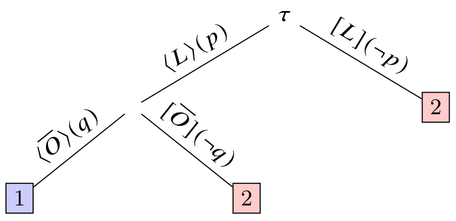


Figure 2 Example of temporal decision tree.

The set $\mathcal{AP}$ is the natural generalization of the set of propositional letters that implicitly emerges in inductive processes from static data; for example, in a static dataset, the propositional letter *fever greater than* 38 *degrees* may emerge. The main difference between the two cases, propositional and temporal, is that in the latter case propositions in $\mathcal{AP}$ are given an interval semantics, that is, they are evaluated over intervals of time; this is a natural choice that depends from the fact that time series describe continuous processes, in which evaluations based on point-wise values have little sense. Intuitively, consider an interval of time $[x, y]$ and an attribute A that varies on it. We can ask the question $A \bowtie a$ over the entire interval, which is positively answered if *every value* of A in the interval $[x, y]$ respects the given constraint; but, to enhance an interval-based semantics we replace this question with $f(A) \bowtie a$, which, in general, allows us to extract more information from the interval $[x, y]$; in the above example, we may have the proposition *average fever greater than 38 degrees*. As it turns out, many dynamic features have been studied in the literature of time series to extract features from *whole* series, and these range from simple functions such as *average, minimum,* or *maximum* to very complex ones such as *number of local minima* or *number of local maxima*; we generalize this concept by applying them to *intervals*, which are, themselves, series. Thus, in temporal decision trees the *univariate split-decisions* (or, simply, *decisions*) that partition a set of instances at a specific node are of the type:

$$\mathcal{S} = \{\langle X\rangle(p) \mid X \in \mathcal{X} \cup \{=\} \text{ and } p \in \mathcal{AP}\},$$

where $\mathcal{X} = \{A, L, B, E, D, O, \overline{A}, \overline{L}, \overline{B}, \overline{E}, \overline{D}, \overline{O}\}$, contains the possible interval-interval relations, whose informal semantics is depicted in Tab. 1 (for a formal definition of the semantics, see [13]). In conclusion, binary *temporal decision trees* τ are formulas of the following grammar:

$$\tau ::= (S \wedge \tau) \vee (\neg S \wedge \tau) \mid C,$$

where $S \in \mathcal{S}$ is a decision and $C \in \mathcal{C}$ is a class. An example of temporal decision tree is depicted in Fig. 2. Recall that time series of length N have $(\{0, 1, \ldots, N-1\}, <)$ as underlying domain of interest. The learning starts by splitting the dataset at the root, where each time series is fixed on the *dummy* interval $[-2, -1]$ so that the only feasible interval-interval operators are $\langle L\rangle$ (for the left branch) and $[L]$ (for the right branch); then, recursively, for the instances that fall into the left branch, the intervals that are *witnesses* for the decision $\langle X\rangle(p)$ are the new intervals to which the time series are fixed to find the next interval-interval relation with a proposition (from $\mathcal{A}P$) as argument; similarly, if the instances fall on the right branch by some decision $[X](\neg p)$, the witnesses remain the same as before the split. What is more, just as static decision trees induce propositional formulas on their branches, temporal decision trees induce *interval temporal logic* [13] formulas. Continuing the parallelism with the static case, a static decision tree may give rise to a branch whose associated formula is *fever greater than 38 degrees and cough as a symptom*, whereas a formula on a temporal

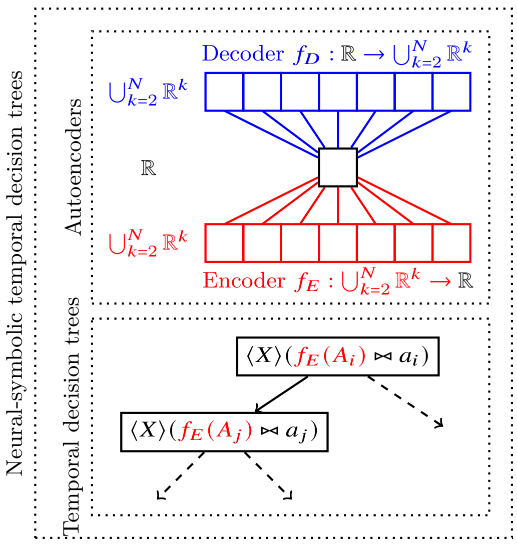

Neural-symbolic temporal decision trees
Autoencoders
Decoder $f_D : \mathbb{R} \to \bigcup_{k=2}^{N} \mathbb{R}^k$
$\bigcup_{k=2}^{N} \mathbb{R}^k$
$\mathbb{R}$
$\bigcup_{k=2}^{N} \mathbb{R}^k$
Encoder $f_E : \bigcup_{k=2}^{N} \mathbb{R}^k \to \mathbb{R}$
Temporal decision trees
$\langle X \rangle (f_E(A_i) \bowtie a_i)$
$\langle X \rangle (f_E(A_j) \bowtie a_j)$

Figure 3 Neural-symbolic temporal decision trees pipeline. At the top level, an autoencoder is trained for each attribute A_i on each of its non-point interval of a multivariate time series. At the bottom level, a temporal decision tree queries the encoder of each trained autoencoder to partition instances in a decision node.

decision tree may be *(a period of) average fever greater than 38 degrees overlapping (a period of) cough as a symptom*; to complete the example, following the left-most branch of the temporal decision tree illustrated in Fig. 2, the corresponding interval temporal logic formula is:

$$\langle L \rangle (p \wedge \langle \overline{O} \rangle (q)),$$

from which, by interpreting p as *average fever greater than 38 degrees* and q as *cough as symptom*, we obtain the above sentence in natural language.

An *autoencoder* is a neural network architecture typically used for extracting significant feature representations from unlabeled data. The feature extraction is achieved by training the model to reproduce its input (i.e., to learn the identity function) while introducing an information bottleneck throughout the model. This generally results in an encoder-decoder architecture where the encoder (that is, the first part of the network), ends with a layer with the smallest number of neurons, which is, then, the only input to the remaining part of the network. Ultimately, the training phase forces the encoder to learn to extract a succinct representation of the input, performing a non-linear dimensionality reduction, and the decoder to learn to retrieve the original information from this representation. After the training phase, the encoder can be used as a feature extractor, that is, a model that provides a succinct abstract description of its input.

In the case of time series, a fundamental distinction among autoencoders is between *time preserving* models, which provide a description that also extends along the time axis, and *non-time preserving* ones, which provide a static description, only consisting of a fixed-size vector of scalar values. The architectures used for time series are typically based on convolutional neural networks, that are effective for shift-invariant pattern recognition, or recurrent neural networks, that are specifically designed for temporal data.

Table 2 Dataset specifications.

Dataset	# train+test instances	# points	# attributes	# classes
Libras	180 + 180 = 360	45	2	15
NATOPS	180 + 180 = 360	51	24	6
RacketSports	151 + 152 = 303	30	6	4

In this first attempt at split hybridization of temporal decision trees, autoencoders are used to derive attribute-specific feature extractors; that is, once an autoencoder is trained, the encoder part, seen as a function whose input is mapped to a real number, plays the same role that the average, minimum, and maximum functions play. With respect to these simpler function, the learned encoder has a black-box nature, and is, therefore, less interpretable; however, as we shall see, it can yield higher specificity for a given attribute, thus providing scalar descriptions that are more relevant. For the purpose of this work, we consider a sequence-to-sequence [37] architecture (referred to as *S2S*), and a transformer-based [39] architecture (referred to as *transformer autoencoder, TSA*); these are two time preserving autoencoders which found fruitful application in many contexts, and represented a major breakthrough in the field of natural language processing. Note that, being time preserving architectures, the descriptions computed by the encoders extend along a time axis, whereas the framework presented above requires a scalar feature extraction. To solve this issue, we operate a choice commonly adopted in contexts where a scalar feature extraction is needed: we only consider a single temporal slice of the description. For both architectures, we considered the one temporal slice that is used by the decoder to reproduce the input series but, because of structural differences between the two architectures, different policies are required: for S2S, the last temporal slice is considered while, for TSA, the first slice is considered. Fig. 3 depicts an abstraction of the structure of the autoencoders in use, and the deployment of the trained encoders within the split-decisions of a temporal decision tree. Recall that the encoder maps its input series (seen as a vector of real numbers) to a single real number, therefore reducing the original size.

4 Experiments

In order to assess the performances of hybrid temporal decision trees, we carried out several experiments using three publicly available datasets that are commonly employed for benchmarking multivariate time series classification models. They are known, respectively, as Libras, NATOPS, and RacketSports [3], and their specifications are shown in Tab. 2. The Libras dataset consists of sensor recordings of hand movements in a bidimensional space, extracted from videos of Brazilian sign language speakers performing different gestures; NATOPS data consists of 3D recordings of hands, elbows, wrists and thumbs of people performing different actions; and RacketSports contains 3D recordings for both a gyroscope and an accelerometer mounted on a smartwatch worn by several subjects while playing badminton and squash games.

All datasets are provided with pre-existent partitioning into training set and test set. The expertiments are conducted in a randomized cross-validation setting with 10 repetitions, where the training and test sets for each repetition are drawn from the union of the original sets, reproducing the class distributions of the original sets, with the sole exception that the first of the 10 repetitions uses the exact original training and test sets; this approach is similar to Ruiz et al. [29]. Six variations of decision trees are compared, which are obtained by using both a static decision tree model and the temporal decision tree one, each, in turn,

Table 3 Test results and training time of the decision tree models in comparison. Values for the performance metrics are shown in percentage points. For each measure, the table reports the average and standard deviation over 10 repetitions. For each dataset, the most performant model is highlighted.

			κ	OA	AA	F1	time (s)
Libras	DT	min, max	35.4 ± 3.4	39.7 ± 3.1	39.7 ± 3.1	39.3 ± 3.2	0.1
		neural	19.0 ± 4.3	24.4 ± 4.0	24.4 ± 4.0	23.8 ± 4.0	0.1
		min, max, neural	40.9 ± 5.9	44.8 ± 5.5	44.8 ± 5.5	44.2 ± 5.6	0.1
	TDT	min, max	54.6 ± 4.3	57.6 ± 4.0	57.6 ± 4.0	57.2 ± 3.8	6.3 ± 1.6
		neural	54.5 ± 3.7	57.5 ± 3.5	57.5 ± 3.5	56.7 ± 4.0	18.0 ± 5.1
		min, max, neural	**55.2 ± 4.1**	**58.2 ± 3.8**	**58.2 ± 3.8**	**57.6 ± 3.9**	**30.7 ± 6.5**
NATOPS	DT	min, max	65.1 ± 3.7	70.9 ± 3.1	70.9 ± 3.1	70.8 ± 3.3	0.7 ± 0.1
		neural	42.8 ± 4.3	52.3 ± 3.6	52.3 ± 3.6	52.1 ± 3.8	0.6 ± 0.1
		min, max, neural	65.7 ± 2.2	71.5 ± 1.8	71.5 ± 1.8	71.4 ± 1.9	1.0 ± 0.1
	TDT	min, max	84.0 ± 2.9	86.7 ± 2.4	86.7 ± 2.4	86.7 ± 2.4	37.0 ± 9.0
		neural	**87.1 ± 3.7**	**89.2 ± 3.1**	**89.2 ± 3.1**	**89.3 ± 3.1**	**118.3 ± 35.8**
		min, max, neural	86.7 ± 2.9	88.9 ± 2.4	88.9 ± 2.4	89.0 ± 2.4	252.1 ± 98.3
RacketSports	DT	min, max	55.4 ± 3.3	66.6 ± 2.4	68.0 ± 2.5	67.4 ± 2.3	0.2
		neural	44.2 ± 3.9	58.4 ± 3.0	59.6 ± 2.8	59.2 ± 3.1	0.2 ± 0.1
		min, max, neural	**57.5 ± 6.9**	**68.2 ± 5.2**	**69.7 ± 5.1**	**69.3 ± 5.3**	**0.3 ± 0.1**
	TDT	min, max	55.0 ± 5.8	66.3 ± 4.3	67.7 ± 4.2	67.5 ± 4.1	1.1 ± 0.9
		neural	56.0 ± 5.6	67.1 ± 4.2	68.2 ± 4.0	68.1 ± 4.3	2.7 ± 1.5
		min, max, neural	56.3 ± 5.8	67.3 ± 4.3	68.6 ± 4.2	68.3 ± 4.1	5.5 ± 5.4

in three versions: original (non-hybrid), split hybrid using only neural features, and split hybrid using both neural and non-neural features. As we have seen in the previous section, temporal decision trees are characterized by taking decisions on intervals of time, and then relating such decisions via temporal logic formulas. This is paradigm-shifting with respect to static ones, which, in the common literature, can only deal with dimensional data (for us, temporal data) by extracting global features for each attribute and then using them as decisions. Such a difference has been maintained in the hybrid version; in fact, in the static case both neural and non-neural features are computed on the whole series. For these experiments, in both the temporal and the static case, the non-neural feature functions were fixed to minimum and maximum. Temporal and static trees were trained using the ModalDecisionTrees.jl open-source Julia package [27], which implements the CART algorithm and its modal extensions.

After a preliminary study in which different decision tree parametrizations are tested, two tree-pruning conditions are fixed, namely, a minimum number of instances at the tree leaves of 2 for RacketSports and Libras, and 4 for NATOPS, and a minimum entropy gain of 0.01, which prevents less informative splits to be performed at any internal node. As neural feature extractors, we use S2S for Libras and TSA for NATOPS and RacketSports. For the training of both neural architectures, we use PyTorch and the following hyperparameters: AdamW optimizer with 10^{-5} learning rate and 10^{-8} epsilon factor, batches of size 256 with accumulation step equal to 4, L1Loss as loss function, 150 epochs, gradient clipping during training, and weights initialized using the default setting of PyTorch. The architecture of S2S is the same proposed in [4] with the difference that we used two Gated Recurrent Unit Networks as encoder and decoder instead of the LSTMs. A simpler architecture is used to

Table 4 Class-wise accuracies and average accuracy (AA) (shown in percentage points) of the decision tree models in comparison. For each entry, the table reports the average over 10 repetitions. The best result of each class is highlighted.

		Libras															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	AA
DT	min, max	29	41	26	45	53	38	34	39	**88**	28	38	38	36	16	**48**	40
	neural	17	12	13	46	38	22	34	8	30	14	28	32	19	22	30	24
	min, max, neural	35	57	25	59	53	31	42	50	87	36	41	43	44	28	42	45
TDT	min, max	48	77	**49**	70	**70**	49	**64**	55	**88**	**42**	**55**	51	41	**58**	**48**	58
	neural	**59**	76	42	68	59	**56**	62	**58**	**88**	35	52	**60**	**50**	50	47	58
	min, max, neural	56	**78**	46	**72**	69	51	62	55	86	39	54	53	53	52	47	**58**

		NATOPS						
		1	2	3	4	5	6	AA
DT	min, max	91	77	65	52	51	90	71
	neural	48	45	40	60	52	69	52
	min, max, neural	91	73	64	57	53	90	72
TDT	min, max	93	87	68	90	90	92	87
	neural	**95**	**88**	**70**	**91**	**94**	92	**89**
	min, max, neural	**95**	87	67	**91**	**94**	**94**	89

		RacketSports				
		1	2	3	4	AA
DT	min, max	50	**55**	83	84	68
	neural	54	36	85	64	60
	min, max, neural	55	52	**88**	85	**70**
TDT	min, max	51	54	80	85	68
	neural	**60**	49	77	**87**	68
	min, max, neural	54	53	82	84	69

build TSA, which is composed of a transformer encoder layer with two heads, both for the encoder and for the decoder. In the encoder, we used Time2Vec [16] to resize each temporal slice from 1 to 168 adding also information about the position of the slices, as done by classical transformers with the positional embeddings. Both the output of the encoder and of the decoder is resized from 168 to 1 with a linear layer. Note that, while static decision trees feature functions are computed on the whole series, features for temporal decision trees are evaluated on *all* the sub-intervals of the series; as such, the autoencoder models are trained using the raw training instances in the first case, and using all the sub-intervals of the training instances in the second case.

Tab. 3 gives an overview of the trained models in terms of the training time required (excluding the prior training of the neural networks), and the performance obtained on the test data. The results are given in terms of mean and standard deviations over the 10 repetitions. The performance itself is measured in terms of κ coefficient (which relativizes the overall accuracy to the probability of a random correct answer), overall accuracy (OA), average accuracy (AA), and average F1-score (F1). Note that the four metrics measure in different ways the overall performance of the models, but they all happen to be in agreement; as such, we focus on the values of the κ coefficients, which is invariant to the number of classes, and varies largely across the three datasets.

At a first look, Libras seems to enclose the hardest problem for the models at hand (κ equal to 55.2%), followed by RacketSports (57.5%) and then by NATOPS (87.2%). Libras and NATOPS reveal a performance gap between static and temporal trees; that is, when the best models for each group is considered, the second group has an average κ higher by ~15 and ~22 percentage points, respectively. Conversely, RacketSports represents a case where temporal decision trees are outperformed by classical decision trees; indeed, in both cases, the best accuracy is achieved using both simple and neural features, but temporal trees achieve an average κ of 56.3%, while classical trees achieve 57.5%. When it comes to neural features, classical DTs only benefit from them when these are used as supplementary information; in fact, for all datasets, classical decision trees using only neural features are

Table 5 Confusion matrixes for the tree generated with seed 6, dataset NATOPS; left: static, non-neural; right: temporal, neural.

	static, non-neural					
	1	2	3	4	5	6
1	**24**	6	0	0	0	0
2	2	**26**	2	0	0	0
3	0	14	**16**	0	0	0
4	1	0	0	**19**	9	1
5	1	0	0	13	**16**	0
6	0	0	0	0	3	**27**

	temporal, neural					
	1	2	3	4	5	6
1	**30**	0	0	0	0	0
2	0	**28**	2	0	0	0
3	0	5	**25**	0	0	0
4	0	0	0	**26**	1	3
5	0	0	0	0	**28**	2
6	3	1	0	0	0	**26**

outperformed by classical DTs based on minimum and maximum, but the latter are in turn always outperformed by mixed trees that use both simple and neural features, which may indicate that our approach is promising. If we look separately at DTs and TDTs within the same experimental settings, we observe, once again, that adding neural features gives a greater advantage in the temporal case; a possible interpretation of this phenomenon is that, being able to perform qualitative temporal reasoning among intervals when partitioning instances, the inductive procedure becomes some kind of *symbolic attention* mechanism; however, as it must be acknowledged, TDTs generally require higher training times due to the interval-interval relations.

A more specific analysis of the trends can be made by inspecting the ability of the models for correctly classifying each of the classes in the three datasets. Tab. 4 reports the class-wise accuracies, as well as the average accuracy; although they provide useful insights, class-wise accuracies are subject to a higher variance than metrics of overall performance, and, as such, reliable conclusions from them can be drawn only when comparing different groups of models. We observe that with the Libras dataset the step from static to temporal learning encompasses a general accuracy improvement over all classes except two (9 and 15), and in some classes, such as 14, such an improvement is impressive (from 28% to 58%); classes 1, 2, 12, and 13, moreover, show a clear benefit of the hybrid temporal-and-neural approach. The improvement due to the temporal approach in NATOPS emerges in all classes except class 1, and in all classes, though in a lesser amount, adding the neural features results in a further improvement. Finally, as for RacketSports, the improvement of the temporal approach is less clear, and only visible in class 1 and 4, and the same holds for the further, slight, improvement given by the addition of the neural features.

In addition to the simple analysis of the numerical performance, we can give a closer look at the generated trees. In order to do so, we consider the dataset NATOPS and a representative seed, namely seed 6, and compare the extracted trees in terms of structure, size, and ability to predict specific classes in two specific cases: the static tree without neural features and the temporal tree with neural features (Fig. 4 and Fig. 5, in which we used m to represent the minimum, M the maximum, and N the encoding by the encoder neural network, applied to an attribute A). As it can be observed, the static one is 40% bigger than the temporal one, the former having 17 leaves, versus the 11 leaves of the latter. Yet, the classification abilities of the static tree are clearly lower than those of the temporal one: on the one side, in average, the static non-neural approach presented 71% accuracy versus the 89% of the temporal one with neural and non-neural features; on the other side, these specific trees present 71% accuracy in the static case versus 91% in the temporal case. Even more interestingly, class 5 labels several leaves in the static tree, and only 1 in the temporal

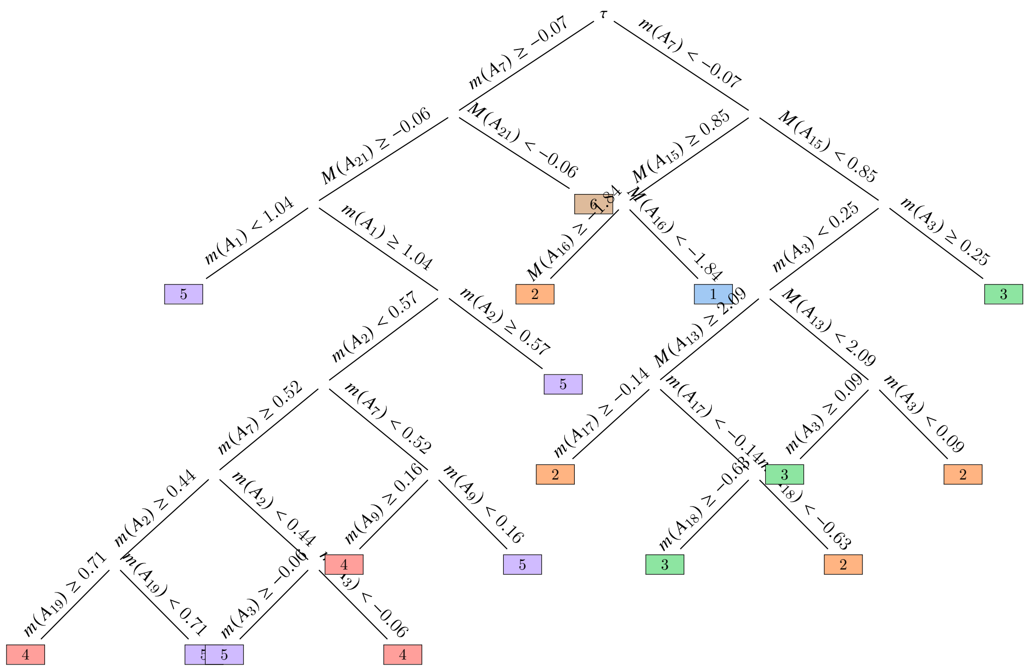


Figure 4 Static tree without neural features, seed 6, NATOPS.

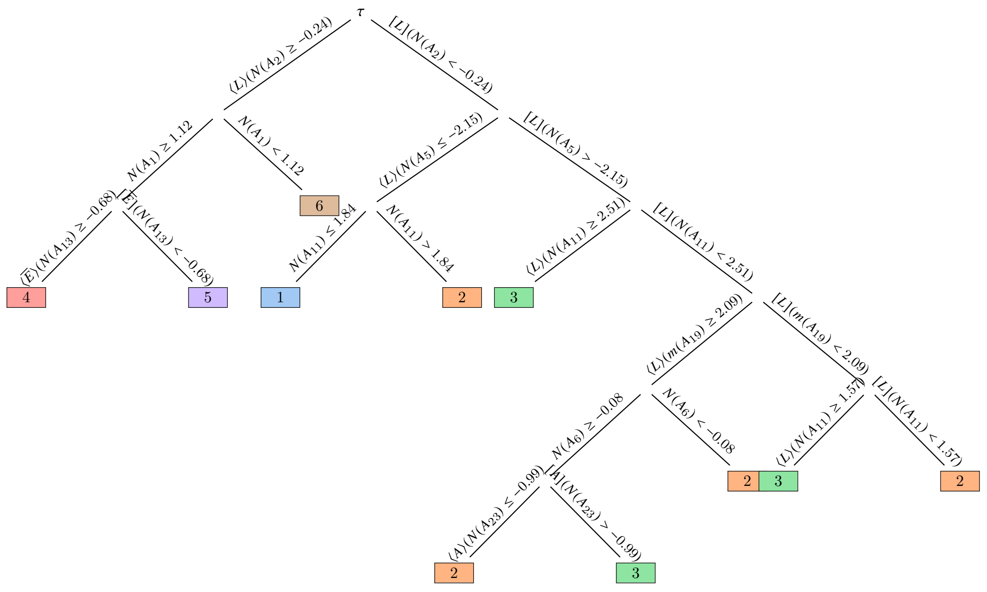


Figure 5 Temporal tree with neural features, seed 6, NATOPS.

one, indicating that the temporal tree was able to extract the essence, in some sense, of this class, in a single formula. Observe that class 5 is classified correctly only 51% of the times in the static tree versus 94% of the times in the temporal one, in average, and 53% in the static case versus 93% in the temporal one for the considered trees. More in particular, as it can be deduced from the observation of the confusion matrices in both cases (Tab. 5), the static tree confuses class 5 with class 4 very often, and, in lesser amount, with class 6, while

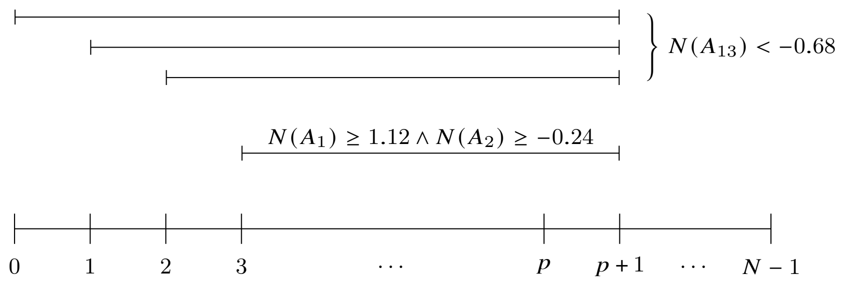


Figure 6 A model for class 5 of the NATOPS dataset.

the temporal neural tree presents similar mistakes a very reduced number of times. In other words, the temporal neural approach was able to extract a smaller and more precise logical description of class 5, which can be summarized into a single temporal formula:

$$\langle L\rangle(N(A_2) \geq -0.24 \wedge N(A_1) \geq 1.12 \wedge [\overline{E}]N(A_{13}) < 0.68) \Rightarrow 5,$$

which, in turn, can be expressed as a temporal model to visualize, in a sense, the class itself (see Fig. 6 and recall our discussion from the previous section on the witnesses for the left/right branch). To finalize this discussion, we observe that, in the static case, 7 variables were involved in the decision tree for class 5, while in the temporal case only 3 variables were sufficient.

5 Conclusions

In this paper we have presented a method for multivariate time series classification that combines the high generalization capacity of trained neural networks with the symbolic nature of temporal decision trees. This method is able to learn the structure of a temporal decision trees from raw multivariate time series, and, internally (at each decision node) performs both a qualitative analysis by means of entity-relation reasoning and a quantitative one by means of neural features extracted from pre-trained neural networks. Although based on a proof-of-concept implementation, our experiments performed on public datasets showed promising results, and allowed us to draw some initial conclusions: *(i)* the hybridization between temporal decision trees and neural networks seems quite natural; *(ii)* the obtained method offers a statistically significant improvement in performances over its constituents; *(iii)* such an improvement seems to be higher in more complicated problems.

As future work, we plan to perform a more systematic experimental benchmark, explore different neural network architectures, which have been successfully applied to the problem of multivariate time series classification, and investigate higher context sizes. More importantly, we plan to explore all different hybridization schemata, and compare them against each other and against the standard approach. Finally, a similar idea can be pursued in the spatial case, where neural networks have shown even more predictive ability than in the temporal one.

References

1 S. Alaniz, D. Marcos, B. Schiele, and Z. Akata. Learning decision trees recurrently through communication. In *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13518–13527, 2021.

2 L. Atlas, R. Cole, Y. Muthusamy, A. Lippman, J. Connor, D. Park, M. El-Sharkawai, and R.J. Marks. A performance comparison of trained multilayer perceptrons and trained classification trees. *Proc. of the IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 78(10):1614–1619, 1990.

3 A. J. Bagnall, J. Lines, A. Bostrom, J. Large, and E. J. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

4 D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2014. `doi:10.48550/arXiv.1409.0473`.

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

6 R. P. Brent. Fast training algorithms for multilayer neural nets. *IEEE Transactions on Neural Networks*, 2(3):346–354, 1991.

7 M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In *Proc. of the 8th Advances in Neural Information Processing Systems (NIPS)*, pages 24–30, 1995. URL: `http://papers.nips.cc/paper/1152-extracting-tree-structured-representations-of-trained-networks`.

8 D. Dancey, D. McLean, and Z. Bandar. Decision tree extraction from trained neural networks. In *Proc. of the 7th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 515–519, 2004.

9 A. S. d'Avila Garcez, M. Gori, L. C. Lamb, L. Serafini, M. Spranger, and S. N. Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4):611–632, 2019.

10 A. S. d'Avila Garcez, L. C. Lamb, and D. M. Gabbay. *Neural-Symbolic Cognitive Reasoning*. Cognitive Technologies. Springer, 2009.

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

12 H. Guo and S. B. Gelfand. Classification trees with neural network feature extraction. *IEEE Transactions on Neural Networks*, 3(6):923–933, 1992.

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

14 G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015. `doi:10.48550/arXiv.1503.02531`.

15 I. Ivanova and M. Kubat. Initialization of neural networks by means of decision trees. *Knowledge-Based Systems*, 8(6):333–344, 1995.

16 Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time, 2019. `doi:10.48550/arXiv.1907.05321`.

17 P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulò. Deep neural decision forests. In *Proc. of the International Conference on Computer Vision (ICCV)*, pages 1467–1475, 2015.

18 R. Krishnan, G. Sivakumar, and P. Bhattacharya. Extracting decision trees from trained neural networks. *Pattern Recognition*, 32(12):1999–2009, 1999.

19 M. Kubat. Decision trees can initialize radial-basis function networks. *IEEE Transactions on Neural Networks*, 9(5):813–821, 1998.

20 M. Längkvist, L. Karlsson, and A. Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

21 T. Li, L. Fang, and A. Jennings. Structurally adaptive self-organizing neural trees. In *Proc. of the International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 329–334, 1992.

22 F. Manzella, G. Pagliarini, G. Sciavicco, and I. E. Stan. Interval temporal random forests with an application to COVID-19 diagnosis. In *Proc. 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.

23 C. Micheloni, A. Rani, S. Kumar, and G. L. Foresti. A balanced neural tree for pattern classification. *Neural Networks*, 27:81–90, 2012.

24 M. Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, 12(2):34–51, 1991.

25 C. Murdock, Z. Li, H. Zhou, and T. Duerig. Blockout: Dynamic model selection for hierarchical deep networks. In *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2583–2591, 2016.

26 V. N. Murthy, V. Singh, T. Chen, R. Manmatha, and D. Comaniciu. Deep decision network for multi-class image classification. In *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2240–2248, 2016.

27 G. Pagliarini, F. Manzella, G. Sciavicco, and I. E. Stan. ModalDecisionTrees.jl: Interpretable models for native time-series & image classification, 2021. `doi:10.5281/zenodo.7040419`.

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

29 A. Pasos Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. J. Bagnall. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449, 2021.

30 G. P. J. Schmitz, C. Aldrich, and F. S. Gouws. ANN-DT: An algorithm for extraction of decision trees from artificial neural networks. *IEEE Transactions on Neural Networks*, 10(6):1392–1401, 1999.

31 G. Sciavicco and I. E. Stan. Knowledge extraction with interval temporal logic decision trees. In *Proc. of the 27th International Symposium on Temporal Representation and Reasoning (TIME)*, volume 178 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

32 I. K. Sethi. Entropy nets: From decision trees to neural networks. *Proc. of the IEEE*, 78(10):1605–1613, 1990.

33 R. Setiono and W. K. Leow. On mapping decision trees and neural networks. *Knowledge-Based Systems*, 12(3):95–99, 1999.

34 R. Setiono and H. Liu. A connectionist approach to generating oblique decision trees. *IEEE Transactions on Systems, Man and Cybernetics – Part B*, 29(3):440–444, 1999.

35 J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–143, 1991.

36 N. Srivastava and R. Salakhutdinov. Discriminative transfer learning with tree-based priors. In *Proc. of the 26th Advances In Neural Information Processing Systems (NIPS)*, pages 2094–2102, 2013.

37 I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proc. of the 27th Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014.

38 G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

39 A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. of the 30th Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, 2017.

40 A. Wan, L. Dunlap, D. Ho, J. Yin, S. Lee, S. Petryk, S. Adel Bargal, and J. E. Gonzalez. NBDT: Neural-Backed Decision Tree. In *Proc. of the 9th International Conference on Learning Representations (ICLR)*, 2021.

41 Z.-H. Zhou and Z. Chen. Hybrid decision tree. *Knowledge-Based Systems*, 15(8):515–528, 2002.

42 Z.-H. Zhou and Y. Jiang. NeC4.5: Neural Ensemble Based C4.5. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):770–773, 2004.

Taming Strategy Logic: Non-Recurrent Fragments

Massimo Benerecetti
University of Napoli "Federico II", Italy

Fabio Mogavero
University of Napoli "Federico II", Italy

Adriano Peron
University of Napoli "Federico II", Italy

Abstract

Strategy Logic (SL for short) is one of the prominent languages for reasoning about the strategic abilities of agents in a multi-agent setting. This logic extends LTL with first-order quantifiers over the agent strategies and encompasses other formalisms, such as ATL* and CTL*. The *model-checking problem* for SL and several of its fragments have been extensively studied. On the other hand, the picture is much less clear on the satisfiability front, where the problem is undecidable for the full logic. In this work, we study two fragments of *One-Goal* SL, where the nesting of sentences within temporal operators is constrained. We show that the *satisfiability problem* for these logics, and for the corresponding fragments of ATL* and CTL*, is ExpSpace and PSpace-complete, respectively.

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

Keywords and phrases Strategic Reasoning, Multi-Agent Systems, Temporal Logics, Satisfiability

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.14

Funding Partially supported by the GNCS 2020 project "Ragionamento Strategico e Sintesi Automatica di Sistemi Multi-Agente".

1 Introduction

A number of extensions of temporal logics specifically tailored to open multi-agent systems and incorporating, implicitly or explicitly, the notion of strategy as a central element, have been proposed in the literature that can also express interesting game-theoretic notions, such as various forms of equilibria in games [22, 5, 24, 25, 6, 26]. *Alternating-Time Temporal Logic* (ATL*, for short) was originally introduced by Alur, Henzinger, and Kupferman [2] and allows for reasoning about *strategic behaviour* of agents with temporal goals. This logic generalises the branching-time temporal logic CTL* [17, 18] by replacing the path quantifiers, *there exists* "E" and *for all* "A", with *strategic modalities* of the form "$\langle\langle A \rangle\rangle$" and "$[[A]]$", for a set A of agents. These modalities can express cooperation and competition among the agents involved towards achieving some required temporal goals. In particular, they allow for selective quantifications over the paths resulting from an infinite game between a coalition of agents and its adversary, the complement coalition. *Strategy Logic* (SL, for short) [10, 34, 11, 32, 33], instead, extends LTL by means of two *strategy quantifiers*, the existential $\exists x$ and the universal $\forall x$, as well as agent bindings (a, x), where a is an agent and x a strategy variable. Intuitively, these elements can be respectively read as *"there exists a strategy x"*, *"for all strategies x"*, and *"bind agent a to the strategy associated with x"*. SL considers strategies as first-class citizens and can express properties requiring an arbitrary alternation of the strategic quantifiers, as opposed to, *e.g.*, ATL*, which only allows for at most one such alternation. From a semantic viewpoint, this entails that SL can encode arbitrary functional dependencies among strategies, which may be crucial to express relevant multi-agent systems and non-trivial game-theoretic notions (see [32, 33]).

The *model-checking problem* for SL and for many of its fragments has been studied with some depth and is relatively well-understood [32, 7, 8, 20, 21]. The picture is, however, much less clear when *satisfiability* is considered. The full logic SL is known to be undecidable [34]. The *one-goal* fragment (SL[1G], for short), where only a single binding prefix is allowed in any sentence, is decidable in 2ExpTime [31]. On the other hand, the *Boolean-Goal* fragment, which allows for Boolean combinations of bindings within a sentence but no nesting of bindings, is already undecidable [33]. Recently, the *flat* fragment of *conjunctive-goal* SL has been studied in [1], which provides a PSpace-complete result for the problem, witnessing the quite rare phenomenon of a language with a satisfiability problem easier than the corresponding model-checking one, which remains 2ExpTime-complete. Such fragment allows for conjunctions of bindings but no nesting of temporal operators within a sentence.

In this work, we widen the picture, by studying larger non-flat fragments of SL[1G]. Specifically, we allow some forms of nesting of temporal operators, but prevent sentences in the first (*resp.*, second) argument of an until (*resp.*, release) operator. Essentially, temporal operators cannot reiterate the request of satisfaction of a sentence arbitrarily many times. The resulting fragment is, thus, called *non-recurrent* SL[1G] ($\text{SL}^{\varnothing}$[1G], for short). We show that the fragment where the first (*resp.*, second) argument of an until (*resp.*, release) is restricted to a pure LTL formula can be decided in ExpSpace. If we further restrict those arguments to Boolean formulae, instead, we obtain a weaker fragment ($\text{WSL}^{\varnothing}$[1G], for short) with a PSpace-complete decision problem. To prove these results, we first introduce a normal form for the models of satisfiable sentences of these fragments. The distinctive property of such models is that, along any of their paths, the number of branching points is linear in the length of the formula. To do that, a sentence is converted into a "skeleton", where it is split into layers at the beginning of each block of strategy quantifiers, and then Skolemized to obtain a set of purely universally-quantified formulas in order to apply techniques from first-order logic [35, 9]. Then, we introduce a novel class of tree automata, called *bounded-fork automata*, accepting trees with bounded-branching. We show that the emptiness problem for these automata, unlike for classic tree automata, can be decided in LogSpace. These results are key to obtaining the complexity bounds. Indeed, we can show that for any sentence φ of the two considered fragments, we can build a bounded-fork automaton of size doubly-exponential (*resp.*, singly-exponential) in the length of φ, accepting all and only its normal models. The ExpSpace and PSpace upper bounds for satisfiability, then, immediately follow from the complexity of the emptiness problem. The results also trickle down to suitable fragments of sublogics of SL such as ATL, ATL*, CTL, and CTL*.

Restrictions similar in vein to the non-recurrent one we study here have been considered in the past for LTL, CTL, and CTL*. In [13] the author introduces *flat*LTL, *flat*CTL, and *flat*CTL*, as fragments of the corresponding temporal logics where the next operator is not allowed and the first argument of both the until and the release operators can only accommodate propositional formulae. In their LTL form, these restrictions have been applied in several contexts, such as temporal logics enriched with constraints over data [12, 14], analysis of discrete pushdown timed systems [28], and the synthesis of hybrid systems [19]. In particular, the LTL fragment considered in [12, 28] is a sublogic of the linear-time logic underlying $\text{WSL}^{\varnothing}$[1G], while the one originally considered in [13] is not comparable to ours, as it restricts the first and not the second argument of the release operator, therefore still allowing for recurrent sentences. While both model-checking and satisfiability problems for flatLTL have been shown to be PSpace-complete [15, 38], to the best of our knowledge, only expressiveness properties have been studied for flatCTL and flatCTL*.

2 Preliminaries

Games. A *concurrent game structure* (CGS, for short) *w.r.t.* finite non-empty sets of *atomic propositions* AP and *agents* Ag is a tuple $\mathfrak{G} \triangleq \langle \mathrm{Ac}, \mathrm{Ps}, \tau, v_I, \lambda \rangle$, where Ac and Ps are countable non-empty sets of *actions* and *positions*, $v_I \in \mathrm{Ps}$ is an *initial position*, and $\lambda \colon \mathrm{Ps} \to 2^{\mathrm{AP}}$ is a *labelling function* mapping every position $v \in \mathrm{Ps}$ to the set of atomic propositions $\lambda(v) \subseteq \mathrm{AP}$ true at that position. A *decision* $d \in \mathrm{Dc} \triangleq \mathrm{Ac}^{\mathrm{Ag}}$ is a function that chooses an action for each agent. A *move function* $\tau \colon \mathrm{Ps} \times \mathrm{Dc} \to \mathrm{Ps}$ maps every position $v \in \mathrm{Ps}$ and decision $d \in \mathrm{Dc}$ to a position $\tau(v, d) \in \mathrm{Ps}$. By abuse of notation, $\tau \subseteq \mathrm{Ps} \times \mathrm{Ps}$ also denotes the *transition relation* between positions such that $(v, w) \in \tau$ *iff* $\tau(v, d) = w$, for some $d \in \mathrm{Dc}$. As usual, τ^{+} (*resp.*, τ^{*}) is the transitive (*resp.*, reflexive and transitive) closure of τ. A *path* $\pi \in \mathrm{Pth} \subseteq \mathrm{Ps}^{\infty} \triangleq \mathrm{Ps}^{*} \cup \mathrm{Ps}^{\omega}$ is a finite or infinite sequence of positions compatible with the move function, *i.e.*, $((\pi)_i, (\pi)_{i+1}) \in \tau$, for each $i \in [0, |\pi| - 1)$. The set $\mathrm{Pth}(v) \triangleq \{\pi \in \mathrm{Pth} \mid |\pi| > 0 \land \mathsf{fst}(\pi) = v\}$ denotes the set of paths starting at a position v. A *history* at v is a finite non-empty path $\rho \in \mathrm{Hst}(v) \triangleq \mathrm{Pth}(v) \cap \mathrm{Ps}^{+}$ starting at that position. Similarly, a *play* $\pi \in \mathrm{Play}(v) \triangleq \mathrm{Pth}(v) \cap \mathrm{Ps}^{\omega}$ at v is an infinite path starting at v. A *strategy* rooted at v is a function $\sigma \in \mathrm{Str}(v) \triangleq \mathrm{Hst}(v) \to \mathrm{Ac}$ mapping histories to actions. A v-rooted *profile* $\xi \in \mathrm{Prf}(v) \triangleq \mathrm{Ag} \to \mathrm{Str}(v)$ associates agents with strategies. A path $\pi \in \mathrm{Pth}(v)$ is *compatible* with a v-rooted profile $\xi \in \mathrm{Prf}(v)$ if, for each $i \in [0, |\pi| - 1)$, it holds that $(\pi)_{i+1} = \tau((\pi)_i, d)$, for the unique decision $d \in \mathrm{Dc}$ such that $d(a) = \xi(a)((\pi)_{\leq i})$, for all agents $a \in \mathrm{Ag}$. A CGS $\mathfrak{G}$ is a *tree* if, for some set X, *1)* Ps is a prefix-closed set of words in X^{*}, *2)* $v_I = \varepsilon$ is the empty word, and *3)* $(v, w) \in \tau$ *iff* $w = v \cdot x$, for all position $v, w \in \mathrm{Ps}$, for some $x \in \mathrm{X}$. As usual, $\tau^{-1} \colon \mathrm{Ps} \setminus \varepsilon \to \mathrm{Ps}$ denotes the *predecessor* function $\tau^{-1}(v \cdot x) \triangleq v$, for all $v \cdot x \in \mathrm{Ps} \setminus \varepsilon$ with $x \in \mathrm{X}$. Finally, a tree CGS $\mathfrak{G}$ is k-*fork*, for some $k \in \mathbb{N}$, if along every path $\pi \in \mathrm{Pth}(v_I)$ there are at most k forks, namely, $|\{i \in \mathbb{N} \mid |\tau((\pi)_i)| > 1\}| \leq k$.

Functions. A *function signature* is a tuple $\mathcal{F} \triangleq \langle \mathrm{Fn}, \mathsf{ar} \rangle$, where Fn is a set of *function symbols* and $\mathsf{ar} \colon \mathrm{Fn} \to \mathbb{N}$ is an *arity function* mapping each symbol $f \in \mathrm{Fn}$ to its arity $\mathsf{ar}(f) \in \mathbb{N}$. An $\mathcal{F}$-structure $\mathfrak{F} \triangleq \langle \mathrm{D}, \cdot^{\mathfrak{F}} \rangle$ is defined by a *domain* D together with an *interpretation* of Fn over D, *i.e.*, every function symbol $f \in \mathrm{Fn}$ is interpreted in a function $f^{\mathfrak{F}} \colon \mathrm{D}^{\mathsf{ar}(f)} \to \mathrm{D}$. The set of terms built over the signature $\mathcal{F}$ and a set of variables Vr is denoted by Tr. A *substitution* is a map $\mu \colon \mathrm{Vr} \to \mathrm{Tr}$ assigning a term to each variable; a *valuation w.r.t.* $\mathfrak{F}$ is a map $\xi \colon \mathrm{Vr} \to \mathrm{D}$ assigning an element of the domain to each variable. Given a term $t \in \mathrm{Tr}$, by t^{μ} we denote the *replacement* of all variables in t with the terms prescribed by the substitution μ; by $t^{\mathfrak{F}, \xi}$ we denote the interpretation of t in $\mathfrak{F}$ under the valuation ξ, *i.e.*, the value assumed by t when each variable x is replaced with the value $\xi(x)$. A set of terms $\mathrm{T} \subseteq \mathrm{Tr}$ unifies if there is a substitution μ such that $t_1^{\mu} = t_2^{\mu}$, for all $t_1, t_2 \in \mathrm{T}$. Similarly, T equalises over $\mathfrak{F}$ if there is a valuation ξ such that $t_1^{\mathfrak{F}, \xi} = t_2^{\mathfrak{F}, \xi}$, for all $t_1, t_2 \in \mathrm{T}$. For more details, we refer to [4, 9].

Automata. A *deterministic* (*resp.*, *nondeterministic*) *word automaton* (DWA (*resp.*, NWA), for short) is a tuple $\langle \Sigma, \mathrm{Q}, \delta, q_I, \mathrm{Q_F} \rangle$, where Σ and Q are the finite non-empty sets of *input symbols* and *states*, $q_I \in \mathrm{Q}$ is the *initial state*, $\mathrm{Q_F} \subseteq \mathrm{Q}$ is the subset of *final states*, and $\delta \colon \mathrm{Q} \times \Sigma \to \mathrm{Q} \cup \{\bot, \top\}$ (*resp.*, $\delta \colon \mathrm{Q} \times \Sigma \to 2^{\mathrm{Q}}$) is the *deterministic* (*resp.*, *nondeterministic*) *transition function* mapping each state $q \in \mathrm{Q}$ and input symbol $\sigma \in \Sigma$ to the successor state (*resp.*, set of successor states) $\delta(q, \sigma)$. A *deterministic* (*resp.*, *nondeterministic*) *tree automaton* (DTA (*resp.*, NTA), for short) is a tuple $\langle \Sigma, \Lambda, \mathrm{Q}, \delta, q_I, \mathrm{Q_F} \rangle$, where all components but Λ and δ are defined as for a word automaton, $\Lambda \subseteq \mathbb{N}_{+}$ is the non-empty set of *node degrees*, and $\delta \colon \mathrm{Q} \times \Sigma \times \Lambda \to \mathrm{Q}^{*} \cup \{\bot, \top\}$ (*resp.*, $\delta \colon \mathrm{Q} \times \Sigma \times \Lambda \to 2^{\mathrm{Q}^{*}}$) is the *deterministic* (*resp.*, *nondeterministic*) *transition function* mapping each state $q \in \mathrm{Q}$, input symbol $\sigma \in \Sigma$, and node degree $d \in \Lambda$ to the tuple of successor states $\delta(q, \sigma, d) \in \mathrm{Q}^{d}$ (*resp.*, set of tuples of successor states $\delta(q, \sigma, d) \subseteq \mathrm{Q}^{d}$), where $\bot$ and $\top$ are two implicit distinguished rejecting and accepting states used to simplify the constructions of this work (these implicit states are not needed in the case of nondeterministic automata). We only consider the Büchi acceptance condition, for both word and tree automata. The notions of *(accepting) run* and *recognised language* are the standard ones. For more details, we refer to [29, 23].

3 Decidable Fragments of Strategy Logic

Strategy Logic [10, 34] extends LTL by allowing to *quantify* over strategies and to assign a strategy to each agent, by *binding* the latter with some quantified variable. A *quantifier prefix* is a finite sequence $\wp \in \mathrm{Qn} \subseteq \{\exists x, \forall x \mid x \in \mathrm{Vr}\}^{*}$ of existential and universal quantifiers $\mathtt{Qn}x$, in which variables $x \in \mathrm{Vr}$ occur at most once. Similarly, a *binding prefix* is a finite sequence $\flat \in \mathrm{Bn} \subseteq (\mathrm{Ag} \times \mathrm{Vr})^{|\mathrm{Ag}|}$ of bindings (a, x), in which each agent $a \in \mathrm{Ag}$ occurs exactly once. By $\mathsf{vr}(\wp), \mathsf{vr}(\flat) \subseteq \mathrm{Vr}$ we denote the sets of variables occurring in $\wp$ and $\flat$.

One-Goal Strategy Logic. *One-Goal Strategy Logic* is one of the largest decidable fragments of SL known to date and is complete for 2ExpTime. Its main constraint *w.r.t.* full SL is that bindings are tightly connected to quantifiers and agents cannot change strategies within the same sentence without quantifying on them again in a nested subsentence.

▶ **Definition 1** (SL[1G] Syntax [31]). SL[1G] formulas *are generated from the sets of atomic propositions* AP, *quantifier prefixes* Qn, *and binding prefixes* Bn *via the following grammar, where* $p \in \mathrm{AP}$, $\wp \in \mathrm{Qn}$, *and* $\flat \in \mathrm{Bn}$, *with* $\mathsf{vr}(\wp) = \mathsf{vr}(\flat)$*:*

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \wp\flat\psi; \qquad \psi := \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathtt{X}\,\psi \mid \psi\,\mathtt{U}\,\psi \mid \psi\,\mathtt{R}\,\psi.$$

SL[1G] *denotes the set of sentences generated by Rule* φ*, while* FSL[1G] $\subset$ SL[1G] *represents the* flat fragment, i.e., *the subset generated by the variant of the grammar where* p *replaces the call to Rule* φ *within Rule* ψ, i.e., *with* ψ *pure* LTL*.*

With $\mathsf{ap}(\varphi) \subseteq \mathrm{AP}$, $\mathsf{vr}(\varphi) \subseteq \mathrm{Vr}$, and $\mathsf{free}(\varphi) \subseteq \mathrm{Vr} \cup \mathrm{Ag}$ we denote, respectively, the sets of *atomic propositions*, *variables*, and *free variables* and *agents* occurring φ. Being a first order language, the semantics of SL formulae is defined *w.r.t.* an assignment, interpreting variables as strategies. This interpretation is extended to agents as well, to take care of bindings assigning strategies to agents. Let $\mathrm{Asg}(v) \triangleq (\mathrm{Vr} \cup \mathrm{Ag}) \rightharpoonup \mathrm{Str}(v)$ denote the set of such assignments. For a set $\mathrm{V} \subseteq (\mathrm{Vr} \cup \mathrm{Ag})$, we also provide, for convenience, the set of assignments defined only over V, *i.e.*, $\mathrm{Asg}(v, \mathrm{V}) \triangleq \{\chi \in \mathrm{Asg}(v) \mid \mathsf{dom}(\chi) = \mathrm{V}\}$, and those defined at least over V as $\mathrm{Asg}_{\subseteq}(v, \mathrm{V}) \triangleq \{\chi \in \mathrm{Asg}(v) \mid \mathrm{V} \subseteq \mathsf{dom}(\chi)\}$. As usual, given an assignment χ, a variable or agent $x \in (\mathrm{Vr} \cup \mathrm{Ag})$ and a strategy $\sigma \in \mathrm{Str}$, we denote with $\chi[x \mapsto \sigma]$, the assignment χ' resulting from assigning σ to x in χ. Since the semantics for the Boolean and temporal operators is practically the classic one (see [32]), we only provide the interpretation of quantifiers and bindings.

▶ **Definition 2** (SL Semantics [32]). *Given a* CGS $\mathfrak{G}$*, for all* SL *formulas* φ*, positions* $v \in \mathrm{Ps}$*, and* v*-rooted assignments* $\chi \in \mathrm{Asg}_{\subseteq}(v, \mathsf{free}(\varphi))$*, the modelling relation* $\mathfrak{G}, v, \chi \models \varphi$ *is inductively defined as follows.*

1. *Atomic propositions, Boolean connectives and temporal operators are interpreted as usual.*
2. *For all* $x \in \mathrm{Vr}$*:*
   - **a.** $\mathfrak{G}, v, \chi \models \exists x.\, \phi$*, if* $\mathfrak{G}, v, \chi[x \mapsto \sigma] \models \phi$*, for some strategy* $\sigma \in \mathrm{Str}(v)$*;*
   - **b.** $\mathfrak{G}, v, \chi \models \forall x.\, \phi$*, if* $\mathfrak{G}, v, \chi[x \mapsto \sigma] \models \phi$*, for all strategies* $\sigma \in \mathrm{Str}(v)$*.*
3. *For all* $a \in \mathrm{Ag}$ *and* $x \in \mathrm{Vr}$*:* $\mathfrak{G}, v, \chi \models (a, x)\phi$*, if* $\mathfrak{G}, v, \chi[a \mapsto \chi(x)] \models \phi$*.*

For a sentence φ, we write $\mathfrak{G}, v \models \varphi$ and $\mathfrak{G} \models \varphi$ instead of $\mathfrak{G}, v, \varnothing \models \varphi$ and $\mathfrak{G}, v_I, \varnothing \models \varphi$.

The existence of a normal model for sentences, as defined in the next section, relies on the notion of *skeleton* that breaks down their nesting structure. The idea is that a skeleton decomposes a sentence φ into a set Φ of simpler sentences of the flat fragment. Essentially, φ is stratified into layers, whose sentences cannot occur within temporal operators. The connection between the layers is achieved by means of auxiliary atomic propositions, used as names of subsentences nested within temporal operators in the original formula. The skeleton is a reminiscent of the technique used in the model-checking algorithms for CTL* [30, 3].

For example, the following sentence $\varphi \triangleq p \wedge \forall x \exists y \forall z (a,x)(b,y)(c,z)(\mathtt{X}\,(q \wedge (\mathtt{X}\,\mathtt{F}\,q)\,\mathtt{U}\,(\phi_1 \wedge \phi_2)))$ with $\phi_1 \triangleq \exists x \forall y (a,x)(b,y)(c,y)(p\,\mathtt{U}\,q)$ and $\phi_2 \triangleq \forall x \exists y (a,y)(b,x)(c,y)(\mathtt{G}\,\neg q)$ can be stratified into 2 layers, using the fresh atomic propositions $\{s, s_1, s_2\}$ as names for the subsentences of φ: $\phi_1 \mapsto s_1$, $\phi_2 \mapsto s_2$ and $\forall x \exists y \forall z (a,x)(b,y)(c,z)(\mathtt{X}\,(q \wedge (\mathtt{X}\,\mathtt{F}\,q)\,\mathtt{U}\,(s_1 \wedge s_2))) \mapsto s$. In the end, the original formula φ is summarised by the positive Boolean formula $\zeta \triangleq p \wedge s$. This idea is formalised by the following definition, where the relation $\prec$ encodes the ordering among the layers and the function ℓ assigns atomic propositions as names of subsentences of φ. We shall denote with BF (*resp.*, BF$^+$) the set of Boolean (*resp.*, positive Boolean) formulae over AP.

▶ **Definition 3** (SL[1G] Skeleton). *An* SL[1G] skeleton *is a tuple* $\eth \triangleq \langle \zeta, \Phi, \ell \rangle$*, where* $\zeta \in \mathrm{BF}^+$ *is a positive Boolean formula,* $\Phi \subseteq$ FSL[1G] *is a finite set of* FSL[1G] *sentences, and* $\ell \colon \Phi \to \mathrm{AP}$ *is an injective function mapping each sentence* $\phi \in \Phi$ *to an atomic proposition* $\ell(\phi) \in \mathrm{AP}$*, for which there is a strict partial order* $\prec \subseteq \Phi \times \Phi$ *such that if* $\ell(\phi) \in \mathsf{ap}(\phi')$ *then* $\phi \prec \phi'$*, for all* $\phi' \in \Phi$*. In addition:* $\eth$ *is* simple *if every atomic proposition* $p \in \mathsf{img}(\ell)$ *occurs in exactly one sentence* $\phi \in \Phi \cup \{\zeta\}$ *and at most once in it;* $\eth$ *is* principal *if it is simple and all sentences* ϕ *in* Φ *have the form* $\wp\flat\psi$*, for some* $\wp \in \mathrm{Qn}$*,* $\flat \in \mathrm{Bn}$*, and* $\psi \in \mathrm{LTL}$*.*

For instance, the skeleton of the example above is indeed a principal one. For a skeleton $\eth$, we denote with $\varphi_\eth$ the sentence derived from ζ by iteratively replacing each atomic proposition $p \in \mathsf{img}(\ell)$ with the corresponding FSL[1G] sentence $\ell^{-1}(p)$ until no atomic proposition in $\mathsf{img}(\ell)$ occurs in the sentence. This effectively reverts the stratification process described above. Note that the strict partial order $\prec$ on Φ ensures termination of the rewriting procedure. While, for convenience, we allow for more liberal forms of skeletons, principal ones suffice, as one such skeleton exists for each sentence, where different occurrences of the same subsentence are mapped to different names by ℓ.

▶ **Proposition 4.** *Each* SL[1G] *sentence* φ *enjoys a principal* SL[1G] *skeleton* $\eth$ *with* $\varphi = \varphi_\eth$*.*

Satisfaction of a skeleton $\eth$ by a CGS $\mathfrak{G}$ over the atomic propositions of $\eth$ is defined quite naturally. Specifically, the initial position of $\mathfrak{G}$ must satisfy locally the Boolean formula ζ, and any sentence in Φ, whose "name" labels a given position v of $\mathfrak{G}$, must be satisfied at v.

▶ **Definition 5** (Skeleton Satisfaction). *A* CGS $\mathfrak{G}$ *satisfies an* SL[1G] *skeleton* $\eth$*, in symbols* $\mathfrak{G} \models \eth$*, if* 1) $\lambda(v_I) \models \zeta$ *and* 2) $\mathfrak{G}, v \models \phi$*, for all* $\phi \in \Phi$ *and* $v \in \mathrm{Ps}$ *with* $\ell(\phi) \in \lambda(v)$*.*

The following result establishes the equisatisfiability of SL[1G] skeletons and their corresponding SL[1G] sentences.

▶ **Theorem 6.** $\varphi_\eth$ *is satisfiable* iff $\eth$ *is satisfied by a tree* CGS*, for every* SL[1G] *skeleton* $\eth$*.*

Non-Recurrent One-Goal Strategy Logics. The main source of complexity for SL[1G], or CTL* and ATL* for that matter, resides in its ability to express properties that request satisfaction of a given sentence an unbounded number of times along a computation, as, *e.g.*, in the CTL formula $\mathtt{EG}\,(\neg p \wedge \mathtt{EX}\,p)$. Given the branching nature of quantifications in SL, this may lead to models with an unbounded number of branching points. In general, such models can be recognised by nondeterministic tree automata with a double exponential number of states [37, 31]. Emptiness for such tree automata is known to be PTime [42], which leads to a 2ExpTime procedure for deciding SL[1G]. Since CTL* is contained in SL[1G], completeness for 2ExpTime follows [16]. To avoid this issue, we restrict the number of times a given sentence can be requested, by preventing sentences in the left-hand (*resp.*, right-hand) argument of the until (*resp.*, release) operator. We call the resulting fragment *non-recurrent*, in that it forbids an unbounded number of requests of the same sentence.

▶ **Definition 7** (SL[1G] Fragments). *Formulas of* non-recurrent fragments *of* SL[1G] *are generated from the sets of atomic propositions* AP*, quantifier prefixes* Qn*, and binding prefixes* Bn *via the following grammar, with* $p \in \mathrm{AP}$*,* $\psi \in \mathrm{LTL}(\mathrm{AP})$*,* $\beta \in \mathrm{BF}(\mathrm{AP})$*,* $\wp \in \mathrm{Qn}$*, and* $\flat \in \mathrm{Bn}$ *such that* $\mathsf{vr}(\wp) = \mathsf{vr}(\flat)$*:*

$$\begin{array}{lll} \mathrm{SL}^{\varnothing}[1\mathrm{G}]: & \varphi := p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \wp\flat\eta \mid \wp\flat\psi; & \eta := \varphi \mid \eta \wedge \eta \mid \eta \vee \eta \mid \psi\,\mathtt{U}\,\varphi \mid \varphi\,\mathtt{R}\,\psi \mid \mathtt{X}\eta \mid \mathtt{X}\psi; \\ \mathrm{WSL}^{\varnothing}[1\mathrm{G}]: & \varphi := p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \wp\flat\eta; & \eta := \varphi \mid \eta \wedge \eta \mid \eta \vee \eta \mid \beta\,\mathtt{U}\,\varphi \mid \varphi\,\mathtt{R}\,\beta \mid \mathtt{X}\eta; \end{array}$$

For each fragment, the rule φ takes care of the first-order (branching) structure of the language, while the rule η handles the temporal portion. The non-recurrence constraint is embedded in the cases for the until and release operators within the rule η, restricting the left-hand (*resp.*, right-hand) argument of the until (*resp.*, release) operators to be a pure LTL formula with no nesting of sentences. The weak fragment further restricts those arguments so that no temporal operators can occur altogether, *i.e.*, they can only accommodate Boolean formulae. No restriction is imposed on the next operator, while negation can only be applied to atomic propositions in AP. By replacing all the occurrences of φ in the two rules η with a positive Boolean formula $\gamma \in \mathrm{BF}^{+}(\mathrm{AP} \cup \mathrm{A})$, for a (possibly empty) distinguished set A of atomic propositions, such that $\mathrm{AP} \cap \mathrm{A} = \emptyset$, we obtain the corresponding flat fragments $\mathrm{FSL}^{\varnothing}_{\mathrm{A}}[1\mathrm{G}]$ and $\mathrm{FWSL}^{\varnothing}_{\mathrm{A}}[1\mathrm{G}]$. The idea is that A contains names of sentences possibly used by the skeletons for the two fragments. In addition, we call *Weak* LTL (WLTL for short), the fragment of LTL that agrees with the rule η of $\mathrm{FWSL}^{\varnothing}[1\mathrm{G}]$.

We can obtain skeletons for the new fragments, by suitably restricting their components to the corresponding flat fragments and requiring that only fresh atomic propositions in A be used as names for sentences. An $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ (*resp.*, $\mathrm{WSL}^{\varnothing}[1\mathrm{G}]$) skeleton $\eth = \langle \zeta, \Phi, \ell \rangle$ is a principal SL[1G] skeleton such that *1)* $\Phi \subseteq \mathrm{FSL}^{\varnothing}_{\mathrm{A}}[1\mathrm{G}]$ (*resp.*, $\Phi \subseteq \mathrm{FWSL}^{\varnothing}_{\mathrm{A}}[1\mathrm{G}]$), and *2)* $\mathsf{img}(\ell) \cap \mathrm{A} = \emptyset$, for some $\mathrm{A} \subseteq \mathrm{AP}$. The analogous of Proposition 4 holds for the two new fragments $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ and $\mathrm{WSL}^{\varnothing}[1\mathrm{G}]$ as well.

▶ **Proposition 8.** *Each* $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ *(*resp.*,* $\mathrm{WSL}^{\varnothing}[1\mathrm{G}]$*) sentence* φ *enjoys an* $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ *(*resp.*,* $\mathrm{WSL}^{\varnothing}[1\mathrm{G}]$*) skeleton* $\eth$ *with* $\varphi = \varphi_{\eth}$*.*

The constraint on the non-recurrence of sentences allows us to strengthen Theorem 6 and show that a sentence is satisfiable *iff* its skeleton can be satisfied by a model where each subsentence is requested at most once. This property is formalised by the definition of *single-time satisfaction* and the following theorem. The result is instrumental to the definition of normal models (see next section) and, ultimately, to the main complexity results.

▶ **Definition 9** (Single-Time Skeleton Satisfaction). *A* CGS $\mathfrak{G}$ single-time satisfies *a skeleton* $\eth$ *if* 1) $\mathfrak{G} \models \eth$ *and* 2) *if* $\ell(\phi) \in \lambda(v)$ *then* $\ell(\phi) \notin \lambda(w)$*, for all* $\phi \in \Phi$*,* $v \in \mathrm{Ps}$*, and* $w \in \tau^{+}(v)$*.*

▶ **Theorem 10.** $\varphi_{\eth}$ *is satisfiable* iff $\eth$ *is single-time satisfied by a tree* CGS*, for every* $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ *skeleton* $\eth$*.*

Assume $\varphi_{\mathfrak{F}}$ is satisfiable. By Theorem 6, $\mathfrak{F}$ is satisfiable as well. Thus, let $\mathfrak{G}$ be one of the tree CGSs satisfying $\mathfrak{F}$. The proof proceeds by induction on the depth k of the strict partial order $\prec$ underlying the skeleton $\mathfrak{F}$.

The idea is the following: starting from $\mathfrak{G}$, we perform a sequence $\mathfrak{G}_{k+1}, \mathfrak{G}_k, \mathfrak{G}_{k-1}, \ldots, \mathfrak{G}_0$ of structure-preserving model transformations, where $\mathfrak{G}_{k+1} \triangleq \mathfrak{G}$. The labelling of the models is modified in such a way that all sentences in Φ at level i in the ordering $\prec$ are single-time satisfied in $\mathfrak{G}_j$, for every $j \leq i$. More specifically, sentences at level k only need to be verified at the root, while those at $i < k$ just need to be checked at the first occurrence of a witness of the until/release operator containing it. Hence, the labelling of $\mathfrak{G}_i$ is obtained from $\mathfrak{G}_{i+1}$, by removing, along each path, every occurrence of the name of a sentence at level i except the one that serves as witness of the corresponding until/release operator. By construction, the name $\ell(\phi)$ of every sentence ϕ in Φ occurs only once along any path of $\mathfrak{G}_0$. Hence, $\mathfrak{G}_0$ single-time satisfies $\mathfrak{F}$.

4 Normal Models

The efficient satisfiability of the non-recurrent fragments relies on the fact that any of their sentences is satisfiable by models of a specific structure, namely, by bounded-fork tree CGS. This can be proven by first extending SL with function symbols, to allow for bindings containing strategy terms, instead of simple variables, which enables us to state a Skolem normal-form theorem for SL[1G]. This result can be used to show that any model for a sentence φ of $\text{SL}^{\varnothing}[1\text{G}]$ in Skolem form can be transformed into a bounded-fork tree satisfying φ, where forks only occur as a result of non-unifying strategy terms within the bindings of φ.

Functions in SL. Given a function signature $\mathcal{F}$, by SL[1G,$\mathcal{F}$] we denote the extension of SL[1G], where we allow agents to be bound with complex terms instead of simple variables. This means that the set of bindings Bn in the syntax gets replaced by its extension $\text{Bn}(\mathcal{F}) \subseteq (\text{Ag} \times \text{Tr})^{|\text{Ag}|}$. A binding prefix is, thus, a finite sequence $\flat \in \text{Bn}(\mathcal{F})$ of bindings (a, t), with $t \in \text{Tr}$, in which each agent $a \in \text{Ag}$ occurs exactly once. $\forall$SL[1G,$\mathcal{F}$] represents the universal fragment of SL[1G,$\mathcal{F}$], where existential quantifiers are forbidden. In order to define the semantics of an SL[1G,$\mathcal{F}$] sentence, we need to provide a *strategy interpretation* for all function symbols in Fn. We do this, via the map $\Im \colon v \in \text{Ps} \mapsto \mathfrak{F}_v$ assigning to each position v an $\mathcal{F}$-structure $\mathfrak{F}_v = \langle \text{Str}(v), \cdot^{\mathfrak{F}_v} \rangle$ whose domain is the set of strategies rooted at v. Given a pair $(\mathfrak{G}, \Im)$ of a CGS $\mathfrak{G}$ and a strategy interpretation $\Im$, called *interpreted* CGS, we can define the modelling relation $(\mathfrak{G}, \Im), v, \chi \models \varphi$ as in Definition 2, where Item 3 gets replaced by the following one:

- for all $a \in \text{Ag}$ and $t \in \text{Tr}$: $(\mathfrak{G}, \Im), v, \chi \models (a, t)\phi$, if $(\mathfrak{G}, \Im), v, \chi[a \mapsto t^{\Im(v),\chi}] \models \phi$,

where agent $a \in \text{Ag}$ is bound to strategy $t^{\Im(v),\chi}$, *i.e.*, the interpretation of term t under the v-rooted assignment $\chi \in \text{Asg}(v)$ in the $\mathcal{F}$-structure $\Im(v) = \langle \text{Str}(v), \cdot^{\Im(v)} \rangle$ associated with v. Intuitively, we assign to agent a a strategy dependent on those associated with the variables occurring in the term t.

An SL[1G,$\mathcal{F}$] sentence φ is *satisfied* by a CGS $\mathfrak{G}$, in symbols $\mathfrak{G} \models \varphi$, if there exists a strategy interpretation $\Im$ such that $(\mathfrak{G}, \Im) \models \varphi$, where the latter stands for $(\mathfrak{G}, \Im), v_I, \varnothing \models \varphi$. In the rest of the work, by $\mathsf{skm} \colon \text{SL}[1\text{G}] \to \forall\text{SL}[1\text{G},\mathcal{F}]$ we denote the function mapping each SL[1G] sentence φ to the corresponding *Skolem normal-form* $\mathsf{skm}(\varphi)$, where each variable x existentially quantified in a subsentence ϕ of φ is replaced by a fresh function symbol applied to the variables universally quantified in ϕ before x. As an example, consider the SL[1G] sentence φ used in the previous section to exemplify the notion of SL[1G] skeleton.

Then, $\mathsf{skm}(\varphi) = p \wedge \forall x \forall z(a,x)(b,f_1(x))(c,z)(\mathtt{X}\,(q \wedge (\mathtt{X}\,\mathtt{F}\,q)\,\mathtt{U}\,(\mathsf{skm}(\phi_1) \wedge \mathsf{skm}(\phi_2))))$, where $\mathsf{skm}(\phi_1) = \forall y(a,f_2)(b,y)(c,y)(p\,\mathtt{U}\,q)$ and $\mathsf{skm}(\phi_2) = \forall x(a,f_3(x))(b,x)(c,f_3(x))(\mathtt{G}\,\neg q)$. In φ, the existential variable y of the outermost sentence is replaced by the term $f_1(x)$, since the strategy chosen by agent b only depends on the strategy used by agent a. A similar reasoning applies to the subsentence ϕ_2. In ϕ_1, instead, the existential variable x is replaced by the constant f_2, as the strategy for agent a does not depend on those of b and c.

In [32] (see Theorem 4.5 and Corollary 4.6), it has been proved that SL enjoys a semantic version of *Skolem normal-form theorem*, where the interpretation of *Skolem functions* is given at the meta-level. Thanks to the introduction of function symbols in the syntax of the logic, this result can now be stated at the object-level in the classic way.

▶ **Theorem 11.** *Let $\mathfrak{G}$ be a* CGS*. An* SL[1G] *sentence φ is satisfied by $\mathfrak{G}$* iff *the* $\forall$SL[1G,$\mathcal{F}$] *sentence* $\mathsf{skm}(\varphi)$ *is satisfied by $\mathfrak{G}$.*

A fundamental property of SL[1G], which allows both its model-checking and satisfiability problem to be elementary decidable [32, 31, 33], is that every satisfiable sentence of this logic is *behaviourally satisfiable* [32] (see Theorem 4.20 and Corollary 4.21), with the intuitive meaning that each action chosen by an agent, for some history of a play, only depends on the actions chosen by the other agents along that history. In other words, an agent does not need to forecast the future to play optimally. At this point, we can formalise this intuition and restate the result proved in [32] as follows. We say that two strategies $\sigma_1, \sigma_2 \in \mathrm{Str}(v)$ are *equal along* history $\rho \in \mathrm{Hst}(v)$ (*ρ-equal*, for short), if, for every history $\rho' \in \mathrm{Hst}(v)$ with $\rho' \leq \rho$, it holds that $\sigma_1(\rho') = \sigma_2(\rho')$, where $\leq$ is the partial order induced by prefixes. This notion immediately lifts to vectors of strategies $\vec{\sigma}_1, \vec{\sigma}_2 \in \mathrm{Str}(v)^k$, of some $k \in \mathbb{N}$, as usual: $\vec{\sigma}_1$ and $\vec{\sigma}_2$ are *ρ-equal* if all their k components $(\vec{\sigma}_1)_i$ and $(\vec{\sigma}_2)_i$ are ρ-equal, with $i \in [k]$. A function between strategies $\mathsf{f}\colon \mathrm{Str}(v)^k \to \mathrm{Str}(v)$, for some dimension $k \in \mathbb{N}$, is *behavioural* if, for every history $\rho \in \mathrm{Hst}$ and pair of ρ-equal k-vectors of strategies $\vec{\sigma}_1, \vec{\sigma}_2 \in \mathrm{Str}(v)^k$, it holds that $\mathsf{f}(\vec{\sigma}_1)(\rho) = \mathsf{f}(\vec{\sigma}_2)(\rho)$. A strategy interpretation $\Im$ *w.r.t.* a given CGS $\mathfrak{G}$ is *behavioural* if the function $f^{\Im(v)}$ is behavioural, for every position $v \in \mathrm{Ps}$ and symbol $f \in \mathrm{Fn}$. An SL[1G,$\mathcal{F}$] sentence φ is *behaviourally satisfied* by a CGS $\mathfrak{G}$ if there exists a behavioural strategy interpretation $\Im$ such that $(\mathfrak{G}, \Im) \models \varphi$.

▶ **Theorem 12.** *For any* CGS $\mathfrak{G}$ *and* SL[1G] *sentence φ, the sentence* $\mathsf{skm}(\varphi)$ *is satisfied by $\mathfrak{G}$* iff *it is behaviourally satisfied by $\mathfrak{G}$.*

Unifying Bindings & Paths. In [35] it has been observed that the decidability of the satisfiability problem for SL[1G] can be attributed to the fact that variables are indivisibly associated with agents by bindings. Here we further exploit that observation to define a normal form for $\mathrm{SL}^{\varnothing}$[1G] models, applying the notions of *Herbrand property* and *quasi-Herbrand structures* devised in [9], so that unifying bindings identify the same paths.

The notion of SL[1G] (*resp.*, $\mathrm{SL}^{\varnothing}$[1G]) skeleton, as well as the corresponding concept of (*resp.*, single-time) skeleton satisfaction, immediately lifts to SL[1G,$\mathcal{F}$] (*resp.*, $\mathrm{SL}^{\varnothing}$[1G,$\mathcal{F}$]) in the obvious way. A skeleton is universal if all formulas in Φ are universal, *i.e.*, $\Phi \subseteq \forall$SL[1G,$\mathcal{F}$]. Given an SL[1G] (*resp.*, $\mathrm{SL}^{\varnothing}$[1G], $\mathrm{WSL}^{\varnothing}$[1G]) skeleton $\eth$, we denote by $\mathsf{skm}(\eth)$ the (universal) $\forall$SL[1G,$\mathcal{F}$] (*resp.*, $\forall\mathrm{SL}^{\varnothing}$[1G,$\mathcal{F}$], $\forall\mathrm{WSL}^{\varnothing}$[1G,$\mathcal{F}$]) skeleton obtained via Skolemisation of all the sentences in Φ, where a different set of Skolem symbols is used for each sentence.

The following result is an easy corollary of what we have derived. Indeed, Theorem 10 ensures that, for every $\mathrm{SL}^{\varnothing}$[1G] skeleton $\eth$, the sentence $\varphi_{\eth}$ is satisfiable *iff* $\eth$ is single-time satisfiable by some tree CGS $\mathfrak{G}$. Now, by Theorem 11, $\mathfrak{G}, v \models \phi$ *iff* $\mathfrak{G}, v \models \mathsf{skm}(\phi)$, for all $\phi \in \Phi$ and $v \in \mathrm{Ps}$ with $\ell(\phi) \in \lambda(v)$. Finally, Theorem 12 allows for a behavioural satisfaction.

▶ **Corollary 13.** *For every* $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ *skeleton* $\eth$*, it holds that* $\varphi_{\eth}$ *is satisfiable* iff $\mathsf{skm}(\eth)$ *is single-time behaviourally satisfiable by a tree* CGS*.*

Bindings $\flat = (a_1, t_1) \cdots (a_k, t_k) \in \mathrm{Bn}(\mathcal{F})$ are sequences of terms over $\mathcal{F}$, one for each agent. Hence, the standard notions of term replacement, interpretation, unification, and equalisation can be lifted to them in the obvious way. Specifically, $\flat^{\mu} \triangleq (a_1, t_1^{\mu}) \cdots (a_k, t_k^{\mu})$ denotes the *replacement* of all the variables in every t_i with the terms prescribed by the substitution μ, while $\flat^{\mathfrak{F},\chi}$ denotes the *interpretation* of $\flat$ in $\mathfrak{F}$ under the assignment χ, *i.e.*, the profile $\flat^{\mathfrak{F},\chi} \in \mathrm{Prf}(v)$ assigning to each agent a_i the strategy $t_i^{\mathfrak{F},\chi}$. A set of bindings $\mathrm{B} \subseteq \mathrm{Bn}(\mathcal{F})$ *unifies* if there is a substitution μ such that $\flat_1^{\mu} = \flat_2^{\mu}$, for all $\flat_1, \flat_2 \in \mathrm{B}$, while B *equalises* over $\mathfrak{F}$ if there is an assignment χ such that $\flat_1^{\mathfrak{F},\chi} = \flat_2^{\mathfrak{F},\chi}$, for all $\flat_1, \flat_2 \in \mathrm{B}$. As an example, consider the bindings $\flat_1 \triangleq (a, x)(b, f_1(x))(c, z)$, $\flat_2 \triangleq (a, f_2)(b, y)(c, y)$, and $\flat_3 \triangleq (a, f_3(x))(b, x)(c, f_3(x))$, previously obtained by Skolemisation. One can see that $\flat_1$ and $\flat_2$ unify in $(a, f_2)(b, f_1(f_2))(c, f_1(f_2))$, while neither $\flat_1$ and $\flat_3$ nor $\flat_2$ and $\flat_3$ unify. By a result in [9] (see Theorem 1) $\flat_1$ and $\flat_2$ also equalise over every structure $\mathfrak{F}$, while there exists a structure $\mathfrak{F}^{\star}$ (quasi-Herbrand *w.r.t.* $\{\flat_1, \flat_2, \flat_3\}$, see Theorem 2 of [9]) over which $\flat_3$ does not equalise with either $\flat_1$ or $\flat_2$.

Every finite set of bindings $\mathrm{B} \subset \mathrm{Bn}(\mathcal{F})$ is associated with its *maximally unifiable coverage* $\mathsf{muc}(\mathrm{B}) \subseteq 2^{\mathrm{B}}$, *i.e.*, the unique set of the subsets of B such that *1)* $\bigcup \mathsf{muc}(\mathrm{B}) = \mathrm{B}$ and *2)* every $\mathrm{C} \in \mathsf{muc}(\mathrm{B})$ is maximally unifiable, *i.e.*, C is unifiable, but $\mathrm{C} \cup \{\flat\}$ is not unifiable, for all $\flat \in \mathrm{B} \setminus \mathrm{C}$. As an example, consider the set of three bindings $\mathrm{B} \triangleq \{\flat_4, \flat_5, \flat_6\}$, where $\flat_4 = (\alpha, u)(\beta, v)(\gamma, u)$, $\flat_5 = (\alpha, w)(\beta, f(w))(\gamma, x)$, and $\flat_6 = (\alpha, y)(\beta, z)(\gamma, g(z))$. Then, $\mathsf{muc}(\mathrm{B})$ contains all the subsets of B of size 2. Indeed, the first two bindings unify in $\flat_{45} \triangleq (\alpha, u)(\beta, f(u))(\gamma, u)$, the first and the last unify in $\flat_{46} \triangleq (\alpha, g(v))(\beta, v)(\gamma, g(v))$, and the last two bindings unify in $\flat_{56} \triangleq (\alpha, w)(\beta, f(w))(\gamma, g(f(w)))$. In addition, the whole set B is not unifiable, as w cannot unify with $g(f(w))$ and, therefore, $\flat_4$ does not unify with $\flat_{56}$ either. As another example, for the set of bindings $\{\flat_1, \flat_2, \flat_3\}$ of the previous paragraph, we have that $\mathsf{muc}(\{\flat_1, \flat_2, \flat_3\}) = \{\{\flat_1, \flat_2\}, \{\flat_3\}\}$.

A *normal model* of a universal skeleton $\eth$ is an interpreted tree CGS $(\mathfrak{G}, \mathfrak{I})$, where the number $|\tau(v)|$ of successors of each position $v \in \mathrm{Ps}$ is dictated solely by the set of bindings $\flat$ of some sentence $\phi \in \Phi$, whose induced play $\pi = \mathsf{play}(\flat^{\mathfrak{I}(w),\chi}, w)$, with $\chi \in \mathrm{Asg}(w)$ and w an ancestor of v satisfying ϕ, passes through v. In other words, each position in a normal model has just enough successors to separate the sets of non-unifying bindings, which may require different paths to satisfy the associated sentences. The underlying idea is the following. Consider a model of a universal skeleton and a position v in the model labelled with propositions s_1 and s_2, which name the subsentences $\forall \flat_1 \psi_1$ and $\forall \flat_2 \psi_2$. This witnesses that both sentences must be satisfied at v. If bindings $\flat_1$ and $\flat_2$ unify, hence equalise, then the corresponding LTL matrices ψ_1 and ψ_2 must necessarily be satisfied along the same paths from v, as the two bindings induce the same paths. If, however, $\flat_1$ and $\flat_2$ do not unify, then ψ_1 and ψ_2 can be satisfied independently along different paths, since the bindings can have different interpretations. Normal models capture this intuition, by keeping track, at each position, of which bindings are paired with which paths from that position. To this end, such models are equipped with three functions: a *global binding* function g that associates with each position v the set of bindings paired with all the paths through v; a *local binding* function l, associating with each position v the set of bindings of the sentences that label v, *i.e.* the sentences that must be satisfied starting from v; and a *routing* map r that, based on (non)unification of the bindings at v, dispatches them along possibly different paths from v.

▶ **Definition 14** (Normal Model). *An interpreted* CGS $(\mathfrak{G}, \Im)$ *satisfying a* $\forall\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ *skeleton* $\eth$ *is* normal *if* 1) $\mathfrak{G}$ *is a tree and* 2) *there exist three maps* $\mathsf{l}, \mathsf{g} \colon \mathrm{Ps} \to 2^{\mathrm{Bn}}$ *and* $\mathsf{r} \colon v \in \mathrm{Ps} \mapsto (\tau(v) \to \mathsf{muc}(\mathsf{g}(v)))$ *enjoying the following properties, for all positions* $v \in \mathrm{Ps}$*:*

a) $\mathsf{r}(v)$ *is a bijective map from* $\tau(v)$ *to* $\mathsf{muc}(\mathsf{g}(v))$*;*

b) $\mathsf{l}(v) = \left\{\flat \in \mathrm{Bn} \,\middle|\, \exists \phi \triangleq \forall \flat \psi \in \Phi .\, \ell(\phi) \in \lambda(v)\right\}$*;*

c) *if* $v = \varepsilon$ *then* $\mathsf{g}(v) = \mathsf{l}(v)$ *else* $\mathsf{g}(v) = \mathsf{l}(v) \cup \mathsf{r}(\tau^{-1}(v))(v)$*;*

d) $\flat \in \mathsf{r}((\pi)_i)((\pi)_{i+1})$*, for all* $\flat \in \mathsf{l}(v)$*,* $\chi \in \mathrm{Asg}(v, \mathsf{vr}(\flat))$*, and* $i \in \mathbb{N}$*, where* $\pi \triangleq \mathsf{play}(\flat^{\Im(v), \chi}, v)$*.*

For each position v, by Item b, the local binding function l identifies the set of bindings of those universal sentences $\phi \in \Phi$ whose atomic proposition $\ell(\phi)$ labels v (note that ϕ holds at v due to Item 2 of Definition 5); by Item c, the global binding function g extends l with the bindings of the sentences satisfied at some ancestor of v; finally, by Item a, the routing map r distributes the bindings collected by g across the successors of v, in such a way that all bindings forming a maximally unifiable set are routed towards the same successor, while different unifying sets are routed towards different successors. Observe that, due to Item d, a path induced by a binding $\flat$ necessarily passes through one of the successors chosen by r for $\flat$ and, *vice versa*, a successor chosen for $\flat$ is traversed by at least one path induced by $\flat$. Hence, Item d captures the requirement that, at each position, normal models keep track of which bindings are paired with which paths from that position.

Thanks to Corollary 13 and the notion of behavioural satisfaction, from the strategy interpretation $\Im(v) = \left\langle \mathrm{Str}(v), \cdot^{\Im(v)} \right\rangle$ at each position $v \in \mathrm{Ps}$ of a tree CGS $\mathfrak{G}$ one can extract infinitely-many action interpretations $\mathfrak{F}^{v}_{\rho} = \left\langle \mathrm{Ac}, \cdot^{\mathfrak{F}^{v}_{\rho}} \right\rangle$, one for each history $\rho \in \mathrm{Hst}(v)$ starting at v in $\mathfrak{G}$. Specifically, for each function symbol $f \in \mathrm{Fn}$ of arity $k \in \mathbb{N}$, we can set $f^{\mathfrak{F}^{v}_{\rho}}(\vec{c}) \triangleq f^{\Im(v)}(\vec{\sigma})(\rho)$, for all k-tuple of strategies $\vec{\sigma} \in \mathrm{Str}(v)^{k}$, where the i-th element $(\vec{c})_i$ of $\vec{c}$ is equal to the action $(\vec{\sigma})_i(\rho)$ chosen by the i-th strategy $(\vec{\sigma})_i$ of $\vec{\sigma}$ at ρ. In a sense, the strategy interpretation $\Im(v)$ can be viewed as a tree of action interpretations $\mathfrak{F}^{v}_{\rho_w}$, one for each descendant w of v, where ρ_w is the history starting in v and leading to w. By exploiting the connection between strategy and action interpretations and using the fact that, for each set of bindings $\mathrm{B} \subseteq \mathrm{Bn}(\mathcal{F})$, there is always an $\mathcal{F}$-structure $\mathfrak{F}^{\star v}_{\rho}$ for which every $\mathrm{X} \subseteq \mathrm{B}$ unifies *iff* X equalises over $\mathfrak{F}^{\star v}_{\rho}$ (see Theorem 2 of [9]), the following result can be obtained.

▶ **Theorem 15.** *For every* $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ *skeleton* $\eth$*, it holds that* $\varphi_{\eth}$ *is satisfiable* iff $\mathsf{skm}(\eth)$ *is single-time normally satisfiable.*

The main result of this section states that every satisfiable $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ sentence has a bounded-fork model. This can be easily derived from the previous theorem by observing the following: *i)* due to the single-time satisfaction property, along any path of the model, there are at most $|\Phi|$ sentences of the form $\mathsf{skm}(\wp\flat\psi)$ that need to be satisfied, since every atomic proposition $\ell(\wp\flat\psi)$ occurs at most once; *ii)* thanks to the normality property, a fork at any given position v of a path is only caused by non-unifying bindings, which occur if new sentences in Φ need to be satisfied at v, as the bindings routed toward v from the ancestors necessarily unify.

▶ **Corollary 16.** *For every* $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ *skeleton* $\eth$*, it holds that* $\varphi_{\eth}$ *is satisfiable* iff $\mathsf{skm}(\eth)$ *is single-time normally satisfied by a* k*-fork* CGS*, for* $k \triangleq \mathsf{max}\{0, |\Phi| - 1\}$*.*

5 New Classes of Automata

In this section, we introduce the novel class of *bounded-fork tree automata* that will be exploited in Section 6 to devise an efficient satisfiability checking algorithm for the non-recurrent fragments of $\mathrm{SL}[1\mathrm{G}]$.

Bounded-Fork Tree Automata. Bounded-fork automata are a restriction of the standard tree automata tailored to accept only trees having a bounded number of fork nodes along each path starting from the root (recall that bounded-fork trees are suitable models for $\text{SL}^{\varnothing}[1\text{G}]$). If at most k forks in a path are allowed, the ability to count the number of occurring forks is obtained by partitioning the set of states Q into $k+1$ subsets $\text{Q}_0, \ldots, \text{Q}_k$. Intuitively, a state $q \in \text{Q}_i$ can observe at most i additional forks along a path. Naturally, the initial states belong to Q_k and only states in Q_0, from where no more forks are admitted, can be involved in the Büchi acceptance condition.

▶ **Definition 17** (Bounded-Fork Automaton). *An* NTA $\mathcal{A} \triangleq \langle \Sigma, \Lambda, \text{Q}, \delta, q_I, \text{Q}_\text{F} \rangle$ *is* k-forking *(*k-NTA*, for short), for some given* $k \in \mathbb{N}$*, if* (i) $1 \in \Lambda$ *and* (ii) *there is a* $(k+1)$*-partition* $(\text{Q}_0, \ldots, \text{Q}_k)$ *of* Q *satisfying the following constraints:* (a) $q_I \in \text{Q}_k$*;* (b) $\text{Q}_\text{F} \subseteq \text{Q}_0$*;* (c) *for all indexes* $i \in [0, k]$*, states* $q \in \text{Q}_i$*, input symbols* $\sigma \in \Sigma$*, and node degrees* $d \in \Lambda$*, if* $d = 1$ *then* $\delta(q, \sigma, d) \subseteq \bigcup_{j=0}^{i} \text{Q}_j$ *else* $\delta(q, \sigma, d) \subseteq (\bigcup_{j=0}^{i-1} \text{Q}_j)^d$*.*

The motivation for using k-NTAs, instead of standard NTAs, is clearly expressed by Theorem 18, which establishes a logarithmic space complexity of the emptiness problem *w.r.t.* the size of the k-NTA. This contrasts with the PTime hardness bound on the same problem for classic NTA [42]. To prove the result, we devise a recursive reachability algorithm that looks for a reachable cycle that includes an accepting state and is completely contained within the partition Q_0 of the k-NTA. The gain in complexity is due to the fact that the algorithm only needs the space required for backtrack along a path to previous fork states, whose number is bounded by k. The emptiness problem can also be solved by reduction to alternating Turing machines, as well as to Büchi games, where the number of turns assigned to the universal player is limited by k [39].

▶ **Theorem 18.** *The emptiness problem for a* k-NTA *with* n *states and transition function of size* m *can be solved in* $\text{Space}(k \cdot \log n + \log^2 n + \log m)$ *and* $\text{ATime}[k\text{-alt}](\log n + \log m)$*.*

Good for Game Automata. One way to solve the satisfiability problem for branching-time logics is to embed a word automaton $\mathcal{W}$, taking care of the linear constraints on the paths, within a tree automaton that, at each step, dispatches copies of $\mathcal{W}$, updated according to the symbol read, along all the possible branching directions [36, 40]. This approach works pretty nicely for deterministic word automata, but not for general nondeterministic ones, as the nondeterministic choice at a given instant of time can be solved by exploiting knowledge about future instants. However, the correctness of the approach can still be recovered if the requirement on determinism is relaxed slightly, allowing for a "controlled" form of non-determinism. This leads to the notion of nondeterministic *good-for-game automata* [27].

Fixed an *a priori* set of node degrees $\Lambda \subseteq \mathbb{N}_+$, the *word-on-tree function* $\mathsf{wot}_\Lambda : \text{NWA} \to \text{NTA}$ maps an NWA $\mathcal{W} = \langle \Sigma, \text{Q}, \delta, q_I, \text{Q}_\text{F} \rangle$ to the NTA $\mathsf{wot}_\Lambda(\mathcal{W}) \triangleq \left\langle \Sigma, \Lambda, \text{Q}, \widehat{\delta}, q_I, \text{Q}_\text{F} \right\rangle$, whose tree transition function $\widehat{\delta}$ is derived from the word transition function δ as follows: $\widehat{\delta}(q, \sigma, d) \triangleq \prod_{i=0}^{d-1} \delta(q, \sigma)$, for all states $q \in \text{Q}$, input symbols $\sigma \in \Sigma$, and node degree $d \in \Lambda$.

▶ **Definition 19** (Good-for-Game Automaton). *Let* $\mathfrak{T}$ *be a class of* Σ*-labelled* Λ*-trees. An* NWA $\mathcal{W} = \langle \Sigma, \text{Q}, \delta, q_I, \text{Q}_\text{F} \rangle$ *is a* good-for-game automaton w.r.t. $\mathfrak{T}$ *(*$\mathfrak{T}$-GFG*, for short) if* $\text{Trc}(\mathcal{T}) \subseteq \mathsf{L}(\mathcal{W})$ *implies* $\mathcal{T} \in \mathsf{L}(\mathsf{wot}_\Lambda(\mathcal{W}))$*, where* $\text{Trc}(\mathcal{T})$ *is the set of* Σ*-traces of* $\mathcal{T}$*, for all trees* $\mathcal{T} \in \mathfrak{T}$*.*

Intuitively, good-for-game automata only use knowledge of the (non-strict) past to determine the next steps and no information on the future (via forks related to nondeterministic guesses). This relates to the notion of strategy in the context of games, where strategies are purely based on histories (*i.e.*, finite traces). In particular, note that the converse direction, $\mathcal{T} \in \mathsf{L}(\mathsf{wot}_{\Lambda}(\mathcal{W}))$ implies $\mathrm{Trc}(\mathcal{T}) \subseteq \mathsf{L}(\mathcal{W})$, always holds true.

The next paragraph introduces a class of word automata that are GFG for k-bounded trees and will allow us to define a satisfiability algorithm for $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$.

Prefix-Deterministic Word Automata. *Prefix-deterministic word automata* are NWAs which behave deterministically on arbitrary prefixes of their runs and behave freely, *i.e.*, nondeterministically, afterwards. The idea is that such automata can be used to encode the checks for compliance of all the paths of a tree *w.r.t.* an LTL property. We shall take advantage of the prefix-determinism to constrain the nondeterministic choices within the automaton to occur only after an initial prefix where all the forks already occurred.

▶ **Definition 20** (Prefix-Deterministic Automaton)**.** *An* NWA $\mathcal{W} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathrm{Q_F} \rangle$ *is* prefix-deterministic *(*PD-NWA*, for short) if there is a deterministic transition function* $\widetilde{\delta} \colon \mathrm{Q} \times \Sigma \to \mathrm{Q} \cup \{\bot, \top\}$ *with* $\widetilde{\delta}(q, \sigma) \in \delta(q, \sigma) \cup \{\bot, \top\}$*, for all states* $q \in \mathrm{Q}$ *and input symbols* $\sigma \in \Sigma$*, such that, for all infinite words* $v \cdot w \in \Sigma^{\omega}$*, it holds that* $v \cdot w \in \mathsf{L}(\mathcal{W})$ iff *either one of the following two conditions holds true, where* $q_v \triangleq \widetilde{\delta}^{*}(q_I, v)$*: 1)* $q_v = \top$*; 2)* $q_v \neq \bot$ *and* $w \in \mathsf{L}(\mathcal{W}_v)$ *with* $\mathcal{W}_v \triangleq \langle \Sigma, \mathrm{Q}, \delta, q_v, \mathrm{Q_F} \rangle$*.*

For every NWA $\mathcal{W}$, we can easily construct a language equivalent PD-NWA, by using a standard subset construction for the determinisation of the initial behaviours of $\mathcal{W}$ and, then, suitably concatenating it to $\mathcal{W}$ itself to complete the behaviours.

▶ **Theorem 21.** *For every* NWA $\mathcal{W}$ *with* n *states, there exists a* PD-NWA $\mathcal{D}$ *with* $n + 2^n$ *states such that* $\mathsf{L}(\mathcal{D}) = \mathsf{L}(\mathcal{W})$*.*

The standard automaton-theoretic construction for LTL [41] can be easily lifted to PD-NWA as stated by Theorem 22. Notice that, when the WLTL fragment of LTL is considered, the automaton construction has to deal with a number of formulas which is only singly-exponential *w.r.t.* the size of the input formula.

▶ **Theorem 22.** *For every* LTL *(resp.* WLTL*) formula* ψ*, there is a* PD-NWA $\mathcal{D}_{\psi}$ *with* $2^{\mathrm{O}\left(2^{|\psi|}\right)}$ *(*resp., $\mathrm{O}\left(2^{|\psi|^3}\right)$*) states such that* $\mathsf{L}(\mathcal{D}_{\psi}) = \mathsf{L}(\psi)$*.*

The following result ensures that every PD-NWA can be embedded within a bounded-fork tree automaton, a result that will be leveraged in the next section.

▶ **Theorem 23.** *Every* PD-NWA *is* GFG w.r.t. *the class of bounded-fork trees.*

6 The Satisfiability Problem

We solve satisfiability for $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ by reducing it to non-emptiness of a bounded-fork automaton. Let φ be an arbitrary $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ sentence and, thanks to Proposition 8, $\eth \triangleq \langle \zeta, \Phi, \ell \rangle$ be its a corresponding Skolem skeleton. Corollary 16 tells us that φ is satisfiable *iff* $\eth$ is single-time normally satisfied by a k-fork CGS, with $k \triangleq \mathsf{max}\{0, |\Phi| - 1\}$. We construct a k-NTA $\mathcal{N}_{\varphi}$ recognising all normal models of $\eth$ (which are also models of φ). The automaton is the product $\mathcal{N}_{\varphi} \triangleq \mathcal{D}_{\zeta} \times \mathcal{D}_{\eth} \times \mathcal{N}_{\Phi}$ of the following three components: *(1)* $\mathcal{D}_{\zeta}$ is a trivial single-state safety automaton checking whether the labelling of the root satisfies the Boolean formula ζ; *(2)* $\mathcal{D}_{\eth}$ is a deterministic safety automaton checking that the structure of the input tree complies with Definition 14; *(2)* the nondeterministic Büchi automaton $\mathcal{N}_{\Phi}$ ensures that all paths identified by the binding $\flat$ of some sentence $\forall\flat\psi$ in Φ satisfy the LTL formula ψ. We focus on the definitions of $\mathcal{D}_{\eth}$ and $\mathcal{N}_{\Phi}$ only, $\mathcal{D}_{\zeta}$ being obvious.

Before proceeding, we need to fix some notation. Let $\mathrm{A} \subseteq \mathrm{AP}$ and $\mathrm{B} \subseteq \mathrm{Bn}(\mathcal{F})$ be the sets of all the atomic propositions and bindings occurring in some sentence of the universal skeleton $\eth$, respectively. The automaton alphabet $\Sigma \subseteq 2^{\mathrm{A} \cup \mathrm{B}}$ is then the set of all those symbols $\sigma \subseteq \mathrm{A} \cup \mathrm{B}$ satisfying the following local coherence condition: if $\ell(\phi) \in \sigma$ then $\flat \in \sigma$, for all universal sentences $\phi \triangleq \forall\flat\psi \in \Phi$. The idea is to recognise all normal models whose labelling is enriched with the bindings of the sentences that are satisfied along some path through each node, as prescribed by Item b of Definition 14 on the global binding function g. In particular, the local coherence condition precisely corresponds to the property required on the local binding function l by Item b of the same definition. Obviously, the branching degree of the tree is bounded by $|\mathsf{muc}(\mathrm{B})|$, thus, the set of node degrees is $\Lambda \triangleq [1, |\mathsf{muc}(\mathrm{B})|]$. Finally, let us consider an arbitrary function $\widehat{\mathsf{r}} \colon \mathrm{X} \in 2^{\mathrm{B}} \mapsto ([0, |\mathsf{muc}(\mathrm{X})| - 1] \to \mathsf{muc}(\mathrm{X}))$ such that, for each set of bindings $\mathrm{X} \subseteq \mathrm{B}$, the associated map $\widehat{\mathsf{f}} \triangleq \widehat{\mathsf{r}}(\mathrm{X}) \colon [0, |\mathsf{muc}(\mathrm{X})| - 1] \to \mathsf{muc}(\mathrm{X})$ is bijective. This function is used in the following construction to ensure Item a of Definition 14.

▶ **Construction 24** (Structure Automaton). *The* structure automaton $\mathcal{D}_{\eth}$ *is the safety* DTA $\langle \Sigma, \Lambda, 2^{\mathrm{B}}, \delta, \emptyset \rangle$ *whose transition function δ is defined as follows: $\delta(\mathrm{X}, \sigma, d) \triangleq \prod_{i=0}^{d-1} \widehat{\mathsf{f}}(i)$, where $\widehat{\mathsf{f}} \triangleq \widehat{\mathsf{r}}(\sigma \cap \mathrm{B})$, if $\mathrm{X} \subseteq \sigma$ and $d = |\mathsf{img}\left(\widehat{\mathsf{f}}\right)|$, and $\delta(\mathrm{X}, \sigma, d) \triangleq \perp$, otherwise, for any set of bindings $\mathrm{X} \subseteq \mathrm{B}$, input symbol $\sigma \in \Sigma$, and node degree $d \in \Lambda$.*

It can be shown that, if we extend any normal model of $\eth$ with the binding labelling dictated by its global binding function, we obtain a tree structure that is accepted by $\mathcal{D}_{\eth}$. *Vice versa*, every tree accepted by $\mathcal{D}_{\eth}$ is the backbone of a tree CGS, whose functions l, g, and r (see Definition 14) can be easily extracted from the labelling. To ensure that this CGS is actually a normal model, we need to verify that the paths labelled by some binding $\flat$ satisfy the corresponding sentences in Φ. This is precisely the goal of $\mathcal{N}_{\Phi}$.

By Theorem 22, for any $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ (*resp.*, $\mathrm{WSL}^{\varnothing}[1\mathrm{G}]$) sentence $\phi \triangleq \wp\flat\psi \in \Phi$, we can always construct a PD-NWA $\mathcal{D}_{\psi}$ with $2^{\mathrm{O}\left(2^{|\psi|}\right)}$ (*resp.*, $\mathrm{O}\left(2^{|\psi|^3}\right)$) states such that $\mathsf{L}(\mathcal{D}_{\psi}) = \mathsf{L}(\psi)$. By using a constant number of additional states, we can turn $\mathcal{D}_{\psi}$ into a PD-NWA $\mathcal{D}_{\widetilde{\phi}}$ recognising all models of the LTL (*resp.*, WLTL) formula $\widetilde{\phi} \triangleq \mathtt{G}\,(\ell(\phi) \to ((\mathtt{X}\,\mathtt{G}\,\neg\ell(\phi)) \wedge (\psi \vee \mathtt{F}\,\neg\flat)))$. This formula ensures that ψ is verified starting from the unique point where the corresponding atomic proposition $\ell(\phi)$ occurs, provided that binding $\flat$ is still active. By turning each PD-NWA $\mathcal{D}_{\widetilde{\phi}}$ into a tree automaton, we obtain the last component of $\mathcal{N}_{\varphi}$.

▶ **Construction 25** (Sentence Automaton). *The* sentence automaton $\mathcal{N}_{\Phi}$ *is obtained as the product $\prod_{\phi \in \Phi} \mathcal{N}_{\phi}$ of the* NTA*s $\mathsf{wot}_{\Lambda}(\mathcal{D}_{\widetilde{\phi}})$ derived from the* PD-NWA *$\mathcal{D}_{\widetilde{\phi}}$, for each $\phi \in \Phi$.*

While neither $\mathcal{D}_{\eth}$ nor $\mathcal{N}_{\Phi}$ taken in isolation is a bounded-fork automaton, their product does satisfy the property. Indeed, $\mathcal{N}_{\Phi}$ ensures that, along a path, the labelling $\ell(\phi)$ of a sentence $\phi \in \Phi$ occurs at most once and $\mathcal{D}_{\eth}$ has a transition with more than one successor at a given node v only if the set of bindings in the labelling of v does not unify. In the remaining part of this section, by $\mathrm{SL}_{k}^{\varnothing}[1\mathrm{G}]$ (*resp.*, $\mathrm{WSL}_{k}^{\varnothing}[1\mathrm{G}]$) we denote the set of those $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ (*resp.*, $\mathrm{WSL}^{\varnothing}[1\mathrm{G}]$) sentences containing at most k subsentences.

▶ **Theorem 26.** *The* NTA *$\mathcal{N}_{\varphi}$ is k-forking, for every $\mathrm{SL}_{k}^{\varnothing}[1\mathrm{G}]$ sentence φ.*

At this point, it is clear that every tree accepted by $\mathcal{N}_\varphi$ corresponds to a normal model for the Skolem skeleton $\mathfrak{d}$ of φ. *Vice versa*, the underlying tree structure of a normal model is accepted by $\mathcal{N}_\varphi$ thanks to the fact that every $\mathcal{D}_{\widetilde{\phi}}$ is a GFG for the class of bounded-fork tree, as shown in Theorem 23. This leads to the following result.

▶ **Theorem 27.** *For every* $\mathrm{SL}_k^{\varnothing}[1\mathrm{G}]$ *(*resp.*,* $\mathrm{WSL}_k^{\varnothing}[1\mathrm{G}]$*) sentence* φ*, there is a* k*-NTA* $\mathcal{N}_\varphi$ *of size* $2^{\mathrm{O}\left(2^{|\varphi|}\right)}$ *(*resp.*,* $2^{|\varphi|^{\Theta(1)}}$*) recognising all and only the normal models of* φ *itself.*

By Theorem 18, we obtain that deciding $\mathrm{WSL}^{\varnothing}[1\mathrm{G}]$ is provably easier than deciding full $\mathrm{SL}[1\mathrm{G}]$, which is known to be 2ExpTime-complete, while the complexity for $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ is lower only if we assume the widely-shared conjecture that ExpSpace $\subset$ 2ExpTime. While PSpace-hardness of $\mathrm{WSL}^{\varnothing}[1\mathrm{G}]$ trivially follows from an obvious encoding of standard modal-logic satisfiability, it is not even known whether $\mathrm{SL}^{\varnothing}[1\mathrm{G}]$ is ExpTime-hard.

▶ **Theorem 28.** $\mathrm{SL}_k^{\varnothing}[1\mathrm{G}]$ *(*resp.*,* $\mathrm{WSL}^{\varnothing}[1\mathrm{G}]$*) satisfiability problem is* AExpTime[k-alt] *(*resp.*,* PSpace-complete*).*

7 Discussion

We have considered efficiently decidable fragments of one-goal SL, called non-recurrent fragments, where satisfaction requests for a sentence can only be iterated a bounded number of times along a computation. This is achieved by restricting the first (*resp.*, second) argument of the until (*resp.*, release) linear temporal operator. Specifically, when these arguments are limited to pure LTL formulae, we obtain that satisfiability is decidable in AExpTime[k-alt] (which is known to be included in ExpSpace), where k is the number of subsentences within the formulae. If, however, those arguments are further restricted to Boolean formulae, a PSpace-complete result is given. Both the non-recurrent fragments, which are strictly included in SL[1G], are still able to express non-trivial game-theoretic problems, such as the automatic synthesis of multi-agent systems, *e.g.*, communication protocols where active participants perform a bounded number of decisions during each session.

On the technical side, we obtain the complexity bounds by means of two main techniques. First, by exploiting a quasi-Herbrand property of a first-order characterisation of the sentences of those fragments, we identify a normal-form for the models of satisfiable sentences, which only admit a bounded number of branching points along any computation. Second, we leverage a novel class of automata, called bounded-fork tree automata, that can recognise normal models and whose language emptiness problem can be checked in LogSpace.

Since SL[1G] strictly includes both ATL* and CTL*, the results immediately applies also to the corresponding non-recurrent fragments of those logics, where only LTL (*resp.*, Boolean) formulae can occur in the first (*resp.*, second) argument of an until (*resp.*, release) operator. In particular, this observation identifies novel fragments for both logics, namely the weak non-recurrent ones, with a satisfiability problem whose complexity, which is PSpace-complete, is strictly lower than the one for the full languages, known to be 2ExpTime-complete.

References

1 E. Acar, M. Benerecetti, and F. Mogavero. Satisfiability in Strategy Logic Can Be Easier than Model Checking. In *AAAI Press19*, pages 2638–2645. AAAI Press, 2019.

2 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002.

3 E.G. Amparore, S. Donatelli, and F. Gallà. A CTL* Model Checker for Petri Nets. In *PETRI NETS'20*, LNCS 12152, pages 403–413. Springer, 2020.

4 F. Baader and W. Snyder. Unification Theory. In *Handbook of Automated Reasoning (vol. 1).*, pages 445–532. Elsevier & MIT Press, 2001.

5 M. Benerecetti, F. Mogavero, and A. Murano. Substructure Temporal Logic. In *LICS'13*, pages 368–377. IEEECS, 2013.

6 M. Benerecetti, F. Mogavero, and A. Murano. Reasoning About Substructures and Games. *TOCL*, 16(3):25:1–46, 2015.

7 P. Bouyer, P. Gardy, and N. Markey. Weighted Strategy Logic with Boolean Goals Over One-Counter Games. In *FSTTCS'15*, LIPIcs 45, pages 69–83. Leibniz-Zentrum fuer Informatik, 2015.

8 P. Bouyer, P. Gardy, and N. Markey. On the semantics of Strategy Logic. *IPL*, 116(2):75–79, 2016.

9 S. Bova and F. Mogavero. Herbrand Property, Finite Quasi-Herbrand Models, and a Chandra-Merlin Theorem for Quantified Conjunctive Queries. In *LICS'17*, pages 1–12. ACM, 2017.

10 K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. In *CONCUR'07*, LNCS 4703, pages 59–73. Springer, 2007.

11 K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. *IC*, 208(6):677–693, 2010.

12 H. Comon and V. Cortier. Flatness Is Not a Weakness. In *CSL'00*, LNCS 1862, pages 262–276. Springer, 2000.

13 D. Dams. Flat Fragments of CTL and CTL*: Separating the Expressive and Distinguishing Powers. *LJIGPL*, 7(1):55–78, 1999.

14 S. Demri, R. Lazic, and D. Nowak. On the Freeze Quantifier in Constraint LTL: Decidability and Complexity. *IC*, 205(1):2–24, 2007.

15 S. Demri and P. Schnoebelen. The Complexity of Propositional Linear Temporal Logics in Simple Cases. *IC*, 174(1):84–103, 2002.

16 E.A. Emerson. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science (vol. B).*, pages 995–1072. MIT Press, 1990.

17 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.

18 E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time. *JACM*, 33(1):151–178, 1986.

19 G.E. Fainekos, S.G. Loizou, and G.J. Pappas. Translating Temporal Logic to Controller Specifications. In *DC'06*, pages 899–904. IEEECS, 2006.

20 P. Gardy, P. Bouyer, and N. Markey. Dependences in Strategy Logic. In *STACS'18*, LIPIcs 96, pages 34:1–15. Leibniz-Zentrum fuer Informatik, 2018.

21 P. Gardy, P. Bouyer, and N. Markey. Dependences in Strategy Logic. *TCS*, 64(3):467–507, 2020.

22 S. Ghosh and R. Ramanujam. Strategies in Games: A Logic-Automata Study. In *ESSLLI'11*, LNCS 7388, pages 110–159. Springer, 2011.

23 E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research.* LNCS 2500. Springer, 2002.

24 J. Gutierrez, P. Harrenstein, and M. Wooldridge. Iterated Boolean Games. In *IJCAI'13*, pages 932–938, 2013.

25 J. Gutierrez, P. Harrenstein, and M. Wooldridge. Reasoning about Equilibria in Game-Like Concurrent Systems. In *KR'14*, pages 408–417. AAAI Press, 2014.

26 J. Gutierrez, P. Harrenstein, and M. Wooldridge. Iterated Boolean Games. *IC*, 242:53–79, 2015.

27 T.A. Henzinger and N. Piterman. Solving Games Without Determinization. In *CSL'06*, LNCS 4207, pages 395–410. Springer, 2006.

28 O.H. Ibarra and Z. Dang. On Removing the Pushdown Stack in Reachability Constructions. In *ISAAC'01*, LNCS 2223, pages 244–256. Springer, 2001.

29 B. Khoussainov and A. Nerode. *Automata Theory and Its Applications.* Birkhauser, 2001.

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

31 F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. What Makes ATL* Decidable? A Decidable Fragment of Strategy Logic. In *CONCUR'12*, LNCS 7454, pages 193–208. Springer, 2012.

32 F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.

33 F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. Reasoning About Strategies: On the Satisfiability Problem. *LMCS*, 13(1:9):1–37, 2017.

34 F. Mogavero, A. Murano, and M.Y. Vardi. Reasoning About Strategies. In *FSTTCS'10*, LIPIcs 8, pages 133–144. Leibniz-Zentrum fuer Informatik, 2010.

35 F. Mogavero and G. Perelli. Binding Forms in First-Order Logic. In *CSL'15*, LIPIcs 41, pages 648–665. Leibniz-Zentrum fuer Informatik, 2015.

36 A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *POPL'89*, pages 179–190. ACM, 1989.

37 S. Schewe. ATL* Satisfiability is 2ExpTime-Complete. In *ICALP'08*, LNCS 5126, pages 373–385. Springer, 2008.

38 P. Schnoebelen. The Complexity of Temporal Logic Model Checking. In *AIML'02*, pages 393–436. College Publications, 2002.

39 L.J. Stockmeyer. The Polynomial-Time Hierarchy. *TCS*, 3(1):1–22, 1976.

40 M.Y. Vardi. Reasoning about The Past with Two-Way Automata. In *ICALP'98*, LNCS 1443, pages 628–641. Springer, 1998.

41 M.Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *LICS'86*, pages 332–344. IEEECS, 1986.

42 M.Y. Vardi and P. Wolper. Automata-Theoretic Techniques for Modal Logics of Programs. *JCSS*, 32(2):183–221, 1986.

Giving Instructions in Linear Temporal Logic

Julian Gutierrez
Monash University, Melbourne, Australia

Sarit Kraus
Bar-Ilan University, Ramat-Gan, Israel

Giuseppe Perelli
Sapienza University of Rome, Italy

Michael Wooldridge
University of Oxford, UK

Abstract

Our aim is to develop a formal semantics for giving instructions to taskable agents, to investigate the complexity of decision problems relating to these semantics, and to explore the issues that these semantics raise. In the setting we consider, agents are given instructions in the form of Linear Temporal Logic (LTL) formulae; the intuitive interpretation of such an instruction is that the agent should act in such a way as to ensure the formula is satisfied. At the same time, agents are assumed to have inviolable and immutable background safety requirements, also specified as LTL formulae. Finally, the actions performed by an agent are assumed to have costs, and agents must act within a limited budget. For this setting, we present a range of interpretations of an instruction to achieve an LTL task Υ, intuitively ranging from "try to do this but only if you can do so with everything else remaining unchanged" up to "drop everything and get this done." For each case we present a formal pre-/post-condition semantics, and investigate the computational issues that they raise.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Automated reasoning

Keywords and phrases Linear Temporal Logic, Synthesis, Game theory, Multi-Agent Systems

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.15

Funding *Giuseppe Perelli*: Giuseppe Perelli is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), by the EU ICT-48 2020 project TAILOR (No. 952215), and by the Sapienza University project "BeGood" (No. AR22117A2DC7CD60).
Michael Wooldridge: Michael Wooldridge was supported by the UKRI under grant (grant EP/W002949/1).

1 Introduction

When we consider AI systems that carry out tasks on our behalf (*agents*), we can distinguish between different types. At one extreme are agents that are hard-wired to carry out a single specific function (e.g., vacuum cleaning robots). At the other extreme are *autonomous* agents, which have some control over their (mental) state and actions: we can *request* an autonomous agent to do something, but whether the agent actually does it is beyond our control. In this work, we are concerned with *taskable* agents, which lie between these two extremes. Taskable agents have a general set of capabilities, which they can draw upon to carry out a wide range of tasks on our behalf (typically within some constrained environment) – see, e.g., [19]. Unlike autonomous agents, however, taskable agents *must* try to carry out the instructions that are presented to them, although we are not in control of how they will try to do so.

In this paper we develop a semantics of instructions for taskable agents: how do we give them instructions, and how should an agent in receipt of these reconfigure itself in light of them? Our work lies within the tradition of reactive synthesis [27] and rational verification [11, 15]. We assume agents are given instructions in the form of Linear Temporal Logic (LTL) formulae. A taskable agent given an LTL instruction Υ must subsequently act so as to ensure Υ is satisfied, if it can do so. At the same time, however, agents are assumed to have inviolable and immutable background *safety requirements*, which are also specified as LTL formulae. *Whatever* an agent does, it must attempt to ensure that its safety requirements are respected: *safety takes priority over obedience.* Actions performed by an agent are assumed to have costs, and agents must act within a given limited budget. For this setting, we present a range of interpretations of an instruction to achieve an LTL task Υ.

A key underlying principle to our work is the *principle of least change*: when acting on a new instruction, an agent should *try to keep everything else as close to how it was before as possible.* This is a *ceteris paribus* condition: keep all other things equal. Thus, the most basic form of instruction we consider intuitively says "try to do this but only if you can do so with everything else remaining unchanged". An agent will adopt such an instruction only if it is able to do so while being able to guarantee its original task is achieved, ensuring that its safety requirement remains satisfied, and staying within its original budget. At the other extreme are instructions of the form "get this done whatever it takes" in which case we want the agent to get the task achieved irrespective of its original task and budget. Between these two extremes are a range of possibilities, some of which we study here. For each case, we present a pre-/post-condition semantics, and investigate the issues the semantics raise. We study the complexity of decision problems relating to our semantics, and find that these range from polynomial time decidable up to 2EXPTIME-complete.

2 The Formal Framework

Where S is a set, we denote the powerset of S by $\mathbf{2}^S$. We use various propositional languages to express properties of the systems we consider. In these languages, we let Φ be a finite and non-empty vocabulary of Boolean variables, with elements $p, q, \ldots$ Where a and b are words (either finite or infinite) we denote the word obtained by concatenating a and b by $a \cdot b$. Where a is a word, we denote by a^ω the infinite repetition of a.

2.1 Environments

We consider environments in which a single agent is acting. We model such environments as nondeterministic finite state machines. We use $\mathcal{CALLIGRAPHIC}$ letters for elements of the environment. Our model assumes that an agent acts in an environment that can be in any of a set $\mathcal{S}$ of possible environment states; the environment is initially in state s_0. The agent has a repertoire $\mathcal{A}$ of actions that it can perform: the result of performing an action $\alpha \in \mathcal{A}$ when the environment is in a state $s \in \mathcal{S}$ is to transform the environment into another state that is nondeterministically taken from $\mathcal{T}(s, \alpha)$; we refer to $\mathcal{T}$ as the state transformer function of the environment. Finally, individual actions have real-valued costs associated with them: the cost of performing α is denoted $\mathcal{C}(\alpha)$ – thus, in the interests of simplicity, costs are fixed, and independent of the state of the system.

Formally an environment is given by a structure $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{L}, s_0)$, where:

- $\mathcal{S}$ is a finite and non-empty set of *environment states*;
- $\mathcal{A}$ is the finite and non-empty set of actions that may be performed in $\mathcal{E}$;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathbf{2}^{\mathcal{S}}$ is the *nondeterministic state transformer function* of $\mathcal{E}$;
- $\mathcal{C} : \mathcal{A} \to \mathbb{R}$ is a *cost function*, which indicates the cost $\mathcal{C}(\alpha)$ of performing action $\alpha \in \mathcal{A}$;
- $\mathcal{L} : \mathcal{S} \to \mathbf{2}^{\Phi}$ is a *labelling function*, which associates with each state $s \in \mathcal{S}$ the set $\mathcal{L}(s) \subseteq \Phi$ of Boolean variables true in s; and finally
- $s_0 \in \mathcal{S}$ is the initial state of the environment.

The environment $\mathcal{E}$ is said to be *deterministic* if $\mathcal{T}(s, \alpha)$ is a singleton set for each environment state $s \in \mathcal{S}$ and action $\alpha \in \mathcal{A}$. As we will see later, deterministic environments greatly simplify many of the decision problems addressed in this paper.

2.2 Strategies

A strategy is a plan that defines how an agent will act in an environment. As is common practice in the theory of iterated games [6], and in work on synthesis and verification [13], we model strategies as finite state machines with output (i.e., Moore machines). Formally, a strategy σ for an environment $\mathcal{E}$ is given by a finite state machine $\sigma = (Q, \tau, \delta, q^0)$ where:

- Q is the set of machine states;
- $\tau : Q \times \mathcal{S} \to Q$ is the state transition function of the strategy;
- $\delta : Q \to \mathcal{A}$ is the action selection function of the strategy, which selects an action $\delta(q)$ for every machine state $q \in Q$; and
- $q^0 \in Q$ is the initial state of the strategy.

We let Σ be the set of all such finite state machine strategies ($\mathcal{E}$ is assumed to be clear from context). A strategy is enacted in an environment, starting from some state $s \in \mathcal{S}$, as follows: the environment starts in state s, and the strategy starts in state q^0. The strategy then selects an action $\alpha_0 = \delta(q^0)$, and changes state to $q' = \tau(q^0, s)$. The environment responds to the performance of α_0 by moving to a state $s_1 \in \mathcal{T}(s, \alpha_0)$; the agent then chooses an action $\delta(q')$, and so on. In this way, the performance of a strategy σ in the environment $\mathcal{E}$ traces out an infinite interleaved sequence of environment states and actions, called a *run*:

$$\rho : s \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \cdots$$

The set of runs that may be generated by the enactment of σ in $\mathcal{E}$ from state $s \in \mathcal{S}$ is denoted by $\mathcal{R}(\mathcal{E}, \sigma, s)$. We write $\mathcal{R}(\mathcal{E}, \sigma)$ as an abbreviation for $\mathcal{R}(\mathcal{E}, \sigma, s_0)$. A run $\rho \in \mathcal{R}(\mathcal{E}, \sigma, s)$ is sometimes said *compatible* with σ in $\mathcal{E}$ – the definition is straightforward, and so we omit it here. Where ρ is a run and $u \in \mathbb{N}$, we write $s(\rho, u)$ to denote the state indexed by u in ρ – so $s(\rho, 0)$ is the first state in ρ, $s(\rho, 1)$ is the second, and so on and so forth. In the same way, we denote the first action in ρ by $\alpha(\rho, 0)$, the second by $\alpha(\rho, 1)$, etc.

Above, we defined the cost function $\mathcal{C}(\cdot)$ with respect to individual actions. In what follows, we find it useful to lift the cost function from individual actions to runs. Since runs are infinite, simply summing costs is not appropriate: instead, we consider the cost of a run to be the *average* cost incurred over the whole run; more precisely, we define the cost $\mathcal{C}(\rho)$ as the *inferior limit of means*: this is a very standard approach in automated formal verification as well as in the theory of iterated games (see, e.g., [6, p.366]). Formally, we have:

$$\mathcal{C}(\rho) = \liminf_{t \to \infty} \frac{1}{t} \sum_{i=0}^{t} \mathcal{C}(\alpha(\rho, i))$$

Given a run ρ, we define the *histories* associated with ρ to be the set of finite and non-empty prefixes of ρ which end in an environment state. We let $\mathcal{H}(\rho)$ denote the histories associated with ρ, and use $h, h', \ldots$ to refer to elements of $\mathcal{H}$. Where h is a history, we denote the word obtained from h by removing the final environment state from it by $\underline{h}$. We denote the final environment state occurring in h by $\mathsf{last}(h)$.

2.3 Linear Temporal Logic

We use Linear Temporal Logic (LTL) to express properties of runs [25, 9]. LTL extends propositional logic with tense operators $\mathbf{X}$ ("in the next state…"), $\mathbf{F}$ ("eventually…"), $\mathbf{G}$ ("always…"), and $\mathbf{U}$ ("…until …"). Formally, the syntax of LTL is defined with respect to a set Φ of Boolean variables by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\, \mathbf{U}\, \varphi$$

where $p \in \Phi$. Other classical connectives ("$\bot$", "$\wedge$", "$\to$", "$\leftrightarrow$") are defined in terms of $\neg$ and $\vee$ in the conventional way. The LTL operators $\mathbf{F}$ and $\mathbf{G}$ are defined in terms of $\mathbf{U}$ as follows: $\mathbf{F}\varphi = \top\, \mathbf{U}\, \varphi$, and $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$. Given a set of variables Ψ, let $LTL(\Psi)$ be the set of LTL formulae over Ψ; where Ψ is clear from the context, we write LTL. We interpret formulae of LTL with respect to pairs (ρ, t) where ρ is a run and $t \in \mathbb{N}$ is a temporal index into ρ. Any given LTL formula may be true at none or multiple time points on a run; for example, a formula $\mathbf{X}p$ will be true at a time point $t \in \mathbb{N}$ on a run ρ if p is true on run ρ at time $t + 1$. We will write $(\rho, t) \models \varphi$ to mean that $\varphi \in LTL$ is true at time $t \in \mathbb{N}$ on run ρ. The rules defining when formulae are true (i.e., the semantics of LTL) are defined as follows:

- $(\rho, t) \models \top$;
- $(\rho, t) \models x$ iff $x \in \mathcal{L}(s(\rho, t))$ (where $x \in \Phi$);
- $(\rho, t) \models \neg\varphi$ iff it is not the case that $(\rho, t) \models \varphi$;
- $(\rho, t) \models \varphi \vee \psi$ iff $(\rho, t) \models \varphi$ or $(\rho, t) \models \psi$;
- $(\rho, t) \models \mathbf{X}\varphi$ iff $(\rho, t + 1) \models \varphi$;
- $(\rho, t) \models \varphi\, \mathbf{U}\, \psi$ iff, for some $t' \geq t$, $(\rho, t') \models \psi$ and $(\rho, t'') \models \varphi$ for all $t \leq t'' < t'$.

We write $\rho \models \varphi$ for $(\rho, 0) \models \varphi$, in which case we say that ρ *satisfies* φ. A formula φ is *satisfiable* if there is some run satisfying φ. A strategy σ *realizes* φ in $\mathcal{E}$, written $\sigma \rhd_{\mathcal{E}} \varphi$, if ρ satisfies φ for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma, s_0)$, that is, if every run compatible with σ satisfies the LTL formula. The formula φ is *realizable* in $\mathcal{E}$, written $\rhd_{\mathcal{E}}\varphi$, if there is a strategy σ such that $\sigma \rhd_{\mathcal{E}} \varphi$. We write $\sigma \rhd_{\mathcal{E},s} \varphi$ to indicate that every run consistent with σ starting from state $s \in \mathcal{S}$ satisfies φ. While LTL satisfiability is PSPACE-complete [31], LTL synthesis/relisability is 2EXPTIME-complete [27]. The following results are useful in what follows:

▶ **Lemma 1.** *Given some $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{L}, s_0)$, a state $s \in \mathcal{S}$, and an LTL formula φ, checking whether there is a run ρ of $\mathcal{E}$ starting in s such that $\rho \models \varphi$ is PSPACE-complete.*

Proof. For membership, we can reduce to LTL satisfiability: we can define a new LTL formula $\psi_{\mathcal{E}}$ which represents the behaviour of the environment, and check whether $\psi_{\mathcal{E}} \wedge \varphi$ is satisfiable. For hardness, we can reduce LTL model checking [31]. ◀

▶ **Lemma 2.** *Given an environment $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{L}, s_0)$, a state $s \in \mathcal{S}$, and an LTL formula φ, checking whether there is a strategy σ such that $\sigma \rhd_{\mathcal{E},s} \varphi$ is 2EXPTIME-complete.*

Proof. For membership, we can reduce to LTL realizability: we can define a new LTL formula $\psi_{\mathcal{E}}$ which represents the behaviour of the environment, and check whether $\psi_{\mathcal{E}} \to \varphi$ is realizable. For hardness, we can reduce LTL synthesis [27]. ◀

2.4 Agents

The agents we consider are taskable: an agent is designed to operate in a specific environment $\mathcal{E}$, but may be assigned different tasks to carry out in the environment. The basic approach we adopt is to specify a task for an agent via an LTL formula Υ: an agent succeeds with the task Υ if it chooses a strategy that realizes Υ. Crucially, tasks for agents are *mutable*: we might initially instruct an agent to achieve a task formula Υ_1, but later *change* our instructions by giving the agent a task Υ_2. Much of the remainder of this paper is concerned with understanding precisely how an agent should interpret a new instruction like this.

We intend our agents to operate safely. To this end, we assume they have a background *safety requirement*, specified as an LTL formula ζ[1]. The absolute priority for an agent is to ensure that ζ is realized. While tasks Υ are mutable, safety requirements ζ are immutable: no matter what instructions the agent receives relating to tasks, the agent *must* ensure that its safety requirement is fulfilled if it can do so in the first place.

Finally, we assume that agents have a budget, which we denote by β. This is a real number which indicates how much cost we are willing for our agent to incur, where the cost of a run is measured using the limit of means approach (see above). Thus, the budget is not an absolute limit, but indicates maximum *average* expenditure that we are willing for an agent to incur. Formally, then, an agent A is defined by a structure $A = (\Upsilon, \zeta, \sigma, \beta)$ where:

- Υ is an LTL formula which specifies the currently assigned task of the agent;
- ζ is an LTL formula representing the safety requirements of the agent;
- $\sigma \in \Sigma$ is the agent's current strategy; and finally,
- $\beta \in \mathbb{R}$ is the agent's budget.

We will say an agent $(\Upsilon, \zeta, \sigma, \beta)$ in an environment $\mathcal{E}$ is:

- *safe* if the strategy ensures the agent's safety requirement is realized: $\sigma \rhd_{\mathcal{E}} \zeta$;
- *sound* if the strategy achieves the task: $\sigma \rhd_{\mathcal{E}} \Upsilon$; and
- *solvent* if the agent is within budget: $\mathcal{C}(\rho) < \beta$, for each $\rho \in \mathcal{R}(\mathcal{E}, \sigma, s_0)$.

An agent that satisfies all of the above conditions is said to be *properly configured*. A properly configured agent is one that has thus been assigned a task Υ, and that has chosen a strategy to achieve Υ that will do so while respecting both the safety requirement ζ and budget β.

▶ **Theorem 3.** *Given an environment $\mathcal{E}$, a safety requirement ζ, a budget β, and a task Υ, checking whether there is a strategy σ such that $A = (\Upsilon, \zeta, \sigma, \beta)$ is properly configured is 2EXPTIME-complete. Moreover, given an environment $\mathcal{E}$ and an agent $A = (\Upsilon, \zeta, \sigma, \beta)$ for $\mathcal{E}$, checking whether A is properly configured with respect to $\mathcal{E}$ is PSPACE-complete.*

Proof. First, note that the problem of finding σ such that the agent is properly configured amounts to solve the synthesis problem for the formula $\Upsilon \wedge \zeta$ with the additional condition of staying within the budget β for every possible generated run in the environment. This problem corresponds to a special case of an equilibrium synthesis with combined qualitative and quantitative objectives, as it is introduced and solved in [14], and for which a 2EXPTIME procedure has been shown. For hardness, we can reduce LTL synthesis: the transformer function $\mathcal{T}$ can be used to encode a transition relation in an instance of LTL synthesis [27]. For the second part, an agent together with an environment will trace out a Kripke structure $\mathcal{K}$ such that $\mathcal{K} \models \zeta \wedge \Upsilon$ if, and only if, the agent is properly configured, thus membership is in PSPACE. For hardness, one can use a reduction from LTL model-checking [31]. ◀

▶ **Theorem 4.** *For a deterministic environment $\mathcal{E}$, a safety requirement ζ, a budget β, and a task Υ, checking whether there exists a strategy σ such that $A = (\Upsilon, \zeta, \sigma, \beta)$ is properly configured is PSPACE-complete. In contrast, for an environment $\mathcal{E}$ and an agent $A = (\Upsilon, \zeta, \sigma, \beta)$ for $\mathcal{E}$, checking whether A is properly configured with respect to $\mathcal{E}$ is solvable in polynomial time.*

[1] A well-known class of LTL formulae express what are technically called safety properties [32], intuitively capturing the idea that "nothing bad happens." We do not *require* our safety requirements ζ to belong to this class, although in practice, many safety requirements will.

Proof. The first part follows from Lemma 1 (note that for deterministic environments, solvency can be checked in polynomial time). For hardness use LTL model checking: the transformer function $\mathcal{T}$ can encode a transition relation in an instance of LTL model checking [31]. For the second part, an agent together with an environment will trace out a path of the form $a \cdot b^\omega$ with a and b finite words. We can compute a and b in polynomial time by executing the strategy in the environment and watching until we find to have repeated a configuration; we then check $\zeta \wedge \Upsilon$ on this path, which can be done in polynomial time [21], and then check solvency, which can be done in polynomial time in the deterministic case (by computing the average cost over the finite path corresponding to b). ◀

2.5 Progressing an LTL Formula

We use the concept of *progressing* an LTL formula [5]. The idea is as follows. Suppose that an agent has been operating to achieve a task Υ, and in this process has generated a (finite) history h. We then give the agent a new instruction Υ'. Now, it may be that the task Υ has already been discharged in the history h, in which case, when the agent reconfigures itself in light of the new instruction Υ', it can disregard the original task Υ. To make this concrete, consider a (rather contrived) agent with a task $\Upsilon = \mathbf{F}\,\mathsf{beer}$ (must eventually drink beer). The agent configures itself, adopting a strategy to eventually drink beer, and succeeds to do so. Sometime later, a user issues a new instruction Υ'. At this point, however, the task Υ has *already been achieved* (beer has been drunk), and nothing the agent does subsequently will change that. As the agent adjusts its configuration in light of the new instruction, it can take account of that. Similar comments apply to the agent's safety requirement ζ: it may be that the requirement has already been discharged within the finite history h. (In general, of course, we do not think of safety requirements operating in this way: they are typically ongoing properties, which must be maintained infinitely into the future. For example, suppose the agent's safety requirement was $\zeta = \mathbf{G}\neg\mathsf{crash}$. Such a requirement cannot be discharged within a finite history: as the agent adjusts its configuration in light of the new instructions, it must take into account these ongoing requirements.)

The notion of progressing a temporal formula φ through a history h captures the idea of transforming φ into a new formula $\mathsf{prog}(\varphi, h)$ so that it captures all those requirements of φ that have not already been satisfied within h. Formally, we have the following [16, Thm 3.2].

▶ **Theorem 5.** *For every LTL formula φ, finite history h, and run ρ such that $s(\rho, 0) = \mathsf{last}(h)$, there exists an LTL formula $\mathsf{prog}(\varphi, h)$ such that $\underline{h} \cdot \rho \models \varphi$ iff $\rho \models \mathsf{prog}(\varphi, h)$.*

We refer the reader to the definition of the function $\mathsf{prog}(\cdots)$ in [16], and note that the function can be implemented in time polynomial in the size of the history and given formula[2].

We now move on to the main focus of the present article: the issue of giving instructions to agents. The precise setting we consider is as follows. We have an agent $A = (\Upsilon, \zeta, \sigma, \beta)$, operating in an environment $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{L}, s_0)$, which has traced out a history h. The agent is then presented with a new instruction Υ'. How then should the agent reconfigure itself – and in particular, what new strategy σ' should the agent adopt, taking into account the safety requirement ζ and the history h to date? As we will see, there are multiple possible answers to this question: we consider a range of possible instruction types, which differ primarily in the strength of the new instruction given to the taskable agent.

[2] The definition in [16] is defined with respect to states rather than histories: the extension to histories simply requires progressing the input formula progressively through each state in the given history h.

As we noted above, a very natural principle when an agent is asked to adopt a new instruction is the one of *least change*: when reconfiguring yourself, first try to do so while keeping everything else as close to how it was before as possible. Thus, on the one hand, we have instructions of the form "take onboard my new instruction Υ' only if you can do so without changing anything else." On the other hand, we have instructions of the form "drop everything else you are doing and get this done whatever it takes." There are several variants between these two extremes. We emphasise, however, that *no* new instructions can result in the agent releasing its safety requirement: this is immutable – but the satisfaction of the safety requirement needs to be considered in the context of the history h traced out so far.

We define the semantics of each type of instruction using pre- and post-conditions. The pre-conditions define the circumstances under which the instruction can be accommodated. If the pre-conditions of an instruction type are not satisfied, then that instruction will fail, in which case the agent is assumed to be unchanged. The post-conditions specify properties of the agent configuration after the agent has adjusted its configuration to accommodate the instruction, *under the assumption that the pre-condition was satisfied.* We consider four different types of instructions, which we refer to as Type I instructions through to Type IV instructions: Type I instructions are the weakest (in the sense that they make the least demands on an agent); Type IV are the strongest type of instructions.

2.6 Instructions of Type I

The first type of instruction we consider is the weakest form of instruction in our setting. In a Type I instruction, we ask an agent to take on a new task Υ' *only* if it can do so safely, without affecting its current task, and within the originally specified budget. This type of instruction thus embodies the notion of "least change" in a very direct way. With this type of instruction, an agent will *retain* its previous task, irrespective of whether the instruction is successful or not. Formally, our Type I semantics are as follows.

Suppose we have an agent $(\Upsilon, \zeta, \sigma, \beta)$ operating in an environment $\mathcal{E}$, having generated a history h. The agent is then presented with a Type I instruction Υ'. The preconditions for this Type I instruction require that there exists a strategy $\sigma' \in \Sigma$ such that:

(I.1) $\underline{h} \cdot \rho \models \zeta \wedge \Upsilon$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \mathsf{last}(h))$;
(I.2) $\sigma' \rhd_{\mathcal{E}, \mathsf{last}(h)} \Upsilon'$;
(I.3) $\mathcal{C}(\rho) \leq \beta$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \mathsf{last}(h))$.

If these pre-conditions are satisfied, then an agent acting upon a Type I instruction will change to a configuration $(\mathsf{prog}(\Upsilon, h) \wedge \Upsilon', \mathsf{prog}(\zeta, h), \sigma', \beta)$ where $\sigma' \in \Sigma$ satisfies conditions (I.1)–(I.3). Several points are in order with respect to this definition. First, note that the only components modified with the new instruction will be the current task of the agent (extended to include the new instruction) and the chosen strategy: both the safety requirement and budget will remain unchanged after updating the agent.

With respect to (I.1), the conditions require that the adopted strategy will ensure that both the original task Υ and the safety requirement are satisfied by σ'. Note that we could alternatively have expressed this condition as:

$$\rho \models \mathsf{prog}(\zeta, h) \wedge \mathsf{prog}(\Upsilon, h) \quad \text{for every } \rho \in \mathcal{R}(\mathcal{E}, \sigma, \mathsf{last}(h))$$

Condition (I.2) requires that σ' ensures the achievement of the new task Υ'. The main point to note here is that, unlike condition (I.1), this condition *does not* take into account the prior history h. The reason for this is as follows: suppose the original task Υ was $\mathbf{F}\mathsf{beer}$ (eventually drink beer), and the agent succeeded to do so within the history h. Then if the new instruction Υ' was $\mathbf{F}\mathsf{beer}$ (intuitively, "drink another beer"), then if we took h into account when considering the new strategy we would regard Υ' as being already discharged. Thus, when we consider the satisfaction of the new task Υ', we need to ignore the history to date and focus on the run that will be generated subsequently.

Finally, observe that the solvency requirement for an agent adopting the new strategy (condition (I.3)) is given with respect to *future* costs only. This may seem counter-intuitive: should we not take into account costs already incurred? The answer stems from the way we measure costs, using the inferior limit of means. The agent will have been running for a finite time when presented with its new instruction, but will run for an *infinite* time subsequently. In the limit, costs already incurred in the finite history h are irrelevant: all that matters are the *future* costs incurred over the infinite suffix to h.

▶ **Example 6.** Consider the floor of an elderly home where a number of guest rooms are arranged around a corridor, and a robot deployed to patrol the surroundings. The robot can move along the corridor or in one of the guest rooms. Moreover, each room can be either occupied or empty, and the floor might be dirty or clean. The robot has two tasks. The first is the safety requirement to eventually tidy up the corridor every time it gets dirty. This can be represented by the LTL formula $\zeta = \mathbf{G}(\mathsf{dirty} \to \mathbf{F}(\mathsf{tidy}\,\mathbf{U}\,\neg\mathsf{dirty}))$. The second is the assigned task to monitor the occupied room and assist a guest in room i in case they send an assistance request. This can be represented by the formula $\Upsilon_i = \mathbf{G}(\mathsf{call}_i \to \mathbf{F}\mathsf{assist}_i)$. Note that requests from the other rooms will be ignored by the agent. This is because they are marked as empty, and so requests from those rooms would happen for reasons that are not relevant to the robot, e.g., a contractor team is testing/maintaining the room.

Once a new guest arrives in room j, the agent needs to be reconfigured to include the task of listening to requests from the corresponding room. This can be done by the Type I instruction of the form $\Upsilon_j = \mathbf{G}(\mathsf{call}_j \to \mathbf{F}\mathsf{assist}_j)$. Without giving full details, for brevity, the setting above could be represented by the following environment $\mathcal{E}$:

- $\mathcal{S} = \mathsf{Pos} \times 2^{\{1,2,3,4\}} \times \{\mathsf{dirty}, \mathsf{not\text{-}dirty}\}$, where Pos represents the possible positions of the robot (i.e., locations in the corridor or one of the rooms), the set $2^{\{1,2,3,4\}}$ represents the set of rooms from which an assistance request has been made, whereas the third component in $\{\mathsf{dirty}, \mathsf{not\text{-}dirty}\}$ represents the fact that the corridor is dirty in some location or clean all around.
- $\mathcal{A} = \{\mathsf{up}, \mathsf{down}, \mathsf{left}, \mathsf{right}, \mathsf{stop}\}$ allows the robot to navigate along the floor by either moving along the four directions or by stopping.
- $\mathcal{C}(\mathsf{up}) = \mathcal{C}(\mathsf{down}) = \mathcal{C}(\mathsf{left}) = \mathcal{C}(\mathsf{right}) = 1$, while $\mathcal{C}(\mathsf{stop}) = 0$, that is, the robot consumes one unit of power only when it moves along the floor.
- $\mathcal{T}$ is deterministic on Pos, meaning that the value in Pos depends only on the current position and the robots action, and not on the environment reaction to this. Whereas $\mathcal{T}$ is nondeterministic on the other components of the states, which depends on whether the environment calls for assistance in any of the room and whether it activates the signal that the corridor is somewhere dirty.
- $\mathcal{L}(\mathsf{room}_i, C, \iota) = \{\mathsf{assist}_i\} \cup C \cup \{\iota\}$ meaning that if the robot is in room room_i, then it is assisting the guest in the same room, while the assistance requests and the cleaning status of the corridor is copied from the other two components of the state.
  $\mathcal{L}(\mathsf{corridor}, C, \iota) = C \cup \{\mathsf{not\text{-}dirty}\}$ meaning that whenever the robot is in any location of the corridor, it is not assisting any guest but efficiently cleaning the corridor.
- $s_0 = (\mathsf{corridor}_0, \emptyset, \mathsf{not\text{-}dirty})$ is an arbitrary taken initial state of the environment.

▶ **Theorem 7.** *Given an environment $\mathcal{E}$ and a properly configured agent $A = (\Upsilon, \zeta, \sigma, \beta)$, together with a history h generated by σ in the environment, checking whether a Type I instruction Υ' allows for a proper reconfiguration of an agent A after history h is 2EXPTIME-complete.*

Proof. Following Theorem 5, the problem can be reduced to checking whether there exists a strategy σ' such that the agent $A' = (\mathsf{prog}(\Upsilon, h) \wedge \Upsilon', \mathsf{prog}(\zeta, h), \sigma', \beta)$ is properly configured over the environment $\mathcal{E}'$ obtained from $\mathcal{E}$ by replacing the initial state s_0 with $\underline{h}$. Following Theorem 3 it holds that such problem can be solved in 2EXPTIME. For hardness, we can encode LTL synthesis [26]. ◀

2.7 Instructions of Type II

Type II instructions are stronger than Type I: whereas in Type I instructions an agent is asked to adopt the new instruction within the original budget, with a Type II instruction an agent is given a new budget – intuitively, more resources to accomplish the task. Otherwise a Type II instruction is as a Type I.

Suppose we have an agent $(\Upsilon, \zeta, \sigma, \beta)$ operating in an environment $\mathcal{E}$, having generated a history h. The agent is then presented with a Type II instruction (Υ', β'). The pre-conditions for this Type II instruction require that there is a strategy $\sigma' \in \Sigma$ such that:

(II.1) $\underline{h} \cdot \rho \models \zeta \wedge \Upsilon$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \mathsf{last}(h))$;
(II.2) $\sigma' \rhd_{\mathcal{E}, \mathsf{last}(h)} \Upsilon'$;
(II.3) $\mathcal{C}(\rho) \leq \beta'$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \mathsf{last}(h))$.

If the pre-conditions are satisfied, then an agent acting upon a Type II instruction will change to a configuration $(\mathsf{prog}(\Upsilon, h) \wedge \Upsilon'), \mathsf{prog}(\zeta, h), \sigma', \beta')$ where $\sigma' \in \Sigma$ satisfies conditions (II.1)–(II.3). Since β' can be set to any value, using a Type II instruction we can in effect say "forget about your budget limits". (We can also use Type II instructions to give *reduced* budgets.)

▶ **Example 8.** Consider again the setting in Example 6. After a given period of time, the components of the robots might deteriorate, including the battery, which might reduce its maximum capacity. Therefore, the budget β must be adjusted to accommodate this requirement. A Type-II instruction is suitable for this purpose. This can be of the form $(\top, \beta')$ in case we are updating the cost requirement only, or of the form (Υ', β') if we are including a new task requirement to the new configuration.

▶ **Theorem 9.** *Given an environment $\mathcal{E}$ and a properly configured agent $A = (\Upsilon, \zeta, \sigma, \beta)$, together with a history h generated by σ in the environment, checking whether a Type II instruction (Υ', β') allows for a proper reconfiguration of an agent A after history h is 2EXPTIME-complete.*

2.8 Instructions of Type III

In a Type III instruction, we ask an agent to take on a task even if that means dropping its original task; but it should stay within its original budget. Suppose we have an agent $(\Upsilon, \zeta, \sigma, \beta)$ operating in an environment $\mathcal{E}$, having generated a history h. The agent is then presented with a Type III instruction Υ'. The pre-conditions for this Type III instruction require that there is a strategy $\sigma' \in \Sigma$ such that:

(III.1) $\underline{h} \cdot \rho \models \zeta$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \mathsf{last}(h))$;
(III.2) $\sigma' \rhd_{\mathcal{E}, \mathsf{last}(h)} \Upsilon'$;
(III.3) $\mathcal{C}(\rho) \leq \beta$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \mathsf{last}(h))$.

If the pre-conditions are satisfied, then an agent acting upon a Type III instruction will change to a configuration $(\Upsilon', \mathsf{prog}(\zeta, h), \sigma', \beta)$ where $\sigma' \in \Sigma$ satisfies conditions (III.1)–(II.3).

▶ **Example 10.** Consider again the setting described in Example 6. Recall that the task requirement is of the form $\Upsilon = \bigwedge_{i \in N'} \Upsilon_i$, for a given subset $N' \subseteq N$ of the rooms and Υ_i being the requirement for room i.

After a guest left the elderly home, say from room j, it is no longer necessary for the robot to monitor such room. We can use a Type III instruction to remove this task requirement. This can be obtained by using the formula $\Upsilon' = \bigwedge_{i \in N' \setminus \{j\}} \mathsf{prog}(\Upsilon_i, \underline{\mathrm{h}})$ that drops room j from monitoring and reinstates the other rooms at the same time.

▶ **Theorem 11.** *Given an environment $\mathcal{E}$ and a properly configured agent $A = (\Upsilon, \zeta, \sigma, \beta)$, together with a history h generated by σ in the environment, checking whether a Type III instruction Υ' allows for a proper reconfiguration of an agent A after history h is 2EXPTIME-complete.*

Proof. The proof is similar to that of Theorem 7, where the synthesis problem is cast by replacing $\mathsf{prog}(\Upsilon, h) \wedge \Upsilon'$ by just Υ'. ◀

2.9 Instructions of Type IV

In a Type IV instruction, we present an agent with a new task and budget: the agent drops its original task and budget completely, and completely replaces them with those newly presented. This represents the strongest type of instruction we consider here: it may result in all components of an agent other than the safety condition (which is progressed through the prior history) being reconfigured. Suppose we have an agent $(\Upsilon, \zeta, \sigma, \beta)$ operating in an environment $\mathcal{E}$, having generated a history h. The agent is then presented with a Type IV instruction (Υ', β'). The pre-conditions for this Type IV instruction require that there is a strategy $\sigma' \in \Sigma$ such that:

(IV.1) $\underline{\mathrm{h}} \cdot \rho \models \zeta$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \mathsf{last}(h))$;
(IV.2) $\sigma' \rhd_{\mathcal{E}, \mathsf{last}(h)} \Upsilon'$;
(IV.3) $\mathcal{C}(\rho) \leq \beta'$, for every $\rho \in \mathcal{R}(\mathcal{E}, \sigma', \mathsf{last}(h))$.

If the pre-conditions are satisfied, then an agent acting upon a Type III instruction will change to a configuration $(\Upsilon', \mathsf{prog}(\zeta, h), \sigma', \beta')$ where $\sigma' \in \Sigma$ satisfies conditions (IV.1)–(IV.3).

▶ **Theorem 12.** *Given an environment $\mathcal{E}$ and a properly configured agent $A = (\Upsilon, \zeta, \sigma, \beta)$, together with a history h generated by σ in the environment, checking whether a Type IV instruction (Υ', β') allows for a proper reconfiguration of an agent A after history h is 2EXPTIME-complete.*

We know from the optimisation and planning literature that giving instructions to an agent as it operates may deliver solutions that are inferior to solutions computed *before* the agent executes any plan. For example, consider the taskable agent presented in [17, 1], where an autonomous agent is required to move in a grid-like world, much like the robot in our running example, and is requested to find the shortest path between two places in the grid, say X and Y, but under the restriction that some elements must be collected before going from X to Y. While the same situation may arise here, our approach ensures that new tasks will be undertaken *safely*.

3 Related Work & Discussion

We hope to have illustrated that the natural and intuitively simple problem of reconfiguring a taskable agent to accommodate new LTL instructions is actually a rather subtle and complex problem. Formally, we have also shown that solving this general problem, in most settings and for most types of instructions we consider, is 2ExpTime-complete, with only a few exceptions where the decision problems are either PSpace-complete or solvable in polynomial time in some of the simplest cases.

Our work is somewhat related to (and influenced by) work in the semantics of speech acts and agent communication. This work traces its origins least as far back to Wittgenstein's 1953 study of language games, which marked the beginning of the *pragmatic* tradition of language understanding [20]. The best-known work in this tradition is the speech acts paradigm, initiated by [4] and [30]. Within AI, Searle's work led to [8] showing how the pre- and post-conditions of speech acts could be formulated using an AI planning formalism, which paved the way for the development of AI systems that could plan to perform speech acts as part of a natural language generation process [3], and then the semantics of agent communication languages [23, 10]. The main difference with our work is that speech act semantics pre-suppose that agents are *autonomous*, in the sense that they have complete control over their own state and actions. For example, the FIPA `request` communicative act represents an attempt to get an agent to perform an action: it is not an *instruction*, which is conceptually and technically different. Our work differs in that we assume agents are *taskable*: we give one of our agents an instruction and they will attempt to accommodate it as per the various different models we have discussed.

Research on human-robot interactions addresses the problem of robots performing tasks requested by a human through natural language [33, 28, 22, 24]. In [2], it is suggested that robots that can understand and perform human instructions will consist of three levels: (*i*) language-based semantic reasoning on the instruction (high level); (*ii*) formulation of goals in robot symbols and planning to achieve them (mid-level); and (*iii*) action execution (low level). We focus on level (*ii*), and our work can exploit previous works that translate high-level natural language tasks to LTL [29, 7]. Another interesting line of research is the use of LTL instructions in Reinforcement Learning [34, 18]. In [34], LTL is used as a language for specifying multiple tasks to speed up learning by generating the tasks in a way that supports the composition of learned skills. We assume that no learning is needed by our agents: our work is closer to the LTL synthesis paradigm (although combining our work with previous work on LTL-based reinforcement learning [34, 18] would present many interesting research questions). Our work has focused on the idea of *directly* instructing agents with regard to tasks. There are of course other *indirect* ways to influence the activities of agents such as sharing information [12] and setting taxation or rewards [35]. The main difference with our work is that we consider agents that are obligated to follow the instructions that they receive, given that they are consistent with their safety requirements; in contrast, [12] and [35] both assume self-interested agents.

So far, we have said nothing about how an agent will *operationalise* our semantics. In the single agent case, one very natural possibility, given an instruction Υ, is to start by attempting to interpret it in the "least intervention" semantics of Type I instructions: try to accomodate it while continuing with your existing goal. If this is not possible, then move on to semantics that result in greater changes. We might, for example, imagine an agent coming back and asking for an increased budget ("if you give me a bigger budget I can get your original task accomplished *and* the new task"). The multi-agent case is more complex, because the operationalisation will have to take into account how other agents respond to the adoption of a new strategy: it would thus be useful to consider the dynamics of a system after an agent reconfigures itself (how agents might respond). Such issues have been investigated in the game theory literature, but not in the settings we consider here. In this latter case, we might, e.g., look at "responsible" agents that take into account the social welfare of a system as they adopt new strategies. The multi-agent case is the most obvious avenue for future research, wherein ideas from both coopeartive and non-cooperative game theory could shed some light into how to approach that more challenging problem.

Finally, in our model, we have a single safety requirement ζ. It is natural to extend our model to support hierarchies of logically-defined safety requirements $(\zeta_1, \ldots, \zeta_k)$, where an agent is required to first ensure that ζ_1 is satisfied, then ζ_2, and so on.

References

1 Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In Doina Precup and Yee Whye Teh, editors, *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 166–175. PMLR, 2017. URL: http://proceedings.mlr.press/v70/andreas17a.html.

2 Alexandre Antunes, Lorenzo Jamone, Giovanni Saponaro, Alexandre Bernardino, and Rodrigo Ventura. From human instructions to robot actions: Formulation of goals, affordances and probabilistic planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5449–5454. IEEE, 2016.

3 D. E. Appelt. *Planning English Sentences.* Cambridge University Press: Cambridge, England, 1985.

4 J. L. Austin. *How to Do Things With Words.* Oxford University Press: Oxford, England, 1962.

5 Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2):123–191, 2000. doi:10.1016/S0004-3702(99)00071-5.

6 K. Binmore. *Fun and Games: A Text on Game Theory.* D. C. Heath and Company: Lexington, MA, 1992.

7 Adrian Boteanu, Thomas Howard, Jacob Arkin, and Hadas Kress-Gazit. A model for verifiable grounding and execution of complex natural language instructions. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2649–2654. IEEE, 2016.

8 P. R. Cohen and C. R. Perrault. Elements of a plan based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.

9 E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 996–1072. Elsevier, 1990.

10 FIPA. The foundation for intelligent physical agents, 2001. See http://www.fipa.org/.

11 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *TACAS*, volume 6015 of *LNCS*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.

12 John Grant, Sarit Kraus, Michael John Wooldridge, and Inon Zuckerman. Manipulating boolean games through communication. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

13 J. Gutierrez, P. Harrenstein, and M. Wooldridge. Iterated Boolean games. *Information and Computation*, 242:53–79, 2015.

14 Julian Gutierrez, Aniello Murano, Giuseppe Perelli, Sasha Rubin, Thomas Steeples, and Michael Wooldridge. Equilibria for games with combined qualitative and quantitative objectives. *Acta Informatica*, pages 1–26, 2020.

15 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. *Artif. Intell.*, 287:103353, 2020. doi:10.1016/j.artint.2020.103353.

16 Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 452–461. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018.

17 Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Teaching multiple tasks to an RL agent using LTL. In Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar, editors, *AAMAS*, pages 452–461. IFAAMAS, Richland, SC, USA / ACM, 2018. URL: `http://dl.acm.org/citation.cfm?id=3237452`.

18 León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 540–550, 2020.

19 Pat Langley, Nishant Trivedi, and Matt Banister. A command language for taskable virtual agents. In G. Michael Youngblood and Vadim Bulitko, editors, *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, October 11-13, 2010, Stanford, California, USA*. The AAAI Press, 2010. URL: `http://aaai.org/ocs/index.php/AIIDE/AIIDE10/paper/view/2153`.

20 S. C. Levinson. *Pragmatics*. Cambridge University Press: Cambridge, England, 1983.

21 Nicolas Markey and Philippe Schnoebelen. Model checking a path. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR 2003 – Concurrency Theory, 14th International Conference, Marseille, France, September 3-5, 2003, Proceedings*, volume 2761 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2003. `doi:10.1007/978-3-540-45187-7_17`.

22 Pedro Henrique Martins, Luís Custódio, and Rodrigo Ventura. A deep learning approach for understanding natural language commands for mobile service robots. *arXiv preprint*, 2018. `arXiv:1807.03053`.

23 J. Mayfield, Y. Labrou, and T. Finin. Evaluating KQML as an agent communication language. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 347–360. Springer-Verlag: Berlin, Germany, 1996.

24 Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300, 2016.

25 A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth IEEE Symposium on the Foundations of Computer Science (FOCS'77)*, pages 46–57. The Society, 1977.

26 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the Sixteenth ACM Symposium on the Principles of Programming Languages (POPL)*, pages 179–190, January 1989.

27 A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programs (ICALP'89)*, pages 652–671, 1989.

28 Pradip Pramanick, Hrishav Bakul Barua, and Chayan Sarkar. Decomplex: Task planning from complex natural instructions by a collocating robot. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6894–6901. IEEE, 2021.

29 Vasumathi Raman, Constantine Lignos, Cameron Finucane, Kenton CT Lee, Mitchell P Marcus, and Hadas Kress-Gazit. Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language. In *Robotics: Science and Systems*, volume 2, pages 2–1. Citeseer, 2013.

30 J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press: Cambridge, England, 1969.

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

32 A. Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects Comput.*, 6(5):495–512, 1994. `doi:10.1007/BF01211865`.

33 Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:25–55, 2020.

34 Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 452–461, 2018.

35 Michael Wooldridge, Ulle Endriss, Sarit Kraus, and Jérôme Lang. Incentive engineering for boolean games. *Artificial Intelligence*, 195:418–439, 2013.