

EPTCS 370

Proceedings of the
**13th International Symposium on
Games, Automata, Logics and Formal
Verification**

Madrid, Spain, September 21-23, 2022

Edited by: Pierre Ganty and Dario Della Monica

Published: 20th September 2022
DOI: 10.4204/EPTCS.370
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Table of Contents	i
Preface	iii
Invited Presentation: Techniques for Unambiguous Systems	v
<i>Wojciech Czerwiński</i>	
Invited Presentation: Learning Languages of Infinite Words	vi
<i>Dana Fisman</i>	
Invited Presentation: State Complexity of Population Protocols	vii
<i>Javier Esparza</i>	
Invited Presentation: Towards Multiset Semantics Database Theory: How I Learned to Stop Worrying and Love Linear Algebra	viii
<i>Jerzy Marcinkowski</i>	
On the Existential Fragments of Local First-Order Logics with Data	1
<i>Benedikt Bollig, Arnaud Sangnier and Olivier Stietel</i>	
Capturing Bisimulation-Invariant Exponential-Time Complexity Classes	17
<i>Florian Bruse, David Kronenberger and Martin Lange</i>	
Complexity through Translations for Modal Logic with Recursion	34
<i>Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza and Anna Ingolfsdottir</i>	
Schema-Based Automata Determinization	49
<i>Joachim Niehren, Momar Sakho and Antonio Al Serhali</i>	
Generating Tokenizers with Flat Automata	66
<i>Hans de Nivelle and Dina Muktubayeva</i>	
Analyzing Robustness of Angluin’s L* Algorithm in Presence of Noise	81
<i>Igor Khmelnitsky, Serge Haddad, Lina Ye, Benoît Barbot, Benedikt Bollig, Martin Leucker, Daniel Neider and Rajarshi Roy</i>	
Parametric Interval Temporal Logic over Infinite Words	97
<i>Laura Bozzelli and Adriano Peron</i>	
Realizable and Context-Free Hyperlanguages	114
<i>Hadar Frenkel and Sarai Sheinvald</i>	

Controller Synthesis for Timeline-based Games	131
<i>Renato Acampora, Luca Geatti, Nicola Gigante, Angelo Montanari and Valentino Picotti</i>	
CryptoSolve: Towards a Tool for the Symbolic Analysis of Cryptographic Algorithms	147
<i>Dalton Chichester, Wei Du, Raymond Kauffman, Hai Lin, Christopher Lynch, Andrew M. Marshall, Catherine A. Meadows, Paliath Narendran, Veena Ravishankar, Luis Rovira and Brandon Rozek</i>	
Adversarial Formal Semantics of Attack Trees and Related Problems	162
<i>Thomas Brihaye, Sophie Pinchinat and Alexandre Terefenko</i>	
Avoid One's Doom: Finding Cliff-Edge Configurations in Petri Nets	178
<i>Giann Karlo Aguirre-Samboní, Stefan Haar, Loïc Paulevé, Stefan Schwoon and Nick Würdemann</i>	
Comparing Channel Restrictions of Communicating State Machines, High-level Message Sequence Charts, and Multiparty Session Types	194
<i>Felix Stutz and Damien Zufferey</i>	
Characterizing the Decidability of Finite State Automata Team Games with Communication	213
<i>Michael Coulombe and Jayson Lynch</i>	

Preface

This volume contains the proceedings of GandALF 2022, the Thirteenth International Symposium on Games, Automata, Logics, and Formal Verification. The symposium was held in Madrid, Spain, September 21-23, 2022.

The GandALF symposium was established by a group of Italian computer scientists to provide an opportunity for researchers interested in logic for computer science, automata theory, game theory, to gather and discuss the application of formal methods to the specification, design, and verification of complex systems. Previous editions of GandALF were held in Padova, Italy (2021); Brussels, Belgium (2020); Bordeaux, France (2019); Saarbrücken, Germany (2018); Rome, Italy (2017); Catania, Italy (2016); Genoa, Italy (2015); Verona, Italy (2014); Borca di Cadore, Italy (2013); Napoli, Italy (2012); and Minori, Italy (2011 and 2010). It is a forum where people from different areas, and possibly with different backgrounds, can fruitfully interact, with a truly international spirit, as witnessed by the composition of the program and steering committee and by the country distribution of the submitted papers.

The program committee selected 14 papers (out of 20 submissions) for presentation at the symposium. Each paper was reviewed by at least two referees, and the selection was based on originality, quality, and relevance to the topics of the call for papers. The scientific program included presentations on automata, logics for computer science and verification, complexity theory, formal methods and languages, games, synthesis algorithms and security. The program included four invited talks, given by Wojciech Czerwiński (University of Warsaw, Poland), Javier Esparza (Technische Universität München, Germany), Dana Fisman (Ben-Gurion University, Israel), and Jerzy Marcinkowski (University of Wrocław, Poland). We are deeply grateful to them for contributing to this year edition of GandALF.

We would like to thank the authors who submitted papers for consideration, the speakers, the program committee members and the additional reviewers for their excellent work. We also thank Eddie Kohler for the HotCRP conference review software, EPTCS and arXiv for hosting the proceedings; in particular, we thank Rob van Glabbeek for the precise and prompt technical support with issues related with the proceeding publication procedure.

Finally we would like to thank the local organisers, and especially Tania Rodríguez and María Alcaraz for making sure the event ran smoothly.

Dario Della Monica and Pierre Ganty

Program Chairs

- Dario Della Monica, University of Udine (Italy)
- Pierre Ganty, IMDEA Software Institute, Madrid (Spain)

Program Committee

- Christel Baier, TU Dresden (Germany)
- Suguman Bansal, University of Pennsylvania (USA)

- Nathalie Bertrand, Inria (France)
- Filippo Bonchi, University of Pisa (Italy)
- Laura Bozzelli, Università degli Studi di Napoli Federico II (Italy)
- Véronique Bruyère, University of Mons (Belgium)
- David de Frutos Escrig, Universidad Complutense de Madrid (Spain)
- Cezara Drăgoi, AWS
- Mohamed Faouzi Atig, Uppsala University (Sweden)
- Adrian Francalanza, University of Malta (Malta)
- Orna Kupferman, The Hebrew University (Israel)
- Konstantinos Mamouras, Rice University (USA)
- Roland Meyer, TU Braunschweig (Germany)
- Fabio Mogavero, Università degli Studi di Napoli Federico II (Italy)
- Paritosh Pandya, IIT Bombay (India)
- Paweł Parys, University of Warsaw (Poland)
- Guillermo Pérez, University of Antwerp (Belgium)
- Pierre-Alain Reynier, LIS, Aix-Marseille University & CNRS (France)
- Andrea Turrini, Institute of Software, Chinese Academy of Sciences (China)
- Georg Zetsche, Max Planck Institute for Software Systems - MPI-SWS (Germany)

Steering Committee

- Luca Aceto, Reykjavik University (Iceland)
- Javier Esparza, University of Munich (Germany)
- Salvatore La Torre, University of Salerno (Italy)
- Angelo Montanari, University of Udine (Italy)
- Mimmo Parente, University of Salerno (Italy)
- Jean-François Raskin, Université libre de Bruxelles (Belgium)
- Martin Zimmermann, University of Liverpool (UK)

External Reviewers

Adriano Peron, Adrien Pommellet, Antoine Mottet, Bartosz Klin, Eryk Kopczyński, Gaëtan Staquet, Giannicola Scarpa, Ji Guan, Joseph Lallemand, Loïc Hélouët, Moses Ganardi, Ramanathan Srinivasan, Sylvain Lombardy, and Wanwei Liu.

Techniques for Unambiguous Systems

Wojciech Czerwiński (University of Warsaw, Poland)

In recent years it became apparent that many decision problems can be solved efficiently on unambiguous systems (systems in which each word is accepted by at most one run). I will show first a classical approach for solving language equivalence by weighted automata. Then I will present novel techniques, which helped us solve the problem for vector addition systems. Finally I plan to advertise an interesting related problem of multiplicity equivalence, which seems to reveal some connections between the unambiguity problem and other fields of mathematics.

Learning Languages of Infinite Words

Dana Fisman (Ben-Gurion University, Israel)

The problem of inferring an automaton model of a black box system has found many applications in formal methods of system design. If the black-box system implements (or can be abstracted as) a regular language of finite words, it can be inferred in polynomial time using query learning. In this talk we'll discuss the problem we face when the black-box system implements a regular language of infinite words (e.g. the black-box is a reactive system).

While there are automata processing infinite words e.g. (Buechi automata) that have the same structure as automata on finite words (DFAs), learning them seems to be a harder problem. During the talk we will get acquainted with the ideas behind learning algorithms for regular languages, the obstacles in devising learning algorithms for regular languages of infinite words, and state-of-the-art results on learnability of acceptors of regular languages of infinite words.

State Complexity of Population Protocols

Javier Esparza (Technische Universität München, Germany)

Population protocols were introduced by Angluin et al. in 2004 to study the theoretical properties of networks of mobile sensors with very limited computational resources. They have also been proposed as a natural computing model, with molecules, cells, or microorganisms playing the role of sensors.

In a population protocol an arbitrary number of indistinguishable, finite-state agents interact randomly in pairs to collectively decide if their initial global configuration satisfies a given property. The property is formalized as a predicate that maps each initial configuration to an output, 0 or 1. Starting from an initial configuration, the agents eventually agree to the correct output almost surely, and continue producing it forever. The protocol is said to stabilize to the correct output.

Population protocols can decide exactly the semilinear predicates, or, equivalently, the predicates expressible in Presburger arithmetic. Current research concentrates on investigating the amount of resources needed to decide a given predicate. The standard resources, time and memory, translate for population protocols into expected time to stabilization and number of states of each agent. In this talk we concentrate on the state complexity of population protocols, for which matching upper and lower bounds have been found in the last few years.

Towards Multiset Semantics Database Theory: How I Learned to Stop Worrying and Love Linear Algebra

Jerzy Marcinkowski (University of Wrocław, Poland)

For the last ten years, or maybe more, I have been a Database Theorist.

Like all (or at least most of) the Database Theorists I modelled the real-world-database relations, and real-world answers to the queries, as sets. If my query was “Exists y Cat(y) and Owns(x,y)”, and my database happened to contain the tuples “Cat(Tibby)” and “Owns(Andreas, Tibby)” then – clearly – the constant “Andreas” was an element of the answer set.

I studied this model, using the methodology of mathematics. My toolbox was simple. It comprised some specific Database Theory concepts (like Chase) and some simple Computation Theory tricks (including maybe some automata-based arguments). Using this toolbox I constructed reasonings, which were occasionally complicated, but always elementary. It was fun.

But then, in early 2021, someone asked me what would happen to my results concerning Query Determinacy (whatever it is) if I considered a more realistic model, in which database relations, and answers to the queries, are multisets rather than sets. So, if my database knew that Andreas actually has six cats, the answer (multi)set would contain the constant “Andreas” with multiplicity six.

We started to think about it (with my very clever students, Jarosław Kwiecień and Piotr Ostropolski-Nalewaja) and soon we realized that this minor change in the assumptions about the model leads to a major change in the tools we need to study it. This felt to us like entering a completely new world. Suddenly the language and the methods from Linear Algebra appear so naturally that it is fair to say that they impose themselves on the researchers.

Clearly, we were not the first Database Theorists to set foot on this new continent. People have attempted to re-examine old Database Theory results, in the multiset-semantics scenario, for about 30 years now. But such attempts were few. And this is (I guess) exactly because in order to make any progress here one needs to learn totally new tools.

To our surprise we learned that this continent had also been visited before by the Mathematicians. They had constructed a number of structures there, some of them of some use for us. That’s a bit frustrating, because they use different terms to call some notions, and it is not always straightforward to understand how to translate their own results to our language.

My talk will be a story by a traveller astonished by what he saw during his first short trip to the little known continent of Multiset-Semantics Database Theory.

To give you a glimpse of the techniques that appear there I will present some results and arguments from our PODS 2022 paper “Determinacy of Real Conjunctive Queries. The Boolean Case” (with Kwiecień and Ostropolski-Nalewaja). But I will also mention the adventures of other travellers to this new land, with particular emphasis on the famous Conjunctive Query Containment Problem.

On the Existential Fragments of Local First-Order Logics with Data

Benedikt Bollig

CNRS, LMF, ENS Paris-Saclay
Université Paris-Saclay, France

Arnaud Sangnier

IRIF, Université Paris Cité
CNRS, France

Olivier Stietel

CNRS, LMF, ENS Paris-Saclay
Université Paris-Saclay, France
IRIF, Université Paris Cité
CNRS, France

We study first-order logic over unordered structures whose elements carry a finite number of data values from an infinite domain which can be compared wrt. equality. As the satisfiability problem for this logic is undecidable in general, in a previous work, we have introduced a family of local fragments that restrict quantification to neighbourhoods of a given reference point. We provide here the precise complexity characterisation of the satisfiability problem for the existential fragments of this local logic depending on the number of data values carried by each element and the radius of the considered neighbourhoods.

1 Introduction

First-order data logic has emerged to specify properties involving infinite data domains. Potential applications include XML reasoning and the specification of concurrent systems and distributed algorithms. The idea is to extend classic mathematical structures by a mapping that associates with every element of the universe a value from an infinite domain. When comparing data values only for equality, this view is equivalent to extending the underlying signature by a binary relation symbol whose interpretation is restricted to an equivalence relation.

Data logics over word and tree structures were studied in [1, 2]. In particular, the authors showed that two-variable first-order logic on words has a decidable satisfiability problem. Other types of data logics allow *two* data values to be associated with an element [12, 13], though they do not assume a linearly ordered or tree-like universe. Again, satisfiability turned out to be decidable for the two-variable fragment of first-order logic. Other notable extensions, either to multiple data values or to totally ordered data domains, include [5, 11, 15, 17].

When considering an arbitrary number of first-order variables, which we do in this paper, the decidability frontier is quickly crossed without further constraints as soon as the number of allowed data in gretar then two [10]. One of the restrictions we consider here is locality, an essential concept in first-order logic. It is well known that first-order logic is only able to express local properties: a first-order formula can always be written as a combination of properties of elements that have limited, i.e., bounded by a given radius, distance from some reference points [8, 9]. In the presence of (several) data values, imposing a corresponding locality restriction on a logic can help ensuring decidability of its satisfiability problem.

In previous work, we considered a local fragment of first-order data logic over structures whose elements (i) are unordered (as opposed to, e.g., words or trees), and (ii) each carries two data values. We showed that the fragment has a decidable satisfiability problem when restricting local properties to radius 1, while it is undecidable for any radius greater than 1.

In the present paper, we study orthogonal local fragments where global quantification is restricted to being existential (while quantification inside a local property is still unrestricted). We obtain decidability for (i) radius 1 and an arbitrary number of data values, and for (ii) radius 2 and two data values. In all cases, we provide tight complexity upper and lower bounds. Moreover, these results mark the exact decidability frontier: satisfiability is undecidable as soon as we consider radius 3 in presence of two data values, or radius 2 together with three data values.

To give a possible application domain of our logic, consider distributed algorithms running on a cloud of processes. Those algorithms are usually designed to be correct independently of the number of processes executing them. Every process gets some inputs and produces some outputs, usually from an infinite domain. These may include process identifiers, nonces, etc. Inputs and outputs together determine the behavior of a distributed algorithm. A simple example is leader election, where every process gets a unique id, whereas the output should be the id of the elected leader and so be the same for all processes. To formalize correctness properties and to define the intended input-output relation, it is hence essential to have suitable data logics at hand.

Outline. The paper is structured as follows. In Section 2, we recall important notions such as structures and first-order logic, and we introduce the local fragments considered in this paper. Section 3 presents the decidable cases, whereas, in Section 4, we show that all remaining cases lead to undecidability.

This work was partly supported by the project ANR FREDDA (ANR-17-CE40-0013).

2 Structures and first-order logic

2.1 Data Structures

We define here the class of models we are interested in. It consists of sets of nodes containing data values with the assumption that each node is labeled by a set of predicates and carries the same number of values. We consider hence Σ a finite set of unary relation symbols (sometimes called unary predicates) and an integer $D \geq 0$. A D -data structure over Σ is a tuple $\mathfrak{A} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1, \dots, f_D)$ (in the following, we simply write $(A, (P_\sigma), f_1, \dots, f_D)$) where A is a nonempty finite set, $P_\sigma \subseteq A$ for all $\sigma \in \Sigma$, and f_i s are mappings $A \rightarrow \mathbb{N}$. Intuitively A represents the set of nodes and $f_i(a)$ is the i -th data value carried by a for each node $a \in A$. For $X \subseteq A$, we let $Val_{\mathfrak{A}}(X) = \{f_i(a) \mid a \in X, i \in \{1, \dots, D\}\}$. The set of all D -data structures over Σ is denoted by $\text{Data}[\Sigma, D]$.

While this representation is often very convenient to represent data values, a more standard way of representing mathematical structures is in terms of binary relations. For every $(i, j) \in \{1, \dots, D\} \times \{1, \dots, D\}$, the mappings f_1, \dots, f_D determine a binary relation $i \sim_j^{\mathfrak{A}} \subseteq A \times A$ as follows: $a \sim_j^{\mathfrak{A}} b$ iff $f_i(a) = f_j(b)$. We may omit the superscript \mathfrak{A} if it is clear from the context and if $D = 1$, as there will be only one relation, we may write \sim for $1 \sim_1$.

2.2 First-Order Logic

Let $\mathcal{V} = \{x, y, \dots\}$ be a countably infinite set of variables. The set $\text{dFO}[\Sigma, D]$ of first-order formulas interpreted over D -data structures over Σ is inductively given by the grammar $\varphi ::= \sigma(x) \mid x \sim_j y \mid x = y \mid \varphi \vee \varphi \mid \neg \varphi \mid \exists x. \varphi$, where x and y range over \mathcal{V} , σ ranges over Σ , and $i, j \in \{1, \dots, D\}$. We use standard abbreviations such as \wedge for conjunction and \Rightarrow for implication. We write $\varphi(x_1, \dots, x_k)$ to indicate that the free variables of φ are among x_1, \dots, x_k . We call φ a *sentence* if it does not contain free variables.

For $\mathfrak{A} = (A, (P_\sigma), f_1, \dots, f_D) \in \text{Data}[\Sigma, D]$ and a formula $\varphi \in \text{dFO}[\Sigma, D]$, the satisfaction relation $\mathfrak{A} \models_I \varphi$ is defined wrt. an interpretation function $I : \mathcal{V} \rightarrow A$. The purpose of I is to assign an interpretation to every (free) variable of φ so that φ can be assigned a truth value. For $x \in \mathcal{V}$ and $a \in A$, the interpretation function $I[x/a]$ maps x to a and coincides with I on all other variables. We then define:

$$\begin{aligned} \mathfrak{A} \models_I \sigma(x) & \text{ if } I(x) \in P_\sigma & \mathfrak{A} \models_I \varphi_1 \vee \varphi_2 & \text{ if } \mathfrak{A} \models_I \varphi_1 \text{ or } \mathfrak{A} \models_I \varphi_2 \\ \mathfrak{A} \models_I x \underset{i}{\sim}_j y & \text{ if } I(x) \underset{i}{\sim}_j^{\mathfrak{A}} I(y) & \mathfrak{A} \models_I \neg \varphi & \text{ if } \mathfrak{A} \not\models_I \varphi \\ \mathfrak{A} \models_I x = y & \text{ if } I(x) = I(y) & \mathfrak{A} \models_I \exists x. \varphi & \text{ if there is } a \in A \text{ s.t. } \mathfrak{A} \models_{I[x/a]} \varphi \end{aligned}$$

Finally, for a data structure $\mathfrak{A} = (A, (P_\sigma), f_1, \dots, f_D)$, a formula $\varphi(x_1, \dots, x_k)$ and $a_1, \dots, a_k \in A$, we write $\mathfrak{A} \models \varphi(a_1, \dots, a_k)$ if there exists an interpretation function I such that $\mathfrak{A} \models_{I[x_1/a_1] \dots [x_k/a_k]} \varphi$. In particular, for a sentence φ , we write $\mathfrak{A} \models \varphi$ if there exists an interpretation function I such that $\mathfrak{A} \models_I \varphi$.

Example 1 Assume a unary predicate $\text{leader} \in \Sigma$. The following formula from $\text{dFO}[\Sigma, 2]$ expresses correctness of a leader-election algorithm: (i) there is a unique process that has been elected leader, and (ii) all processes agree, in terms of their output values (their second data), on the identity (the first data) of the leader: $\exists x. (\text{leader}(x) \wedge \forall y. (\text{leader}(y) \Rightarrow y = x)) \wedge \forall y. \exists x. (\text{leader}(x) \wedge x \underset{1}{\sim}_2 y)$.

We are interested here in the satisfiability problem for these logics. Let \mathcal{F} denote a generic class of first-order formulas, parameterized by Σ and D . In particular, for $\mathcal{F} = \text{dFO}$, we have that $\mathcal{F}[\Sigma, D]$ is the class $\text{dFO}[\Sigma, D]$. The satisfiability problem for \mathcal{F} wrt. D -data structures is defined as follows:

$\text{DATASAT}(\mathcal{F}, D)$	
Input:	A finite set Σ and a sentence $\varphi \in \mathcal{F}[\Sigma, D]$.
Question:	Is there $\mathfrak{A} \in \text{Data}[\Sigma, D]$ such that $\mathfrak{A} \models \varphi$?

The following negative result (see [10, Theorem 1]) calls for restrictions of the general logic.

Theorem 1 [10] *The problem $\text{DATASAT}(\text{dFO}, 2)$ is undecidable, even when we require that $\Sigma = \emptyset$ and we do not use $\underset{1}{\sim}_2$ and $\underset{2}{\sim}_1$ in the considered formulas.*

2.3 Local First-Order Logic and its existential fragment

We are interested in logics combining the advantages of $\text{dFO}[\Sigma, D]$, while preserving decidability. With this in mind, we have introduced in [3], for the case of two data values, a *local* restriction, where the scope of quantification in the presence of free variables is restricted to the view of a given element. We present now the definition of such restrictions adapted to the case of many data values.

First, the view of a node a includes all elements whose distance to a is bounded by a given radius. It is formalized using the notion of a Gaifman graph (for an introduction, see [14]). We use here a variant that is suitable for our setting and that we call *data graph*. Given a data structure $\mathfrak{A} = (A, (P_\sigma), f_1, \dots, f_D) \in \text{Data}[\Sigma, D]$, we define its *data graph* $\mathcal{G}(\mathfrak{A}) = (V_{\mathcal{G}(\mathfrak{A})}, E_{\mathcal{G}(\mathfrak{A})})$ with set of vertices $V_{\mathcal{G}(\mathfrak{A})} = A \times \{1, \dots, D\}$ and set of edges $E_{\mathcal{G}(\mathfrak{A})} = \{((a, i), (b, j)) \in V_{\mathcal{G}(\mathfrak{A})} \times V_{\mathcal{G}(\mathfrak{A})} \mid a = b \text{ or } a \underset{i}{\sim}_j b\}$. Figure 1a provides an example of the graph $\mathcal{G}(\mathfrak{A})$ for a data structure with 2 data values.

We then define the distance $d^{\mathfrak{A}}((a, i), (b, j)) \in \mathbb{N} \cup \{\infty\}$ between two elements (a, i) and (b, j) from $A \times \{1, \dots, D\}$ as the length of the shortest path from (a, i) to (b, j) in $\mathcal{G}(\mathfrak{A})$. For $a \in A$ and $r \in \mathbb{N}$, the *radius- r -ball around a* is the set $B_r^{\mathfrak{A}}(a) = \{(b, j) \in V_{\mathcal{G}(\mathfrak{A})} \mid d^{\mathfrak{A}}((a, i), (b, j)) \leq r \text{ for some } i \in \{1, \dots, D\}\}$. This ball contains the elements of $V_{\mathcal{G}(\mathfrak{A})}$ that can be reached from $(a, 1), \dots, (a, D)$ through a path of length at most r . On Figure 1a the blue nodes represent $B_2^{\mathfrak{A}}(a)$.

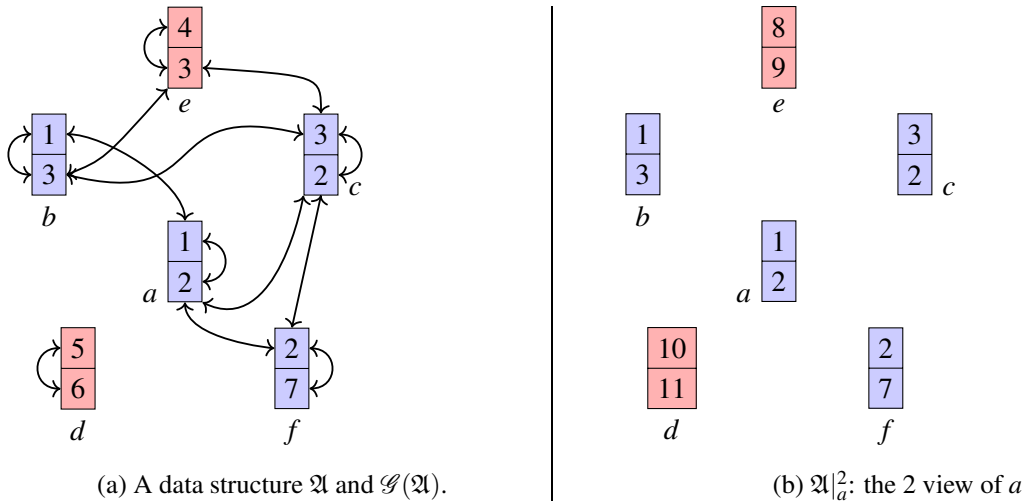


Figure 1

We now define the r -view of an element a in the D -data structure \mathfrak{A} . Intuitively it is a D -data structure with the same elements as \mathfrak{A} but where the data values which are not in the radius- r ball around a are changed with new values all different one from each other. Let $f_{\text{new}} : A \times \{1, \dots, D\} \rightarrow \mathbb{N} \setminus \text{Val}_{\mathfrak{A}}(A)$ be an injective mapping. The r -view of a in \mathfrak{A} is the structure $\mathfrak{A}_a^r = (A, (P_\sigma), f'_1, \dots, f'_n) \in \text{Data}[\Sigma, D]$ where its universe is the same as the one of \mathfrak{A} and the unary predicates stay the same and $f'_i(b) = f_i(b)$ if $(b, i) \in B_r^{\mathfrak{A}}(a)$, and $f'_i(b) = f_{\text{new}}((b, i))$ otherwise. On Figure 1b, the structure \mathfrak{A}_a^2 is depicted where the values of the red nodes, not belonging to $B_2^{\mathfrak{A}}(a)$ have been replaced by fresh values not in $\{1, \dots, 7\}$.

We are now ready to present the logic r -Loc-dFO $[\Sigma, D]$, where $r \in \mathbb{N}$, interpreted over structures from $\text{Data}[\Sigma, D]$. It is given by the grammar

$$\varphi ::= \langle\langle \psi \rangle\rangle_x^r \mid x = y \mid \exists x. \varphi \mid \varphi \vee \varphi \mid \neg \varphi$$

where ψ is a formula from dFO $[\Sigma, D]$ with (at most) one free variable x . This logic uses the *local modality* $\langle\langle \psi \rangle\rangle_x^r$ to specify that the formula ψ should be interpreted over the r -view of the element associated to the variable x . For $\mathfrak{A} \in \text{Data}[\Sigma, D]$ and an interpretation function I , we have indeed $\mathfrak{A} \models_I \langle\langle \psi \rangle\rangle_x^r$ iff $\mathfrak{A}_{I(x)}^r \models_I \psi$.

Example 2 We now illustrate what can be specified by formulas in the logic 1-Loc-dFO $[\Sigma, 2]$. We can rewrite the formula from Example 1 so that it falls into our fragment as follows: $\exists x. (\langle\langle \text{leader}(x) \rangle\rangle_x^1 \wedge \forall y. (\langle\langle \text{leader}(y) \rangle\rangle_y^1 \Rightarrow x = y)) \wedge \forall y. \langle\langle \exists x. \text{leader}(x) \wedge y \text{ }_2 \sim_1 x \rangle\rangle_y^1$. The next formula specifies an algorithm in which all processes suggest a value and then choose a new value among those that have been suggested at least twice: $\forall x. \langle\langle \exists y. \exists z. y \neq z \wedge x \text{ }_2 \sim_1 y \wedge x \text{ }_2 \sim_1 z \rangle\rangle_x^1$. We can also specify partial renaming, i.e., two output values agree only if their input values are the same: $\forall x. \langle\langle \forall y. (x \text{ }_2 \sim_2 y \Rightarrow x \text{ }_1 \sim_1 y) \rangle\rangle_x^1$. Conversely, the formula $\forall x. \langle\langle \forall y. (x \text{ }_1 \sim_1 y \Rightarrow x \text{ }_2 \sim_2 y) \rangle\rangle_x^1$ specifies partial fusion of equivalences classes.

In [3], we have studied the decidability status of the satisfiability problem for r -Loc-dFO $[\Sigma, 2]$ with $r \geq 1$ and we have shown that $\text{DATASAT}(2\text{-Loc-dFO}, 2)$ is undecidable and that $\text{DATASAT}(1\text{-Loc-dFO}, 2)$ is decidable when restricting the formulas (and the view of elements) to binary relations belonging to the set $\{1 \sim_1, 2 \sim_2, 1 \sim_2\}$. Whether $\text{DATASAT}(1\text{-Loc-dFO}, 2)$ in its full generality is decidable or not remains an open problem.

We wish to study here the existential fragment of r -Loc-dFO $[\Sigma, D]$ (with $r \geq 1$ and $D \geq 1$) and establish when its satisfiability problem is decidable. This fragment, denoted by \exists - r -Loc-dFO $[\Sigma, D]$, is given by the grammar

$$\varphi ::= \langle\langle \psi \rangle\rangle_x^r \mid x = y \mid \neg(x = y) \mid \exists x. \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

where ψ is a formula from dFO $[\Sigma, D]$ with (at most) one free variable x . The quantifier free fragment qf- r -Loc-dFO $[\Sigma, D]$ is defined by the grammar $\varphi ::= \langle\langle \psi \rangle\rangle_x^r \mid x = y \mid \neg(x = y) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$.

Remark 1 *Note that for both these fragments, we do not impose any restrictions on the use of quantifiers in the formula ψ located under the local modality $\langle\langle \psi \rangle\rangle_x^r$.*

3 Decidability results

We show here decidability of DATASAT(\exists -2-Loc-dFO, 2) and, for all $D \geq 0$, DATASAT(\exists -1-Loc-dFO, D).

3.1 Preliminary results: 0 and 1 data values

We introduce two preliminary results we shall use in this section to obtain new decidability results. First, note that formulas in dFO $[\Sigma, 0]$ (i.e. where no data is considered) correspond to first order logic formulas with a set of predicates and equality test as a unique relation. As mentioned in Chapter 6.2.1 of [4], these formulas belong to the *Löwenheim class with equality* also called as the relational monadic formulas, and their satisfiability problem is in NEXP. Furthermore, thanks to [6] (Theorem 11), we know that this latter problem is NEXP-hard even if one considers formulas which use only two variables.

Theorem 2 DATASAT(dFO, 0) is NEXP-complete.

In [16], the authors study the satisfiability problem for Hybrid logic over Kripke structures where the transition relation is an equivalence relation, and they show that it is N2EXP-complete. Furthermore in [7], it is shown that Hybrid logic can be translated to first-order logic in polynomial time and this holds as well for the converse translation. Since 1-data structures can be interpreted as Kripke structures with one equivalence relation, altogether this allows us to obtain the following preliminary result about the satisfiability problem of dFO $[\Sigma, 1]$.

Theorem 3 DATASAT(dFO, 1) is N2EXP-complete.

3.2 Two data values and balls of radius 2

In this section, we prove that the satisfiability problem for the existential fragment of local first-order logic with two data values and balls of radius two is decidable. To obtain this result we provide a reduction to the satisfiability problem for first-order logic over 1-data structures. Our reduction is based on the following intuition. Consider a 2-data structure $\mathfrak{A} = (A, (P_\sigma), f_1, f_2) \in \text{Data}[\Sigma, 2]$ and an element $a \in A$. If we take an element b in $B_2^{\mathfrak{A}}(a)$, the radius-2-ball around a , we know that either $f_1(b)$ or $f_2(b)$ is a common value with a . In fact, if b is at distance 1 of a , this holds by definition and if b is at distance 2 then b shares an element with c at distance 1 of a and this element has to be shared with a as well so b ends to be at distance 1 of a . The trick consists then in using extra-labels for elements sharing a value with a that can be forgotten and to keep only the value of b not present in a , this construction leading to a 1-data structure. It remains to show that we can ensure that a 1-data structure is the fruit of this

construction in a formula of $\text{dFO}[\Sigma', 1]$ (where Σ' is obtained from Σ by adding extra predicates).

The first step for our reduction consists in providing a characterisation for the elements located in the radius-1-ball and the radius-2-ball around another element.

Lemma 1 *Let $\mathfrak{A} = (A, (P_\sigma), f_1, f_2) \in \text{Data}[\Sigma, 2]$ and $a, b \in A$ and $j \in \{1, 2\}$. We have:*

1. $(b, j) \in B_1^{\mathfrak{A}}(a)$ iff there is $i \in \{1, 2\}$ such that $a \sim_j^{\mathfrak{A}} b$.
2. $(b, j) \in B_2^{\mathfrak{A}}(a)$ iff there exists $i, k \in \{1, 2\}$ such that $a \sim_k^{\mathfrak{A}} b$.

Proof: We show both statements:

1. Since $(b, j) \in B_1^{\mathfrak{A}}(a)$, by definition we have either $b = a$ and in that case $a \sim_j^{\mathfrak{A}} b$ holds, or $b \neq a$ and necessarily there exists $i \in \{1, 2\}$ such that $a \sim_j^{\mathfrak{A}} b$.
2. First, if there exists $i, k \in \{1, 2\}$ such that $a \sim_k^{\mathfrak{A}} b$, then $(b, k) \in B_1^{\mathfrak{A}}(a)$ and $(b, j) \in B_2^{\mathfrak{A}}(a)$ by definition. Assume now that $(b, j) \in B_2^{\mathfrak{A}}(a)$. Hence there exists $i \in \{1, 2\}$ such that $d^{\mathfrak{A}}((a, i), (b, j)) \leq 2$. We perform a case analysis on the value of $d^{\mathfrak{A}}((a, i), (b, j))$.
 - **Case** $d^{\mathfrak{A}}((a, i), (b, j)) = 0$. In that case $a = b$ and $i = j$ and we have $a \sim_j^{\mathfrak{A}} b$.
 - **Case** $d^{\mathfrak{A}}((a, i), (b, j)) = 1$. In that case, $((a, i), (b, j))$ is an edge in the data graph $\mathcal{G}(\mathfrak{A})$ of \mathfrak{A} which means that $a \sim_j^{\mathfrak{A}} b$ holds.
 - **Case** $d^{\mathfrak{A}}((a, i), (b, j)) = 2$. Note that we have by definition $a \neq b$. Furthermore, in that case, there is $(c, k) \in A \times \{1, 2\}$ such that $((a, i), (c, k))$ and $((c, k), (b, j))$ are edges in $\mathcal{G}(\mathfrak{A})$. If $c \neq a$ and $c \neq b$, this implies that $a \sim_k^{\mathfrak{A}} c$ and $c \sim_j^{\mathfrak{A}} b$, so $a \sim_j^{\mathfrak{A}} b$ and $d^{\mathfrak{A}}((a, i), (b, j)) = 1$ which is a contradiction. If $c = a$ and $c \neq b$, this implies that $a \sim_j^{\mathfrak{A}} b$. If $c \neq a$ and $c = b$, this implies that $a \sim_k^{\mathfrak{A}} b$.

□

We consider a formula $\varphi = \exists x_1 \dots \exists x_n. \varphi_{qf}(x_1, \dots, x_n)$ of \exists -2-Loc-dFO $[\Sigma, 2]$ in prenex normal form, i.e., such that $\varphi_{qf}(x_1, \dots, x_n) \in \text{qf-2-Loc-dFO}[\Sigma, 2]$. We know that there is a structure $\mathfrak{A} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2)$ in $\text{Data}[\Sigma, 2]$ such that $\mathfrak{A} \models \varphi$ if and only if there are $a_1, \dots, a_n \in A$ such that $\mathfrak{A} \models \varphi_{qf}(a_1, \dots, a_n)$.

Let $\mathfrak{A} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2)$ be a structure in $\text{Data}[\Sigma, 2]$ and a tuple $\vec{a} = (a_1, \dots, a_n)$ of elements in A^n . We shall present the construction of a 1-data structure $[[\mathfrak{A}]]_{\vec{a}}$ in $\text{Data}[\Sigma', 1]$ (with $\Sigma \subseteq \Sigma'$) with the same set of nodes as \mathfrak{A} , but where each node carries a single data value. In order to retrieve the data relations that hold in \mathfrak{A} while reasoning over $[[\mathfrak{A}]]_{\vec{a}}$, we introduce extra-predicates in Σ' to establish whether a node shares a common value with one of the nodes among a_1, \dots, a_n in \mathfrak{A} .

We now explain formally how we build $[[\mathfrak{A}]]_{\vec{a}}$. Let $\Gamma_n = \{a_p[i, j] \mid p \in \{1, \dots, n\}, i, j \in \{1, 2\}\}$ be a set of new unary predicates and $\Sigma' = \Sigma \cup \Gamma_n$. For every element $b \in A$, the predicates in Γ_n are used to keep track of the relation between the data values of b and the one of a_1, \dots, a_n in \mathfrak{A} . Formally, we define $P_{a_p[i, j]} = \{b \in A \mid \mathfrak{A} \models a_p \sim_j b\}$. We now define a data function $f : A \rightarrow \mathbb{N}$. We recall for this matter that $\text{Val}_{\mathfrak{A}}(\vec{a}) = \{f_1(a_1), f_2(a_1), \dots, f_1(a_n), f_2(a_n)\}$ and let $f_{\text{new}} : A \rightarrow \mathbb{N} \setminus \text{Val}_{\mathfrak{A}}(A)$ be an injection. For every $b \in A$, we set:

$$f(b) = \begin{cases} f_2(b) & \text{if } f_1(b) \in \text{Val}_{\mathfrak{A}}(\vec{a}) \text{ and } f_2(b) \notin \text{Val}_{\mathfrak{A}}(\vec{a}) \\ f_1(b) & \text{if } f_1(b) \notin \text{Val}_{\mathfrak{A}}(\vec{a}) \text{ and } f_2(b) \in \text{Val}_{\mathfrak{A}}(\vec{a}) \\ f_{\text{new}}(b) & \text{otherwise} \end{cases}$$

Hence depending if $f_1(b)$ or $f_2(b)$ is in $\text{Val}_{\mathfrak{A}}(\vec{a})$, it splits the elements of \mathfrak{A} in four categories. If $f_1(b)$ and $f_2(b)$ are in $\text{Val}_{\mathfrak{A}}(\vec{a})$, the predicates in Γ_n allow us to retrieve all the data values of b . Given $j \in \{1, 2\}$,

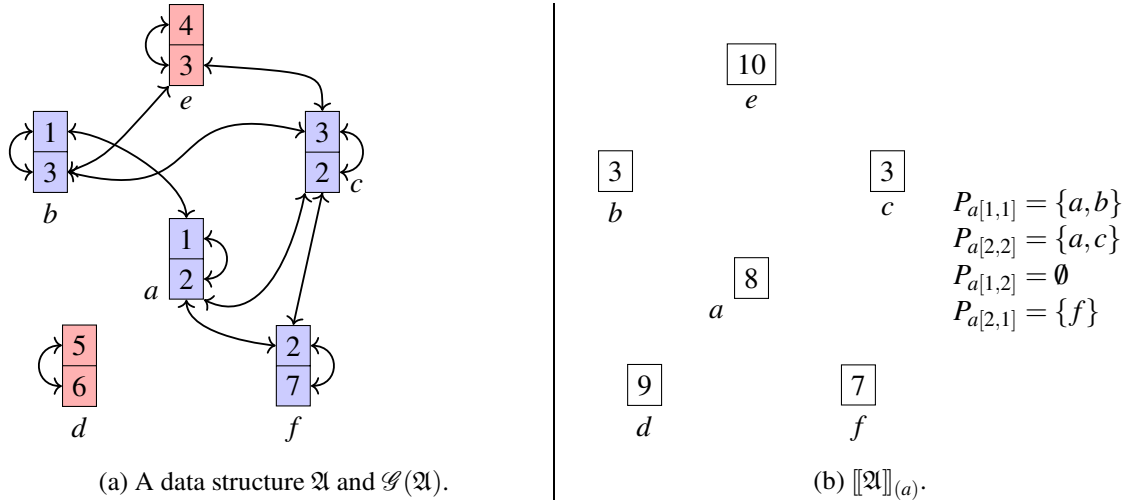


Figure 2

if $f_j(b)$ is in $\text{Val}_{\mathfrak{A}}(\vec{a})$ but $f_{3-j}(b)$ is not, the new predicates will give us the j -th data value of b and we have to keep track of the $(3-j)$ -th one, so we save it in $f(b)$. Lastly, if neither $f_1(b)$ nor $f_2(b)$ is in $\text{Val}_{\mathfrak{A}}(\vec{a})$, we will never be able to see the data values of b in φ_{q_f} (thanks to Lemma 1), so they do not matter to us. Finally, we have $[[\mathfrak{A}]]_{\vec{a}} = (A, (P_{\sigma})_{\sigma \in \Sigma}, f)$. Figure 2b provides an example of $\text{Val}_{\mathfrak{A}}(\vec{a})$ for the data structures depicted on Figure 2a and $\vec{a} = (a)$.

The next lemma formalizes the connection existing between \mathfrak{A} and $[[\mathfrak{A}]]_{\vec{a}}$ with $\vec{a} = (a_1, \dots, a_n)$.

Lemma 2 *Let $b, c \in A$ and $j, k \in \{1, 2\}$ and $p \in \{1, \dots, n\}$. The following statements then hold.*

1. *If $(b, j) \in B_1^{\mathfrak{A}}(a_p)$ and $(c, k) \in B_1^{\mathfrak{A}}(a_p)$ then $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$ iff there is $i \in \{1, 2\}$ s.t. $b \in P_{a_p[i,j]}$ and $c \in P_{a_p[i,k]}$.*
2. *If $(b, j) \in B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$ and $(c, k) \in B_1^{\mathfrak{A}}(a_p)$ then $\mathfrak{A}|_{a_p}^2 \not\models b \sim_k c$.*
3. *If $(b, j), (c, k) \in B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$ then $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$ iff either $b \sim_1^{[[\mathfrak{A}]]_{\vec{a}}} c$ or there exists $p' \in \{1, \dots, n\}$ and $\ell \in \{1, 2\}$ such that $b \in P_{a_{p'}[\ell,j]}$ and $c \in P_{a_{p'}[\ell,k]}$.*
4. *If $(b, j) \notin B_2^{\mathfrak{A}}(a_p)$ and $(c, k) \in B_2^{\mathfrak{A}}(a_p)$ then $\mathfrak{A}|_{a_p}^2 \not\models b \sim_k c$.*
5. *If $(b, j) \notin B_2^{\mathfrak{A}}(a_p)$ and $(c, k) \notin B_2^{\mathfrak{A}}(a_p)$ then $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$ iff $b = c$ and $j = k$.*

Proof: We suppose that $\mathfrak{A}|_{a_p}^2 = (A, (P_{\sigma})_{\sigma}, f_1^p, f_2^p)$.

1. Assume that $(b, j) \in B_1^{\mathfrak{A}}(a_p)$ and $(c, k) \in B_1^{\mathfrak{A}}(a_p)$. It implies that $f_j^p(b) = f_j(b)$ and $f_k^p(c) = f_k(c)$. Then assume that $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$. As $(b, j) \in B_1^{\mathfrak{A}}(a_p)$, thanks to Lemma 1.1 it means that there is a $i \in \{1, 2\}$ such that $a_p \sim_j^{\mathfrak{A}} b$. So we have $f_k(c) = f_k^p(c) = f_j^p(b) = f_j(b) = f_i(a_p)$, that is $a_p \sim_k^{\mathfrak{A}} c$. Hence by definition, $b \in P_{a_p[i,j]}$ and $c \in P_{a_p[i,k]}$. Conversely, let $i \in \{1, 2\}$ such that $b \in P_{a_p[i,j]}$ and $c \in P_{a_p[i,k]}$. This means that $a_p \sim_j^{\mathfrak{A}} b$ and $a_p \sim_k^{\mathfrak{A}} c$. So $f_j^p(b) = f_j(b) = f_i(a_p) = f_k(c) = f_k^p(c)$, that is $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$.
2. Assume that $(b, j) \in B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$ and $(c, k) \in B_1^{\mathfrak{A}}(a_p)$. It implies that $f_j^p(b) = f_j(b)$ and $f_k^p(c) = f_k(c)$. Thanks to Lemma 1.1, $(c, k) \in B_1^{\mathfrak{A}}(a_p)$ implies that $f_k(c) \in \{f_1(a_p), f_2(a_p)\}$ and $(b, j) \notin B_1^{\mathfrak{A}}(a_p)$ implies that $f_j(b) \notin \{f_1(a_p), f_2(a_p)\}$. So $\mathfrak{A}|_{a_p}^2 \not\models b \sim_k c$.

3. Assume that $(b, j), (c, k) \in B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$. As previously, we have that $f_j(b) \notin \{f_1(a_p), f_2(a_p)\}$ and $f_k(c) \notin \{f_1(a_p), f_2(a_p)\}$, and thanks to Lemma 1.2, we have $f_{3-j}(b) \in \{f_1(a_p), f_2(a_p)\}$ and $f_{3-k}(b) \in \{f_1(a_p), f_2(a_p)\}$. There is then two cases:
- Suppose there does not exist $p' \in \{1, \dots, n\}$ such that $f_j(b) \in \{f_1(a_{p'}), f_2(a_{p'})\}$. This allows us to deduce that $f_j^p(b) = f_j(b) = f(b)$ and $f_k^p(c) = f_k(c)$. If $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$, then necessarily there does not exist $p' \in \{1, \dots, n\}$ such that $f_k(c) \in \{f_1(a_{p'}), f_2(a_{p'})\}$ so we have $f_k^p(c) = f_k(c) = f(c)$ and $f(b) = f(c)$, consequently $b \sim_1^{\llbracket \mathfrak{A} \rrbracket_{\bar{a}}} c$. Similarly assume that $b \sim_1^{\llbracket \mathfrak{A} \rrbracket_{\bar{a}}} c$, this means that $f(b) = f(c)$ and either $b = c$ and $k = j$ or $b \neq c$ and by injectivity of f , we have $f_j(b) = f(b) = f_k(c)$. This allows us to deduce that $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$.
 - If there exists $p' \in \{1, \dots, n\}$ such that $f_j(b) = f_\ell(a_{p'})$ for some $\ell \in \{1, 2\}$. Then we have $b \in P_{a_{p'}[\ell, j]}$. Consequently, we have $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$ iff $c \in P_{a_{p'}[\ell, k]}$.
4. We prove the case 4 and 5 at the same time. Assume that $(b, j) \notin B_2^{\mathfrak{A}}(a_p)$. It means that in order to have $f_j^p(b) = f_k^p(c)$, we must have $(b, j) = (c, k)$. So if $(c, k) \in B_2^{\mathfrak{A}}(a_p)$, we can not have $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$ which ends case 4. And if $(c, k) \notin B_2^{\mathfrak{A}}(a_p)$, we have that $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$ iff $b = c$ and $j = k$.

□

We shall now see how we translate the formula $\varphi_{qf}(x_1, \dots, x_n)$ into a formula $\llbracket \varphi_{qf} \rrbracket(x_1, \dots, x_n)$ in $\text{dFO}[\Sigma', 1]$ such that \mathfrak{A} satisfies $\varphi_{qf}(a_1, \dots, a_n)$ if, and only if, $\llbracket \mathfrak{A} \rrbracket_{\bar{a}}$ satisfies $\llbracket \varphi_{qf} \rrbracket(a_1, \dots, a_n)$. Thanks to the previous lemma we know that if $\mathfrak{A}|_{a_p}^2 \models b \sim_k c$ then (b, j) and (c, k) must belong to the same set among $B_1^{\mathfrak{A}}(a_p)$, $B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$ and $A \setminus B_2^{\mathfrak{A}}(a_p)$ and we can test in $\llbracket \mathfrak{A} \rrbracket_{\bar{a}}$ whether (b, j) is a member of $B_1^{\mathfrak{A}}(a_p)$ or $B_2^{\mathfrak{A}}(a_p)$. Indeed, thanks to Lemmas 1.1 and 1.2, we have $(b, j) \in B_1^{\mathfrak{A}}(a_p)$ iff $b \in \bigcup_{i=1,2} P_{a_p[i, j]}$ and $(b, j) \in B_2^{\mathfrak{A}}(a_p)$ iff $b \in \bigcup_{i=1,2}^{j'=1,2} P_{a_p[i, j']}$. This reasoning leads to the following formulas in $\text{dFO}[\Sigma', 1]$ with $p \in \{1, \dots, n\}$ and $j \in \{1, 2\}$:

- $\varphi_{j, B_1(a_p)}(y) := a_p[1, j](y) \vee a_p[2, j](y)$ to test if the j -th field of an element belongs to $B_1^{\mathfrak{A}}(a_p)$
- $\varphi_{B_2(a_p)}(y) := \varphi_{1, B_1(a_p)}(y) \vee \varphi_{2, B_1(a_p)}(y)$ to test if a field of an element belongs to $B_2^{\mathfrak{A}}(a_p)$
- $\varphi_{j, B_2(a_p) \setminus B_1(a_p)}(y) := \varphi_{B_2(a_p)}(y) \wedge \neg \varphi_{j, B_1(a_p)}(y)$ to test that the j -th field of an element belongs to $B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$

We shall now present how we use these formulas to translate atomic formulas of the form $y \sim_k z$ under some $\langle\langle - \rangle\rangle_{x_p}^2$. For this matter, we rely on the three following formulas of $\text{dFO}[\Sigma', 1]$:

- The first formula asks for (y, j) and (z, k) to be in $B_1^{\mathfrak{A}}(a_p)$ (where here we abuse notations, using variables for the elements they represent) and for these two data values to coincide with one data value of a_p , it corresponds to Lemma 2.1:

$$\varphi_{j, k, a_p}^{r=1}(y, z) := \varphi_{j, B_1(a_p)}(y) \wedge \varphi_{k, B_1(a_p)}(z) \wedge \bigvee_{i=1,2} a_p[i, j](y) \wedge a_p[i, k](z)$$

- The second formula asks for (y, j) and (z, k) to be in $B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$ and checks either whether the data values of y and z in $\llbracket \mathfrak{A} \rrbracket_{\bar{a}}$ are equal or whether there exist p' and ℓ such that y belongs to $a_{p'}[\ell, j](y)$ and z belongs to $a_{p'}[\ell, k](z)$, it corresponds to Lemma 2.3:

$$\varphi_{j, k, a_p}^{r=2}(y, z) := \varphi_{j, B_2(a_p) \setminus B_1(a_p)}(y) \wedge \varphi_{k, B_2(a_p) \setminus B_1(a_p)}(z) \wedge (y \sim z \vee \left(\bigvee_{p'=1}^n \bigvee_{\ell=1}^2 a_{p'}[\ell, j](y) \wedge a_{p'}[\ell, k](z) \right))$$

- The third formula asks for (y, j) and (z, k) to not belong to $B_2^{\mathfrak{A}}(a_p)$ and for $y = z$, it corresponds to Lemma 2.5:

$$\varphi_{j,k,a_p}^{r>2}(y,z) := \begin{cases} \neg\varphi_{B_2(a_p)}(y) \wedge \neg\varphi_{B_2(a_p)}(z) \wedge y = z & \text{if } j = k \\ \perp & \text{otherwise} \end{cases}$$

Finally, here is the inductive definition of the translation $\llbracket - \rrbracket$ which uses sub transformations $\llbracket - \rrbracket_{x_p}$ in order to remember the centre of the ball and leads to the construction of $\llbracket \varphi_{qf} \rrbracket(x_1, \dots, x_n)$:

$$\begin{aligned} \llbracket \varphi \vee \varphi' \rrbracket &= \llbracket \varphi \rrbracket \vee \llbracket \varphi' \rrbracket \\ \llbracket x_p = x'_p \rrbracket &= x_p = x'_p \\ \llbracket \neg \varphi \rrbracket &= \neg \llbracket \varphi \rrbracket \\ \llbracket \langle \langle \psi \rangle \rangle_{x_p}^2 \rrbracket &= \llbracket \psi \rrbracket_{x_p} \\ \llbracket y \underset{j}{\sim} \underset{k}{z} \rrbracket_{x_p} &= \varphi_{j,k,a_p}^{r=1}(y,z) \vee \varphi_{j,k,a_p}^{r=2}(y,z) \vee \varphi_{j,k,a_p}^{r>2}(y,z) \\ \llbracket \sigma(x) \rrbracket_{x_p} &= \sigma(x) \\ \llbracket x = y \rrbracket_{x_p} &= x = y \\ \llbracket \varphi \vee \varphi' \rrbracket_{x_p} &= \llbracket \varphi \rrbracket_{x_p} \vee \llbracket \varphi' \rrbracket_{x_p} \\ \llbracket \neg \varphi \rrbracket_{x_p} &= \neg \llbracket \varphi \rrbracket_{x_p} \\ \llbracket \exists x. \varphi \rrbracket_{x_p} &= \exists x. \llbracket \varphi \rrbracket_{x_p} \end{aligned}$$

Lemma 3 We have $\mathfrak{A} \models \varphi_{qf}(\vec{a})$ iff $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \llbracket \varphi_{qf} \rrbracket(\vec{a})$.

Proof: Because of the inductive definition of $\llbracket \varphi \rrbracket$ and that only the atomic formulas $y \underset{j}{\sim} \underset{k}{z}$ change, we only have to prove that given $b, c \in A$, we have $\mathfrak{A}|_{a_p}^2 \models b \underset{j}{\sim} \underset{k}{c}$ iff $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \llbracket y \underset{j}{\sim} \underset{k}{z} \rrbracket_{x_p}(b, c)$.

We first suppose that $\mathfrak{A}|_{a_p}^2 \models b \underset{j}{\sim} \underset{k}{c}$. Using Lemma 2, it implies that (b, j) and (c, k) belong to same set between $B_1^{\mathfrak{A}}(a_p)$, $B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$ and $A \setminus B_2^{\mathfrak{A}}(a_p)$. We proceed by a case analysis.

- If $(b, j), (c, k) \in B_1^{\mathfrak{A}}(a_p)$ then by lemma 2.1 we have that $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \varphi_{j,k,a_p}^{r=1}(b, c)$ and thus $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \llbracket y \underset{j}{\sim} \underset{k}{z} \rrbracket_{x_p}(b, c)$.
- If $(b, j), (c, k) \in B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$ then by lemma 2.3 we have that $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \varphi_{j,k,a_p}^{r=2}(b, c)$ and thus $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \llbracket y \underset{j}{\sim} \underset{k}{z} \rrbracket_{x_p}(b, c)$.
- If $(b, j), (c, k) \in A \setminus B_2^{\mathfrak{A}}(a_p)$ then by lemma 2.5 we have that $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \varphi_{j,k,a_p}^{r>2}(b, c)$ and thus $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \llbracket y \underset{j}{\sim} \underset{k}{z} \rrbracket_{x_p}(b, c)$.

We now suppose that $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \llbracket y \underset{j}{\sim} \underset{k}{z} \rrbracket_{x_p}(b, c)$. It means that $\llbracket \mathfrak{A} \rrbracket_{\vec{a}}$ satisfies at least $\varphi_{j,k,a_p}^{r=1}(b, c)$, $\varphi_{j,k,a_p}^{r=2}(b, c)$ or $\varphi_{j,k,a_p}^{r>2}(b, c)$. If $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \varphi_{j,k,a_p}^{r=1}(b, c)$, it implies that (b, j) and (c, k) are in $B_1^{\mathfrak{A}}(a_p)$, and we can then apply lemma 2.1 to deduce that $\mathfrak{A}|_{a_p}^2 \models b \underset{j}{\sim} \underset{k}{c}$. If $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \varphi_{j,k,a_p}^{r=2}(b, c)$, it implies that (b, j) and (c, k) are in $B_2^{\mathfrak{A}}(a_p) \setminus B_1^{\mathfrak{A}}(a_p)$, and we can then apply lemma 2.3 to deduce that $\mathfrak{A}|_{a_p}^2 \models b \underset{j}{\sim} \underset{k}{c}$. If $\llbracket \mathfrak{A} \rrbracket_{\vec{a}} \models \varphi_{j,k,a_p}^{r>2}(b, c)$, it implies that (b, j) and (c, k) are in $A \setminus B_2^{\mathfrak{A}}(a_p)$, and we can then apply lemma 2.5 to deduce that $\mathfrak{A}|_{a_p}^2 \models b \underset{j}{\sim} \underset{k}{c}$. \square

To provide a reduction from DATASAT(\exists -2-Loc-dFO, 2) to DATASAT(dFO, 1), having the formula $\llbracket \varphi_{qf} \rrbracket(x_1, \dots, x_n)$ is not enough because to use the result of the previous Lemma, we need to ensure that there exists a model \mathfrak{B} and a tuple of elements (a_1, \dots, a_n) such that $\mathfrak{B} \models \llbracket \varphi_{qf} \rrbracket(a_1, \dots, a_n)$ and as well that there exists $\mathfrak{A} \in \text{Data}[\Sigma, 2]$ such that $\mathfrak{B} = \llbracket \mathfrak{A} \rrbracket_{\vec{a}}$. We explain now how we can ensure this last point.

Now, we want to characterize the structures of the form $\llbracket \mathfrak{A} \rrbracket_{\vec{a}}$. Given $\mathfrak{B} = (A, (P_\sigma)_{\sigma \in \Sigma'}, f) \in \text{Data}[\Sigma', 1]$ and $\vec{a} \in A$, we say that (\mathfrak{B}, \vec{a}) is *well formed* iff there exists a structure $\mathfrak{A} \in \text{Data}[\Sigma, 2]$ such that $\mathfrak{B} =$

$[[\mathfrak{A}]]_{\vec{a}}$. Hence (\mathfrak{B}, \vec{a}) is *well formed* iff there exist two functions $f_1, f_2 : A \rightarrow \mathbb{N}$ such that $[[\mathfrak{A}]]_{\vec{a}} = [[(A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2)]]_{\vec{a}}$. We state three properties on (\mathfrak{B}, \vec{a}) , and we will show that they characterize being well formed.

1. (Transitivity) For all $b, c \in A$, $p, q \in \{1, \dots, n\}$, $i, j, k, \ell \in \{1, 2\}$ if $b \in P_{a_p[i, j]}$, $c \in P_{a_p[i, \ell]}$ and $b \in P_{a_q[k, j]}$ then $c \in P_{a_q[k, \ell]}$.
2. (Reflexivity) For all p and i , we have $a_p \in P_{a_p[i, i]}$
3. (Uniqueness) For all $b \in A$, if $b \in \bigcap_{j=1,2} \bigcup_{p=1, \dots, n}^{i=1,2} P_{a_p[i, j]}$ or $b \notin \bigcup_{j=1,2} \bigcup_{p=1, \dots, n}^{i=1,2} P_{a_p[i, j]}$ then for any $c \in B$ such that $f(c) = f(b)$ we have $c = b$.

Each property can be expressed by a first order logic formula, which we respectively name φ_{tran} , φ_{refl} and φ_{uniq} and we denote by φ_{wf} their conjunction:

$$\begin{aligned} \varphi_{tran} &= \forall y \forall z. \bigwedge_{p,q=1}^n \bigwedge_{i,j,k,\ell=1}^2 \left(a_p[i, j](y) \wedge a_p[i, \ell](z) \wedge a_q[k, j](y) \Rightarrow a_q[k, \ell](z) \right) \\ \varphi_{refl}(x_1, \dots, x_n) &= \bigwedge_{p=1}^n \bigwedge_{i=1}^2 a_p[i, i](x_p) \\ \varphi_{uniq} &= \forall y. \left(\bigwedge_{j=1}^2 \bigvee_{p=1}^n \bigvee_{i=1}^2 a_p[i, j](y) \vee \bigwedge_{j=1}^2 \bigwedge_{p=1}^n \bigwedge_{i=1}^2 \neg a_p[i, j](y) \right) \Rightarrow (\forall z. y \sim z \Rightarrow y = z) \\ \varphi_{wf}(x_1, \dots, x_n) &= \varphi_{tran} \wedge \varphi_{refl}(x_1, \dots, x_n) \wedge \varphi_{uniq} \end{aligned}$$

The next lemma expresses that the formula φ_{wf} allows to characterise precisely the 1-data structures in $\text{Data}[\Sigma', 1]$ which are well-formed.

Lemma 4 *Let $\mathfrak{B} \in \text{Data}[\Sigma', 1]$ and a_1, \dots, a_n elements of \mathfrak{B} , then (\mathfrak{B}, \vec{a}) is well formed iff $\mathfrak{B} \models \varphi_{wf}(\vec{a})$.*

Proof: First, if (\mathfrak{B}, \vec{a}) is well formed, then there exists $\mathfrak{A} \in \text{Data}[\Sigma, 2]$ such that $\mathfrak{B} = [[\mathfrak{A}]]_{\vec{a}}$ and by construction we have $[[\mathfrak{A}]]_{\vec{a}} \models \varphi_{wf}(\vec{a})$. We now suppose that $\mathfrak{B} = (A, (P_\sigma)_{\sigma \in \Sigma'}, f)$ and $\mathfrak{B} \models \varphi_{wf}(\vec{a})$. In order to define the functions $f_1, f_2 : A \rightarrow \mathbb{N}$, we need to introduce some objects.

We first define a function $g : \{1, \dots, n\} \times \{1, 2\} \rightarrow \mathbb{N} \setminus \text{Im}(f)$ (where $\text{Im}(f)$ is the image of f in \mathfrak{B}) which verifies the following properties:

- for all $p \in \{1, \dots, n\}$ and $i \in \{1, 2\}$, we have $a_p \in P_{a_p[i, 3-i]}$ iff $g(p, 1) = g(p, 2)$;
- for all $p, q \in \{1, \dots, n\}$ and $i, j \in \{1, 2\}$, we have $a_q \in P_{a_p[i, j]}$ iff $g(p, i) = g(q, j)$.

We use this function to fix the two data values carried by the elements in $\{a_1, \dots, a_n\}$. We now explain why this function is well founded, it is due to the fact that $\mathfrak{B} \models \varphi_{tran} \wedge \varphi_{refl}(a_1, \dots, a_n)$. In fact, since $\mathfrak{B} \models \varphi_{refl}(a_1, \dots, a_n)$, we have for all $p \in \{1, \dots, n\}$ and $i \in \{1, 2\}$, $a_p \in P_{a_p[i, i]}$. Furthermore if $a_p \in P_{a_p[i, j]}$ then $a_p \in P_{a_p[j, i]}$ thanks to the formula φ_{tran} ; indeed since we have $a_p \in P_{a_p[i, j]}$ and $a_p \in P_{a_p[i, i]}$ and $a_p \in P_{a_p[j, j]}$, we obtain $a_p \in P_{a_p[j, i]}$. Next, we also have that if $a_q \in P_{a_p[i, j]}$ then $a_p \in P_{a_q[j, i]}$ again thanks to φ_{tran} ; indeed since we have $a_q \in P_{a_p[i, j]}$ and $a_p \in P_{a_p[i, i]}$ and $a_q \in P_{a_q[j, j]}$, we obtain $a_p \in P_{a_q[j, i]}$.

We also need a natural d_{out} belonging to $\mathbb{N} \setminus (\text{Im}(g) \cup \text{Im}(f))$. For $j \in \{1, 2\}$, we define f_j as follows for all $b \in A$:

$$f_j(b) = \begin{cases} g(p, i) & \text{if for some } p, i \text{ we have } b \in P_{a_p[i, j]} \\ f(b) & \text{if for all } p, i \text{ we have } b \notin P_{a_p[i, j]} \text{ and for some } p, i \text{ we have } b \in P_{a_p[i, 3-j]} \\ d_{out} & \text{if for all } p, i, j', \text{ we have } b \notin P_{a_p[i, j']} \end{cases}$$

Here again, we can show that since $\mathfrak{B} \models \varphi_{tran} \wedge \varphi_{refl}(a_1, \dots, a_n)$, the functions f_1 and f_2 are well founded. Indeed, assume that $b \in P_{a_p[i, j]} \cap P_{a_q[k, j]}$, then we have necessarily that $g(p, i) = g(q, k)$. For this we need to show that $a_p \in a_q[k, i]$ and we use again the formula φ_{tran} . This can be obtained because we have $b \in P_{a_p[i, j]}$ and $a_p \in P_{a_p[i, i]}$ and $b \in P_{a_q[k, j]}$.

We then define \mathfrak{A} as the 2-data-structures $(A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2)$. It remains to prove that $\mathfrak{B} = \llbracket \mathfrak{A} \rrbracket_{\vec{a}}$.

First, note that for all $b \in A$, $p \in \{1, \dots, n\}$ and $i, j \in \{1, 2\}$, we have $b \in P_{a_p[i,j]}$ iff $a_p \dot{\sim}_j^{\mathfrak{A}} b$. Indeed, we have $b \in P_{a_p[i,j]}$, we have that $f_j(b) = g(p, i)$ and since $a_p \in P_{a_p[i,j]}$ we have as well that $f_i(a_p) = g(p, i)$, as a consequence $a_p \dot{\sim}_j^{\mathfrak{A}} b$. In the other direction, if $a_p \dot{\sim}_j^{\mathfrak{A}} b$, it means that $f_j(b) = f_i(a_p) = g(p, i)$ and thus $b \in P_{a_p[i,j]}$. Now to have $\mathfrak{B} = \llbracket \mathfrak{A} \rrbracket_{\vec{a}}$, one has only to be careful in the choice of function f_{new} while building $\llbracket \mathfrak{A} \rrbracket_{\vec{a}}$. We recall that this function is injective and is used to give a value to the elements $b \in A$ such that neither $f_1(b) \in \text{Val}_{\mathfrak{A}}(\vec{a})$ and $f_2(b) \notin \text{Val}_{\mathfrak{A}}(\vec{a})$ nor $f_1(b) \notin \text{Val}_{\mathfrak{A}}(\vec{a})$ and $f_2(b) \in \text{Val}_{\mathfrak{A}}(\vec{a})$. For these elements, we make f_{new} matches with the function f and the fact that we define an injection is guaranteed by the formula φ_{uniq} . \square

Using the results of Lemma 3 and 4, we deduce that the formula $\varphi = \exists x_1 \dots \exists x_n. \varphi_{qf}(x_1, \dots, x_n)$ of \exists -2-Loc-dFO $[\Sigma, 2]$ is satisfiable iff the formula $\psi = \exists x_1 \dots \exists x_n. \llbracket \varphi_{qf} \rrbracket(x_1, \dots, x_n) \wedge \varphi_{wf}(x_1, \dots, x_n)$ is satisfiable. Note that ψ can be built in polynomial time from φ and that it belongs to dFO $[\Sigma', 1]$. Hence, thanks to Theorem 3, we obtain that DATASAT(\exists -2-Loc-dFO, 2) is in N2EXP.

We can as well obtain a matching lower bound thanks to a reduction from DATASAT(dFO, 1). For this matter we rely on two crucial points. First in the formulas of \exists -2-Loc-dFO $[\Sigma, 2]$, there is no restriction on the use of quantifiers for the formulas located under the scope of the $\langle\langle \cdot \rangle\rangle_x^2$ modality and consequently we can write inside this modality a formula of dFO $[\Sigma, 1]$ without any modification. Second we can extend a model dFO $[\Sigma, 1]$ into a 2-data structure such that all elements and their values are located in the same radius-2-ball by adding everywhere a second data value equal to 0. More formally, let φ be a formula in dFO $[\Sigma, 1]$ and consider the formula $\exists x. \langle\langle \varphi \rangle\rangle_x^2$ where we interpret φ over 2-data structures (this formula simply never mentions the values located in the second fields). We have then the following lemma.

Lemma 5 *There exists $\mathfrak{A} \in \text{Data}[\Sigma, 1]$ such that $\mathfrak{A} \models \varphi$ if and only if there exists $\mathfrak{B} \in \text{Data}[\Sigma, 2]$ such that $\mathfrak{B} \models \exists x. \langle\langle \varphi \rangle\rangle_x^2$.*

Proof: Assume that there exists $\mathfrak{A} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1)$ in $\text{Data}[\Sigma, 1]$ such that $\mathfrak{A} \models \varphi$. Consider the 2-data structure $\mathfrak{B} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2)$ such that $f_2(a) = 0$ for all $a \in A$. Let $a \in A$. It is clear that we have $\mathfrak{B}|_a^2 = \mathfrak{B}$ and that $\mathfrak{B}|_a^2 \models \varphi$ (because $\mathfrak{A} \models \varphi$ and φ never mentions the second values of the elements since it is a formula in dFO $[\Sigma, 1]$). Consequently $\mathfrak{B} \models \exists x. \langle\langle \varphi \rangle\rangle_x^2$.

Assume now that there exists $\mathfrak{B} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2)$ in $\text{Data}[\Sigma, 2]$ such that $\mathfrak{B} \models \exists x. \langle\langle \varphi \rangle\rangle_x^2$. Hence there exists $a \in A$ such that $\mathfrak{B}|_a^2 \models \varphi$, but then by forgetting the second value in $\mathfrak{B}|_a^2$ we obtain a model in $\text{Data}[\Sigma, 1]$ which satisfies φ . \square

Since DATASAT(dFO, 1) is N2EXP-hard (see Theorem 3), we obtain the desired lower bound.

Theorem 4 *The problem DATASAT(\exists -2-Loc-dFO, 2) is N2EXP-complete.*

3.3 Balls of radius 1 and any number of data values

Let $D \geq 1$. We first show that DATASAT(\exists -1-Loc-dFO, D) is in NEXP by providing a reduction towards DATASAT(dFO, 0). This reduction uses the characterisation of the radius-1-ball provided by Lemma 1 and is very similar to the reduction provided in the previous section. In fact, for an element b located in the radius-1-ball of another element a , we use extra unary predicates to explicit which are the values of b that are common with the values of a . We provide here the main step of this reduction whose proof follows the same line as the one of Theorem 4.

We consider a formula $\varphi = \exists x_1 \dots \exists x_n. \varphi_{qf}(x_1, \dots, x_n)$ of \exists -1-Loc-dFO $[\Sigma, D]$ in prenex normal form, i.e., such that $\varphi_{qf}(x_1, \dots, x_n) \in \text{qf-1-Loc-dFO}[\Sigma, D]$. We know that there is a structure $\mathfrak{A} = (A, (P_\sigma)_{\sigma \in \Sigma},$

f_1, f_2, \dots, f_D) in $\text{Data}[\Sigma, D]$ such that $\mathfrak{A} \models \varphi$ if and only if there are $a_1, \dots, a_n \in A$ such that $\mathfrak{A} \models \varphi_{qf}(a_1, \dots, a_n)$. Let then $\mathfrak{A} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2, \dots, f_D)$ in $\text{Data}[\Sigma, D]$ and a tuple $\vec{a} = (a_1, \dots, a_n)$ of elements in A^n . Let $\Omega_n = \{a_p[i, j] \mid p \in \{1, \dots, n\}, i, j \in \{1, \dots, D\}\}$ be a set of new unary predicates and $\Sigma' = \Sigma \cup \Omega_n$. For every element $b \in A$, the predicates in Ω_n are used to keep track of the relation between the data values of b and the one of a_1, \dots, a_n in \mathfrak{A} . Formally, we have $P_{a_p[i, j]} = \{b \in A \mid \mathfrak{A} \models a_p i \sim_j b\}$. Finally, we build the 0-data-structure $[[\mathfrak{A}]]'_{\vec{a}} = (A, (P_\sigma)_{\sigma \in \Sigma'})$. Similarly to Lemma 2, we have the following connection between \mathfrak{A} and $[[\mathfrak{A}]]'_{\vec{a}}$.

Lemma 6 *Let $b, c \in A$ and $j, k \in \{1, \dots, D\}$ and $p \in \{1, \dots, n\}$. The following statements hold:*

1. *If $(b, j) \in B_1^{\mathfrak{A}}(a_p)$ and $(c, k) \in B_1^{\mathfrak{A}}(a_p)$ then $\mathfrak{A}|_{a_p}^1 \models b j \sim_k c$ iff there is $i \in \{1, 2\}$ s.t. $b \in P_{a_p[i, j]}$ and $c \in P_{a_p[i, k]}$.*
2. *If $(b, j) \notin B_1^{\mathfrak{A}}(a_p)$ and $(c, k) \in B_1^{\mathfrak{A}}(a_p)$ then $\mathfrak{A}|_{a_p}^1 \not\models b j \sim_k c$*
3. *If $(b, j) \notin B_1^{\mathfrak{A}}(a_p)$ and $(c, k) \notin B_1^{\mathfrak{A}}(a_p)$ then $\mathfrak{A}|_{a_p}^1 \models b j \sim_k c$ iff $b = c$ and $j = k$.*

We shall now see how we translate the formula $\varphi_{qf}(x_1, \dots, x_n)$ into a formula $[[\varphi_{qf}]]'(x_1, \dots, x_n)$ in $\text{dFO}[\Sigma', 0]$ such that \mathfrak{A} satisfies $\varphi_{qf}(a_1, \dots, a_n)$ if, and only if, $[[\mathfrak{A}]]'_{\vec{a}}$ satisfies $[[\varphi_{qf}]]'(a_1, \dots, a_n)$. As in the previous section, we introduce the following formula in $\text{dFO}[\Sigma', 0]$ with $p \in \{1, \dots, n\}$ and $j \in \{1, \dots, D\}$ to test if the j -th field of an element belongs to $B_1^{\mathfrak{A}}(a_p)$:

$$\varphi_{j, B_1(a_p)}(y) := \bigvee_{i \in \{1, \dots, D\}} a_p[i, j](y)$$

We now present how we translate atomic formulas of the form $y j \sim_k z$ under some $\langle\langle - \rangle\rangle_{x_p}^1$. For this matter, we rely on two formulas of $\text{dFO}[\Sigma', 0]$ which can be described as follows:

- The first formula asks for (y, j) and (z, k) to be in $B_1^{\mathfrak{A}}(a_p)$ (here we abuse notations, using variables for the elements they represent) and for these two data values to coincide with one data value of a_p , it corresponds to Lemma 6.1:

$$\psi_{j, k, a_p}^{r=1}(y, z) := \varphi_{j, B_1(a_p)}(y) \wedge \varphi_{k, B_1(a_p)}(z) \wedge \bigvee_{i=1}^D a_p[i, j](y) \wedge a_p[i, k](z)$$

- The second formula asks for (y, j) and (z, k) to not belong to $B_1^{\mathfrak{A}}(a_p)$ and for $y = z$, it corresponds to Lemma 6.3:

$$\psi_{j, k, a_p}^{r>1}(y, z) := \begin{cases} \bigwedge_{i=1}^D (\neg \varphi_{i, B_1(a_p)}(y) \wedge \neg \varphi_{i, B_1(a_p)}(z)) \wedge y = z & \text{if } j = k \\ \perp & \text{otherwise} \end{cases}$$

Finally, as before we provide an inductive definition of the translation $[[-]]'$ which uses subtransformations $[[-]]'_{x_p}$ in order to remember the centre of the ball and leads to the construction of $[[\varphi_{qf}]]'(x_1, \dots, x_n)$. We only detail the case

$$[[y j \sim_k z]]'_{x_p} = \psi_{j, k, a_p}^{r=1}(y, z) \vee \psi_{j, k, a_p}^{r>1}(y, z)$$

as the other cases are identical as for the translation $[[-]]$ shown in the previous section. This leads to the following lemma (which is the pendant of Lemma 3).

Lemma 7 *We have $\mathfrak{A} \models \varphi_{qf}(\vec{a})$ iff $[[\mathfrak{A}]]'_{\vec{a}} \models [[\varphi_{qf}]]'(\vec{a})$.*

As we had to characterise the well-formed 1-data structure, a similar trick is necessary here. For this matter, we use the following formulas:

$$\begin{aligned} \Psi_{tran} &= \forall y \forall z. \bigwedge_{p,q=1}^n \bigwedge_{i,j,k,\ell=1}^D \left(a_p[i,j](y) \wedge a_p[i,\ell](z) \wedge a_q[k,j](y) \Rightarrow a_q[k,\ell](z) \right) \\ \Psi_{refl}(x_1, \dots, x_n) &= \bigwedge_{p=1}^n \bigwedge_{i=1}^D a_p[i,i](x_p) \\ \Psi_{wf}(x_1, \dots, x_n) &= \Psi_{tran} \wedge \Psi_{refl}(x_1, \dots, x_n) \end{aligned}$$

Finally with the same reasoning as the one given in the previous section, we can show that the formula $\varphi = \exists x_1 \dots \exists x_n. \varphi_{qf}(x_1, \dots, x_n)$ of \exists -1-Loc-dFO $[\Sigma, D]$ is satisfiable iff the formula $\exists x_1 \dots \exists x_n. \llbracket \varphi_{qf} \rrbracket'(x_1, \dots, x_n) \wedge \Psi_{wf}(x_1, \dots, x_n)$ is satisfiable. Note that this latter formula can be built in polynomial time from φ and that it belongs to dFO $[\Sigma', 0]$. Hence, thanks to Theorem 2, we obtain that DATASAT(\exists -1-Loc-dFO, D) is in NEXP. The matching lower bound is as well obtained the same way by reducing DATASAT(dFO, 0) to DATASAT(\exists -1-Loc-dFO, D) showing that a formula φ in dFO $[\Sigma, 0]$ is satisfiable iff the formula $\exists x. \langle\langle \varphi \rangle\rangle_x^1$ in \exists - D -Loc-dFO $[\Sigma, 1]$ is satisfiable.

Theorem 5 *For all $D \geq 1$, the problem DATASAT(\exists -1-Loc-dFO, D) is NEXP-complete.*

4 Undecidability results

We show here DATASAT(\exists -3-Loc-dFO, 2) and DATASAT(\exists -2-Loc-dFO, 3) are undecidable. To obtain this we provide reductions from DATASAT(dFO, 2) and we use the fact that any 2-data structure can be interpreted as a radius-3-ball of a 2-data structure or respectively as a radius-2-ball of a 3-data structure.

4.1 Radius 3 and two data values

In order to reduce DATASAT(dFO, 2) to DATASAT(\exists -3-Loc-dFO, 2), we show that we can transform slightly any 2-data structure \mathfrak{A} into an other 2-data structure \mathfrak{A}_{ge} such that \mathfrak{A}_{ge} corresponds to the radius-3-ball of any element of \mathfrak{A}_{ge} and this transformation has some kind of inverse. Furthermore, given a formula $\varphi \in$ dFO $[\Sigma, 2]$, we transform it into a formula $T(\varphi)$ in \exists -3-Loc-dFO $[\Sigma', 2]$ such that \mathfrak{A} satisfies φ iff \mathfrak{A}_{ge} satisfies $T(\varphi)$. What follows is the formalisation of this reasoning.

Let $\mathfrak{A} = (A, (P_\sigma)_\sigma, f_1, f_2)$ be a 2-data structure in Data $[\Sigma, 2]$ and ge be a fresh unary predicate not in Σ . From \mathfrak{A} we build the following 2-data structure $\mathfrak{A}_{ge} = (A', (P'_\sigma)_\sigma, f'_1, f'_2) \in$ Data $[\Sigma \cup \{ge\}, 2]$ such that:

- $A' = A \uplus Val_{\mathfrak{A}}(A) \times Val_{\mathfrak{A}}(A)$,
- for $i \in \{1, 2\}$ and $a \in A$, $f'_i(a) = f_i(a)$ and for $(d_1, d_2) \in Val_{\mathfrak{A}}(A) \times Val_{\mathfrak{A}}(A)$, $f_i((d_1, d_2)) = d_i$,
- for $\sigma \in \Sigma$, $P'_\sigma = P_\sigma$,
- $P_{ge} = Val_{\mathfrak{A}}(A) \times Val_{\mathfrak{A}}(A)$.

Hence to build \mathfrak{A}_{ge} from \mathfrak{A} we have added to the elements of \mathfrak{A} all pairs of data presented in \mathfrak{A} and in order to recognise these new elements in the structure we use the new unary predicate ge . We add these extra elements to ensure that all the elements of the structure are located in the radius-3-ball of any element of \mathfrak{A}_{ge} . We have then the following property.

Lemma 8 $\mathfrak{A}_{ge}|_a^3 = \mathfrak{A}_{ge}$ for all $a \in A'$.

Proof: Let $b \in A'$ and $i, j \in \{1, 2\}$. We show that $d^{\mathfrak{A}_{\text{ge}}}((a, i), (b, j)) \leq 3$. i.e. that there is a path of length at most 3 from (a, i) to (b, j) in the data graph $\mathcal{G}(\mathfrak{A}_{\text{ge}})$. By construction of \mathfrak{A}_{ge} , there is an element $c \in A'$ such that $f_1(c) = f_i(a)$ and $f_2(c) = f_j(b)$. So we have the path $(a, i), (c, 1), (c, 2), (b, j)$ of length at most 3 from (a, i) to (b, j) in $\mathcal{G}(\mathfrak{A}_{\text{ge}})$. \square

Conversely, to $\mathfrak{A} = (A, (P_\sigma)_\sigma, f_1, f_2) \in \text{Data}[\Sigma \cup \{\text{ge}\}, 2]$, we associate $\mathfrak{A}_{\setminus \text{ge}} = (A', (P'_\sigma)_\sigma, f'_1, f'_2) \in \text{Data}[\Sigma, 2]$ where:

- $A' = A \setminus P_{\text{ge}}$,
- for $i \in \{1, 2\}$ and $a \in A'$, $f'_i(a) = f_i(a)$,
- for $\sigma \in \Sigma$, $P'_\sigma = P_\sigma \setminus P_{\text{ge}}$.

Finally we inductively translate any formula $\varphi \in \text{dFO}[\Sigma, 2]$ into $T(\varphi) \in \text{dFO}[\Sigma \cup \{\text{ge}\}, 2]$ by making it quantify over elements not labeled with ge: $T(\sigma(x)) = \sigma(x)$, $T(x_i \sim_j y) = x_i \sim_j y$, $T(x = y) = (x = y)$, $T(\exists x. \varphi) = \exists x. \neg \text{ge}(x) \wedge T(\varphi)$, $T(\varphi \vee \varphi') = T(\varphi) \vee T(\varphi')$ and $T(\neg \varphi) = \neg T(\varphi)$.

Lemma 9 *Let φ be a sentence in $\text{dFO}[\Sigma, 2]$, $\mathfrak{A} \in \text{Data}[\Sigma, 2]$ and $\mathfrak{B} \in \text{Data}[\Sigma \cup \{\text{ge}\}, 2]$. The two following properties hold:*

- $\mathfrak{A} \models \varphi$ iff $\mathfrak{A}_{\text{ge}} \models T(\varphi)$
- $\mathfrak{B}_{\setminus \text{ge}} \models \varphi$ iff $\mathfrak{B} \models T(\varphi)$.

Proof: As for any $\mathfrak{A} \in \text{Data}[\Sigma, 2]$ we have $(\mathfrak{A}_{\text{ge}})_{\setminus \text{ge}} = \mathfrak{A}$, it is sufficient to prove the second point. We reason by induction on φ . Let $\mathfrak{A} = (A, (P_\sigma)_\sigma, f_1, f_2) \in \text{Data}[\Sigma \cup \{\text{ge}\}, 2]$ and let $\mathfrak{A}_{\setminus \text{ge}} = (A', (P'_\sigma)_\sigma, f'_1, f'_2) \in \text{Data}[\Sigma, 2]$. The inductive hypothesis is that for any formula $\varphi \in \text{dFO}[\Sigma, 2]$ (closed or not) and any context interpretation function $I: \mathcal{V} \rightarrow A'$ we have $\mathfrak{A}_{\setminus \text{ge}} \models_I \varphi$ iff $\mathfrak{A} \models_I T(\varphi)$. Note that the inductive hypothesis is well founded in the sense that the interpretation I always maps variables to elements of the structures.

We prove two cases: when φ is a unary predicate and when φ starts by an existential quantification, the other cases being similar. First, assume that $\varphi = \sigma(x)$ where $\sigma \in \Sigma$. $\mathfrak{A}_{\setminus \text{ge}} \models_I \sigma(x)$ holds iff $I(x) \in P'_\sigma$. As $I(x) \in A \setminus P_{\text{ge}}$, we have $I(x) \in P'_\sigma$ iff $I(x) \in P_\sigma$, which is equivalent to $\mathfrak{A} \models_I T(\sigma(x))$. Second assume $\varphi = \exists x. \varphi'$. Suppose that $\mathfrak{A}_{\setminus \text{ge}} \models_I \exists x. \varphi'$. Thus, there is a $a \in A'$ such that $\mathfrak{A}_{\setminus \text{ge}} \models_{I[x/a]} \varphi'$. By inductive hypothesis, we have $\mathfrak{A} \models_{I[x/a]} T(\varphi')$. As $a \in A' = A \setminus P_{\text{ge}}$, we have $\mathfrak{A} \models_{I[x/a]} \neg \text{ge}(x)$, so $\mathfrak{A} \models_I \exists x. \neg \text{ge}(x) \wedge T(\varphi')$ as desired. Conversely, suppose that $\mathfrak{A} \models_I T(\exists x. \varphi')$. It means that there is a $a \in A$ such that $\mathfrak{A} \models_{I[x/a]} \neg \text{ge}(x) \wedge T(\varphi')$. So we have that $a \in A' = A \setminus P_{\text{ge}}$, which means that $I[x/a]$ takes values in A and we can apply the inductive hypothesis to get that $\mathfrak{A}_{\setminus \text{ge}} \models_{I[x/a]} \varphi'$. So we have $\mathfrak{A}_{\setminus \text{ge}} \models_I \exists x. \varphi'$. \square

From Theorem 1, we know that $\text{DATASAT}(\text{dFO}, 2)$ is undecidable. From a closed formula $\varphi \in \text{dFO}[\Sigma, 2]$, we build the formula $\exists x. \langle\langle T(\varphi) \rangle\rangle_x^3 \in \exists\text{-3-Loc-dFO}[\Sigma \cup \{\text{ge}\}, 2]$. Now if φ is satisfiable, it means that there exists $\mathfrak{A} \in \text{Data}[\Sigma, 2]$ such that $\mathfrak{A} \models \varphi$. By Lemma 9, $\mathfrak{A}_{\text{ge}} \models T(\varphi)$. Let a be an element of \mathfrak{A} , then thanks to Lemma 8, we have $\mathfrak{A}_{\text{ge}}|_a^3 \models T(\varphi)$. Finally by definition of our logic, $\mathfrak{A}_{\text{ge}} \models \exists x. \langle\langle T(\varphi) \rangle\rangle_x^3$. So $\exists x. \langle\langle T(\varphi) \rangle\rangle_x^3$ is satisfiable. Now assume that $\exists x. \langle\langle T(\varphi) \rangle\rangle_x^3$ is satisfiable. So there exist $\mathfrak{A} \in \text{Data}[\Sigma \cup \{\text{ge}\}, 2]$ and an element a of \mathfrak{A} such that $\mathfrak{A}|_a^3 \models T(\varphi)$. Using Lemma 9, we obtain $(\mathfrak{A}|_a^3)_{\setminus \text{ge}} \models \varphi$. Hence φ is satisfiable. This shows that we can reduce $\text{DATASAT}(\text{dFO}, 2)$ to $\text{DATASAT}(\exists\text{-3-Loc-dFO}, 2)$.

Theorem 6 *The problem $\text{DATASAT}(\exists\text{-3-Loc-dFO}, 2)$ is undecidable.*

4.2 Radius 2 and three data values

We provide here a reduction from $\text{DATASAT}(\text{dFO}, 2)$ to $\text{DATASAT}(\exists\text{-2-Loc-dFO}, 3)$. The idea is similar to the one used in the proof of Lemma 5 to show that $\text{DATASAT}(\exists\text{-2-Loc-dFO}, 2)$ is N2EXP-hard by reducing $\text{DATASAT}(\text{dFO}, 1)$. Indeed we have the following Lemma.

Lemma 10 *Let φ be a formula in $\text{dFO}[\Sigma, 2]$. There exists $\mathfrak{A} \in \text{Data}[\Sigma, 2]$ such that $\mathfrak{A} \models \varphi$ if and only if there exists $\mathfrak{B} \in \text{Data}[\Sigma, 3]$ such that $\mathfrak{B} \models \exists x. \langle\langle \varphi \rangle\rangle_x^2$.*

Proof: Assume that there exists $\mathfrak{A} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2)$ in $\text{Data}[\Sigma, 2]$ such that $\mathfrak{A} \models \varphi$. Consider the 3-data structure $\mathfrak{B} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2, f_3)$ such that $f_3(a) = 0$ for all $a \in A$. Let $a \in A$. It is clear that we have $\mathfrak{B}|_a^2 = \mathfrak{A}$ and that $\mathfrak{B}|_a^2 \models \varphi$ (because $\mathfrak{A} \models \varphi$ and φ never mentions the third values of the elements since it is a formula in $\text{dFO}[\Sigma, 1]$). Consequently $\mathfrak{B} \models \exists x. \langle\langle \varphi \rangle\rangle_x^2$.

Assume now that there exists $\mathfrak{B} = (A, (P_\sigma)_{\sigma \in \Sigma}, f_1, f_2, f_3)$ in $\text{Data}[\Sigma, 3]$ such that $\mathfrak{B} \models \exists x. \langle\langle \varphi \rangle\rangle_x^2$. Hence there exists $a \in A$ such that $\mathfrak{B}|_a^2 \models \varphi$, but then by forgetting the third value in $\mathfrak{B}|_a^2$ we obtain a model in $\text{Data}[\Sigma, 2]$ which satisfies φ . \square

Using Theorem 1, we obtain the following result.

Theorem 7 *The problem $\text{DATASAT}(\exists\text{-2-Loc-dFO}, 3)$ is undecidable.*

References

- [1] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick & L. Segoufin (2011): *Two-variable logic on data words*. *ACM Trans. Comput. Log.* 12(4), pp. 27:1–27:26, doi:10.1145/1970398.1970403.
- [2] M. Bojanczyk, A. Muscholl, T. Schwentick & L. Segoufin (2009): *Two-variable logic on data trees and XML reasoning*. *J. ACM* 56(3), doi:10.1145/1516512.1516515.
- [3] Benedikt Bollig, Arnaud Sangnier & Olivier Stietel (2021): *Local First-Order Logic with Two Data Values*. In: *FSTTCS'21, LIPIcs* 213, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 39:1–39:15, doi:10.4230/LIPIcs.FSTTCS.2021.39.
- [4] Egon Börger, Erich Grädel & Yuri Gurevich (1997): *The Classical Decision Problem*. *Perspectives in Mathematical Logic*, Springer, doi:10.1023/A:1008334715902.
- [5] N. Decker, P. Habermehl, M. Leucker & D. Thoma (2014): *Ordered Navigation on Multi-attributed Data Words*. In Paolo Baldan & Daniele Gorla, editors: *CONCUR'14, Lecture Notes in Computer Science* 8704, Springer, pp. 497–511, doi:10.1007/978-3-662-44584-6_34.
- [6] Kousha Etessami, Moshe Y. Vardi & Thomas Wilke (2002): *First-Order Logic with Two Variables and Unary Temporal Logic*. *Inf. Comput.* 179(2), pp. 279–295, doi:10.1006/inco.2001.2953.
- [7] Melvin Fitting (2012): *Torben Braüner, Hybrid Logic and its Proof-Theory, Applied Logic Series Volume 37, Springer, 2011, pp. XIII+231. ISBN: 978-94-007-0001-7. Stud Logica* 100(5), pp. 1051–1053, doi:10.1007/s11225-012-9439-2.
- [8] H. Gaifman (1982): *On local and nonlocal properties*. In J. Stern, editor: *Logic Colloquium '81*, North-Holland, pp. 105–135, doi:10.1016/S0049-237X(08)71879-2.
- [9] W. Hanf (1965): *Model-theoretic methods in the study of elementary logic*. In J.W. Addison, L. Henkin & A. Tarski, editors: *The Theory of Models*, North Holland, pp. 132–145, doi:10.2307/2271017.
- [10] A. Janiczak (1953): *Undecidability of some simple formalized theories*. *Fundamenta Mathematicae* 40, pp. 131–139, doi:10.2307/2964197.
- [11] A. Kara, T. Schwentick & T. Zeume (2010): *Temporal Logics on Words with Multiple Data Values*. In Kamal Lodaya & Meena Mahajan, editors: *FSTTCS'10, LIPIcs* 8, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 481–492, doi:10.4230/LIPIcs.FSTTCS.2010.481.

- [12] E. Kieronski (2005): *Results on the Guarded Fragment with Equivalence or Transitive Relations*. In C.-H. Luke Ong, editor: *CSL'05, Lecture Notes in Computer Science 3634*, Springer, pp. 309–324, doi:10.1007/11538363_22.
- [13] E. Kieronski & L. Tendera (2009): *On Finite Satisfiability of Two-Variable First-Order Logic with Equivalence Relations*. In: *LICS'09*, IEEE, pp. 123–132, doi:10.1109/LICS.2009.39.
- [14] L. Libkin (2004): *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/978-3-662-07003-1.
- [15] A. Manuel & T. Zeume (2013): *Two-Variable Logic on 2-Dimensional Structures*. In Simona Ronchi Della Rocca, editor: *CSL'13, LIPIcs 23*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 484–499, doi:10.4230/LIPIcs.CSL.2013.484.
- [16] Martin Mundhenk & Thomas Schneider (2009): *The Complexity of Hybrid Logics over Equivalence Relations*. *J. Log. Lang. Inf.* 18(4), pp. 493–514, doi:10.1007/s10849-009-9089-6.
- [17] T. Tan (2014): *Extending two-variable logic on data trees with order on data values and its automata*. *ACM Trans. Comput. Log.* 15(1), pp. 8:1–8:39, doi:10.1145/2559945.

Capturing Bisimulation-Invariant Exponential-Time Complexity Classes

Florian Bruse

University of Kassel
Kassel, Germany

florian.bruse@uni-kassel.de

David Kronenberger

University of Kassel
Kassel, Germany

Martin Lange

University of Kassel
Kassel, Germany

martin.lange@uni-kassel.de

Otto's Theorem characterises the bisimulation-invariant PTIME queries over graphs as exactly those that can be formulated in the polyadic μ -calculus, hinging on the Immerman-Vardi Theorem which characterises PTIME (over ordered structures) by First-Order Logic with least fixpoints. This connection has been extended to characterise bisimulation-invariant EXPTIME by an extension of the polyadic μ -calculus with functions on predicates, making use of Immerman's characterisation of EXPTIME by Second-Order Logic with least fixpoints.

In this paper we show that the bisimulation-invariant versions of all classes in the exponential time hierarchy have logical counterparts which arise as extensions of the polyadic μ -calculus by higher-order functions. This makes use of the characterisation of k -EXPTIME by Higher-Order Logic (of order $k + 1$) with least fixpoints, due to Freire and Martins.

1 Introduction

Descriptive complexity theory aims at characterising complexity classes – usually defined via computational resources like time or space consumption – by means of *logical* resources. Its central notion is that of a complexity class \mathcal{C} being *captured* by a logic \mathcal{L} in the sense that the properties which can be checked in complexity \mathcal{C} are exactly those that can be defined in \mathcal{L} . This provides a characterisation of computational complexity that is independent of a machine model. Instead, computational difficulty is characterised by the need for particular logical resources like k -order quantifiers or fixpoints of a particular type. It is widely believed that this provides a more promising line of attack for notoriously difficult problems of separating complexity classes; at least it makes machinery that is traditionally used for measuring the expressive power of logics available for such tasks.

Here we adopt terminology from database theory as this is traditionally close to descriptive complexity, and speak of *queries* being *answered* instead of problems being solved or languages being decided, different names for the same thing.

Ever since Fagin's seminal work showing that the complexity class NP is captured by Existential Second-Order Logic [5], descriptive complexity has provided logical characterisations of many standard complexity classes. The Abiteboul-Vianu Theorem for instance states that PSPACE is captured by First-Order Logic with Partial Fixpoint Operators [1]. The proofs of these results rely on the existence of a total order on the structure at hand. This is not a restriction for characterisations of complexity classes including and above NP as the resources available there are sufficient to construct such an order.

There is no known way to define or construct such a total order in deterministic polynomial time which is a major obstacle for capturing the complexity class P. There is some belief that such a logic should exist, possibly in the form of first-order logic with additional operators like fixpoints, counting and others, cf. [7]. This is grounded in the characterisation of the complexity class of *Ordered Polynomial*

Time – i.e. those queries that can be answered in polynomial time on structures that are equipped with a total order – by First-Order Logic with Least Fixpoints [20, 8].

An interesting result was then found by Otto who considered another restriction of the class P, namely that of *bisimulation-invariant* queries (on graphs, naturally). He showed that this class, denoted P/\sim is captured by the polyadic μ -calculus \mathcal{L}_μ^ω [17], a generalisation of the well-known modal μ -calculus to interpretations of formulas not in states but in tuples of states of fixed arity [2]. This is particularly interesting as it shifts the borderline at which the availability of an order becomes critical, from between P and NP to between P/\sim and $NLOGSPACE/\sim$.

This opens up the question of further capturing results of bisimulation-invariant complexity classes by (modal) logics. Indeed, characterisations have been found for $EXPTIME/\sim$ and $PSPACE/\sim$ in terms of an extension of \mathcal{L}_μ^ω by first-order functions from predicates to predicates [15]. The logic capturing $EXPTIME/\sim$ is coined PHFL¹ – Polyadic Higher-Order Fixpoint Logic of order 1. The higher-order extension is borrowed from HFL which extends the modal μ -calculus with a simply typed λ -calculus [21]. A syntactical restriction called *tail-recursiveness* [4] has been identified that captures $PSPACE/\sim$ [15], and when applying this restriction to \mathcal{L}_μ^ω or, likewise, PHFL⁰, one captures $NLOGSPACE/\sim$ (with the help of a particular partial order only, though) [15].

Considering bisimulation-invariant complexity classes above NP/\sim does not serve the same purpose as it does for smaller ones as one of the key motivations for moving to the bisimulation-invariant world is to avoid the need for a total order. On the other hand, for two complexity classes $\mathcal{C}, \mathcal{C}'$ that have complete and bisimulation-invariant problems we have $\mathcal{C} \neq \mathcal{C}'$ iff $\mathcal{C}/\sim \neq \mathcal{C}'/\sim$. Most of the standard complexity classes possess such problems, for example (1-letter) NFA universality for NP, resp. PSPACE, unbounded tree automaton intersection for EXPTIME, etc. Moreover, separating bisimulation-invariant classes may be easier due to their close connection to modal logics where separating their expressiveness is routinely done (not for the relatively complex higher-order modal fixpoint logics mentioned here, though).

In this paper we extend the characterisation of bisimulation-invariant time complexity classes to all the levels of the exponential time hierarchy. We show that k -EXPTIME/ \sim is captured by PHFL^k, the polyadic version of the higher-order extension of the modal μ -calculus with functions up to type order k . We remark that a similar characterisation of the space complexity classes k -EXSPACE is also possible [13] but needs to be omitted for lack of space and is therefore left for a future publication.

The paper is organised as follows. In Sect. 2 we recall the necessary preliminaries, mainly about the logics studied here. In Sect. 3 we provide the easy half of the capturing result by putting together known results and constructions which witness that any PHFL^k query can be answered in k -fold exponential time. In Sect. 4 we prepare for the more difficult and other half of the capturing result. We rely on a logical characterisation of k -EXPTIME in terms of Higher-Order Predicate Logic with Fixpoints [6], and a key step in expressing such (bisimulation-invariant) queries in Higher-Order Modal Fixpoint is the modelling of higher-order quantification using an enumeration technique. In Sect. 5 we put this to use for showing that any k -EXPTIME/ \sim -query is definable in PHFL^k. In Sect. 6 we conclude with remarks on further work etc.

2 Preliminaries

Let \mathbf{P} and \mathbf{A} be finite sets of *propositions*, resp. *actions*. A labelled transition system (LTS) is a tuple $T = (S, \{\xrightarrow{a}\}_{a \in \mathbf{A}}, \ell)$ where S is a nonempty set of *states*, $\xrightarrow{a} \subseteq S \times S$ is a *transition relation* for each action $a \in \mathbf{A}$, and $\ell: S \rightarrow 2^{\mathbf{P}}$ labels the states with those propositions that are true in them. We write $s \xrightarrow{a} t$ instead of $(s, t) \in \xrightarrow{a}$.

For $d \geq 1$, a d -pointed LTS is a pair $T, (s_1, \dots, s_d)$ of an LTS and a d -tuple of states in it. It is also simply called *pointed* when d is clear from the context. Note that in the case of $d = 1$, this coincides with the usual notion of a pointed LTS. Given some tuple $\bar{s} = (s_1, \dots, s_d)$ and $i \leq d$, we write $\bar{s}[t/i]$ to denote the tuple $(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_d)$.

2.1 Polyadic Higher-Order Fixpoint Logic

We assume familiarity with the modal μ -calculus \mathcal{L}_μ [12]. Polyadic Higher-Order Fixpoint Logic PHFL [15] extends \mathcal{L}_μ in several ways: (I) HFL [21] adds to it a simply typed λ -calculus; (II) the polyadic $\mathcal{L}_\mu, \mathcal{L}_\mu^\omega$ [2, 17] is obtained by lifting the interpretation of formulas in states to tuples of states of fixed arity. Now PHFL merges both extensions. Its introduction requires a few technicalities.

Types. Types are used to govern the syntax of PHFL, especially those parts that denote functions. They are derived from the grammar

$$\tau ::= \bullet \mid \tau^v \rightarrow \tau$$

where \bullet is a ground type for propositions, $v \in \{+, -, 0\}$ are *variances* denoting whether a function is monotonically increasing, monotonically decreasing, or constant in its argument. Variances are only needed to check well-typedness; we often do not display them for the sake of readability.

The *order* ord of a type is defined via $ord(\bullet) = 0$ and $ord(\tau_1 \rightarrow \tau_2) = \max(ord(\tau_1) + 1, ord(\tau_2))$. Types associate to the right whence every type is of the form $\tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \bullet$.

Given some LTS $T = (S, \{\xrightarrow{a}\}_{a \in \mathbf{A}}, \ell)$ and some $0 < d \in \mathbb{N}$, the semantics $\llbracket \tau \rrbracket^T$ of a type τ is a complete lattice, defined inductively as follows:

$$\llbracket \bullet \rrbracket^T = (2^{S^d}, \subseteq) \qquad \llbracket \tau_2^v \rightarrow \tau_1 \rrbracket^T = (\llbracket \tau_2 \rrbracket^T \rightarrow \llbracket \tau_1 \rrbracket^T, \sqsubseteq_{\tau_2^v \rightarrow \tau_1})$$

where we tacitly identify a lattice with its domain and where the order $\sqsubseteq_{\tau_2^v \rightarrow \tau_1}$ is given by

$$f \sqsubseteq_{\tau_2^v \rightarrow \tau_1} g \iff \text{for all } x \in \llbracket \tau_2 \rrbracket^T \text{ we have } \begin{cases} f(x) \sqsubseteq_{\tau_1} g(x), & \text{if } v = + \\ g(x) \sqsubseteq_{\tau_1} f(x), & \text{if } v = - \\ f(x) = g(x), & \text{if } v = 0. \end{cases}$$

Hence, the semantics of \bullet is the powerset lattice over S^d and $\tau_2^v \rightarrow \tau_1$ is the lattice of all monotonically increasing, monotonically decreasing or constant functions from $\llbracket \tau_2 \rrbracket^T$ to $\llbracket \tau_1 \rrbracket^T$, depending on v . Such a set together with the above point-wise order forms a complete lattice if $\llbracket \tau_1 \rrbracket^T$ is one. Hence, monotone functions in such a lattice always have a least and greatest fixpoint due to the Knaster-Tarski-Theorem. We write $\bigsqcup_{\tau_2^v \rightarrow \tau_1}$ to denote the join operator in this lattice.

Note that, technically, $\llbracket \tau_1 \rrbracket^T$ is also parameterised in d . However, since d is usually clear from context, we do not display it to avoid clutter.

Syntax. Let \mathbf{P} and \mathbf{A} be as above and let \mathbf{F} and \mathbf{L} be finite sets of *fixpoint variables*, resp. *lambda variables*. We use upper case letters X, Y, \dots for the former and lower case letters x, y, f, g for the latter.

Let $d \geq 1$. By $[d]$ we denote the set $1, \dots, d$. The set of – potentially non-well-formed – formulas of the d -adic fragment of PHFL, called PHFL $_d$, is derived via

$$\varphi ::= p_i \mid \varphi \vee \varphi \mid \neg \varphi \mid \langle a_i \rangle \varphi \mid \{ \sigma \} \varphi \mid \lambda(x: \tau). \varphi \mid x \mid (\varphi \varphi) \mid \mu(X: \tau). \varphi \mid X$$

$$\begin{array}{c}
\frac{}{\Gamma \vdash p_i : \bullet} \quad \frac{\Gamma \vdash \varphi_1 : \bullet \quad \Gamma \vdash \varphi_2 : \bullet}{\Gamma \vdash \varphi_1 \vee \varphi_2 : \bullet} \quad \frac{\bar{\Gamma} \vdash \varphi : \bullet}{\Gamma \vdash \neg \varphi : \bullet} \quad \frac{\Gamma \vdash \varphi : \bullet}{\Gamma \vdash \langle a_i \rangle \varphi : \bullet} \quad \frac{\Gamma \vdash \varphi : \bullet}{\Gamma \vdash \{\sigma\} \varphi : \bullet} \\
\\
\frac{\Gamma, x^v : \tau_1 \vdash \varphi : \tau_2}{\Gamma \vdash \lambda(x^v : \tau_1). \varphi : \tau_1^v \rightarrow \tau_2} \quad \frac{v \in \{+, 0\}}{\Gamma, x^v : \tau \vdash x : \tau} \quad \frac{\Gamma, X^+ : \tau \vdash \varphi : \tau}{\Gamma \vdash \mu(X : \tau). \varphi : \tau} \quad \frac{}{\Gamma, X^+ : \tau \vdash X : \tau} \\
\\
\frac{\Gamma \vdash \varphi_1 : \tau_2^+ \rightarrow \tau_1 \quad \Gamma \vdash \varphi_2 : \tau_2}{\Gamma \vdash (\varphi_1 \varphi_2) : \tau_1} \quad \frac{\Gamma \vdash \varphi_1 : \tau_2^- \rightarrow \tau_1 \quad \bar{\Gamma} \vdash \varphi_2 : \tau_2}{\Gamma \vdash (\varphi_1 \varphi_2) : \tau_1} \\
\\
\frac{\Gamma \vdash \varphi_1 : \tau_2^0 \rightarrow \tau_1 \quad \Gamma \vdash \varphi_2 : \tau_2 \quad \bar{\Gamma} \vdash \varphi_2 : \tau_2}{\Gamma \vdash \varphi_1 \varphi_2 : \tau_1}
\end{array}$$

Figure 1: The PHFL typing system.

where $x \in \mathbf{L}$, $X \in \mathbf{F}$, $p \in \mathbf{P}$, $a \in \mathbf{A}$, $1 \leq i \leq d$ and $\sigma : [d] \rightarrow [d]$ is a mapping on so-called *indices*. We assume that standard connectives such as ff , \wedge , $[a]$ and $v(X : \tau)$ are available via the obvious dualities if needed. We will also allow ourselves to use more convenient notation for the definition of (more complex) functions and their applications. For instance, $(\dots((\varphi \psi_1) \psi_2) \dots) \psi_n$ is simply written as $\varphi(\psi_1, \dots, \psi_n)$, and $\lambda(x_1 : \tau_1). \lambda(x_2 : \tau_2). \dots \lambda(x_n : \tau_n). \psi$ is written as $\lambda(x_1 : \tau_1, \dots, x_n : \tau_n). \psi$, or even $\lambda(x_1, \dots, x_n : \tau). \psi$ if $\tau = \tau_i$ for all $1 \leq i \leq n$.

The notions of free and bound variables in a formula are as usual, with $\lambda(x : \tau). \varphi$ binding x and with $\mu(X : \tau). \varphi$ binding X .

Over an LTS T , a PHFL_d formula intuitively defines a set of d -tuples in T , or a function that transforms such a set into such a set, or into a function etc., depending on the formula's type which will be explained shortly. Before that we briefly introduce the intuitive meaning of the operators in the syntax. The first five listed in the grammar above all define a subset of S^d . For example, p_i denotes all tuples such that p holds at their i th state. The operator $\{\sigma\} \varphi$ rearranges the positions in tuples in such a subset. The modality $\langle a_i \rangle \varphi$ expresses that φ holds on a tuple after replacing the i th state in it by some a -successor. A formula of the form $\lambda(x : \tau^v). \varphi$ defines a function that consumes an argument of type τ and is monotonically increasing, monotonically decreasing, or constant in this argument, depending on v . A formula of the form $(\varphi \psi)$ denotes the application of the semantics of φ to the object defined by ψ . Finally, fixpoints can now also define higher-order functions.

Obviously, not all formulas that can be derived from the above grammar can be given a semantics in a meaningful way; consider e.g. $(p_1 p_2)$. Moreover, as is typical in a situation involving least and greatest fixpoints, the use of negation has to be restricted, cf. the example $\mu(X : \bullet). \neg X$. Hence, PHFL has a type system to filter out formulas that cannot be endowed with a proper semantics.

A finite sequence Γ of *hypotheses* of the form $X^v : \tau$ or $x^v : \tau$ in which each variable occurs at most once is called a *context*. The dual context $\bar{\Gamma}$ is obtained from Γ by replacing all the hypotheses of the form $X^+ : \tau$ by $X^- : \tau$ and vice versa, and doing the same for lambda variables. We say that φ has type τ in the context Γ if the statement $\Gamma \vdash \varphi : \tau$ can be derived from the rules in Fig. 1. A formula without free variables is *well-typed* if the statement $\emptyset \vdash \varphi : \bullet$ can be derived from these rules. We tacitly assume that each fixpoint variable and each lambda variable is bound at most once in a well-typed formula, and that no variable occurs both freely and bound in a formula. Hence, each variable has a unique type in the context of a given, well-formed formula. If the type information is clear from context or not important,

we drop it from binders, simply writing $\lambda x. \varphi$ and $\mu X. \varphi$ for better readability.

A formula is said to be of order k if the maximal order of the type of any subformula in φ is k . By PHFL_d^k we denote the set of well-typed formulas in PHFL_d that are of order k . Note that PHFL does indeed constitute an extension of other known formalisms, namely

- PHFL_1^k is the same as HFL^k for any $k \geq 0$ and, thus $\text{PHFL}_1 = \text{HFL}$, and in particular
- PHFL_1^0 is just the modal μ -calculus \mathcal{L}_μ while
- PHFL_d^0 is the d -dimensional polyadic μ -calculus.

Semantics. Let $T = (S, \{\xrightarrow{a}\}_{a \in \mathbf{A}}, \ell)$ be an LTS and let φ be a well-typed formula. An *environment* is a function η that assigns to each fixpoint variable and each lambda variable of type τ an element of $\llbracket \tau \rrbracket^T$.

Let $d \geq 1$. The semantics $\llbracket \varphi : \tau \rrbracket_\eta^T$ of a PHFL_d formula φ of type τ , relative to an LTS T and an environment η , is an object of $\llbracket \tau \rrbracket^T$, defined recursively as follows.

$$\begin{aligned}
\llbracket \Gamma \vdash p_i : \bullet \rrbracket_\eta^T &= \{(s_1, \dots, s_d \in S^d \mid p \in \ell(s_i))\} \\
\llbracket \Gamma \vdash \varphi_1 \vee \varphi_2 : \bullet \rrbracket_\eta^T &= \llbracket \Gamma \vdash \varphi_1 : \bullet \rrbracket_\eta^T \cup \llbracket \Gamma \vdash \varphi_2 : \bullet \rrbracket_\eta^T \\
\llbracket \Gamma \vdash \neg \varphi : \bullet \rrbracket_\eta^T &= S^d \setminus \llbracket \Gamma \vdash \varphi : \bullet \rrbracket_\eta^T \\
\llbracket \Gamma \vdash \langle a_i \rangle \varphi : \bullet \rrbracket_\eta^T &= \{(s_1, \dots, s_d) \in S^d \mid \text{ex. } t \text{ s.t.} \\
&\quad (s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_d) \in \llbracket \Gamma \vdash \varphi : \bullet \rrbracket_\eta^T \text{ and } s_i \xrightarrow{a} t\} \\
\llbracket \Gamma \vdash \{\sigma\} \varphi : \bullet \rrbracket_\eta^T &= \{(s_1, \dots, s_d) \in S^d \mid (s_{\sigma(1)}, \dots, s_{\sigma(d)}) \in \llbracket \Gamma \vdash \varphi : \bullet \rrbracket_\eta^T\} \\
\llbracket \Gamma \vdash \lambda(x^v : \tau_2). \varphi : \tau_2^v \rightarrow \tau_1 \rrbracket_\eta^T &= f \in \llbracket \tau_2^v \rightarrow \tau_1 \rrbracket^T \text{ s.t. f.a. } y \in \llbracket \tau_2 \rrbracket^T. f(y) = \llbracket \Gamma, x^v : \tau_2 \vdash \varphi : \tau_1 \rrbracket_{\eta[x \mapsto y]}^T \\
\llbracket \Gamma \vdash x : \tau \rrbracket_\eta^T &= \eta(x) \\
\llbracket \Gamma \vdash (\varphi_1 \varphi_2) : \tau_1 \rrbracket_\eta^T &= \llbracket \Gamma \vdash \varphi_1 : \tau_2^v \rightarrow \tau_1 \rrbracket_\eta^T (\llbracket \Gamma \vdash \varphi_2 : \tau_2 \rrbracket_\eta^T) \\
\llbracket \Gamma \vdash \mu(X : \tau). \varphi : \tau \rrbracket_\eta^T &= \bigsqcap_{\tau \rightarrow \tau} \{d \in \llbracket \tau \rrbracket_\eta^T \mid \llbracket \Gamma, X : \tau^+ \vdash \varphi : \tau \rrbracket_{\eta[X \mapsto d]}^T \sqsubseteq \tau d\} \\
\llbracket \Gamma \vdash X : \tau \rrbracket_\eta^T &= \eta(X)
\end{aligned}$$

If the type is clear from context, or not important, we simply write $\llbracket \varphi \rrbracket_\eta^T$. We write $T, (s_1, \dots, s_d) \models_\eta \varphi$ if $\varphi : \bullet$ and $(s_1, \dots, s_d) \in \llbracket \varphi \rrbracket_\eta^T$. We say that two formulas φ and ψ are *equivalent*, written $\varphi \equiv \psi$, if for all T and all η we have $\llbracket \varphi \rrbracket_\eta^T = \llbracket \psi \rrbracket_\eta^T$.

It is well-known that the semantics of HFL and, hence, PHFL is invariant under β -reduction and admits the fixpoint unfolding principle, i.e. $\mu X. \varphi \equiv \varphi[\mu X. \varphi/X]$ where substitution is defined as usual.

Bisimilarity. A *bisimulation* R on an LTS $T = (S, \{\xrightarrow{a}\}_{a \in \mathbf{A}}, \ell)$ is a symmetric relation $R \subseteq S \times S$ satisfying the following for all $(s, t) \in R$.

- $\ell(s) = \ell(t)$,
- if there is $a \in \mathbf{A}$ and $s' \in S$ s.t. $s \xrightarrow{a} s'$ then there is $t' \in S$ with $t \xrightarrow{a} t'$ and $(s', t') \in R$,
- if there is $a \in \mathbf{A}$ and $t' \in S$ s.t. $t \xrightarrow{a} t'$ then there is $s' \in S$ with $s \xrightarrow{a} s'$ and $(s', t') \in R$.

Two states s, t are bisimilar, written $s \sim t$, if there is a bisimulation R with $(s, t) \in R$.

Let $d \geq 1$. A set $T \subseteq S$ is called *bisimulation-invariant* if for all $\bar{s} = (s_1, \dots, s_d), \bar{t} = (t_1, \dots, t_d) \in S^d$ such that $s_i \sim t_i$ for all $i \in [d]$, we have $\bar{s} \in T$ iff $\bar{t} \in T$. The notion of bisimulation-invariance can straight-forwardly be lifted to objects of type $\llbracket \tau \rrbracket^T$ for types $\tau \neq \bullet$, cf. [21].

It is well-known that modal logics cannot distinguish bisimilar models, and it is not surprising that the extensions beyond pure modal logic that are available in PHFL do not break this property.

Proposition 1 ([21, 15]). *Let $d \geq 1$, T be an LTS with state set S and φ be a closed PHFL $_d$ formula of type \bullet . Then $\llbracket \varphi \rrbracket^T \subseteq S^d$ is bisimulation-invariant.*

Example 2 ([17]). A standard example shows that bisimilarity itself is definable in PHFL $_2^0$, provided that \mathbf{P} and \mathbf{A} are finite. The PHFL $_2^0$ formula

$$\varphi_{\sim} := \nu(X : \bullet). \left(\bigwedge_{p \in \mathbf{P}} p_1 \leftrightarrow p_2 \right) \wedge \left(\bigwedge_{a \in \mathbf{A}} [a_1] \langle a_2 \rangle X \right) \wedge \{1 \mapsto 2, 2 \mapsto 1\} X$$

is satisfied by a pair (s, t) of some T iff $s \sim t$. The formula essentially states that (s, t) needs to belong to the largest set X of states (s', t') that agree on all propositions (first conjunct) and for which t' can match any a -transition out of s' , for any $a \in \mathbf{A}$, to a pair $(s'', t'') \in X$ (second conjunct). Moreover, X needs to be a symmetric relation (third conjunct).

To exemplify the use of higher-orderness (here: first-order functions) we can use the definability of finite-trace equivalence in PHFL $_2^1$.

Example 3. Two states s, t are finite-trace equivalent if whenever there is a sequence $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} s_n$ then there are t_1, \dots, t_n s.t. $t \xrightarrow{a_1} t_1 \xrightarrow{a_2} t_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} t_n$ and vice-versa.

$$\varphi_{\text{fte}} := \left(\nu F(x, y). (x \leftrightarrow y) \wedge \bigwedge_{a \in \mathbf{A}} F(\langle a_1 \rangle x, \langle a_2 \rangle y) \right) (\text{tt}, \text{tt})$$

is then satisfied by a pair (s, t) iff s and t are finite-trace equivalent in the sense above. The greatest fixpoint in the formula expresses an infinite conjunction over all finite paths, these can be thought of to be built step-wise via fixpoint unfolding; the n th unfolding expresses that all finite traces of length n are available in the first component of the tuple (and built via $\langle a_1 \rangle$) iff they are available in the second component (built via $\langle a_2 \rangle$). For better readability we have omitted the types in the formula. The types of x, y are \bullet and that of F is, consequently, $\bullet^0 \rightarrow \bullet^0 \rightarrow \bullet$.

We remark that it is easily possible to extend the formula to also check for matching atomic propositions along the traces emerging from s and t . Cf. [16] for further examples of how various other process equivalences and preorders can be expressed in PHFL 1 .

2.2 Higher-Order Logic with Least Fixpoints

We introduce Higher-Order Logic with Least Fixpoints (HO(LFP)) to make use of the characterisation of k -EXPTIME over the class of ordered structures as the queries definable in order- $(k+1)$ HO(LFP), due to Immerman and Vardi [8, 20, 9], resp. Freire and Martins [6].

Types. Types for Higher-Order Logic with Least Fixpoints are constructed, similar to those for PHFL, from a single base type and one constructor: $\tau' ::= \odot \mid (\tau', \dots, \tau')$. Here, however, \odot is the type of *individuals* (like states rather than sets of states), and the tuple type is used to denote (higher-order) relations. We define the order 1 of a type via $\text{ord}(\odot) = 1$ and $\text{ord}(\tau'_1, \dots, \tau'_n) = 1 + \max\{\text{ord}(\tau'_1), \dots, \text{ord}(\tau'_n)\}$.

Let $T = (S, \{\xrightarrow{a}\}_{a \in \mathbf{A}}, \ell)$ be an LTS. This induces a set-theoretic interpretation of types via $\llbracket \odot \rrbracket^T = S$ and $\llbracket (\tau'_1, \dots, \tau'_n) \rrbracket^T = 2^{\llbracket \tau'_1 \rrbracket^T \times \dots \times \llbracket \tau'_n \rrbracket^T}$.

¹Note the discrepancy in the traditional ways to assign numerals to orders in the two logics considered here: order 1 in HO(LFP) refers to, like in “First-Order Logic”, individual elements and order 2 is for relations like sets thereof. In PHFL, order 1 refers to the order of a function, i.e. one that takes sets of arguments. This explains why PHFL k corresponds to the fragment of HO(LFP) of order $k+1$, see also the right column in Fig. 2.

Syntax. Let $\mathbf{V} = \{X, \dots\}$ be a countable set of *higher-order variables*, each implicitly equipped with a type. Let \mathbf{P}, \mathbf{A} be sets of propositions, resp. actions. The syntax of HO(LFP) formulas is derived from the following grammar:

$$\varphi ::= p(X) \mid a(X, Y) \mid X(Y_1, \dots, Y_N) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists(X: \tau'). \varphi \mid (\text{lfp}(X, Y_1, \dots, Y_N). \varphi)(Z_1, \dots, Z_n)$$

where $p \in \mathbf{P}$, $a \in \mathbf{A}$ and $X, Y_1, \dots, Y_n, Z_1, \dots, Z_n \in \mathbf{V}$. Again, dual operators such as \wedge and *gfp* (for greatest fixpoints) are available via the obvious dualities. The variable X is bound in $\exists(X: \tau'). \varphi$ and X, Y_1, \dots, Y_n are bound in $(\text{lfp}(X, Y_1, \dots, Y_N). \varphi)(Z_1, \dots, Z_n)$. A formula is *well-formed* if each variable is bound at most once, and, moreover, variables occur only in a way that matches their type. For example, only a variable of type \odot can occur in a subformula of the form $p(X)$, and if a subformula of the form $X(Y_1, \dots, Y_n)$ occurs, then X has type $(\tau'_1, \dots, \tau'_n)$ for some τ'_1, \dots, τ'_n and Y_i has type τ'_i for all $1 \leq i \leq n$. Moreover, in a subformula of the form $(\text{lfp}(X, Y_1, \dots, Y_N). \varphi)(Z_1, \dots, Z_n)$, the variable X occurs only under an even number of negations in φ . For a more detailed introduction into Higher-Order Logic including formal ways to define well-formedness of formulas, cf. [19]. In a well-formed formula each variable has a unique type. An HO(LFP) formula φ has order k if the order of the highest type of a variable in φ is at most k . We write $\text{HO}^k(\text{LFP})$ for the set of HO(LFP) formulas of order at most k .

Semantics. Let $T = (S, \{\xrightarrow{a}\}_{a \in \mathbf{A}}, \ell)$ be an LTS. A variable assignment α is a function that maps each variable in \mathbf{V} of type τ' into $\llbracket \tau' \rrbracket^T$.

Let φ be an HO(LFP) formula with free variables in X, Y_1, \dots, Y_n such that X occurs only under an even number of negations in φ and let $\tau' = (\tau'_1, \dots, \tau'_n)$ be the type of X in φ while τ'_i is the type of Y_i in φ for $1 \leq i \leq n$. Then, given some variable assignment α , the formula φ defines a monotone function $f: \llbracket \tau' \rrbracket^T \rightarrow \llbracket \tau' \rrbracket^T$ via

$$f(M) \mapsto \{(m_1, \dots, m_n) \in \llbracket \varphi \rrbracket_{\alpha[Y_1 \mapsto m'_1, \dots, Y_n \mapsto m'_n]}^T \mid (m'_1, \dots, m'_n) \in M\}.$$

By the Knaster-Tarski-Theorem [11, 18] this function has a least fixpoint denoted by $\text{LFP}(X, Y_1, \dots, Y_n)\varphi$. Note that we suppress T and α here since they will always be clear from context.

The satisfaction relation between an LTS T , a variable assignment α and an HO(LFP) formula φ is defined inductively as follows.

$$\begin{aligned} T, \alpha &\models p(X) \text{ iff } p \in \ell(\alpha(X)) \\ T, \alpha &\models a(X, Y) \text{ iff } \alpha(X) \xrightarrow{a} \alpha(Y) \\ T, \alpha &\models X(Y_1, \dots, Y_n) \text{ iff } (\alpha(Y_1), \dots, \alpha(Y_n)) \in \alpha(X) \\ T, \alpha &\models \neg\varphi \text{ iff } T, \alpha \not\models \varphi \\ T, \alpha &\models \varphi_1 \vee \varphi_2 \text{ iff } T, \alpha \models \varphi_1 \text{ or } T, \alpha \models \varphi_2 \\ T, \alpha &\models \exists(X: \tau') \text{ iff ex. } d \in \llbracket \tau' \rrbracket^T \text{ s.t. } T, \alpha[X \mapsto d] \models \varphi \\ T, \alpha &\models (\text{lfp}(X, Y_1, \dots, Y_N). \varphi)(Z_1, \dots, Z_n) \text{ iff } (\alpha(Z_1), \dots, \alpha(Z_n)) \in \text{LFP}(X, Y_1, \dots, Y_n)\varphi. \end{aligned}$$

2.3 Descriptive Complexity

A *query* (of dimension d) is a set Q of pairs $(T, (s_1, \dots, s_d))$ s.t. each T is finite. It is expressed by the HO(LFP) formula φ with free variables X_1, \dots, X_d , if $Q = \{(T, (s_1, \dots, s_d)) \mid T, [X_1 \mapsto s_1, \dots, X_d \mapsto s_d] \models \varphi\}$. Let $k \geq 0$. The complexity class k -EXPTIME is defined as $\text{DTIME}(2_k^{m^{\sigma(1)}})$ where $2_0^m := m$ and $2_{k+1}^m := 2^{2_k^m}$. Note that 0-EXPTIME equals P.

For a complexity class \mathcal{C} , a \mathcal{C} -query is one that can be decided within the resource bounds given by \mathcal{C} . We write \mathcal{C}/\sim for the complexity class of \mathcal{C} -queries that are bisimulation-invariant.

We say that a logic \mathcal{L} captures a complexity class \mathcal{C} if the model checking problem for \mathcal{L} is in \mathcal{C} and each \mathcal{C} -query can be expressed in \mathcal{L} .

The Immerman-Vardi Theorem characterises the P-queries over ordered structures as those expressible in $\text{HO}^1(\text{LFP})$, resp. first-order logic with least fixpoints. It's generalisation is the following:

Proposition 4 ([8, 20, 9, 6]). *For each $k \geq 0$, $\text{HO}^{k+1}(\text{LFP})$ captures $k\text{-EXPTIME}$ over the class of ordered structures.*

The ordering is only important for the case of $k = 0$ as such an ordering can be defined in second-order logic, i.e. as soon as $k \geq 1$. It is an open problem whether a logic exists that captures P over the class of all structures (cf. e.g. [7]).

The first capturing result for a bisimulation-invariant class is given by Otto's Theorem.

Proposition 5 ([17]). *The polyadic modal μ -calculus \mathcal{L}_μ^0 , or, equivalently PHFL^0 , captures P/\sim , or equivalently, $0\text{-EXPTIME}/\sim$.*

We give a quick sketch of its proof in order to prepare for the technical developments in Sects. 4 and 5. The model checking problem for PHFL^0 is readily seen to be in P (see also [14]). The interesting part is to show that every bisimulation-invariant P query can be expressed in PHFL^0 . In principle, this could be done by encoding runs of polynomially time-bounded Turing machines, but it is not immediately clear how bisimulation-invariance of the query in question can be used. Instead, the proof for this rests on a key observation: non-bisimilarity of two states can be expressed in PHFL_2^0 by a least-fixpoint formula, since bisimilarity can be defined via the greatest fixpoint formula in Ex. 2. Hence, non-bisimilarity of two states, as a least fixpoint, has a well-founded reason, i.e. one that can be found in finitely many fixpoint unfoldings. Ordering the atomic types in an arbitrary way entails a total order on the bisimulation-equivalence classes, and this order can be defined in PHFL_2^0 . Hence, the LTS in question is ordered, and the Immerman-Vardi Theorem is available, whence the problem reduces to showing that every query defined by a bisimulation-invariant $\text{HO}^1(\text{LFP})$ -query can be expressed equivalently in PHFL^0 .

This latter reduction now follows from a rather straightforward translation from $\text{HO}^1(\text{LFP})$ formulas to PHFL^0 formulas. Variables of type \odot are emulated through polyadicity and variables of type (\odot, \dots, \odot) are represented as order-0 PHFL^0 -variables. Since, in the bisimulation-invariant setting, one can always assume that the LTS in question already is its own bisimulation quotient, bisimilarity and equality coincide. Hence, a subformula of the form $a(X_i, X_j)$ can be replaced by the statement that the i th component of the relation defined has an a -successor that is bisimilar and, hence equal to the j th component. Moreover, since all states in such a bisimulation quotient of a pointed LTS are reachable from a distinguished state, existential quantification can be replaced by reachability of a suitable state. It remains to translate least fixpoints in $\text{HO}^1(\text{LFP})$ into order-0 fixpoints of PHFL^0 .

Using reasoning along similar lines, Otto's Theorem has also been generalised by one order.

Proposition 6 ([15]). *PHFL^1 captures $\text{EXPTIME}/\sim$.*

3 Upper Bounds

Capturing a complexity class \mathcal{C} , defined by some restricted resource consumption, by a logic \mathcal{L} contains two parts: what is commonly seen as the lower bound consists of showing that every query which can be evaluated in complexity \mathcal{C} can also be defined in the logic \mathcal{L} . The upper bound is established by

showing the contrary. This is relatively easy as it suffices to show that queries definable in \mathcal{L} can be evaluated in complexity \mathcal{C} , in other words that the model checking problem for \mathcal{L} belongs to class \mathcal{C} .

Here we do this for the fragments of PHFL of arbitrary but fixed arity d and arbitrary order k , w.r.t. the classes k -EXPTIME of the exponential time hierarchy. We do so by extending the reduction of the model checking problem for a polyadic logic to that of its monadic fragment [14]. Note that PHFL_1^k equals HFL^k – the fragment of (non-polyadic) Higher-Order Fixpoint Logic of formulas of type order at most k . The complexity of model checking such fragments is known:

Proposition 7 ([3]). *Let $k \geq 1$. The model checking problem for HFL^k is k -EXPTIME-complete.*

The corresponding result for PHFL_d^k , first stated without proof in [15], follows via a reduction:

Theorem 8. *Let $k, d \geq 1$. The model checking problem for PHFL_d^k is in k -EXPTIME.*

Proof. By a polynomial reduction to the model checking problem for HFL. Let $k, d \geq 1$ and an LTS $T = (S, \{\xrightarrow{a}\}_{a \in \mathbf{A}}, \ell)$ over \mathbf{A} and \mathbf{P} be given. We construct its d -product T^d over the action set $\mathbf{A}^d := \{a_i \mid a \in \mathbf{A}, i \in [d]\} \cup \{\sigma \mid \sigma : [d] \rightarrow [d]\}$ and atomic propositions $\mathbf{P}^d := \{q_i \mid q \in \mathbf{P}, i \in [d]\}$ as $(S^d, \{\xrightarrow{x}\}_{x \in \mathbf{A}_d^d}, \ell')$ where, for all $q \in \mathbf{P}$, $i \in [d]$, $a \in \mathbf{A}$, $s_1, \dots, s_d, t_1, \dots, t_d \in S$ we have

- $q_i \in \ell'(s_1, \dots, s_d)$ iff $q \in \ell(s_i)$,
- $(s_1, \dots, s_d) \xrightarrow{a_i} (t_1, \dots, t_d)$ iff $s_i \xrightarrow{a} t_i$ and $t_j = s_j$ for all $j \neq i$,
- $(s_1, \dots, s_d) \xrightarrow{\sigma} (t_1, \dots, t_d)$ iff $t_j = \sigma(s_j)$ for all $j \in [d]$.

Next, we translate a PHFL_d^k formula φ inductively into a PHFL_1 formula $\widehat{\varphi}$ as follows. The operation $\widehat{\cdot}$ acts homomorphically on all operators apart from the following three cases.

$$\widehat{\langle \sigma \rangle \psi} := \langle \sigma \rangle \widehat{\psi} \quad , \quad \widehat{\langle a_i \rangle \psi} := \langle a_i \rangle \widehat{\psi} \quad , \quad \widehat{p_i} := p_i .$$

The latter two cases may be confusing as $\widehat{\cdot}$ seems to not change those operators either. However, in e.g. the second case, $\langle a_i \rangle$ on the left side is a polyadic modality combining the action $a \in \mathbf{A}$ with the index $i \in [d]$. On the right side, $\langle a_i \rangle$ is a monadic modality over the action $a_i \in \mathbf{A}^d$. Likewise in the third case.

This equality in syntax for two technically different modal operators, resp. atomic formulas is intended because of the following connection: the d -tuple (s_1, \dots, s_d) of states in T satisfies the PHFL_d formula $\langle a_i \rangle \psi$ iff the state (s_1, \dots, s_d) of T^d satisfies the PHFL_1 formula $\langle a_i \rangle \widehat{\psi}$. A similar statement can be made for atomic propositions and these can easily be generalised to show by induction on the syntax of PHFL that for all PHFL_d formulas φ , all $s_1, \dots, s_d \in S$ and all environments η we have: $(s_1, \dots, s_d) \in \llbracket \varphi \rrbracket_\eta^T$ iff $(s_1, \dots, s_d) \in \llbracket \widehat{\varphi} \rrbracket_\eta^{T^d}$.

This establishes correctness of the reduction. Note that $\widehat{\varphi}$ is a PHFL_1^k , i.e. HFL^k formula whenever $\varphi \in \text{PHFL}_d^k$. Moreover, both T^d and $\widehat{\varphi}$ are easily seen to be constructible in polynomial time (for fixed d). Thus, by Prop. 7 model checking PHFL_d^k is also in k -EXPTIME. \square

4 Higher-Order Quantification

To show that every bisimulation-invariant k -EXPTIME query can be expressed in PHFL^k it suffices, due to Prop. 4, to show that every $\text{HO}^{k+1}(\text{LFP})$ -query can be translated into a PHFL^k formula. The main challenge here is to deal with existential quantification, which has no obvious equivalent in PHFL. In [17], first-order existential quantification is replaced by reachability of a suitable state, which is sufficient in the bisimulation-invariant setting. Higher-order quantification does not have such an obvious

correspondent - there is no notion of a set, or a set of sets, etc. being reachable. Moreover, PHFL does only have a type for sets of tuples of states, not for sets of sets etc. We solve this problem by replacing higher-order types by a variant of their characteristic function, something that fits quite naturally into the PHFL world. We then lift the order on the states inherited from [17] to sets of tuples of states and then to said characteristic functions by ordering them lexicographically. We can enumerate sets, functions and so on alongside this order, and hence, we can mimic existential quantification over some $\text{HO}(\text{LFP})$ type by an enumeration of corresponding characteristic functions.

The increased complexity of this approach compared to simple reachability requires two adaptations: first, since we use the order of sets, functions etc. present in the bisimulation-invariant setting, we often have to compare two such objects w.r.t. this order. Comparing e.g. two sets of d -ary tuples, however, requires a formula containing types of width $2d$. Also, $\text{HO}(\text{LFP})$ -formulas can define a query of some width, yet contain types, resp. quantification over objects of much higher width. Hence, in order to keep the presentation simple, the formulas we develop subsequently will be of some unspecified, yet generally quite high arity, i.e. they will be in PHFL_d^k for some d that is large compared with the width of the original query. The exact value of d will be given towards the end of the translation.

Since we have already agreed to blow up the width used in our translated formulas, we can also make things easier by reserving certain positions in the tuples we work with for special tasks. We generally use the last two positions in our tuples (those with indices d and $d - 1$) to compare individual states w.r.t. the order from [17], and we will use the next r positions from the right, i.e. those with indices $d - r - 1, \dots, d - 2$ for some r , to keep copies of states such that the whole LTS is reachable from at least one of these states, in order to keep the pattern for first-order quantification valid. This will be made formal after we revisit the pattern for existential quantification just below. We then subsequently expand quantification towards sets and characteristic functions. The final translation is then given in Sec. 5.

Reachable States. In [17], existential quantification of first-order logic was replaced by reachability of a suitable state in the LTS in question, using the pattern given subsequently. First, we recall the role of the substitution operator: Let $\sigma^{i \leftarrow j}$ be defined by $\sigma^{i \leftarrow j}(i) = j$ and $\sigma^{i \leftarrow j}(i') = i'$ if $i' \neq i$. Then $(\bar{s} \in \llbracket \{\sigma^{i \leftarrow j}\} \varphi \rrbracket_\eta^T$ iff $\bar{s}[s_j/i] = (s_1, \dots, s_{i-1}, s_j, s_{i+1}, \dots, s_d) \in \llbracket \varphi \rrbracket_\eta^T$ for all T and η .

Now let \mathbf{A} be a set of actions, let $d > 2$, $r \leq d - 2$ and $i \leq d - r - 2$. Recall that, for the time being, we assume that every tuple we work with is such that all states in an LTS are reachable from one of the states at indices $d - r - 1, \dots, d - 2$ of the tuple and that we reserve the last two positions for comparisons (see below). Consider the formula

$$\exists_i \varphi := \bigvee_{j=d-r-1}^{d-2} \{ \sigma^{i \leftarrow j} \} (\mu(X: \bullet). \varphi \vee \bigvee_{a \in \mathbf{A}} \langle a_i \rangle X).$$

for some $\varphi \in \text{PHFL}_d$. We have the following:

Observation 9. Let T be an LTS over \mathbf{A} and let $s = (s_1, \dots, s_d)$ such that all states in T are reachable from at least one state in $s_{d-r-1}, \dots, s_{d-2}$. Then $T, \bar{s} \models \exists_i \varphi$ iff there is a state t in T such that $T, \bar{s}[t/i] \models \varphi$.

As said before, quantification over higher-order types is more complicated, but we can replace reachability by enumeration in lexicographical order for higher-order types. Towards this, note that for all $d \geq 2$ there is some PHFL_2^0 -formula $\varphi_<$ defining a transitive and irreflexive relation $<$ such that, for all LTS T and d -tuples $\bar{s} = (s_1, \dots, s_d)$ we have that $T, \bar{s} \models \varphi_<$ iff $s_{d-1} < s_d$. This formula is defined in [17] using a variant of the negation of the formula from Ex. 2. The actual position of the tuple elements that are compared is not important, we choose to fix it here for consistency.

A crucial ingredient for the correctness of the quantification pattern above is that every state in the LTS is reachable from the states in positions $d - r - 1, \dots, d - 2$. For the first-order case, this can be guaranteed by never manipulating the components with the respective indices. In the higher-order setting, this does not suffice as one deals with arbitrary sets, functions, etc. Hence, for the remainder of the section all formulas are assumed to have a free lambda variable e of type \bullet that is as follows:

Definition 10. Let T be an LTS and let r be fixed. Then an interpretation η is *good* if $\eta(e)$ is a set of the form $M \times \{s_{d-r-1}\} \times \dots \times \{s_{d-2}\} \times S^2$ such that $\emptyset \neq M \subseteq S^{d-2-r}$ and each state of T is reachable from one of the $s_{d-r-1}, \dots, s_{d-2}$.

We do not make this free variable explicit, since it is always assumed to be there. We will see in Sect. 5 how this intended interpretation can be enforced. This stipulation formalises the informal idea given above. Note that in [17], it was assumed that all states of the LTS in question are reachable from a singular state, but this is not a necessary requirement for the argument to work.

Finally, let w and d be such that $2w + r + 2 \leq d$. The intuition here is that, in order to translate from HO(LFP), we will have to deal with sets and higher-order sets of arity at most w .

Quantification for Sets. We now define a similar pattern to that for the first-order case which allows us to iterate over all sets of w -tuples in an LTS. This is not the same as enumerating $\llbracket \bullet \rrbracket^T$ since $w < d$.

Let σ_i be defined via $\sigma_i(d-1) = i$, $\sigma_i(d) = i + w$ and $\sigma_i(j) = j$ if $j < d - 1$. Then $\bar{s} \in \llbracket \{\sigma_i\} \varphi \rrbracket_\eta^T$ iff $\bar{s}[s_i/d-1, s_{i+w}/d] \in \llbracket \varphi \rrbracket_\eta^T$. The intended use for this substitution is to compare the elements at indices i and $i + w$ w.r.t. to the order induced by $\varphi_{<}$. Remember that this formula always compares the last two elements of the tuple. Moreover, let $\sigma_{\rightarrow w}$ be defined by $\sigma_{\rightarrow w}(i) = w + i$ for $i \leq w$ and $\sigma_{\rightarrow w}(j) = j$ for $j > w$. Then $(s_1, \dots, s_d) \in \llbracket \{\sigma_{\rightarrow w}\} \varphi \rrbracket_\eta^T$ iff $(s_1, \dots, s_w, s_1, \dots, s_w, s_{2w+1}, \dots, s_d) \in \llbracket \varphi \rrbracket_\eta^T$. The intended use here is to shift the first w elements of a formula to the right to make room for another tuple at the first w positions such that the two tuples can be compared lexicographically.

Consider the formula $\exists^{(w)}x. \varphi := (\mu(F : \bullet \rightarrow \bullet). \lambda(x : \bullet). \varphi \vee F(\text{next}^{(w)}(x))) \text{ff}$ where

$$\begin{aligned} \varphi_{<}^w &:= \bigvee_{i=1}^w \{\sigma_i\} \varphi_{<} \wedge \bigwedge_{j=1}^{i-1} \{\sigma_j\} \neg \varphi_{<} \\ \varphi_{<}^{(w)}(x, y) &:= \exists_1 \dots \exists_w. y \wedge \neg x \wedge \{\sigma_{\rightarrow w}\} (\forall_1 \dots \forall_w. \varphi_{<}^w \rightarrow x \rightarrow y). \\ \text{next}^{(w)}(x) &:= \lambda(x : \bullet). e \wedge \neg x \wedge \{\sigma_{\rightarrow w}\} (\forall_1 \dots \forall_w. \varphi_{<}^w \rightarrow x) \\ &\quad \vee e \wedge x \wedge \{\sigma_{\rightarrow w}\} (\exists_1 \dots \exists_w. \varphi_{<}^w \wedge \neg x) \end{aligned}$$

The first formula $\varphi_{<}^w$ implements lexicographical comparison of the first w elements in a tuple to the second w elements. The second formula $\varphi_{<}^{(w)}$ lifts the order from tuples to sets of tuples via the lexicographical order induced by the order on the tuples. Finally, the formula $\text{next}^{(w)}$ is a predicate transformer that consumes a set of tuples. It returns a set of tuples that is the lexicographical successor of the input in the order induced by the order on the first w elements of the individual tuples: the output contains a tuple iff either the input does contain it, but not all lexicographically smaller tuples, or if the input does not contain it, but all lexicographically smaller tuples. Also note the role of e that filters out all tuples that do not adhere to our stipulation that the whole LTS be reachable from one of the states at indices $d - r - 1, \dots, d - 2$.

Lemma 11. Let T be an LTS with state set S and let η be good. Then $T, \bar{s} \models_\eta \exists^{(w)}x. \varphi$ iff there is $M \subseteq S^w$ such that $T, \bar{s} \models_{\eta[x \rightarrow M \times S^{d-w} \cap \eta(e)]} \varphi$.

The proof consists of verifying the informal intuition above. We write $\forall^{(w)}x. \varphi$ to denote $\neg \exists^{(w)}x. \neg \varphi$. We write $\exists^{(w)}x_1, \dots, x_n. \varphi$ for $\exists^{(w)}x_1 \dots \exists^{(w)}x_n. \varphi$, and similarly for $\forall^{(w)}x. \varphi$.

Generalised Higher-Order Quantification. We have just seen how existential quantification can be emulated for individual states in an LTS and, with some restrictions, for sets of w -tuples of an LTS. For other types that commonly appear in HO(LFP), i.e. relations of higher order, there is no immediate PHFL equivalent, since all types beyond \bullet are function types. However, we can use these function types to emulate the HO(LFP) types to a sufficient degree. For the sake of simplicity, we only consider types of a special form; we argue in Sect. 5 why this is not a restriction.

Let $\tau_{w,k}$ be inductively defined via $\tau_{w,0} = \bullet$ and $\tau_{w,i+1} = \tau_{w,i} \rightarrow \dots \rightarrow \tau_{w,i} \rightarrow \bullet$ where $\tau_{w,i}$ is repeated w many times. Given T , let $\llbracket \tau_{w,i}^\circ \rrbracket^T = \llbracket \tau_{w,i} \rrbracket^T$ for $i \leq 1$ and let

$$\llbracket \tau_{w,k}^\circ \rrbracket^T = \{f \in \llbracket \tau_{w,k} \rrbracket^T \mid f(f_1, \dots, f_w) = S \text{ or } f(f_1, \dots, f_w) = \emptyset \text{ f.a. } f_1, \dots, f_w \in \llbracket \tau_{w,k-1}^\circ \rrbracket^T\}$$

for $k \geq 2$. The important distinction here is that $\llbracket \tau_{w,k}^\circ \rrbracket^T$ is the restriction of $\llbracket \tau_{w,k} \rrbracket^T$ to those functions that always return either the full set of states or the empty set, at least on inputs from $\llbracket \tau_{w,k-1}^\circ \rrbracket^T$. This is desirable since we want to use functions in $\llbracket \tau_{w,k}^\circ \rrbracket^T$ to emulate higher-order variables of a special form. Given x, x_1, \dots, x_w of the appropriate type, the question whether $\bar{s} \in \llbracket x(x_1, \dots, x_w) \rrbracket_\eta^T$ does not depend on \bar{s} (as in modal logics), but is uniform over the while LTS. However, since $\llbracket \tau_{w,k} \rrbracket^T$ also contains functions that are not uniform starting from PHFL-order 2, we restrict ourselves to functions that are uniform on the necessary inputs (i.e. those that are themselves sufficiently uniform).

Consider the following formulas for $k \geq 1$, where $\bar{x} = x_1, \dots, x_w$ and $\bar{y} = y_1, \dots, y_w$:

$$\begin{aligned} \varphi_{<}^{w,k-1}(x_1, \dots, x_w, y_1, \dots, y_w) &:= \bigvee_{i=1}^w \varphi_{<}^{(w),k-1}(x_i, y_i) \wedge \bigwedge_{j=1}^{i-1} \neg \varphi_{<}^{(w),k-1}(x_j, y_j) \\ \varphi_{<}^{(w),k}(x, y) &:= \exists^{w,k-1} \bar{x}. y(\bar{x}) \wedge \neg x(\bar{x}) \\ &\quad \wedge \forall^{w,k-1} \bar{y}. \varphi_{<}^{w,l}(\bar{y}, \bar{x}) \rightarrow x(\bar{y}) \rightarrow y(\bar{y}) \\ \mathbf{ff}_{(w),k} &:= \lambda(\bar{x}: \tau_{w,k-1}). \mathbf{ff} \\ \mathbf{next}^{w,k}(x) &:= \lambda(x: \tau_{w,k}). \lambda(\bar{x}: \tau_{w,k-1}). \\ &\quad (\neg x(\bar{x}) \wedge \forall^{w,k} \bar{y}. \varphi_{<}^{w,k}(\bar{y}, \bar{x}) \rightarrow x(\bar{y})) \\ &\quad \vee (x(\bar{x}) \wedge \exists^{w,k} \bar{y}. \varphi_{<}^{w,k}(\bar{y}, \bar{x}) \wedge \neg x(\bar{y})) \\ \exists^{w,k}(x). \varphi &:= (\mu(F: \tau_{w,k} \rightarrow \tau_{w,k}). \lambda(x: \tau_{w,k}). \varphi \vee F(\mathbf{next}^{w,k} x)) \mathbf{ff}_{(w),k} \end{aligned}$$

where $\varphi_{<}^{(w),k-1} = \varphi_{<}^{(w)}$ in case $k = 1$ and $\exists^{w,k-1} x. \varphi = \exists^{(w)} x. \varphi$ if $k = 1$.

Similarly as in the definitions given before Lemma 11, these formulas lift quantification up by one level on the type hierarchy. The formula $\varphi_{<}^{w,k-1}$ compares width- w -tuples of functions of type $\tau_{w,k-1}^\circ$ lexicographically using the previously defined formula $\varphi_{<}^{(w),k-1}$ that compares individual such functions. The formula $\varphi_{<}^{(w),k}$ then lifts this to individual functions of the next type, using existential and universal quantification. The formula $\mathbf{next}^{w,k}$ again consumes a function of type $\tau_{w,k}^\circ$ and returns the lexicographically next one using the standard definition of binary incrementation, while $\exists^{w,k}$ implements existential quantification for $\tau_{w,k}^\circ$ by iterating through all possible candidates using $\mathbf{next}^{w,k}$.

Lemma 12. *Let T be an LTS and let η be good. Then $T, \bar{s} \models_\eta \exists^{w,k} x. \varphi$ iff there is $f \in \llbracket \tau_{w,k}^\circ \rrbracket^T$ such that $T, \bar{s} \models_{\eta[x \mapsto f]} \varphi$.*

The proof follows the same pattern as that of Lemma 11 by verifying that the individual formulas do what is claimed above.

Lemmas 11 and 12 justify the use of HO(LFP)-style quantification symbols for individual states, sets of type $M \times S^{d-w} \cap \eta(e)$ for $M \subseteq S^s$ and for $\tau_{w,k}^\circ$ for all $k \geq 2$. Note that the latter do neither coincide with HO(LFP) types nor with the respective $\tau_{w,k}$.

5 Lower Bounds

Homogeneous Types. In order to simplify the translation from HO(LFP) to PHFL, we restrict the set of types that can be used in HO(LFP) formulas. Let $w \geq 2$ be fixed but arbitrary. Define $\tau'_{w,k}$ as $\tau'_{w,1} = \odot$, $\tau'_{w,i+1} = (\tau'_{w,i}, \dots, \tau'_{w,i})$ with w many repetitions of $\tau'_{w,i}$.

Lemma 13. *If $\varphi \in \text{HO}^k(\text{LFP})$ defines a query Q , then there is $\varphi' \in \text{HO}^k(\text{LFP})$ that defines the same query, but the only types used in φ' are $\tau'_{w,0}, \dots, \tau'_{w,k}$ for some w .*

Proof. There are two principles that are used here: If φ contains a type of the form $\tau'' = (\tau', \dots, \tau')$ with $w' \leq w$ many repetitions of w , we can replace τ'' by (τ', \dots, τ') with exactly w many repetitions of τ' by changing the respective type everywhere in the formula and requiring at quantifiers that, e.g., the last $w - w'$ components are equal to the $w - w' - 1$ st in every tuple contained in a set. Hence, every type can be assumed to have width exactly w .

It remains to deal with inhomogeneous types, e.g. those of the form (τ', τ'', \dots) such that $\text{ord}(\tau') \neq \text{ord}(\tau'')$. This can be remedied by increasing the type of the lower order by one order, and requiring at quantification steps that the only tuple in the set be a singleton of the form (M, \dots, M) of width w . This procedure needs to be chained if the orders of constituent types diverge by more than one. \square

Type Correspondence. Now that we can assume w.l.o.g. that all $\text{HO}^k(\text{LFP})$ -definable queries are defined by an $\text{HO}^k(\text{LFP})$ formula which uses only the types $\tau'_{w,i}$ for $i \leq k$ we can observe that these types are quite similar to the types $\tau_{w,k}$ defined in the previous section. In fact, we want to emulate the type $\tau'_{w,k}$ with $k > 1$ by $\tau_{w,k-2}^\circ$. The type $\tau'_{s,1}$ will be handled by polyadicity as in [17].²

Let T be an LTS with state set S and let $d \geq w \geq 2$. For each $k \geq 2$, we define a translation $\text{tptr}_k^T : \llbracket \tau'_{w,k} \rrbracket^T \rightarrow \llbracket \tau_{w,k-2}^\circ \rrbracket^T$ via $\text{tptr}_2^T(M) = M \times S^{d-w}$ and $\text{tptr}_{i+1}^T(M) = f \in \llbracket \tau_{w,k-2} \rrbracket^T$ s.t.

$$f(f_1, \dots, f_n) = \begin{cases} S, & \text{if } f_j = \text{tptr}_i^T(x_i) \text{ f.a. } 0 \leq j \leq w \text{ and } (x_1, \dots, x_w) \in M \\ \emptyset, & \text{otherwise.} \end{cases}$$

Hence, a variable assignment α such that all variables are of types $\tau'_{w,2}, \dots, \tau'_{w,k}$ induces an environment η_α with definitions for all those variables of order ≥ 2 via $\eta_\alpha(X) = \text{tptr}_i^T(X)$ where i is the order of X .

The Translation. We are now ready to extend the translation given in [17] to $\text{HO}^k(\text{LFP})$ with $k \geq 2$. Towards this, we present a syntactical translation trans from $\text{HO}^k(\text{LFP})$ with $k \geq 2$ into PHFL_{k-1}^d for some $d \geq 2$.

Lemma 14. *Let $\varphi \in \text{HO}^k(\text{LFP})$ be bisimulation invariant and have free first-order variables X_1, \dots, X_r and, hence define a query of width r . Moreover, let $\tau'_{w,0}, \dots, \tau'_{w,k}$ be the only types in φ . W.l.o.g. let $2w \geq r$. Let $d = 2w + r + 2$. Then there is $\varphi' \in \text{PHFL}_{k-1}^d$ such that, for all LTS T that are a bisimulation quotient, we have $T, \alpha \models \varphi$ iff $T, (\alpha(X_1), \dots, \alpha(X_r)) \models_{\eta_\alpha[e \mapsto M]} \varphi'$ where $\emptyset \neq M' \subseteq S^r$ and $M = M' \times S^{d-2r-2} \times \{\alpha(X_1)\} \times \dots \times \{\alpha(X_r)\} \times S^2$.*

²It would also be possible to completely eliminate this type from $\text{HO}^k(\text{LFP})$ for $k \geq 2$; cf. similar constructions in the context of MSO and automata theory.

Proof. Let *trans* be given via

$$\begin{aligned}
\text{trans}(p(X_i)) &:= p_i \\
\text{trans}(a(X_i, X_j)) &:= \langle a_i \rangle \{ \sigma_{i,j} \} \varphi_{\sim} \\
\text{trans}(\varphi_1 \vee \varphi_2) &:= \text{trans}(\varphi_1) \vee \text{trans}(\varphi_2) \\
\text{trans}(\neg \varphi) &:= \neg \text{trans}(\varphi) \\
\text{trans}(\exists(X_i : \odot). \varphi) &:= \exists_i. \text{trans}(\varphi) \\
\text{trans}(\exists(X : \tau'_{w,1}). \varphi) &:= \exists^{(w)}(x). \text{trans}(\varphi) \\
\text{trans}(\exists(X : \tau'_{w,k}). \varphi) &:= \exists^{w,k}(x). \text{trans}(\varphi) \text{ if } k \geq 2 \\
\text{trans}(X(X_{i_1}, \dots, X_{i_w})) &:= \{ \sigma \} x \text{ if } (X : \tau'_{w,1}) \\
\text{trans}(X(X_1, \dots, X_w)) &:= x(x_1, \dots, x_w) \text{ if } (X : \tau'_{w,k}) \text{ and } k \geq 2 \\
\text{trans}(\text{lfp}(X, Y_1, \dots, Y_w). \varphi)(Z_{i_1}, \dots, Z_{i_w}) &:= \{ \sigma \} \mu(F : \bullet \rightarrow \bullet). \text{trans}(\varphi) \text{ if } (X : \tau_{w,2}) \\
\text{trans}(\text{lfp}(X, Y_1, \dots, Y_w). \varphi)(Z_1, \dots, Z_w) &:= (\mu(F : \tau_{w,k-2} \rightarrow \tau_{w,k-2}). \lambda(y_1, \dots, y_w : \tau_{w,k-3}). \\
&\quad \text{trans}(\varphi))(Z_1, \dots, Z_w) \text{ if } (X : \tau_{w,k}) \text{ and } k \geq 3
\end{aligned}$$

where $\sigma_{i,j}$ is defined via $\sigma_{i,j}(d-1) = i$, $\sigma_{i,j}(d) = j$ and $\sigma_{i,j}(k) = k$ for $k < d-1$, and φ_{\sim} is the formula from Ex. 2, resp. [17] that holds iff the states in positions $d-1$ and d are bisimilar, and, finally σ is defined via $\sigma(j) = i_j$ for $0 \leq j \leq w$ and $\sigma(k) = k$ for $k \geq w$. The substitutions $\sigma_{i,j}$, resp. σ serve to swap the elements i and j to positions $d-1$ and d , resp. to reorder the elements according to the variable order used on the left side of the translation.

The claim now follows by induction over the syntax tree of φ . The first five cases are as in [17]. The next four cases are by Lemmas 11 and 12, resp. the definition of η_α . The fixpoint cases are by an induction over the individual stages of the fixpoint iteration; note that the Knaster-Tarski-based semantics of the fixpoints in HO(LFP) and PHFL can be equivalently replaced by semantics based on the Kleene Fixpoint Theorem [10]. This mirrors the proof of the fixpoint case in [17]. \square

Theorem 15. *PHFL^k captures k-EXPTIME/_~ for all $k \geq 0$.*

Proof. The cases of $k = 0$ and $k = 1$ are of course already known [17, 15]. For $k \geq 2$, the upper bound is shown in Thm. 8. For the lower bound, it suffices, due to Prop. 4, to show that for any bisimulation-invariant query defined by an HO^{k+1}(LFP) formula there is an equivalent formula in PHFL^k. Let φ be a formula defining such a query Q of width $r \geq 1$, and w.l.o.g. φ contains only the types $\tau_{w,0}, \dots, \tau_{w,k}$.

Let φ' be the formula obtained from the translation in Lemma 14 and let σ be defined via $\sigma(d-r-2+i) = i$ for $1 \leq i \leq r$, and $\sigma(j) = j$ for $j \leq d-r-2$ or $j \geq d-1$. Note that σ simply copies the first r many states in a tuple to the positions $d-r-1, \dots, d-2$ where they serve to retain goodness (cf. Def. 10). Then $\psi = (\lambda(e : \bullet)\varphi)\{\sigma\}\text{tt}$ defines the query $\{(T, (s_1, \dots, s_d)) \mid (T, (s_1, \dots, s_r)) \in Q\}$. Towards this, note that $\llbracket \{\sigma\}\text{tt} \rrbracket^T = \{(s_1, \dots, s_d) \mid s_i = s_{d-r-2+i} \text{ for } 1 \leq i \leq r\} = M$. Hence, $\llbracket \psi \rrbracket^T = \llbracket \varphi' \rrbracket_\eta^T$ where $\eta = \emptyset[e \mapsto M]$. Since η is good, Lemma 14 is applicable.

What remains is to argue that a query that, on each LTS T , returns all tuples in $M \times S^{d-r}$ where $M \subseteq S^r$, is in fact a query that defines tuples of width r . This can either be done formally by expanding the semantics of the substitution operator $\{\sigma\}$ to return tuples in the width of its co-domain, or by simply considering queries of the above form to be of width r , cf. [17]. \square

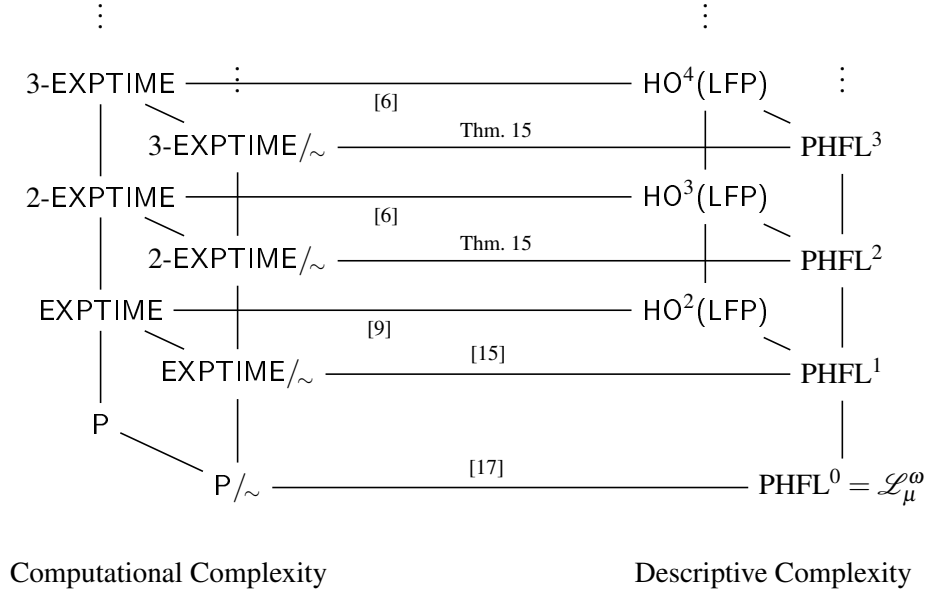


Figure 2: Capturing results for time complexity classes - an overview.

6 Conclusion

We have extended the descriptive complexity of bisimulation-invariant queries, as started by Otto [17] with the capturing of the class P/\sim with the logic \mathcal{L}_μ^ω , to further time complexity classes, namely the classes $k\text{-EXPTIME}/\sim$ of bisimulation-invariant queries that can be answered in k -fold exponential time. These turn out to be exactly those that can be defined in PHFL, the polyadic extension of the higher-order modal fixpoint logic HFL. This is genuinely an extension as \mathcal{L}_μ^ω coincides with PHFL^0 , the fragment with no higher-order constructs. Moreover, the level in the exponential-time hierarchy corresponds to the type order used in formulas: $k\text{-EXPTIME}/\sim = \text{PHFL}^k$.

The resulting picture of descriptive time complexity is shown in Fig. 2. A natural way to extend this is of course to integrate results about the space complexity classes $k\text{-EXPSPACE}/\sim$ sitting in between $k\text{-EXPTIME}/\sim$ and $(k+1)\text{-EXPTIME}/\sim$. Here we can only state that it is possible to capture these in terms of natural fragments of PHFL as well but the proof requires a few more ingredients than the time-complexity case. Here we could use the already known characterisation of $k\text{-EXPTIME}$ by Higher-Order Predicate Logics with Least Fixpoints [6]. For the space complexity classes, we need to first develop such characterisations that extend the Abiteboul-Vianu Theorem. This can be done [13], but due to space restrictions here its detailed presentation is left for a future publication.

A noteworthy observation for the capturing of $k\text{-EXPTIME}/\sim$ with $k \geq 1$ is that, unlike in the case of $k = 0$, the requirement of structures being ordered is not necessary for the invocation of the (generalised) Immerman-Vardi Theorem. However, starting from $k \geq 1$ the order being provided in the bisimulation-invariant setting plays a crucial role to emulate existential quantification from the $\text{HO}(\text{LFP})$ side: the LTS in question coming with a PHFL_2^0 -definable order allows us to enumerate sufficiently many sets, functions, etc. to emulate existential quantification. It remains to see in further detail whether this is an artifact of the proof strategy or a genuine and inherent part of the correspondence between complexity classes and logics in the bisimulation-invariant framework.

References

- [1] S. Abiteboul & V. Vianu (1987): *A Transaction Language Complete for Database Update and Specification*. In: *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, San Diego, CA, pp. 260–268, doi:10.1145/28659.28688.
- [2] H. R. Andersen (1994): *A Polyadic Modal μ -Calculus*. Technical Report ID-TR: 1994-195, Dept. of Computer Science, Technical University of Denmark, Copenhagen, doi:10.1.1.42.1859.
- [3] R. Axelsson, M. Lange & R. Somla (2007): *The Complexity of Model Checking Higher-Order Fixpoint Logic*. *Logical Methods in Computer Science* 3, pp. 1–33, doi:10.2168/LMCS-3(2:7)2007.
- [4] F. Bruse, M. Lange & É. Lozes (2017): *Space-Efficient Fragments of Higher-Order Fixpoint Logic*. In: *Proc. 11th Int. Workshop on Reachability Problems, RP'17, LNCS 10506*, pp. 26–41, doi:10.1007/978-3-319-67089-8_3.
- [5] R. Fagin (1974): *Generalized First-Order Spectra and Polynomial-Time Recognizable Sets*. *Complexity and Computation* 7, pp. 43–73.
- [6] C. M. Freire & A. T. Martins (2011): *The Descriptive Complexity of the Deterministic Exponential Time Hierarchy*. In: *Proc. 5th Workshop on Logical and Semantic Frameworks with Applications, LSFA'10*, 269, pp. 71–82, doi:10.1016/j.entcs.2011.03.006.
- [7] M. Grohe (2008): *The Quest for a Logic Capturing PTIME*. In: *Proc. 23rd Annual IEEE Symp. on Logic in Computer Science, LICS'08*, IEEE Computer Society, pp. 267–271, doi:10.1109/LICS.2008.11.
- [8] N. Immerman (1986): *Relational Queries Computable in Polynomial Time*. *Information and Control* 68(1-3), pp. 86–104, doi:10.1016/S0019-9958(86)80029-8.
- [9] N. Immerman (1987): *Languages That Capture Complexity Classes*. *SIAM Journal of Computing* 16(4), pp. 760–778, doi:10.1137/0216051.
- [10] S. C. Kleene (1938): *On Notation for Ordinal Numbers*. *Journal of Symbolic Logic* 3(4), pp. 150–155, doi:10.2307/2267778.
- [11] B. Knaster (1928): *Un théorème sur les fonctions d'ensembles*. *Annals Soc. Pol. Math* 6, pp. 133–134.
- [12] D. Kozen (1983): *Results on the Propositional μ -calculus*. *Theor. Comp. Sci.* 27, pp. 333–354, doi:10.1016/0304-3975(82)90125-6.
- [13] D. Kronenberger (2018): *Capturing Bisimulation-Invariant Complexity Classes by Polyadic Higher-Order Fixpoint Logic*. Master's thesis, University of Kassel.
- [14] M. Lange & É. Lozes (2012): *Model Checking the Higher-Dimensional Modal μ -Calculus*. In: *Proc. 8th Workshop on Fixpoints in Computer Science, FICS'12, Electr. Proc. in Theor. Comp. Sc.* 77, pp. 39–46, doi:10.4204/EPTCS.77.
- [15] M. Lange & É. Lozes (2014): *Capturing Bisimulation-Invariant Complexity Classes with Higher-Order Modal Fixpoint Logic*. In: *Proc. 8th Int. IFIP Conf. on Theoretical Computer Science, TCS'14, LNCS 8705*, Springer, pp. 90–103, doi:10.1007/978-3-662-44602-7.
- [16] M. Lange, É. Lozes & M. Vargas Guzmán (2014): *Model-Checking Process Equivalences*. *Theor. Comp. Sci.* 560, pp. 326–347, doi:10.1016/j.tcs.2014.08.020.
- [17] M. Otto (1999): *Bisimulation-invariant PTIME and higher-dimensional μ -calculus*. *Theor. Comput. Sci.* 224(1-2), pp. 237–265, doi:10.1016/S0304-3975(98)00314-4.
- [18] A. Tarski (1955): *A Lattice-theoretical Fixpoint Theorem and its Application*. *Pacific Journal of Mathematics* 5, pp. 285–309, doi:10.2140/pjm.1955.5.285.
- [19] J. van Benthem & K. Doets (1983): *Higher-Order Logic*. In D. Gabbay & F. Guenther, editors: *Handbook of Philosophical Logic, Volume I: Elements of Classical Logic*, D. Reidel Publishing Co., pp. 275–329, doi:10.1007/978-94-009-7066-3_4.

- [20] M. Y. Vardi (1982): *The Complexity of Relational Query Languages (Extended Abstract)*. In: *Proc. 14th Symp. on Theory of Computing, STOC'82*, ACM, San Francisco, CA, USA, pp. 137–146, doi:10.1145/800070.
- [21] M. Viswanathan & R. Viswanathan (2004): *A Higher Order Modal Fixed Point Logic*. In: *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04, LNCS 3170*, Springer, pp. 512–528, doi:10.1007/978-3-540-28644-8_33.

Complexity through Translations for Modal Logic with Recursion*

Luca Aceto

Department of Computer Science
Reykjavik University
Reykjavik, Iceland
Gran Sasso Science Institute
L'Aquila, Italy
luca@ru.is

Antonis Achilleos

Department of Computer Science
Reykjavik University
Reykjavik, Iceland
antonios@ru.is

Elli Anastasiadi

Department of Computer Science
Reykjavik University
Reykjavik, Iceland
elli19@ru.is

Adrian Francalanza

Department of Computer Science, ICT
University of Malta
Msida, Malta
adrian.francalanza@um.edu.mt

Anna Ingólfssdóttir

Department of Computer Science
Reykjavik University
Reykjavik, Iceland
annai@ru.is

This paper studies the complexity of classical modal logics and of their extension with fixed-point operators, using translations to transfer results across logics. In particular, we show several complexity results for multi-agent logics via translations to and from the μ -calculus and modal logic, which allow us to transfer known upper and lower bounds. We also use these translations to introduce a terminating tableau system for the logics we study, based on Kozen’s tableau for the μ -calculus, and the one of Fitting and Massacci for modal logic.

1 Introduction

We introduce a family of multi-modal logics with fixed-point operators that are interpreted on restricted classes of Kripke models. One can consider these logics as extensions of the usual multi-agent logics of knowledge and belief [14] by adding recursion to their syntax or of the μ -calculus [21] by interpreting formulas on different classes of frames and thus giving an epistemic interpretation to the modalities. We define *translations* between these logics, and we demonstrate how one can rely on these translations to prove finite-model theorems, complexity bounds, and tableau termination for each logic in the family.

Modal logic comes in several variations [5]. Some of these, such as multi-modal logics of knowledge and belief [14], are of particular interest to Epistemology and other application areas. Semantically, the classical modal logics used in epistemic (but also other) contexts result from imposing certain restrictions on their models. On the other hand, the modal μ -calculus [21] can be seen as an extension of the smallest normal modal logic **K** with greatest and least fixed-point operators, νX and μX respectively. We explore the situation where one allows both recursion (*i.e.* fixed-point) operators in a multi-modal language and imposes restrictions on the semantic models.

We are interested in the complexity of satisfiability for the resulting logics. Satisfiability for the μ -calculus is known to be EXP-complete [21], while for the modal logics between **K** and **S5** the problem

*This work has been funded by the projects “Open Problems in the Equational Logic of Processes (OPEL)” (grant no. 196050), “Epistemic Logic for Distributed Runtime Monitoring” (grant no. 184940), “Mode(1)s of Verification and Monitorability” (MoVeMent) (grant no 217987) of the Icelandic Research Fund, and the project “Runtime and Equational Verification of Concurrent Programs” (ReVoCoP) (grant 222021) of the Reykjavik University Research Fund.

is PSPACE-complete or NP-complete, depending on whether they have Negative Introspection [18, 22]. In the multi-modal case, satisfiability for those modal logics becomes PSPACE-complete, and is EXP-complete with the addition of a common knowledge operator [17].

There is plenty of relevant work on the μ -calculus on restricted frames, mainly in its single-agent form. Alberucci and Facchini examine the alternation hierarchy of the μ -calculus over reflexive, symmetric, and transitive frames in [2]. D’Agostino and Lenzi have studied the μ -calculus on different classes of frames in great detail. In [11], they reduce the μ -calculus over finite transitive frames to first-order logic. In [9], they prove that $\mathbf{S5}^\mu$ -satisfiability is NP-complete, and that the two-agent version of $\mathbf{S5}^\mu$ does not have the finite model property. In [12], they consider finite symmetric frames, and they prove that \mathbf{B}^μ -satisfiability is in 2EXP, and EXP-hard. They also examine planar frames in [10], where they show that the alternation hierarchy of the μ -calculus over planar frames is infinite.

Our primary method of proving complexity results is through translations to and from the multi-modal μ -calculus. We show that we can use surprisingly simple translations from modal logics without recursion to the base modal logic \mathbf{K}_n , reproving the PSPACE upper bound for these logics (Theorem 8 and Corollary 9). These translations and our constructions to prove their correctness do not generally transfer to the corresponding logics with recursion. We present translations from specific logics to the μ -calculus and back, and we discuss the remaining open cases. We discover, through the properties of our translations, that several behaviors induced on the transitions do not affect the complexity of the satisfiability problem. As a result, we prove that all logics with axioms among D , T , and 4, and the least-fixed-point fragments of logics that also have B , have their satisfiability in EXP, and a matching lower bound for the logics with axioms from D, T, B (Corollaries 15 and 16). Finally, we present tableaux for the discussed logics, based on the ones by Kozen for the μ -calculus [21], and by Fitting and Massacci for modal logic [16, 24]. We give tableau-termination conditions for every logic with a finite model property (Theorem 19).

The addition of recursive operators to modal logic increases expressiveness. An important example is that of *common knowledge* or *common belief*, which can be expressed with a greatest fixed-point thus: $\nu X.(\varphi \wedge \bigwedge_\alpha [\alpha]X)$. But the combination of epistemic logics and fixed-points can potentially express more interesting epistemic concepts. For instance, the formula $\mu X. \bigvee_\alpha ([\alpha]\varphi \vee [\alpha]X)$, in the context of a belief interpretation, can be thought to claim that there is a rumour of φ . It would be interesting to see what other meaningful sentences of epistemic interest one can express using recursion. Furthermore, the family of logics we consider allows each agent to behave according to a different logic. This flexibility allows one to mix different interpretations of modalities, such as a temporal interpretation for one agent and an epistemic interpretation for another. Such logics can even resemble hyper-logics [8] if a set of agents represents different streams, and combinations of epistemic and temporal or hyper-logics have recently been used to express safety and privacy properties of systems [7].

The paper is organized as follows. Section 2 gives the necessary background and an overview of current results. Section 3 defines a class of translations that provide us with several upper and lower bounds, and identifies conditions under which they can be composed. In Section 4 we finally give tableaux for our multi-modal logics with recursion. We conclude in Section 5 with a set of open questions and directions. Omitted proofs can be found in the full version of the paper [23].

2 Definitions and Background

This section introduces the logics that we study and the necessary background on the complexity of modal logic and the μ -calculus.

2.1 The Multi-Modal Logics with Recursion

We start by defining the syntax of the logics.

Definition 1. We consider formulas constructed from the following grammar:

$$\begin{array}{l} \varphi, \psi \in L ::= p \quad | \neg p \quad | \text{tt} \quad | \text{ff} \quad | X \quad | \varphi \wedge \psi \quad | \varphi \vee \psi \\ \quad | \langle \alpha \rangle \varphi \quad | [\alpha] \varphi \quad | \mu X. \varphi \quad | \nu X. \varphi, \end{array}$$

where X comes from a countable set of logical (or fixed-point) variables, LVAR , α from a finite set of agents, AG , and p from a finite set of propositional variables, PVAR . When $\text{AG} = \{\alpha\}$, $\square \varphi$ stands for $[\alpha] \varphi$, and $\diamond \varphi$ for $\langle \alpha \rangle \varphi$. We also write $[A] \varphi$ to mean $\bigwedge_{\alpha \in A} [\alpha] \varphi$ and $\langle A \rangle \varphi$ for $\bigvee_{\alpha \in A} \langle \alpha \rangle \varphi$.

A formula is closed when every occurrence of a variable X is in the scope of recursive operator νX or μX . Henceforth we consider only closed formulas, unless we specify otherwise.

Moreover, for recursion-free closed formulas we associate the notion of *modal depth*, which is the nesting depth of the modal operators¹. The modal depth of φ is defined inductively as:

- $md(p) = md(\neg p) = md(\text{tt}) = md(\text{ff}) = 0$, where $p \in \text{PVAR}$,
- $md(\varphi \vee \psi) = md(\varphi \wedge \psi) = \max(md(\varphi), md(\psi))$, and
- $md([\alpha] \varphi) = md(\langle \alpha \rangle \varphi) = 1 + md(\varphi)$, where $\alpha \in \text{AG}$.

We assume that in formulas, each recursion variable X appears in a unique fixed-point formula $\text{fx}(X)$, which is either of the form $\mu X. \varphi$ or $\nu X. \varphi$. If $\text{fx}(X)$ is a least-fixed-point (*resp.* greatest-fixed-point) formula, then X is called a least-fixed-point (*resp.* greatest-fixed-point) variable. We can define a partial order on fixed-point variables, such that $X \leq Y$ iff $\text{fx}(X)$ is a subformula of $\text{fx}(Y)$, and $X < Y$ when $X \leq Y$ and $X \neq Y$. If X is \leq -minimal among the free variables of φ , then we define the *closure* of φ to be $cl(\varphi) = cl(\varphi[\text{fx}(X)/X])$, where $\varphi[\psi/X]$ is the usual substitution operation, and if φ is closed, then $cl(\varphi) = \varphi$.

We define $\text{sub}(\varphi)$ as the set of subformulas of φ , and $|\varphi| = |\text{sub}(\varphi)|$ is bounded by the length of φ as a string of symbols. Negation, $\neg \varphi$, and implication, $\varphi \rightarrow \psi$, can be defined in the usual way. Then, we define $\overline{\text{sub}(\varphi)} = \text{sub}(\varphi) \cup \{\neg \psi \in L \mid \psi \in \text{sub}(\varphi)\}$.

Semantics We interpret formulas on the states of a *Kripke model*. A Kripke model, or simply model, is a quadruple $M = (W, R, V)$ where W is a nonempty set of states, $R \subseteq W \times \text{AG} \times W$ is a transition relation, and $V : W \rightarrow 2^{\text{PVAR}}$ determines on which states a propositional variable is true. (W, R) is called a *frame*. We usually write $(u, v) \in R_\alpha$ or $uR_\alpha v$ instead of $(u, \alpha, v) \in R$, or uRv , when AG is a singleton $\{\alpha\}$.

Formulas are evaluated in the context of an *environment* $\rho : \text{LVAR} \rightarrow 2^W$, which gives values to the logical variables. For an environment ρ , variable X , and set $S \subseteq W$, we write $\rho[X \mapsto S]$ for the environment that maps X to S and all $Y \neq X$ to $\rho(Y)$. The semantics for our formulas is given through a function $\llbracket - \rrbracket_{\mathcal{M}}$, defined in Table 1. The semantics of $\neg \varphi$ are constructed as usual, where $\llbracket \neg X, \rho \rrbracket_{\mathcal{M}} = W \setminus \rho(X)$.

We sometimes use $\mathcal{M}, s \models_{\rho} \varphi$ for $s \in \llbracket \varphi, \rho \rrbracket_{\mathcal{M}}$, and as the environment has no effect on the semantics of a closed formula φ , we often drop it from the notation and write $\mathcal{M}, s \models \varphi$ or $s \in \llbracket \varphi \rrbracket_{\mathcal{M}}$. If $\mathcal{M}, s \models \varphi$, we say that φ is true, or satisfied, in s . When the particular model does not matter, or is clear from the context, we may omit it.

¹The modal depth of recursive formulas can be either zero, or infinite. However, this is not relevant for the spectrum of this work.

$$\begin{aligned}
\llbracket \text{tt}, \rho \rrbracket &= W, & \llbracket \text{ff}, \rho \rrbracket &= \emptyset, & \llbracket p, \rho \rrbracket &= \{s \mid p \in V(s)\}, & \llbracket \neg p, \rho \rrbracket &= W \setminus \llbracket p, \rho \rrbracket, \\
\llbracket [\alpha]\varphi, \rho \rrbracket &= \{s \mid \forall t. sR_\alpha t \text{ implies } t \in \llbracket \varphi, \rho \rrbracket\}, & \llbracket \varphi_1 \wedge \varphi_2, \rho \rrbracket &= \llbracket \varphi_1, \rho \rrbracket \cap \llbracket \varphi_2, \rho \rrbracket, \\
\llbracket \langle \alpha \rangle \varphi, \rho \rrbracket &= \{s \mid \exists t. sR_\alpha t \text{ and } t \in \llbracket \varphi, \rho \rrbracket\}, & \llbracket \varphi_1 \vee \varphi_2, \rho \rrbracket &= \llbracket \varphi_1, \rho \rrbracket \cup \llbracket \varphi_2, \rho \rrbracket, \\
\llbracket \mu X. \varphi, \rho \rrbracket &= \bigcap \{S \mid S \supseteq \llbracket \varphi, \rho[X \mapsto S] \rrbracket\}, & \llbracket X, \rho \rrbracket &= \rho(X), \\
\llbracket \nu X. \varphi, \rho \rrbracket &= \bigcup \{S \mid S \subseteq \llbracket \varphi, \rho[X \mapsto S] \rrbracket\}.
\end{aligned}$$

Table 1: Semantics of modal formulas on a model $\mathcal{M} = (W, R, V)$. We omit \mathcal{M} from the notation.

Depending on how we further restrict our syntax and the model, we can describe several logics. Without further restrictions, the resulting logic is the μ -calculus [21]. The max-fragment (resp. min-fragment) of the μ -calculus is the fragment that only allows the νX (resp. the μX) recursive operator. If $|\text{AG}| = k$ and we allow no recursive operators (or recursion variables), then we have the basic modal logic \mathbf{K}_k (or \mathbf{K} , if $k = 1$), and further restrictions on the frames can result in a wide variety of modal logics (see [6]). We give names to the following frame conditions, or frame constraints, for the case where $\text{AG} = \{\alpha\}$. These conditions correspond to the usual axioms for normal modal logics — see [5, 6, 14], which we will revisit in Section 3.

D: R is serial: $\forall s. \exists t. sRt$;

4: R is transitive: $\forall s, t, r. (sRtRr \Rightarrow sRr)$;

T: R is reflexive: $\forall s. sRs$;

5: R is euclidean: $\forall s, t, r.$ if sRt and sRr ,

B: R is symmetric: $\forall s, t. (sRt \Rightarrow tRs)$;

then tRr .

We consider modal logics that are interpreted over models that satisfy a combination of these constraints for each agent. D , which we call Consistency, is a special case of T , called Factivity. Constraint 4 is Positive Introspection and 5 is called Negative Introspection.² Given a logic \mathbf{L} and constraint c , $\mathbf{L} + c$ is the logic that is interpreted over all models with frames that satisfy all the constraints of \mathbf{L} and c . The name of a single-agent logic is a combination of the constraints that apply to its frames, including K , if the constraints are among 4 and 5. Therefore, logic \mathbf{D} is $\mathbf{K} + D$, \mathbf{T} is $\mathbf{K} + T$, \mathbf{B} is $\mathbf{K} + B$, $\mathbf{K4} = \mathbf{K} + 4$, $\mathbf{D4} = \mathbf{K} + D + 4 = \mathbf{D} + 4$, and so on. We use the special names $\mathbf{S4}$ for $\mathbf{T4}$ and $\mathbf{S5}$ for $\mathbf{T45}$. We define a (multi-agent) logic \mathbf{L} on AG as a map from agents to single-agent logics. \mathbf{L} is interpreted on Kripke models of the form (W, R, V) , where for every $\alpha \in \text{AG}$, (W, R_α) is a frame for $\mathbf{L}(\alpha)$.

For a logic \mathbf{L} , \mathbf{L}^μ is the logic that results from \mathbf{L} after we allow recursive operators in the syntax — in case they were not allowed in \mathbf{L} . Furthermore, if for every $\alpha \in \text{AG}$, $\mathbf{L}(\alpha)$ is the same single-agent logic \mathbf{L} , we write \mathbf{L} as \mathbf{L}_k , where $|\text{AG}| = k$. Therefore, the μ -calculus is \mathbf{K}_k^μ .

From now on, unless we explicitly say otherwise, by a logic, we mean one of the logics we have defined above. We call a formula satisfiable for a logic \mathbf{L} , if it is satisfied in some state of a model for \mathbf{L} .

Example 1. For a formula φ , we define $\text{Inv}(\varphi) = \nu X. (\varphi \wedge [\text{AG}]X)$. $\text{Inv}(\varphi)$ asserts that φ is true in *all* reachable states, or, alternatively, it can be read as an assertion that φ is common knowledge. We dually define $\text{Eve}(\varphi) = \mu X. (\varphi \vee \langle \text{AG} \rangle X)$, which asserts that φ is true in *some* reachable state.

²These are names for properties or axioms of a logic. When we refer to these conditions as conditions of a frame or model, we may refer to them with the name of the corresponding relation condition: seriality, reflexivity, symmetry, transitivity, and euclidicity.

2.2 Known Results

For logic \mathbf{L} , the satisfiability problem for \mathbf{L} , or \mathbf{L} -satisfiability is the problem that asks, given a formula φ , if φ is satisfiable. Similarly, the model checking problem for \mathbf{L} asks if φ is true at a given state of a given finite model.

Ladner [22] established the classical result of PSPACE-completeness for the satisfiability of \mathbf{K} , \mathbf{T} , \mathbf{D} , $\mathbf{K4}$, $\mathbf{D4}$, and $\mathbf{S4}$ and NP-completeness for the satisfiability of $\mathbf{S5}$. Halpern and Rêgo later characterized the NP–PSPACE gap for one-action logics by the presence or absence of Negative Introspection [18], resulting in Theorem 1. Later, Rybakov and Shkatov [27] proved the PSPACE-completeness of \mathbf{B} and \mathbf{TB} . For formulas with fixed-point operators, D’Agostino and Lenzi in [9] show that satisfiability for single-agent logics with constraint 5 is also NP-complete.

Theorem 1 ([18,22,27]). *If $\mathbf{L} \in \{\mathbf{K}, \mathbf{T}, \mathbf{D}, \mathbf{B}, \mathbf{TB}, \mathbf{K4}, \mathbf{D4}, \mathbf{S4}\}$, then \mathbf{L} -satisfiability is PSPACE-complete; and $\mathbf{L} + 5$ -satisfiability and $(\mathbf{L} + 5)^\mu$ -satisfiability is NP-complete.*

Theorem 2 ([17]). *If $k > 1$ and \mathbf{L} has a combination of constraints from $\mathbf{D}, \mathbf{T}, 4, 5$ and no recursive operators, then \mathbf{L}_k -satisfiability is PSPACE-complete.*

Remark 1. Note that Halpern and Moses in [17] prove these bounds for the cases of $\mathbf{K}_k, \mathbf{T}_k, \mathbf{S4}_k, \mathbf{KD45}_k$, and $\mathbf{S5}_k$ only. Similarly, D’Agostino and Lenzi in [9] only prove the NP-completeness of satisfiability for $\mathbf{S5}^\mu$. However, it is not hard to see that their respective methods also work for the rest of the logics of Theorems 1 and 2. ■

Theorem 3 ([21]). *The satisfiability problem for the μ -calculus is EXP-complete.*

Theorem 4 ([13]). *The model checking problem for the μ -calculus is in $\text{NP} \cap \text{coNP}$.³*

Finally we have the following initial known results about the complexity of satisfiability, when we have recursive operators. Theorems 5 and 6 have already been observed in [1].

Theorem 5. *The satisfiability problem for the min- and max-fragments of the μ -calculus is EXP-complete, even when $|\text{AG}| = 1$.*

Proof sketch. It is known that satisfiability for the min- and max-fragments of the μ -calculus (on one or more action) is EXP-complete. It is in EXP due to Theorem 3, and these fragments suffice [26] to describe the PDL formula that is constructed by the reduction used in [15] to prove EXP-hardness for PDL. Therefore, that reduction can be adjusted to prove that satisfiability for the min- and max-fragments of the μ -calculus is EXP-complete. □

It is not hard to express in logics with both frame constraints and recursion operators that formula φ is common knowledge, with formula $\forall X. \varphi \wedge [\text{AG}]X$. Since validity for \mathbf{L}_k with common knowledge (and without recursive operators) and $k > 1$ is EXP-complete [17]⁴, \mathbf{L}_k^μ is EXP-hard.

Proposition 6. *Satisfiability for \mathbf{L}_k^μ , where $k > 1$, is EXP-hard.*

3 Complexity through Translations

In this section, we examine \mathbf{L} -satisfiability. We use formula translations to reduce the satisfiability of one logic to the satisfiability of another. We investigate the properties of these translations and how they compose with each other, and we achieve complexity bounds for several logics.

³In fact, the problem is known to be in $\text{UP} \cap \text{coUP}$ [20].

⁴Similarly to Remark 1, [17] does not explicitly cover all these cases, but the techniques can be adjusted.

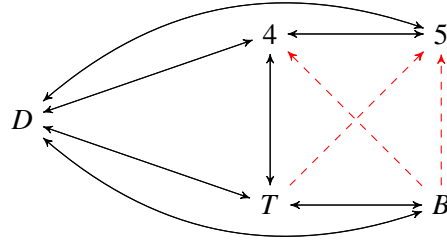


Figure 1: The frame property hierarchy

In the context of this paper, a formula translation from logic \mathbf{L}_1 to logic \mathbf{L}_2 is a mapping f on formulas such that each formula φ is \mathbf{L}_1 -satisfiable if and only if $f(\varphi)$ is \mathbf{L}_2 -satisfiable. We only consider translations that can be computed in polynomial time, and therefore, our translations are polynomial-time reductions, transferring complexity bounds between logics.

According to Theorem 3, \mathbf{K}_k^μ -satisfiability is EXP-complete, and therefore for each logic \mathbf{L} , we aim to connect \mathbf{K}_k^μ and \mathbf{L} via a sequence of translations in either direction, to prove complexity bounds for \mathbf{L} -satisfiability.

3.1 Translating Towards \mathbf{K}_k

We begin by presenting translations from logics with more to logics with fewer frame conditions. To this end, we study how taking the closure of a frame under one condition affects any other frame conditions.

3.1.1 Composing Frame Conditions

We now discuss how the conditions for frames affect each other. For example, to construct a transitive frame, one can take the transitive closure of a possibly non-transitive frame. The resulting frame will satisfy condition 4. As we see, taking the closure of a frame under condition x may affect whether that frame maintains condition y , depending on x and y . In the following we observe that one can apply the frame closures in certain orders that preserve the properties one acquires with each application.

Let $F = (W, R)$ be a frame, $\alpha \in A \subseteq \text{AG}$, and x a frame restriction among $T, B, 4, 5$. Then, $\overline{R_\alpha}^x$ is the closure of R_α under x , $\overline{R}^{x,A}$ is defined by $\overline{R}_\beta^{x,A} = \overline{R}_\beta^x$, if $\beta \in A$, and $\overline{R}_\beta^{x,A} = R_\beta$, otherwise. Then, $\overline{F}^{x,A} = (W, \overline{R}^{x,A})$. We make the following observation.

Lemma 7. *Let x be a frame restriction among $D, T, B, 4, 5$, and y a frame restriction among $T, B, 4, 5$, such that $(x, y) \neq (4, B), (5, T), (5, B)$. Then, for every frame F that satisfies x , \overline{F}^y also satisfies x .*

According to Lemma 7, frame conditions are preserved as seen in Figure 1. In Figure 1, an arrow from x to y indicates that property x is preserved though the closure of a frame under y . Dotted red arrows indicate one-way arrows. For convenience, we define $\overline{F}^D = (W, \overline{R}^D)$, where $\overline{R}^D = R \cup \{(a, a) \in W^2 \mid \nexists (a, b) \in R\}$.

Remark 2. We note that, in general, not all frame conditions are preserved through all closures under another condition. For example, the accessibility relation $\{(a, b), (b, b)\}$ is euclidean, but its reflexive closure $\{(a, b), (b, b), (a, a)\}$ is not.

There is at least one linear ordering of the frame conditions $D, T, B, 4, 5$, such that all preceding conditions are preserved by closures under the following conditions. We call such an order a closure-preserving order. We use the linear order $D, T, B, 4, 5$ in the rest of the paper.

3.1.2 Modal Logics

We start with translations that map logics without recursive operators to logics with fewer constraints. As mentioned in Subsection 2.2, all of the logics $\mathbf{L} \in \{\mathbf{K}, \mathbf{T}, \mathbf{D}, \mathbf{K4}, \mathbf{D4}, \mathbf{S4}\}$ and $\mathbf{L} + 5$ with one agent have known completeness results, and the complexity of modal logic is well-studied for multi-agent modal logics as well. The missing cases are very few and concern the combination of frame conditions (other than 5) as well as the multi-agent case. However we take this opportunity to present an intuitive introduction to our general translation method. In fact, the translations that we use for logics without recursion are surprisingly straightforward. Each frame condition that we introduced in Section 2 is associated with an axiom for modal logic, such that whenever a model has the condition, every substitution instance of the axiom is satisfied in all worlds of the model (see [5, 6, 14]). We give for each frame condition x and agent α , the axiom ax_α^x :

$$\begin{array}{lll} \text{ax}_\alpha^D: \langle \alpha \rangle \text{tt} & \text{ax}_\alpha^B: \langle \alpha \rangle [\alpha] p \rightarrow p & \text{ax}_\alpha^5: \langle \alpha \rangle [\alpha] p \rightarrow [\alpha] p \\ \text{ax}_\alpha^T: [\alpha] p \rightarrow p & \text{ax}_\alpha^4: [\alpha] p \rightarrow [\alpha][\alpha] p & \end{array}$$

For each formula φ and $d \geq 0$, let $\text{Inv}_d(\varphi) = \bigwedge_{i \leq d} [\text{AG}]^i \varphi$. Our first translations are straightforwardly defined from the above axioms.

Translation 1 (One-step Translation). Let $A \subseteq \text{AG}$ and let x be one of the frame conditions. For every formula φ , let $d = \text{md}(\varphi)$ if $x \neq 4$, and $d = \text{md}(\varphi)|\varphi|$, if $x = 4$. We define:

$$F_A^x(\varphi) = \varphi \wedge \text{Inv}_d \left(\bigwedge_{\substack{\psi \in \text{sub}(\varphi) \\ \alpha \in A}} \text{ax}_\alpha^x[\psi/p] \right).$$

Theorem 8. *Let $A \subseteq \text{AG}$, x be one of the frame conditions, and let $\mathbf{L}_1, \mathbf{L}_2$ be logics without recursion operators, such that $\mathbf{L}_1(\alpha) = \mathbf{L}_2(\alpha) + x$ when $\alpha \in A$, and $\mathbf{L}_2(\alpha)$ otherwise, and $\mathbf{L}_2(\alpha)$ only includes frame conditions that precede x in the fixed order of frame conditions. Then, φ is \mathbf{L}_1 -satisfiable if and only if $F_A^x(\varphi)$ is \mathbf{L}_2 -satisfiable.*

We present here a short proof sketch of this theorem.

Proof sketch. The proof of the “only if” direction is straightforward, as for any agent α with frame condition x , ax_α^x is valid in \mathbf{L}_1 . Thus the translation holds on any \mathbf{L}_1 -model that satisfies φ .

The other, and more involved direction requires the construction of an \mathbf{L}_1 -model for φ from an \mathbf{L}_2 -model for $F_A^x(\varphi)$. We make use of the observation that no modal logic formula can describe a model at depth more than the constant d . Therefore, we use the unfolding of the \mathbf{L}_2 -model, to keep track of the path that one takes to reach a certain state, and that path’s length. We then carefully reapply the necessary closures on the accessibility relations and we use induction on its subformulas, to prove that φ is true in the constructed model. We do this as a separate case for each axiom. It is worth noting that we pay special care for the case of $x = 4$ to account for the fact that the translation does not allow the modal depth of the relevant subformulas to decrease with each transition during the induction; and that we needed to include the negations of subformulas of φ in the conjunction of Translation 1, only for the case of $x = 5$. \square

Corollary 9. *The satisfiability problem for every logic without fixed-point operators is in PSPACE.*

3.1.3 Modal Logics with Recursion

In the remainder of this section we will modify our translations and proof technique, in order to lift our results to logics with fixed-point operators. It is not clear whether the translations of Subsection 3.1.2 can be extended straightforwardly in the case of logics with recursion, by using unbounded invariance Inv , instead of the bounded Inv_d .

Example 2. Let $\varphi_f = \mu X.\Box X$, which requires all paths in the model to be finite, and thus it is not satisfiable in reflexive frames. In Subsection 3.1.2, to translate formulas from reflexive models, we did not need to add the negations of subformulas as conjuncts. In this case, such a translation would give

$$\varphi_t := \varphi_f \wedge Inv((\Box\varphi_f \rightarrow \varphi_f) \wedge (\Box\Box\varphi_f \rightarrow \Box\varphi_f)).$$

Indeed, on reflexive frames, the formulas $\Box\varphi_f \rightarrow \varphi_f$ and $\Box\Box\varphi_f \rightarrow \Box\varphi_f$ are valid, and therefore φ_t is equivalent to φ_f , which is \mathbf{K} -satisfiable. This was not an issue in Subsection 3.1.2, as the finiteness of the paths in a model cannot be expressed without recursion.

One would then naturally wonder whether conjoining over $\overline{\text{sub}}(\varphi_f)$ in the translation would make a difference. The answer is affirmative, as the translation

$$\varphi_f \wedge Inv\left(\bigwedge_{\psi \in \overline{\text{sub}}(\varphi_f)} \Box\psi \rightarrow \psi\right)$$

would then yield a formula that is not satisfiable. However, our constructions would not work to prove that such a translation preserves satisfiability. For example, consider $\mu X.\Box(p \rightarrow (r \wedge (q \rightarrow X)))$, whose translation is satisfied on a pointed model that satisfies at the same time p and q . We invite the reader to verify the details.

The only case where the approach that we used for the logics without recursion can be applied is for the case of seriality (condition D), as $Inv(\langle\alpha\rangle\tau\tau)$ directly ensures the seriality of a model.

Translation 2.

$$F_A^{D^\mu}(\varphi) = \varphi \wedge Inv\left(\bigwedge_{\alpha \in A} \langle\alpha\rangle\tau\tau\right).$$

Theorem 10. *Let $A \subseteq \text{AG}$ and $|\text{AG}| = k$, and let \mathbf{L} be a logic, such that $\mathbf{L}(\alpha) = \mathbf{D}$ when $\alpha \in A$, and \mathbf{K} otherwise. Then, φ is \mathbf{L} -satisfiable if and only if $F_A^{D^\mu}(\varphi)$ is \mathbf{K}_k^μ -satisfiable.*

For the cases of reflexivity and transitivity, our simple translations substitute the modal subformulas of a formula to implicitly enforce the corresponding condition.

Translation 3. The operation $F_A^{T^\mu}(-)$ is defined to be such that

- $F_A^{T^\mu}([\alpha]\varphi) = [\alpha]F_A^{T^\mu}(\varphi) \wedge F_A^{T^\mu}(\varphi)$;
- $F_A^{T^\mu}(\langle\alpha\rangle\varphi) = \langle\alpha\rangle F_A^{T^\mu}(\varphi) \vee F_A^{T^\mu}(\varphi)$;
- and it commutes with all other operations.

Theorem 11. *Let $\emptyset \neq A \subseteq \text{AG}$, and let $\mathbf{L}_1, \mathbf{L}_2$ be logics, such that $\mathbf{L}_1(\alpha) = \mathbf{L}_2(\alpha) + T$ when $\alpha \in A$, and $\mathbf{L}_2(\alpha)$ otherwise, and $\mathbf{L}_2(\alpha)$ at most includes frame condition D . Then, φ is \mathbf{L}_1 -satisfiable if and only if $F_A^{T^\mu}(\varphi)$ is \mathbf{L}_2 -satisfiable.*

Proof sketch. The “only if” direction is proven by taking the appropriate reflexive closure and showing by induction that the subformulas of φ are preserved. \square

Translation 4. The operation $F_A^{4\mu}(-)$ is defined to be such that

- $F_A^{4\mu}([\alpha]\psi) = Inv([\alpha](F_A^{4\mu}(\psi)))$,
- $F_A^{4\mu}(\langle\alpha\rangle\psi) = Eve(\langle\alpha\rangle(F_A^{4\mu}(\psi)))$,
- $F_A^{4\mu}(-)$ commutes with all other operations.

Theorem 12. Let $\emptyset \neq A \subseteq AG$, and let L_1, L_2 be logics, such that $L_1(\alpha) = L_2(\alpha) + 4$ when $\alpha \in A$, and $L_2(\alpha)$ otherwise, and $L_2(\alpha)$ at most includes frame conditions D, T, B . Then, φ is L_1 -satisfiable if and only if $F_A^{4\mu}(\varphi)$ is L_2 -satisfiable.

Proof. If $F_A^{4\mu}(\varphi)$ is satisfied in a model $M = (W, R, V)$, let $M' = (W, R^+, V)$, where R_α^+ is the transitive closure of R_α , if $\alpha \in A$, and $R_\alpha^+ = R_\alpha$, otherwise. It is now not hard to use induction on ψ to show that for every (possibly open) subformula ψ of φ , for every environment ρ , $\llbracket F_A^{4\mu}(\psi), \rho \rrbracket_{\mathcal{M}} = \llbracket \psi, \rho \rrbracket_{\mathcal{M}'}$. The other direction is more straightforward. \square

In order to produce a similar translation for symmetric frames, we needed to use a more intricate type of construction. Moreover, we only prove the correctness of the following translation for formulas without least-fixed-point operators.

Translation 5. The operation $F_A^{B\mu}(-)$ is defined as

$$F_A^{B\mu}(\varphi) = \varphi \wedge Inv([\alpha]\langle\alpha\rangle p \wedge \bigwedge_{\psi \in \overline{\text{sub}}(\varphi)} (\psi \rightarrow [\alpha][\alpha](p \rightarrow \psi))),$$

where p is a new propositional variable, not occurring in φ .

Theorem 13. Let $\emptyset \neq A \subseteq AG$, and let L_1, L_2 be logics, such that $L_1(\alpha) = L_2(\alpha) + B$ when $\alpha \in A$, and $L_2(\alpha)$ otherwise, and $L_2(\alpha)$ at most includes frame conditions D, T . Then, a formula φ that has no μX operators is L_1 -satisfiable if and only if $F_A^{B\mu}(\varphi)$ is L_2 -satisfiable.

Remark 3. A translation for euclidean frames and for the full syntax on symmetric frames would need different approaches. D’Agostino and Lenzi show in [9] that $S5_2^\mu$ does not have a finite model property, and their result can be easily extended to any logic L with fixed-point operators, where there are at least two distinct agents α, β , such that $L(\alpha)$ and $L(\beta)$ have constraint B or 5 . Therefore, as our constructions for the translations to K_k^μ guarantee the finite model property to the corresponding logics, they do not apply to multimodal logics with B or 5 .

3.2 Embedding K_n^μ

In this subsection, we present translations from logics with fewer frame conditions to ones with more conditions. This will allow us to prove EXP-completeness in the following subsection. Let p, q be distinguished propositional variables that do not appear in our formulas. We let \vec{p} range over $p, \neg p, p \wedge q, p \wedge \neg q$, and $\neg p \wedge q$.

Definition 2 (function *next*). *next*($p \wedge q$) = $p \wedge \neg q$, *next*($p \wedge \neg q$) = $\neg p \wedge q$, and *next*($\neg p \wedge q$) = $p \wedge q$; and *next*(p) = $\neg p$ and *next*($\neg p$) = p .

We use a uniform translation from K_k^μ to any logic with a combination of conditions D, T, B .

Translation 6. The operation $F_A^{\mathbf{K}^\mu}(-)$ on formulas is defined such that:

- $F_A^{\mathbf{K}^\mu}(\langle \alpha \rangle \psi) = \bigwedge_{\vec{p}} (\vec{p} \rightarrow \langle \alpha \rangle (\text{next}(\vec{p}) \wedge F_A^{\mathbf{K}^\mu}(\psi)))$, if $\alpha \in A$;
- $F_A^{\mathbf{K}^\mu}([\alpha] \psi) = \bigwedge_{\vec{p}} (\vec{p} \rightarrow [\alpha] (\text{next}(\vec{p}) \rightarrow F_A^{\mathbf{K}^\mu}(\psi)))$, if $\alpha \in A$;
- $F_A^{\mathbf{K}^\mu}(-)$ commutes with all other operations.

We note that there are simpler translations for the cases of logics with only D or T as a constraint, but the $F_A^{\mathbf{K}^\mu}(-)$ is uniform for all the logics that we consider in this subsection.

Theorem 14. *Let $\emptyset \neq A \subseteq \text{AG}$, $|\text{AG}| = k$, and let \mathbf{L} be such that $\mathbf{L}(\alpha)$ includes only frame conditions from $\mathbf{D}, \mathbf{T}, \mathbf{B}$ when $\alpha \in A$, and $\mathbf{L}(\alpha) = \mathbf{K}$ otherwise. Then, φ is \mathbf{K}_k^μ -satisfiable if and only if $F_A^{\mathbf{K}^\mu}(\varphi)$ is \mathbf{L} -satisfiable.*

The proof of Theorem 14 can be found in [23]. It is worth noting that the “if” direction uses the symmetric closure to construct a new model, while the “only if” direction requires the introduction of new states that behave as each original state in the model.

3.3 Complexity results

We observe that our translations all result in formulas of size at most linear with respect to the original. The exceptions are Translations 1 and 5, which have a quadratic cost.

Corollary 15. *If \mathbf{L} only has frame conditions D, T , then its satisfiability problem is EXP-complete; if \mathbf{L} only has frame conditions $D, T, 4$, then its satisfiability problem is in EXP.*

Proof. Immediately from Theorems 10, 11, 12, and 14. □

Corollary 16. *If \mathbf{L} only has frame conditions D, T, B , then*

1. *\mathbf{L} -satisfiability is EXP-hard; and*
2. *the restriction of \mathbf{L} -satisfiability on formulas without μX operators is EXP-complete.*

Proof. Immediately from Theorems 10, 11, 13, and 14. □

4 Tableaux for \mathbf{L}_k^μ

We give a sound and complete tableau system for logic \mathbf{L} . Furthermore, if \mathbf{L} has a finite model property, then we give terminating conditions for its tableau. The system that we give in this section is based on Kozen’s tableaux for the μ -calculus [21] and the tableaux of Fitting [16] and Massacci [24] for modal logic. We can use Kozen’s finite model theorem [21] to help us ensure the termination of the tableau for some of these logics.

Theorem 17 ([21]). *There is a computable $\kappa : \mathbb{N} \rightarrow \mathbb{N}$, such that every \mathbf{K}_k^μ -satisfiable formula φ is satisfied in a model with at most $\kappa(|\varphi|)$ states.⁵*

Corollary 18. *If \mathbf{L} only has frame conditions $D, T, 4$, then there is a computable $\kappa : \mathbb{N} \rightarrow \mathbb{N}$, such that every \mathbf{L} -satisfiable formula φ is satisfied in a model with at most $\kappa(|\varphi|)$ states.*

⁵The tableau in [21] yields an upper bound of $2^{2^{O(n^3)}}$ for $\kappa_0(n)$, but that bound is not useful to obtain a “good” decision procedure. The purpose of this section is not to establish any good upper bound for satisfiability testing, which is done in Section 3.

$$\begin{array}{cccc}
\frac{\sigma \pi X. \varphi}{\sigma \varphi} \text{ (fix)} & \frac{\sigma X}{\sigma \text{fx}(X)} \text{ (X)} & \frac{\sigma \varphi \vee \psi}{\sigma \varphi \mid \sigma \psi} \text{ (or)} & \frac{\sigma \varphi \wedge \psi}{\sigma \varphi} \text{ (and)} \\
\frac{\sigma [\alpha] \varphi}{\sigma. \alpha \langle \psi \rangle \varphi} \text{ (B)} & \frac{\sigma \langle \alpha \rangle \varphi}{\sigma. \alpha \langle \varphi \rangle \varphi} \text{ (D)} & \frac{\sigma [\alpha] \varphi}{\sigma. \alpha \langle \varphi \rangle \varphi} \text{ (d)} & \frac{\sigma [\alpha] \varphi}{\sigma. \alpha \langle \psi \rangle [\alpha] \varphi} \text{ (4)}
\end{array}$$

where, for rules (B) and (4), $\sigma. \alpha \langle \psi \rangle$ has already appeared in the branch; and for (D), σ is not α -flat.

$$\begin{array}{cccc}
\frac{\sigma. \alpha \langle \psi \rangle [\alpha] \varphi}{\sigma [\alpha] \varphi} \text{ (B5)} & \frac{\sigma. \alpha \langle \psi \rangle \langle \alpha \rangle \varphi}{\sigma. \alpha \langle \psi \rangle. \alpha \langle \varphi \rangle \varphi} \text{ (D5)} & \frac{\sigma. \alpha \langle \psi \rangle [\alpha] \varphi}{\sigma \varphi} \text{ (b)} & \frac{\sigma [\alpha] \varphi}{\sigma \varphi} \text{ (t)} \\
\frac{\sigma. \alpha \langle \psi \rangle [\alpha] \varphi}{\sigma. \alpha \langle \psi' \rangle [\alpha] \varphi} \text{ (B55)} & \frac{\sigma. \alpha \langle \psi \rangle. \alpha \langle \psi' \rangle \langle \alpha \rangle \varphi}{\sigma. \alpha \langle \psi \rangle. \alpha \langle \varphi \rangle \varphi} \text{ (D55)} & \frac{\sigma. \alpha \langle \psi \rangle [\alpha] \varphi}{\sigma [\alpha] \varphi} \text{ (b4)} &
\end{array}$$

where, for rule (B55), $\sigma. \alpha \langle \psi' \rangle$ has already appeared in the branch; for rule (D5), σ is not α -flat, and $\sigma \langle \alpha \rangle \varphi$ does not appear in the branch; for rule (D55), $\sigma \langle \alpha \rangle \varphi$ does not appear in the branch.

Table 2: The tableau rules for $\mathbf{L} = \mathbf{L}_n^{\mu}$

Proof. Immediately, from Theorems 17, 10, 11, and 12, and Lemma 7. \square

Remark 4. We note that not all modal logics with recursion have a finite model property – see Remark 3.

Intuitively, a tableau attempts to build a model that satisfies the given formula. When it needs to consider two possible cases, it branches, and thus it may generate several branches. Each branch that satisfies certain consistency conditions, which we define below, represents a corresponding model.

Our tableaux use *prefixed formulas*, that is, formulas of the form $\sigma \varphi$, where $\sigma \in (\text{AG} \times L)^*$ and $\varphi \in L$; σ is the prefix of φ in that case, and we say that φ is prefixed by σ . We note that we separate the elements of σ with a dot. We say that the prefix σ is α -flat when α has axiom 5 and $\sigma = \sigma'. \alpha \langle \psi \rangle$ for some ψ . Each prefix possibly represents a state in a corresponding model, and a prefixed formula $\sigma \varphi$ declares that φ is satisfied in the state represented by σ . As we will see below, the prefixes from $(\text{AG} \times L)^*$ allow us to keep track of the diamond formula that generates a prefix through the tableau rules. For agents with condition 5, this allows us to restrict the generation of new prefixes and avoid certain redundancies, due to the similarity of euclidean binary relations to equivalence relations [18, 25].

The tableau rules that we use appear in Table 2. These include fixed-point and propositional rules, as well as rules that deal with modalities. Depending on the logic that each agent α is based on, a different set of rules applies for α : for rule (d), $\mathbf{L}(\alpha)$ must have condition *D*; for rule (t), $\mathbf{L}(\alpha)$ must have condition *T*; for rule (4), $\mathbf{L}(\alpha)$ must have condition 4; for rule (B5), (D5), and (D55), $\mathbf{L}(\alpha)$ must have condition 5; for (b) $\mathbf{L}(\alpha)$ must have condition *B*; and for (b4) $\mathbf{L}(\alpha)$ must have both *B* and 4. Rule (or) is the only rule that splits the current tableau branch into two. A tableau branch is propositionally closed when σff or both σp and $\sigma \neg p$ appear in the branch for some prefix σ . For each prefix σ that appears in a fixed tableau branch, let $\Phi(\sigma)$ be the set of formulas prefixed by σ in that branch. We use the notation $\sigma \prec \sigma'$ to mean that $\sigma' = \sigma. \sigma''$ for some σ'' , in which case σ is an ancestor of σ' .

We define the relation \xrightarrow{X} on prefixed formulas in a tableau branch as $\chi_1 \xrightarrow{X} \chi_2$, if $\frac{\chi_1}{\chi_2}$ is a tableau rule and χ_1 is not of the form σY , where $X < Y$; then, $\xrightarrow{X^+}$ is the transitive closure of \xrightarrow{X} and $\xrightarrow{X^*}$ is its reflexive and transitive closure. We can also extend this relation to prefixes, so that $\sigma \xrightarrow{X} \sigma'$, if and only if $\sigma \psi \xrightarrow{X} \sigma' \psi'$, for some $\psi \in \Phi(\sigma)$ and $\psi' \in \Phi(\sigma')$. If in a branch there is a \xrightarrow{X} -sequence where X is

a least fixed-point and appears infinitely often, then the branch is called fixed-point-closed. A branch is closed when it is either fixed-point-closed or propositionally closed; if it is not closed, then it is called open.

Now, assume that there is a $\kappa : \mathbb{N} \rightarrow \mathbb{N}$, such that every \mathbf{L} -satisfiable formula φ is satisfied in a model with at most $\kappa(|\varphi|)$ states. An open tableau branch is called (*resp. locally*) *maximal* when all tableau rules (*resp. the tableau rules that do not produce new prefixes*) have been applied. A branch is called *sufficient* for φ when it is locally maximal and for every $\sigma \psi$ in the branch, for which a rule can be applied and has not been applied to $\sigma \psi$, $|\sigma| > |\text{AG}| \cdot \kappa(|\varphi|)^{|\varphi|^2} \cdot 2^{2|\varphi|+1}$. A tableau is called maximal when all of its open branches are maximal, and closed when all of its branches are closed. It is called sufficiently closed for φ if it is propositionally closed, or for some least fixed-point variable X , it has a \xrightarrow{X} -path, where X appears at least $\kappa(|\varphi|) + 1$ times. A sufficient branch for φ that is not sufficiently closed is called sufficiently open for φ .

A tableau for φ starts from $\varepsilon \varphi$ and is built using the tableau rules of Table 2. A tableau proof for φ is a closed tableau for the negation of φ .

Theorem 19 (Soundness, Completeness, and Termination of \mathbf{L}_k^μ -Tableaux). *From the following, the first two are equivalent for any formula $\varphi \in L$ and any logic \mathbf{L} . Furthermore, if there is a $\kappa : \mathbb{N} \rightarrow \mathbb{N}$, such that every \mathbf{L} -satisfiable formula φ is satisfied in a model with at most $\kappa(|\varphi|)$ states, then all the following are equivalent.*

1. φ has a maximal \mathbf{L} -tableau with an open branch;
2. φ is \mathbf{L} -satisfiable; and
3. φ has an \mathbf{L} -tableau with a sufficiently open branch for φ .

Proof sketch. The direction from 1 to 2 uses the usual model construction, but where one needs to take into account the fixed-point formulas; the direction from 2 to 3 uses techniques and results from [21, 28], including Corollary 18; and the direction from 3 to 1 shows how to detect appropriate parts of the branch to repeat until we safely get a maximal branch. \square

Corollary 20. *\mathbf{L} -tableaux are sound and complete for \mathbf{L} .*

Example 3. Let $\text{AG} = \{a, b\}$ and \mathbf{L} be a logic, such that $\mathbf{L}(a) = \mathbf{K}^\mu$ and $\mathbf{L}(b) = \mathbf{K5}^\mu$. Let

$$\varphi_1 = (p \wedge \langle a \rangle p) \wedge \mu X. (\neg p \vee [a]X) \quad \text{and} \quad \varphi_2 = \langle b \rangle p \wedge \mu X. ([b] \neg p \vee [b]X).$$

As we see in Figure 2, the tableau for φ_1 produces an open branch, while the one for φ_2 has all of its branches closed, the leftmost one due to an infinite \xrightarrow{X} -sequence.

5 Conclusions

We studied multi-modal logics with recursion. These logics mix the frame conditions from epistemic modal logic, and the recursion of the μ -calculus. We gave simple translations among these logics that connect their satisfiability problems. This allowed us to offer complexity bounds for satisfiability and to prove certain finite model results. We also presented a sound and complete tableau that has termination guarantees, conditional on a logic's finite model property.

$$\begin{array}{c}
\frac{\frac{\frac{\frac{\frac{\frac{\varepsilon (p \wedge \langle a \rangle p) \wedge \mu X.(\neg p \vee [a]X)}{\varepsilon \mu X.(\neg p \vee [a]X)} \\ \frac{\frac{\frac{\varepsilon p \wedge \langle a \rangle p}{\varepsilon p}}{\varepsilon \langle a \rangle p} \text{ (fix)}}{\varepsilon \neg p \vee [a]X} \\ \frac{\frac{\frac{\varepsilon [a]X}{a \langle p \rangle p} \text{ (D)}}{a \langle p \rangle X} \text{ (B)}}{a \langle p \rangle \mu X.(\neg p \vee [a]X)} \text{ (X)}}{a \langle p \rangle \neg p \vee [a]X} \text{ (fix)}}{a \langle p \rangle [a]X \quad a \langle p \rangle \neg p} \text{ x} \\
\frac{\frac{\frac{\frac{\frac{\frac{\varepsilon \langle b \rangle p \wedge \mu X.([b]\neg p \vee [b]X)}{\varepsilon \mu X.([b]\neg p \vee [b]X)} \\ \frac{\frac{\frac{\frac{\varepsilon \langle b \rangle p}{b \langle p \rangle p} \text{ (D)}}{b \langle p \rangle p} \text{ (fix)}}{\varepsilon [b]\neg p \vee [b]X} \\ \frac{\frac{\frac{\frac{\varepsilon [b]X}{b \langle p \rangle X} \text{ (B)}}{b \langle p \rangle \mu X.([b]\neg p \vee [b]X)} \text{ (X)}}{b \langle p \rangle [b]\neg p \vee [b]X} \text{ (fix)}}{\frac{b \langle p \rangle [b]X}{\varepsilon [b]X} \text{ (B5)} \quad \frac{b \langle p \rangle [b]\neg p}{\varepsilon [b]\neg p} \text{ (B5)}}{\frac{b \langle p \rangle [b]X}{\varepsilon [b]X} \text{ (B)} \quad \frac{b \langle p \rangle [b]\neg p}{\varepsilon [b]\neg p} \text{ (B)}}{b \langle p \rangle X \quad b \langle p \rangle \neg p} \text{ x} \\
\vdots
\end{array}$$

Figure 2: Tableaux for φ_1 and φ_2 . The dots represent that the tableau keeps repeating as from the identical node above. The x mark represents a propositionally closed branch.

Conjectures and Future Work We currently do not possess full translations for the cases of symmetric and euclidean frames. What is interesting is that we also do not have a counterexample to prove that the translations that we already have, as well as other attempts, are *not* correct. In the case of symmetric frames, we have managed to prove that our construction works for formulas without least-fixed-point operators. A translation for euclidean frames and for the full syntax on symmetric frames is left as future work. We know that we cannot use the same model constructions that preserve the finiteness of the model as in Subection 3.1.3 (see Remark 3).

We do not prove the finite model property on all logics. We note that although it is known that logics with recursion with at least two agents with either B or 5 do not have this property (see 3, [9]), the situation is unclear if there is only one such agent.

We further conjecture that it is not possible to prove EXP-completeness for all the single-agent cases. Specifically, we expect $\mathbf{K4}^\mu$ -satisfiability to be in PSPACE, similarly to how $\mathbf{K5}^\mu$ -satisfiability is in NP [9]. As such, we do not expect Translation 6 to be correct for these cases.

The model checking problem for the μ -calculus is an important open problem. The problem does not depend on the frame restrictions of the particular logic, though one may wonder whether additional frame restrictions would help solve the problem more efficiently. We are not aware of a way to use our translations to solve model checking more efficiently.

As, to the best of our knowledge, most of the logics described in this paper have not been explicitly defined before, with notable exceptions such as [3, 9, 11], they also lack any axiomatizations and completeness theorems. We do expect the classical methods from [17, 21, 22] and others to work out in these cases as well. However it would be interesting to see if there are any unexpected situations that arise.

Given the importance of common knowledge for epistemic logic and the fact that it has been known that common knowledge can be thought of as a (greatest) fixed-point already from [4, 19], we consider the logics that we presented to be natural extensions of modal logic. Besides the examples given in Section 2, we are interested in exploring what other natural concepts can be defined with this enlarged language.

References

- [1] Luca Aceto, Antonis Achilleos, Adrian Francalanza & Anna Ingólfssdóttir (2020): *The complexity of identifying characteristic formulae*. *J. Log. Algebraic Methods Program.* 112, p. 100529. Available at <https://doi.org/10.1016/j.jlamp.2020.100529>.
- [2] Luca Alberucci & Alessandro Facchini (2009): *The modal μ -calculus hierarchy over restricted classes of transition systems*. *The Journal of Symbolic Logic* 74(4), p. 1367–1400, doi:10.2178/jsl/1254748696.
- [3] Luca Alberucci & Alessandro Facchini (2009): *The modal μ -calculus hierarchy over restricted classes of transition systems*. *The Journal of Symbolic Logic* 74(4), pp. 1367–1400, doi:10.2178/jsl/1254748696.
- [4] Jon Barwise (1988): *Three views of common knowledge*. In: *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning About Knowledge, TARK '88*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 365–379. Available at <http://dl.acm.org/citation.cfm?id=1029718.1029753>.
- [5] Patrick Blackburn, Johan van Benthem & Frank Wolter (2006): *Handbook of Modal Logic*. *Studies in Logic and Practical Reasoning* 3, Elsevier Science.
- [6] Patrick Blackburn, Maarten de Rijke & Yde Venema (2001): *Modal Logic*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, doi:10.1017/cbo9781107050884.
- [7] Laura Bozzelli, Bastien Maubert & Sophie Pinchinat (2014): *Unifying Hyper and Epistemic Temporal Logic*. [abs/1409.2711](https://arxiv.org/abs/1409.2711), doi:10.1007/978-3-662-46678-0_11. arXiv:1409.2711.
- [8] Michael R. Clarkson & Fred B. Schneider (2010): *Hyperproperties*. *Journal of Computer Security* 18(6), p. 1157–1210, doi:10.3233/JCS-2009-0393.
- [9] Giovanna D'Agostino & Giacomo Lenzi (2013): *On modal μ -calculus in S5 and applications*. *Fundamenta Informaticae* 124(4), pp. 465–482, doi:10.3233/FI-2013-844.
- [10] Giovanna D'Agostino & Giacomo Lenzi (2018): *The μ -Calculus Alternation Depth Hierarchy is infinite over finite planar graphs*. *Theoretical Computer Science* 737, pp. 40–61, doi:10.1016/j.tcs.2018.04.009. Available at <https://www.sciencedirect.com/science/article/pii/S0304397518302317>.
- [11] Giovanna D'Agostino & Giacomo Lenzi (2010): *On the μ -calculus over transitive and finite transitive frames*. *Theoretical Computer Science* 411(50), pp. 4273–4290, doi:10.1016/j.tcs.2010.09.002. Available at <https://www.sciencedirect.com/science/article/pii/S030439751000469X>.
- [12] Giovanna D'Agostino & Giacomo Lenzi (2015): *On the modal μ -Calculus over finite symmetric graphs*. *Mathematica Slovaca* 65(4), pp. 731–746, doi:10.1515/ms-2015-0052.
- [13] E Allen Emerson, Charanjit S Jutla & A Prasad Sistla (2001): *On model checking for the μ -calculus and its fragments*. *Theoretical Computer Science* 258(1-2), pp. 491–522, doi:10.1016/S0304-3975(00)00034-7.
- [14] Ronald Fagin, Joseph Y. Halpern, Yoram Moses & Moshe Y. Vardi (1995): *Reasoning About Knowledge*. The MIT Press, doi:10.7551/mitpress/5803.001.0001.
- [15] Michael J. Fischer & Richard E. Ladner (1979): *Propositional dynamic logic of regular programs*. *Journal of computer and system sciences* 18(2), pp. 194–211, doi:10.1016/0022-0000(79)90046-1.
- [16] Melvin Fitting (1972): *Tableau methods of proof for modal logics*. *Notre Dame Journal of Formal Logic* 13(2), pp. 237–247, doi:10.1305/ndjfl/1093894722.
- [17] Joseph Y. Halpern & Yoram Moses (1992): *A guide to completeness and complexity for modal logics of knowledge and belief*. *Artificial Intelligence* 54(3), pp. 319–379, doi:10.1016/0004-3702(92)90049-4.
- [18] Joseph Y. Halpern & Leandro Chaves Rêgo (2007): *Characterizing the NP-PSPACE gap in the satisfiability problem for modal logic*. *Journal of Logic and Computation* 17(4), pp. 795–806, doi:10.1093/logcom/exm029.
- [19] Gilbert Harman (1977): *Review of linguistic behavior by Jonathan Bennett*. *Language* 53, pp. 417–424, doi:10.1353/lan.1977.0036.
- [20] Marcin Jurdziński (1998): *Deciding the winner in parity games is in $UP \cap co-UP$* . *Information Processing Letters* 68(3), pp. 119–124, doi:10.1016/S0020-0190(98)00150-1.

- [21] Dexter Kozen (1983): *Results on the propositional μ -calculus*. *Theoretical Computer Science* 27(3), pp. 333–354, doi:10.1016/0304-3975(82)90125-6.
- [22] Richard E. Ladner (1977): *The computational complexity of provability in systems of modal propositional logic*. *SIAM Journal on Computing* 6(3), pp. 467–480, doi:10.1137/0206033.
- [23] Elli Anastasiadi Adrian Francalanza Anna Ingólfssdóttir Luca Aceto, Antonis Achilleos: *Complexity through Translations for Modal Logic with Recursion*. Available at <http://icetcs.ru.is/movemnt/papers/ComplexityTranslationsMLRecFullGand.pdf>.
- [24] Fabio Massacci (1994): *Strongly analytic tableaux for normal modal logics*. In: *CADE*, pp. 723–737, doi:10.1007/3-540-58156-1_52.
- [25] Michael C. Nagle & S. K. Thomason (1985): *The extensions of the modal logic K5*. *Journal of Symbolic Logic* 50(1), pp. 102—109, doi:10.2307/2273793.
- [26] Vaughan R. Pratt (1981): *A decidable mu-calculus: Preliminary report*. In: *22nd Annual Symposium on Foundations of Computer Science (SFCS 1981)*, IEEE, doi:10.1109/sfcs.1981.4.
- [27] Mikhail Rybakov & Dmitry Shkatov (2018): *Complexity of finite-variable fragments of propositional modal logics of symmetric frames*. *Logic Journal of the IGPL* 27(1), pp. 60–68, doi:10.1093/jigpal/jzy018.
- [28] Wieslaw Zielonka (1998): *Infinite games on finitely coloured graphs with applications to automata on infinite trees*. *Theoretical Computer Science* 200(1-2), pp. 135–183, doi:10.1016/S0304-3975(98)00009-7.

Schema-Based Automata Determinization

Joachim Niehren

Inria, France Université de Lille
joachim.niehren@inria.fr

Momar Sakho

Inria, France Université de Lille
momar.sakho@inria.fr

Antonio Al Serhali

Inria, France Université de Lille
antonio.al-serhali@inria.fr

We propose an algorithm for schema-based determinization of finite automata on words and of stepwise hedge automata on nested words. The idea is to integrate schema-based cleaning directly into automata determinization. We prove the correctness of our new algorithm and show that it is always more efficient than standard determinization followed by schema-based cleaning. Our implementation permits to obtain a small deterministic automaton for an example of an XPath query, where standard determinization yields a huge stepwise hedge automaton for which schema-based cleaning runs out of memory.

1 Introduction

Nested words are words enhanced with well-nested parenthesis. They generalize over trees, unranked trees, and sequences of thereof that are also called hedges or forests. Nested words provide a formal way to represent semi-structured textual documents of XML and JSON format.

Regular queries for nested words can be defined by finite state automata. We will use stepwise hedge automata (SHAs) for this purpose [19], which combine finite state automata for words and trees in a natural manner. SHAs refine previous notions of hedge automata from the sixties [24, 9] in order to obtain a decent notion of left-to-right and bottom-up determinism. They extend on stepwise tree automata [8], so that they can not only be applied to unranked trees but also to hedges. Any SHA defines a forest algebra [5] based on its transition relation. Furthermore, SHAs can always be determinized and have the same expressiveness as (deterministic) nested word automaton (NWA) [16, 6, 2, 21]. Note, however, that SHAs do not provide any form of top-down determinism in contrast to NWAs.

Efficient compilers from regular XPATH queries to SHAs exist [19], possibly using NWAs as intermediates [4, 17, 10]. Our main motivation is to determinize the SHAs of regular XPATH queries since deterministic automata are crucial for various algorithmic querying tasks. In particular, determinism reduce the complexity of universality or inclusion checking from EXP-completeness to P-time, both for the classes of deterministic SHAs or NWAs. In turn, universality checking is relevant for the earliest query answering of XPath queries on XML streams [14]. Furthermore, determinism is needed for efficient in-memory answer enumeration of regular queries [22].

Automata determinization may take exponential time in the worst case, so it may not always be feasible in practice. For SHAs compiled from the XPATH queries of the XPathMark benchmark [12], however, it was shown to be unproblematic. This changes for the XPATH benchmark collected by Lick and Schmitz [15]: for 37% of its regular XPATH queries, SHA determinization does require more than 100 seconds, in which case it produces huge deterministic automata [1]. An example is:

```
(QN7)     /a/b//(* | @* | comment() | text())
```

This XPath query selects all nodes of an XML document that are descendants of a *b*-element below an *a*-element at the root. The nodes may have any XML type: element, attribute, comment, or text. The

nondeterministic SHA for QN7 has 145 states and an overall size of 348. Its determinization however leads to an automaton with 10.005 states and an overall size of 1.634.122.

A kick-off question is how to reduce the size of deterministic automata. One approach beside of minimization is to apply schema-based cleaning [19], where the schema of a query defines to which nested words the query can be applied. Schemas are always given by deterministic automata while the automata for queries may be nondeterministic. The idea of schema-based automaton cleaning is to keep only those states and transition rules of the automaton, that are needed to recognize some nested word satisfying the schema. The needed states and rules can be found by building the product of automata for the query and the schema. For XPATH queries selecting nodes, we have the schema one^x that states that a single node is selected for a fixed variable x by any answer of the query. The second schema expresses which nested words satisfy the XML data model. With the intersection of these two schemas, the schema-based cleaning of the deterministic SHA for QN7 indeed has only 74 states and 203 rules. When applying SHA minimization afterwards, the size of the automaton goes down to 27 states and 71 transition rules. However, our implementation of schema-based cleaning, runs out of memory for larger automata with say more than 1000 states. Therefore, we cannot compute the schema-based cleaning from the deterministic SHA obtained from QN7. Neither can we minimize it with our implementation of deterministic SHA minimization. The question of how to produce small deterministic automaton for queries as simple as QN7 thus needs a different answer.

Given the relevance of schemas, one naive approach could be to determinize the product of the automata for the query and schema. This may look questionable at first sight, given that the schema-product may be bigger than the original automaton, so why could it make determinization more efficient? But in the case of QN7, the determinization of the schema-product yields a deterministic automata with only 92 states and 325 transition rules, and can be computed efficiently. This observation is very promising, motivating three general questions:

1. Why are schemas so important for automata determinization?
2. Can this be established by some complexity result?
3. Is there a way to compute the schema-based cleaning of the determinization of an SHA more efficiently than be schema-based cleaning followed by determinization?

Our main result is a novel algorithm for schema-based determinization of NFAS and SHAs, that integrates schema-based cleaning directly into the usual determinization algorithm. This algorithm answers question 3 positively. Its idea is to keep only those subsets of states of the automaton during the determinization, that can be aligned to some state of the schema. In our Theorem 2, we prove that schema-based determinization always produces the same deterministic automaton than schema-free determinization followed by schema-based cleaning. By schema-based determinization we could compute the schema-based cleaning of the determinization of QN7 in less than three seconds. In contrast, the schema-based cleaning of the determinization does not terminate after a few hours. In the general case, the worst case complexity of schema-based determinization is lower than schema-less determinization followed by schema-based cleaning.

We also provide a more precise complexity upper bound in Proposition 13. Given an nondeterministic SHA A let $det(A)$ be its determinization, and given a deterministic SHA S for the schema, let $A \times S$ the accessible part of the schema-product, and $scl_S(A)$ the schema-based cleaning of S with respect to schema S . We show that the upper bound for the maximal computation time of $scl_S(det(A))$ depends quadratically on the number of states of $S \times det(A)$, which is often way smaller than for $det(A)$ since S is deterministic. This complexity result shows why the schema is so relevant for determinization

(questions 1 and 2), and why computing the schema-based determinization is often more efficient than determinization followed by schema-based cleaning (question 3).

To see that $S \times \det(A)$ is often way smaller than $\det(A)$ for deterministic S we first note that $\det(A \times S) = \det(A) \times S$ since S is deterministic.¹ So for the many states $Q = \{q_1 \dots q_n\}$ of $\det(A)$ there may not exist any state s of S such that $(Q, s) \in \det(A) \times S$, because this requires all states q_i can be aligned to s , i.e. that (q_i, s) in $A \times S$ for all $1 \leq i \leq n$. Furthermore, $\det(A) \times S$ is equal to $\det_S(A) \times S$, so that $\det(A \times S) = \det_S(A) \times S$. Hence any size bound for the schema-based determinization $\det_S(A)$ implies a size bound for the determinization of the schema-product. Also, in our experiments $\det_S(A) \times S$ is almost by a factor of 2 bigger than $\det_S(A)$. So the size of the determinization of the schema-product is closely tied to the size of the schema-based determinization.

We also present a experimental evaluation of our implementation of schema-based determinization of SHAs. We consider a scalable family of SHAs obtained from a scalable family of XPATH queries. Our experiments confirm the very large improvement implied by the usage of schemas for determinization. For this, we implemented the algorithm for schema-based SHA determinization in Scala. Furthermore, we applied the XSLT compiler from regular forward XPATH queries to SHAs from [19], as well as the datalog implementations of SHA minimization and schema-based cleaning from there.

A large scale experiment on practical XPATH queries was provided in follow-up work [1] where schema-based algorithms were applied to the regular XPATH queries collected by Lick and Schmitz [15] from real word XQuery and XSLT programs. Small deterministic SHAs could be obtained by schema-based determinization for all regular XPATH queries in this corpus. In contrast, standard determinization in 37% of the cases fails with a timeout of 100 seconds. Without this timeout, determinization either runs out of memory or produces very large automata.

Outline. We start with related work on automata for nested words, determinization for XPATH queries (Section 2). In Section 3, we recall the definition NFAs and discuss how to use them as schemas and queries on words. In Section 4, we recall schema-based cleaning for NFAs. In Section 5, we contribute our schema-based determinization algorithm in the case of NFAs and show its correctness. In Section 6, we recall the notion of SHAs for defining languages of nested words. In Section 7, we lift schema-based determinization to SHAs. Full proofs can be found in the Appendix of the long version [20].

2 Related Work

We focus on automata for nested words, even though our results are new for NFAs too.

Nested word automata. As recalled in the survey of Okhotin and Salomaa [21], Alur’s et al. [2] NWAs were first introduced in the eighties under the name of input driven automata by Mehlhorn [16], and then reinvented several times under different names. In particular, they were called visibly pushdown automata [3], pushdown forest automata [18], and streaming tree automata [13]. The determinization algorithm for NWAs was first invented in the eighties by von Braunmühl and Verbeek in the journal version of [6] and then rediscovered various times later on too.

Determinization algorithms. The usual determinization algorithms for NFAs relies on the well-known subset construction. The determinization algorithms of bottom-up tree automata and SHAs are straightforward extensions thereof. The determinization algorithm for NWAs, in contrast, is more complicated, since having to deal with pushdowns. Subsets of *pairs* of states are to be considered there and not

¹If $\{(q_1, s_1) \dots (q_n, s_n)\} \in \det(A \times S)$ then there exists a tree that can go into all states $q_1 \dots q_n$ with A and into all states s_1, \dots, s_n with S . Since S is deterministic, we have $s_1 = \dots = s_n$. So there exists a tree going into $\{q_1, \dots, q_n\}$ with $\det(A)$ and also into all s_i . So $(\{q_1, \dots, q_n\}, s_i)$ is a state of $\det(A) \times S$.

$$\begin{array}{c}
\frac{I^A \neq \emptyset}{I^A \in I^{\det(A)} \quad I^A \in \mathcal{Q}^{\det(A)}} \quad \frac{Q \in \mathcal{Q}^{\det(A)} \quad Q \cap F^A \neq \emptyset}{Q \in F^{\det(A)}} \\
\frac{Q \in \mathcal{Q}^{\det(A)} \quad Q' = \{q' \in \mathcal{Q}^A \mid q \xrightarrow{a} q' \in \Delta^A, q \in Q\} \neq \emptyset}{Q \xrightarrow{a} Q' \in \Delta^{\det(A)} \quad Q' \in \mathcal{Q}^{\det(A)}} \\
\det(A) = (\Sigma, \mathcal{Q}^{\det(A)}, \Delta^{\det(A)}, I^{\det(A)}, F^{\det(A)})
\end{array}$$

Figure 1: The accessible determinization $\det(A)$ of NFA A .

only subsets of states as with the usual automata determinization algorithm. We also notice that general pushdown automata with nonvisible stacks can even not always be determinized.

Application to XPATH. Debarbieux et al. [10] noticed that the determinization algorithm for NWA_s often behaves badly when applied to NWA_s obtained from XPath queries as simple as $//a/b$. Niehren and Sakho [19] observed more recently that the situation is different for the determinization of SHAs: It works out nicely for the SHA of $//a/b$ and also for all other SHAs obtained by compilation from forward navigational XPath queries in the XPathMark benchmark [12]. Even more surprisingly, the same good behavior could be observed for the determinization algorithm of NWA when restricted to NWA_s with the weak-single entry property.

Weak single-entry NWA_s versus SHAs. The weak-single entry property implies that an NFA cannot memoize anything in its state when moving top-down. So it can only pass information left-to-right and bottom-up, similarly to an SHA. This property failed for the NWA_s considered by Debarbieux et al. and the determinization of their NWA_s thus required top-down determinization. This quickly led to the size explosion described above. On the other hand side, the weak single-entry property can always be established in quadratic time by compiling NWA_s to SHAs forth and back. Or else, one can avoid top-down determinization all over by directly working with SHAs as we do here.

3 Finite Automata on Words, Schemas, and Queries

In this section, we discuss how to use NFAs for defining schemas and queries on words.

Let \mathbb{N} be the set of natural numbers including 0. The set of words over a finite alphabet Σ is $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$. A word $(a_1, \dots, a_n) \in \Sigma^n$ is written as $a_1 \dots a_n$. We denote by ε the empty word, i.e., the unique element of Σ^0 and by $w_1 \cdot w_2 \in \Sigma^*$ the concatenation of two words $w_1, w_2 \in \Sigma^*$. For example, if $\Sigma = \{a, b\}$ then $aa \cdot bb = aabb = a \cdot a \cdot b \cdot b$.

Definition 1. A NFA is a tuple $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ such that \mathcal{Q} is a finite set of states, the alphabet Σ is a finite set, $I, F \subseteq \mathcal{Q}$ are subsets of initial and final states, and $\Delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is the set of transition rules.

The size of a NFA is $|A| = |\mathcal{Q}| + |\Delta|$. A transition rule $(q, a, q') \in \Delta$ is denoted by $q \xrightarrow{a} q' \in \Delta$. We define transitions $q \xrightarrow{w} q'$ wrt Δ for arbitrary words $w \in \Sigma^*$ by the following inference rules:

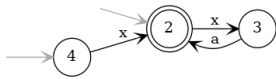
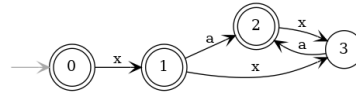
$$\frac{q \in \mathcal{Q}}{q \xrightarrow{\varepsilon} q \text{ wrt } \Delta} \quad \frac{q \xrightarrow{a} q' \in \Delta}{q \xrightarrow{a} q' \text{ wrt } \Delta} \quad \frac{q_0 \xrightarrow{w_1} q_1 \text{ wrt } \Delta \quad q_1 \xrightarrow{w_2} q_2 \text{ wrt } \Delta}{q_0 \xrightarrow{w_1 \cdot w_2} q_2 \text{ wrt } \Delta}$$

The language of words recognized by a NFA then is $\mathcal{L}(A) = \{w \in \Sigma^* \mid q \xrightarrow{w} q' \text{ wrt } \Delta, q \in I, q' \in F\}$.

```

1 fun det(A) =
2   let Store = hashset.new(0) and Agenda = list.new() and Rules = hashset.new(0)
3   if initA ≠ ∅ then Agenda.add(initA)
4   while Agenda.notEmpty() do
5     let Q = Agenda.pop()
6     let h be an empty hash table with keys from Σ.
7     // the values will be nonempty hash subsets of QA
8     for q  $\xrightarrow{a}$  q' ∈ ΔA such that q ∈ Q do
9       if h.get(a) = undef then h.add(a, hashset.new(0))
10      (h.get(a)).add(q')
11     for (a, Q') in h.toList() do Rules.add(Q  $\xrightarrow{a}$  Q')
12     if not Store.member(Q') then Store.add(Q') Agenda.push(Q')
13 let initdet(A) = {Q | Q ∈ Store, Q ∩ initA ≠ ∅} and Fdet(A) = {Q | Q ∈ Store, Q ∩ FA ≠ ∅}
14 return (Σ, Store.toSet(), Rules.toSet(), initdet(A), Fdet(A))

```

Figure 2: A program computing the accessible determinization of an NFA A from Figure 1.Figure 3: The NFA A_0 for the regular expression $(x + \varepsilon).(x.a)^*$ Figure 4: The accessible determinization $\det(A_0)$ up to the renaming of states $[\{2, 4\}/0, \{2, 3\}/1, \{2\}/2, \{3\}/3]$.

A NFA A is called *deterministic* or equivalently a DFA, if it has at most one initial state, and for every pair $(q, a) \in \mathcal{Q} \times \Sigma$ there is at most one state $q' \in \mathcal{Q}^A$ such that $q \xrightarrow{a} q' \in \Delta^A$. Any NFA A can be converted into a DFA that recognizes the same language by the usual subset construction. The accessible determinization $\det(A)$ of $A = (\Sigma, \mathcal{Q}^A, \Delta^A, I^A, F^A)$ is defined by the inference rules in Figure 1. It works like the usual subset construction, except that only accessible subsets are created. It is well known that $\mathcal{L}(A) = \mathcal{L}(\det(A))$. Since only accessible subsets of states are added, we have $\mathcal{Q}^{\det(A)} \subseteq 2^{\mathcal{Q}^A}$. Therefore, the accessible determinization may even reduce the size of the automaton and often avoid the exponential worst case where $\mathcal{Q}^{\det(A)} = 2^{\mathcal{Q}^A}$.

Proposition 2 (Folklore). *The accessible determinization $\det(A)$ of a NFA A can be computed in expected amortized time $O(|\mathcal{Q}^{\det(A)}| |\Delta^A| + |A|)$.*

Proof sketch. The algorithm for accessible determinization with this complexity is somehow folklore. We sketch it nevertheless, since we need to refined it for schema-based determinization later on. A set of inference rules for accessible determinization is given in Figure 1, and an algorithm computing the fixed point of these inference rules is presented in Figure 2. It uses dynamic perfect hashing [11] for implementing hash sets, so that set inserting and membership can be done in randomized amortized time $O(1)$. The algorithm has a hash set $Store$ to save all discovered states $\mathcal{Q}^{\det(A)}$ and a hash set $Rules$ to collect all transition rules. Furthermore, it has a stack $Agenda$ to process all new states $Q \in \mathcal{Q}^{\det(A)}$. \square

As a running example, we consider the NFA A_0 for the regular expression $(x + \varepsilon).(x.a)^*$ that is drawn as a labeled digraph in Figure 3: the nodes of the graph are the states and the labeled edges represent the transitions rules. The initial states are indicated by an ingoing arrow and the final state are doubly circled. The graph of the DFA $\det(A_0)$ obtained by accessible determinization is shown in Figure 4. It is

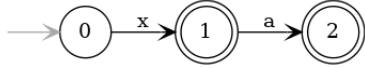


Figure 5: The schema-based cleaning of $\det(A_0)$ with schema $words-one_\Sigma^x$.

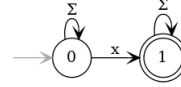
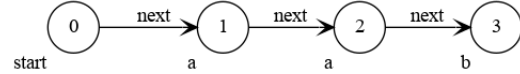


Figure 6: Schema $words-one_\Sigma^x$ with alphabet $\Sigma \uplus \{x\}$.

given up to a renaming of the states that is given in the caption. Note that only 4 out of the $2^3 = 8$ subsets are accessible, so the size increases only by a single state and two transitions rules in this example.

A *regular schema* over Σ is a DFA with the alphabet Σ . We next show how to use automata to define regular queries on words. For this, any word is seen as a labeled digraph. The labeled digraph of the word aab , for instance, is drawn to the right. The set of nodes of the graph is the set of positions of the word $pos(w) = \{0, \dots, n\}$ where n is the length of w . Position 0 is labeled by *start*, while all other positions are labeled by a single letter in Σ . A monadic query function on words with alphabet Σ is a total function \mathbf{Q} that maps some words $w \in \Sigma^*$ to a subset of position $\mathbf{Q}(w) \subseteq pos(w)$. We say that a position $\pi \in pos(w)$ is selected by \mathbf{Q} if $w \in dom(\mathbf{Q})$ and $\pi \in \mathbf{Q}(w)$.



Let us fix a single variable x . Given a position π of a word $w \in \Sigma^*$ let $w * [\pi/x]$ be the word obtained from w by inserting x after position π . We note that all words of the form $w * [\pi/x]$ contain a single occurrence of x . Such words are also called V -structures where $V = \{x\}$ (see e.g [23]).

The set of all V -structures can be defined by the schema $words-one_\Sigma^x$ over $\Sigma \uplus \{x\}$ in Figure 6. It is natural to identify any total monadic query function \mathbf{Q} with the language of V -structures $L_{\mathbf{Q}} = \{w * [\pi/x] \mid w \in \Sigma^*, \pi \in \mathbf{Q}(w)\}$. This view permits us to define a subclass of total monadic query functions by automata. A (*monadic*) *query automaton* over Σ is a NFA A with alphabet $\Sigma \uplus \{x\}$. It defines the unique total monadic query function \mathbf{Q} such that $L_{\mathbf{Q}} = \mathcal{L}(A) \cap \mathcal{L}(words-one_\Sigma^x)$. A position π of a word $w \in \Sigma^*$ is thus selected by the query \mathbf{Q} on w if and only if the V -structure $w * [\pi/x]$ is recognized by A , i.e.:

$$\pi \in \mathbf{Q}(w) \Leftrightarrow w * [\pi/x] \in \mathcal{L}(A)$$

A query function is called regular if it can be defined by some NFA. It is well-known from the work of Büchi in the sixties [7] that the same class of regular query functions can be defined equivalently by monadic second-order logic.

We note that only the words satisfying the schema $words-one_\Sigma^x$ (the V -structures) are relevant for the query function \mathbf{Q} of a query automaton A . The query automaton A_0 in Figure 3 for instance, defines the query function that selects the start position of the words ε and a and no other positions elsewhere. This is since the subset of V -structures recognized by A_0 is $x + x.a$. Note that the words ε and xxa do also belong to $\mathcal{L}(A_0)$, but are not V -structures, and thus are irrelevant for the query function \mathbf{Q} .

4 Schema-Based Cleaning

Schema-based cleaning was introduced only recently [19] in order to reduce the size of automata on nested words. The idea is to remove all rules and states from an automaton that are not used to recognize any word satisfying the schema. Schema-based cleaning can be based on the accessible states of the product of the automaton with the schema. While this product may be larger than the automaton, the schema-based cleaning will always be smaller.

$$\frac{q \in I^A \quad s \in I^S}{(q, s) \in I^{A \times S}} \quad \frac{q \in F^A \quad s \in F^S \quad (q, s) \in \mathcal{Q}^{A \times S}}{(q, s) \in F^{A \times S}}$$

$$\frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad (q_1, s_1) \in \mathcal{Q}^{A \times S}}{(q_1, s_1) \xrightarrow{a} (q_2, s_2) \in \Delta^{A \times S} \quad (q_2, s_2) \in \mathcal{Q}^{A \times S}}$$

Figure 7: Accessible product $A \times S = (\Sigma, \mathcal{Q}^{A \times S}, I^{A \times S}, F^{A \times S}, \Delta^{A \times S})$.

For illustration, the schema-based cleaning of NFA $\det(A_0)$ in Figure 4 with respect to schema $words-one_\Sigma^x$ is given in Figure 5. The only words recognized by both $\det(A_0)$ and $words-one_\Sigma^x$ are x and xa . For recognizing these two words, the automaton $\det(A_0)$ does not need states 2 and 3, so they can be removed with all their transitions rules. Thereby, the word xxa violating the schema is no more recognized after schema-based cleaning, while it was recognized by $\det(A_0)$. Furthermore, note that the state 0 needs no more to be final after schema-based cleaning. Therefore the word ε , which is recognized by the automaton but not by the schema, is no more recognized after schema-based cleaning. So schema-based cleaning may change the language of the automaton but only outside of the schema.

Interestingly, the NFA A_0 in Figure 3 is schema-clean for schema $words-one_\Sigma^x$ too, even though it is not perfect, in that it recognizes the words ε and xxa which are rejected by the schema. The reason is that for recognizing the words x and xa , which both satisfy the schema, all 3 states and all 4 transition rules of A_0 are needed. In contrast, we already noticed that the accessible determinization $\det(A_0)$ in Figure 4 is not schema-clean for schema $words-one_\Sigma^x$. This illustrates that accessible determinization does not always preserve schema-cleanliness. In other words, schema-based cleaning may have a stronger cleaning effect after determinization than before.

The schema-based cleaning of an automaton can be defined based on the accessible product of the automaton with the schema. The accessible product $A \times S$ of two NFAs A and S with alphabet Σ is defined in Figure 7. This is the usual product, except that only accessible states are admitted. Clearly, $\mathcal{L}(A \times S) = \mathcal{L}(A) \cap \mathcal{L}(S)$. Let $\Pi_A(A \times S)$ be obtained from the accessible product by projecting away the second component. The schema-based cleaning of A with respect to schema S is this projection.

Definition 3. $scl_S(A) = \Pi_A(A \times S)$.

The fact that $A \times S$ is restricted to accessible states matches our intuition that all states of $scl_S(A)$ can be used to read some word in $\mathcal{L}(A)$ that satisfies schema S . This can be proven formally under the condition that all states of $A \times S$ are also co-accessible. Clearly, $scl_S(A)$ is obtained from A by removing states, initial states, final states, and transitions rules. So it is smaller or equal in size $|scl_S(A)| \leq |A|$ and language $\mathcal{L}(scl_S(A)) \subseteq \mathcal{L}(A)$. Still, schema-based cleaning preserves the language within the schema.

Proposition 4 ([19]). $\mathcal{L}(A) \cap \mathcal{L}(S) = \mathcal{L}(scl_S(A)) \cap \mathcal{L}(S)$.

Schema-clean deterministic automata may still not be perfect, in that they may recognize some words outside the schema. This happens for DFAs if some state of is reached, both, by a word satisfying the schema and another word that does not satisfy the schema. An example for a DFA that is schema-clean but not perfect for $words-one_\Sigma^x$ is given in Figure 8. It is not perfect since it accepts the non V -structure $xaxa$. The problem is that state 1 can be reached by the words a and xa , so one cannot infer from being in state 1 whether some x was read or not. If one wants to avoid this, one can use the accessible product of the DFA with the schema instead. In the example, this yields the DFA in Figure 9 that is schema-clean and perfect for $words-one_\Sigma^x$.

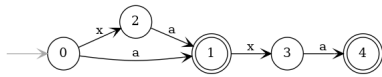


Figure 8: A DFA that is schema-clean but not perfect for $words-one_{\Sigma}^x$.

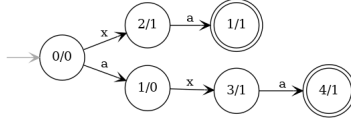


Figure 9: The accessible product with $words-one_{\Sigma}^x$ is schema-clean and perfect for $words-one_{\Sigma}^x$.

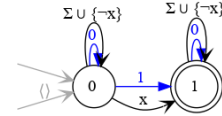


Figure 10: The dSHA one_{Σ}^x with alphabet $\Sigma \uplus \{x, \neg x\}$.

Proposition 5 (Folklore). *For any two DFAs A and S with alphabet Σ the accessible product $A \times S$ can be computed in expected amortized time $O(|\mathcal{Q}^{A \times S}| |\Sigma| + |A| + |S|)$.*

Proof. An algorithm to compute the fixed points of the inference rules for the accessible product $A \times S$ in Figure 7 can be organized such that only accessible states are considered (similarly to semi-naive datalog evaluation). This algorithm is presented in Figure 11. It dynamically generates the set of rules *Rules* by using perfect dynamic hashing [11]. Testing set membership is in time $O(1)$ and the addition of elements to the set is in expected amortized time $O(1)$. The algorithm uses a stack, *Agenda*, to memoize all new pairs $(q_1, s_1) \in \mathcal{Q}^{A \times S}$ that need to be processed, and a hash set *Store* that saves all processed states $\mathcal{Q}^{A \times S}$. We aim not to push the same pair more than once in the *Agenda*. For this, membership to the *Store* is checked before an element is pushed to the *Agenda*. For each pair popped from the stack *Agenda*, the algorithm does the following: for each letter $a \in \Sigma$ it computes the sets $Q = \{q_2 \mid q_1 \xrightarrow{a} q_2 \in \Delta^A\}$ and $R = \{s_2 \mid s_1 \xrightarrow{a} s_2 \in \Delta^S\}$ and then adds the subset of states of $Q \times R$ that were not stored in the hash set *Store* to the agenda. Since A and S are deterministic, there is at most one such pair, so the time for treating one pair on the agenda is in expected amortized time $O(|\Sigma|)$. The overall number of elements in the agenda will be $|\mathcal{Q}^{A \times S}|$. Note that Q and R can be computed in $O(1)$ after preprocessing A and S in time $O(|A| + |S|)$. Therefore, we will have a total time of the algorithm in $O(|\mathcal{Q}^{A \times S}| |\Sigma| + |A| + |S|)$. \square

Corollary 6. *For any two DFAs A and S with alphabet Σ schema-based cleaning $scl_S(A)$ can be computed in expected amortized time $O(|\mathcal{Q}^{A \times S}| |\Sigma| + |A| + |S|)$.*

Proof. By Definition 3 it is sufficient to compute the projection of the accessible product $A \times S$. By Proposition 5 the product can be computed in time $O(|\mathcal{Q}^{A \times S}| |\Sigma| + |A| + |S|)$. Its size cannot be larger than its computation time. The projection can be computed in linear time in the size of $A \times S$, so the overall time is in $O(|\mathcal{Q}^{A \times S}| |\Sigma| + |A| + |S|)$ too. \square

5 Schema-Based Determinization

Schema-based cleaning after determinization becomes impossible in practice if the automaton obtained by determinization is too big. We therefore show next how to integrate schema-based cleaning into automata determinization directly.

The schema-based determinization of A with respect to schema S extends on accessible determinization $\det(A)$. The idea is to run the schema S in parallel with $\det(A)$, in order to keep only those state $Q \in \mathcal{Q}^{\det(A)}$ that can be aligned to some state $s \in \mathcal{Q}^S$. In this case we write $Q \sim s$.

```

1 fun A × S =
2   let Store = hashset.new(0) and Agenda = list.new() and Rules = hashset.new(0)
3   if initA = {q0} and initS = {s0} then Agenda.add((q0, s0))
4   while Agenda.notEmpty() do
5     let (q1, s1) = Agenda.pop()
6     for a ∈ Σ do
7       let Q = {q2 | q1  $\xrightarrow{a}$  q2 ∈ ΔA} R = {s2 | s1  $\xrightarrow{a}$  s2 ∈ ΔS}
8       for q2 ∈ Q and s2 ∈ R do
9         Rules.add((q1, s1)  $\xrightarrow{a}$  (q2, s2))
10        if not Store.member((q2, s2))
11          then Store.add((q2, s2)) Agenda.push((q2, s2))
12 let initA×S = {(q0, s0) | (q0, s0) ∈ Store} and FA×S = {(q, s) | (q, s) ∈ Store, q ∈ FA, s ∈ FS}
13 return (Σ, Store.toSet(), Rules.toSet(), initA×S, FA×S)

```

Figure 11: An algorithm computing the accessible product of DFAs A and S .

$$\begin{array}{c}
\frac{Q \in I^{\det(A)} \quad I^S = \{s\}}{Q \in I^{\det_S(A)} \quad Q \sim s} \quad \frac{Q \sim s}{Q \in \mathcal{Q}^{\det_S(A)}} \quad \frac{Q \in F^{\det(A)} \quad s \in F^S}{Q \in F^{\det_S(A)}} \quad \frac{Q \sim s}{Q \sim s} \\
\frac{Q \xrightarrow{a} Q' \in \Delta^{\det(A)} \quad Q \sim s \quad s \xrightarrow{a} s' \in \Delta^S}{Q \xrightarrow{a} Q' \in \Delta^{\det_S(A)} \quad Q' \sim s'}
\end{array}$$

Figure 12: Schema-based determ. $\det_S(A) = (\Sigma, \mathcal{Q}^{\det_S(A)}, \Delta^{\det_S(A)}, I^{\det_S(A)}, F^{\det_S(A)})$.

The schema-determinization $\det_S(A)$ is defined in Figure 12. The automaton $\det_S(A)$ permits to go from any subset $Q \in \mathcal{Q}^{\det(A)}$ and letter $a \in \Sigma$ to the set of states $Q' = a^{\Delta^{\det(A)}}(Q)$, under the condition that there exists schema states $s, s' \in \mathcal{Q}^S$ such that $Q \sim s$ and $s \xrightarrow{a} s'$. In this case $Q' \sim s'$ is inferred.

Theorem 1 (Correctness). $\det_S(A) = scl_S(\det(A))$ for any NFA A and DFA S with the same alphabet.

The theorem states that schema-based determinization yields the same result as accessible determinization followed by schema-based cleaning.

For the correctness proof we collapse the two systems of inference rules for accessible products and projection into a single rule system. This yields the rule systems for schema-based cleaning in Figure 13.

The rules there define the automaton $\widehat{scl}_S(A)$, that we annotate with a hat, in order to distinguish it from the previous automaton $scl_S(A)$. The rules also infer judgements $(q, s) \in \mathcal{Q}^{A \widehat{\times} S}$ that we distinguish by a hat from the previous judgments $(q, s) \in \mathcal{Q}^{A \times S}$ of the accessible product. The next proposition shows that the system of collapsed inference rules indeed redefines the schema-based cleaning.

Proposition 7. For any two NFAs A and S with the same alphabet:

$$scl_S(A) = \widehat{scl}_S(A) \quad \text{and} \quad \mathcal{Q}^{A \times S} = \mathcal{Q}^{A \widehat{\times} S}$$

Proof of Correctness Theorem 1. Instantiating the system of collapsed rules for schema-based cleaning from Figure 13 with $\det(A)$ for A yields the rule system in Figure 15. We can identify the instantiated collapsed system for $\widehat{scl}_S(\det(A))$ with that for $\det_S(A)$ in Figure 12, by identifying the judgements $(Q, s) \in \mathcal{Q}^{\det(A) \widehat{\times} S}$ with judgments $Q \sim s$. After renaming the predicates, the inference rules for the corresponding judgments are the same. Hence $\widehat{scl}_S(\det(A)) = \det_S(A)$, so that Proposition 7 implies $scl_S(\det(A)) = \det_S(A)$. \square

$$\begin{array}{c}
\frac{q \in I^A \quad s \in I^S}{q \in \widehat{I}^{scl_S(A)} \quad (q, s) \in \mathcal{Q}^{A \times S}} \quad \frac{q \in F^A \quad s \in F^S \quad (q, s) \in \mathcal{Q}^{A \times S}}{q \in \widehat{F}^{scl_S(A)}} \\
\frac{(q, s) \in \mathcal{Q}^{A \times S}}{q \in \widehat{\mathcal{Q}}^{scl_S(A)}} \quad \frac{q_1 \xrightarrow{a} q_2 \in \Delta^A \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad (q_1, s_1) \in \mathcal{Q}^{A \times S}}{q_1 \xrightarrow{a} q_2 \in \widehat{\Delta}^{scl_S(A)} \quad (q_2, s_2) \in \mathcal{Q}^{A \times S}} \\
\widehat{scl}_S(A) = (\Sigma, \widehat{\mathcal{Q}}^{scl_S(A)}, \widehat{\Delta}^{scl_S(A)}, \widehat{I}^{scl_S(A)}, \widehat{F}^{scl_S(A)})
\end{array}$$

Figure 13: A collapsed rule systems for schema-based cleaning $\widehat{scl}_S(A)$.

```

1 fun detS(A, S) =
2   let Store = hashset.new(0) and Agenda = list.new() and Rules = hashset.new(0)
3   if initA ≠ ∅ and initS = {s0} then Agenda.add(initA ~ s0)
4   while Agenda.notEmpty() do
5     let (Q1 ~ s1) = Agenda.pop()
6     for a ∈ Σ do
7       let P = {Q2 | Q1  $\xrightarrow{a}$  Q2 ∈ Δdet(A)} and R = {s2 | s1  $\xrightarrow{a}$  s2 ∈ ΔS}
8       for Q2 ∈ P and s2 ∈ R do Rules.add(Q1  $\xrightarrow{a}$  Q2)
9       if not Store.member(Q2 ~ s2)
10        then Store.add(Q2 ~ s2) Agenda.push(Q2 ~ s2)
11   let initdetS(A)} = {Q | Q ~ s ∈ Store, Q ∩ initA ≠ ∅} and FdetS(A)} = {Q | Q ~ s ∈ Store, Q ∩ FA ≠ ∅}
12   return (Σ, Store.toSet(), Rules.toSet(), initdetS(A)}, FdetS(A)})

```

Figure 14: An algorithm for schema-based determinization $det_S(A)$ of an NFA A and a DFA schema S .

Proposition 8. *The schema-based determinization $det_S(A)$ for a NFA A and a DFA S over Σ can be computed in expected amortized time $O(|\mathcal{Q}^{\det(A) \times S}| |\Sigma| + |\mathcal{Q}^{\det_S(A)}| |\Delta^A| + |A| + |S|)$.*

Proof. An algorithm computing the fixed points of the inference rules of schema-based determinization from Figure 12 is given in Figure 14. It refines the algorithm computing the accessible product with on-the-fly determinization and projection.

On the stack *Agenda*, the algorithm stores alignments $Q \sim s$ such that $(Q, s) \in \mathcal{Q}^{\det(A) \times S}$ that were not considered before. Transition rules of $det_S(A)$ are collected in hash set *Rules*, using the dynamic perfect hashing aforementioned. The alignments $Q_1 \sim s_1$ popped from the agenda are processed as follows: For any letter $a \in \Sigma$, the sets $R = \{Q_2 \mid Q_1 \xrightarrow{a} Q_2 \in \Delta^{\det(A)}\}$ and $P = \{s_2 \mid s_1 \xrightarrow{a} s_2 \in \Delta^S\}$ are computed. One then pushes all new pairs $Q_2 \sim s_2$ with $Q_2 \in P$ and $s_2 \in R$ into the agenda, and adds $Q_1 \xrightarrow{a} Q_2$ to the set *Rules*. Since S and $\det(A)$ are deterministic there is at most one pair $(Q, s) \in P \times R$ for Q_1 and s_1 . So the time for treating one pair on the agenda is in $O(|\Sigma|)$ plus the time for building the needed transition rules of $\det(A)$ from Δ^A on the fly. The time for the on the fly computation of transition rules of $\det(A)$ is in time $O(|\mathcal{Q}^{\det_S(A)}| |\Delta^A|)$. The overall number of pairs on the agenda is at most $|\mathcal{Q}^{\det(A) \times S}|$ so the main while loop of the algorithm requires time in $O(|\mathcal{Q}^{\det(A) \times S}| |\Sigma|)$ apart from on the fly determinization. \square

By Proposition 2, computing $\det(A)$ requires time $O(|\mathcal{Q}^{\det(A)}| |\Delta^A| + |A|)$. Therefore, with Proposition 5, the accessible product $\det(A) \times S$ can be computed from A and S in time $O(|\mathcal{Q}^{\det(A) \times S}| |\Sigma| + |\mathcal{Q}^{\det(A)}| |\Delta^A| + |A| + |S|)$. Since $\mathcal{Q}^{\det_S(A)} \subseteq \mathcal{Q}^{\det(A)}$ the proposition shows that schema-based determinization is at most as efficient in the worst case as accessible determinization followed by schema-based cleaning. If $|\mathcal{Q}^{\det(A) \times S}| |\Sigma| < |\mathcal{Q}^{\det(A)}| |\Delta^A|$ then it is more efficient, since schema-based determinization

$$\begin{array}{c}
\frac{Q \in I^{\det(A)} \quad s \in I^S}{Q \in \widehat{I}^{\text{scls}(\det(A))} \quad (Q, s) \in \mathcal{Q}^{\det(A) \times S}} \quad \frac{Q \in F^{\det(A)} \quad s \in F^S \quad (Q, s) \in \mathcal{Q}^{\det(A) \times S}}{Q \in \widehat{F}^{\text{scls}(\det(A))}} \\
\frac{(Q, s) \in \mathcal{Q}^{\det(A) \times S}}{Q \in \widehat{\mathcal{Q}}^{\text{scls}(\det(A))}} \quad \frac{Q_1 \xrightarrow{a} Q_2 \in \Delta^{\det(A)} \quad s_1 \xrightarrow{a} s_2 \in \Delta^S \quad (Q_1, s_1) \in \mathcal{Q}^{\det(A) \times S}}{Q_1 \xrightarrow{a} Q_2 \in \widehat{\Delta}^{\text{scls}(\det(A))} \quad (Q_2, s_2) \in \mathcal{Q}^{\det(A) \times S}} \\
\widehat{\text{scls}}(\det(A)) = (\Sigma, \widehat{\mathcal{Q}}^{\text{scls}(\det(A))}, \widehat{\Delta}^{\text{scls}(\det(A))}, \widehat{I}^{\text{scls}(\det(A))}, \widehat{F}^{\text{scls}(\det(A))})
\end{array}$$

Figure 15: Instantiation of the collapsed rules for schema-based cleaning from Figure 13 with $\det(A)$.

avoids the computation of $\det(A)$ all over. Instead, it only computes the accessible product $\det(A) \times S$, which may be way smaller, since exponentially many states of $\det(A)$ may not be aligned to any state of S . Sometimes, however, the accessible product may be bigger. In this case, schema-based determinization may be more costly than pure accessible determinization, not followed by schema-based cleaning.

6 Stepwise Hedge Automata for Nested Words

We next recall SHAs [19] for defining languages of nested words, regular schemas and queries. Nested words generalize on words by adding parenthesis that must be well-nested. While containing words natively, they also generalize on unranked trees, and hedges. We restrict ourselves to nested words with a single pair of opening and closing parenthesis \langle and \rangle . Nested words over a finite alphabet Σ of internal letters have the following abstract syntax.

$$w, w' \in \mathcal{N}_\Sigma ::= \varepsilon \mid a \mid \langle w \rangle \mid w \cdot w' \quad \text{where } a \in \Sigma$$

We assume that concatenation \cdot is associative and that the empty word ε is a neutral element, that is $w \cdot (w' \cdot w'') = (w \cdot w') \cdot w''$ and $\varepsilon \cdot w = w = w \cdot \varepsilon$. Nested words can be identified with hedges, i.e., words of unranked trees and letters from Σ . Seen as a graph, the inner nodes are labeled by the tree constructor $\langle \rangle$ and the leafs by symbols in Σ or the tree constructor. For instance $\langle a \cdot \langle b \rangle \cdot \varepsilon \rangle \cdot c \cdot \langle d \cdot \langle \varepsilon \rangle \rangle$ corresponds to the hedge on the right. A nested word of type *tree* has the form $\langle h \rangle$. Note that dangling parentheses are ruled out and that labeled parentheses can be simulated by using internal letters. *XML* documents are labeled unranked trees, for instance: $\langle a \text{ name} = \text{"uff"} \rangle \langle b \text{ isgaga} \langle d \rangle \langle b \rangle \langle c \rangle \langle a \rangle$. Labeled unranked trees satisfying the *XML* data model can be represented as nested words over an alphabet that contains the *XML* node-types (*elem, attr, text, ...*), the *XML* names of the document (a, \dots, d, name), and the characters of the data values, say UTF8. For the above example, we get the nested word $\langle \text{elem} \cdot a \cdot \langle \text{attr} \cdot \text{name} \cdot \text{u} \cdot \text{f} \cdot \text{f} \rangle \langle \text{elem} \cdot b \cdot \langle \text{text} \cdot \text{i} \cdot \text{s} \cdot \text{g} \cdot \text{a} \cdot \text{g} \cdot \text{a} \rangle \langle \text{elem} \cdot d \rangle \langle \text{elem} \cdot c \rangle \rangle$

Definition 9. A SHA is a tuple $A = (\Sigma, \mathcal{Q}, \Delta, I, F)$ where $\Delta = (\Delta', @^\Delta, \diamond^\Delta)$ such that $(\Sigma, \mathcal{Q}, \Delta', I, F)$ is a NFA, $\diamond^\Delta \subseteq \mathcal{Q}$ is a set of tree initial states and $@^\Delta \subseteq \mathcal{Q}^3$ a set of apply rules.

SHAs can be drawn as graphs while extending on the graphs of NFAs. A tree initial state $q \in \diamond^\Delta$ is drawn as a node $\overset{\diamond}{\circlearrowleft} q$ with an incoming tree arrow. An applyrule $(q_1, q, q_2) \in @^\Delta$ is drawn as a blue edge $\overset{a}{\circlearrowleft} q_1 \rightarrow q_2$ that is labeled by a state $q \in \mathcal{Q}$ rather than a letter $a \in \Sigma$. It states that a nested word in state q_1 can be extended by a tree in state q and become a nested word in state q_2 .

For instance, the SHA one_Σ^x is drawn graphically in Figure 10. It accepts all nested words over $\Sigma \uplus \{x, \neg x\}$ that contain exactly one occurrence of letter x . Compared to the NFA $\text{words-one}_{\Sigma \uplus \{x\}}^x$ from

$$\frac{\diamond^{\Delta^A} \neq \emptyset}{\diamond^{\Delta^A} \in \mathcal{Q}^{\det(A)}} \quad \frac{\begin{array}{c} Q_1 \in \mathcal{Q}^{\det(A)} \quad Q_2 \in \mathcal{Q}^{\det(A)} \\ Q' = \{q' \in \mathcal{Q}^A \mid q_1 @ q_2 \rightarrow q' \in \Delta^A, q_1 \in Q_1, q_2 \in Q_2\} \neq \emptyset \\ Q_1 @ Q_2 \rightarrow Q' \in \Delta^{\det(A)} \quad Q' \in \mathcal{Q}^{\det(A)} \end{array}}{\diamond^{\Delta^A} \in \mathcal{Q}^{\det(A)}}$$

Figure 16: Accessible determinization $\det(A)$ lifted from NFAs to SHAs.

Figure 6, the SHA one_{Σ}^x contains three additional apply rules $(0,0,0)$, $(0,1,1)$, $(1,0,1) \in @^{\Delta^{one_{\Sigma}^x}}$ for reading the states assigned to subtrees. The state 0 is chosen as the single tree initial state.

Transitions for NFAs on words can be lifted to transitions for SHAs of the form $q \xrightarrow{w} q'$ wrt Δ where $w \in \mathcal{N}_{\Sigma}$ and $q, q' \in \mathcal{Q}$. For this, we add the following inference rule to the previous rules for NFAs:

$$\frac{q' \in \diamond^{\Delta} \quad q' \xrightarrow{w} q \text{ wrt } \Delta \quad (q_1, q, q_2) \in @^{\Delta}}{q_1 \xrightarrow{\langle w \rangle} q_2 \text{ wrt } \Delta}$$

The rule says that a tree $\langle w \rangle$ can transit from a state q_1 to a state q_2 if there is an apply rule $(q_1, q, q_2) \in @^{\Delta}$ so that w can transit from some tree initial state $q' \in \diamond^{\Delta}$ to q . Otherwise, the language $\mathcal{L}(A)$ of nested words accepted by a SHA A is defined as in the case of NFAs.

Definition 10. A SHA $(\Sigma, \mathcal{Q}, \Delta, I, F)$ is deterministic or equivalently a dSHA if it satisfies:

- I and \diamond^{Δ} both contain at most one element,
- a^{Δ} is a partial function from \mathcal{Q} to \mathcal{Q} for all $a \in \Sigma$, and
- $@^{\Delta}$ is a partial function from $\mathcal{Q} \times \mathcal{Q}$ to \mathcal{Q} .

Note that if A is a dSHA and $\Delta = (\Delta', @^{\Delta}, \diamond^{\Delta})$ then $A' = (\Sigma, \mathcal{Q}, \Delta', I, F)$ is a DFA. Conversely any DFA A' defines a dSHA with $@^{\Delta} = \emptyset$ and $I = \emptyset$. For instance, the SHA one_{Σ}^x in Figure 10 contains the DFA $words-one_{\Sigma \uplus \{-x\}}^x$ from Figure 6 with Σ instantiated by $\Sigma \uplus \{x\}$.

A schema for nested words over Σ is a dSHA over Σ . Note that schemas for nested words generalize over schemas of words, since dSHAs generalize on DFAs. The rules for the accessible determinization $\det(A)$ of a SHA A in Figure 16 extend on those for NFAs in Figure 1. As for words, $\det(A)$ is always deterministic, recognizes the same language as A , and contains only accessible states. The complexity of accessible determinization in case of SHA go similarly to DFA, however, the apply rules will introduce quadratic factor in the number of states.

Proposition 11. *The accessible determinization of a SHA can be computed in expected amortized time $O(|\mathcal{Q}^{\det(A)}|^2 |\Delta^A| + |A|)$.*

The notions of monadic query functions \mathbf{Q} can be lifted from words to nested words, so that it selects nodes of the graph of a nested word. For this, we have to fix one of manner possible manners to define identifiers for these nodes. The set of nodes of a nested word w is denoted by $nod(w) \subseteq \mathbb{N}$.

For indicating the selection of node $\pi \in nod(w)$, we insert the variable x into the sequence of letters following the opening parenthesis of π . If we don't want to select π , we insert the letter $\neg x$ instead. For any nested word w with alphabet Σ , the nested word $w[\pi/x]$ obtained by insertion of x or $\neg x$ at a node $\pi \in nod(w)$ has alphabet $\Sigma \uplus \{x, \neg x\}$. As before, we define $L_{\mathbf{Q}} = \{w * [\pi/x] \mid w \in \mathcal{N}_{\Sigma}, \pi \in \mathbf{Q}(w)\}$.

The notion of a query automata can now be lifted from words to nested words straightforwardly: a query automaton for nested words over Σ is a SHA A with alphabet $\Sigma \cup \{x, \neg x\}$. It defines the unique total query \mathbf{Q} such that $L_{\mathbf{Q}} = \mathcal{L}(A) \cap \mathcal{L}(one_{\Sigma}^x)$.

$$\frac{q \in \diamond^{\Delta^A} \quad s \in \diamond^{\Delta^S}}{(q, s) \in \diamond^{\Delta^{A \times S}}} \quad \frac{(q_1, s_1) \in \mathcal{Q}^{A \times S} \quad (q, s) \in \mathcal{Q}^{A \times S}}{(q_1, s_1) @ (q, s) \rightarrow (q_2, s_2) \in \Delta^{A \times S}} \quad \frac{q_1 @ q \rightarrow q_2 \in \Delta^A \quad s_1 @ s \rightarrow s_2 \in \Delta^S}{(q_2, s_2) \in \mathcal{Q}^{A \times S}}$$

Figure 17: Lifting accessible products to SHAs.

$$\frac{\diamond^{\Delta^S} = \{s\}}{\diamond^{\Delta^A} \in \diamond^{\Delta^{det_S(A)}} \quad \diamond^{\Delta^A} \sim s} \quad \frac{s_1 @ s_2 \rightarrow s' \in \Delta^S \quad Q_1 \sim s_1 \quad Q_2 \sim s_2 \quad Q_1 @ Q_2 \rightarrow Q' \in \Delta^{det(A)}}{Q_1 @ Q_2 \rightarrow Q' \in \Delta^{det_S(A)} \quad Q' \sim s'}$$

Figure 18: Extension of schema-based determinization to SHAs.

7 Schema-Based Determinization for SHAs

We can lift all previous algorithms from NFAs to SHAs while extending the system of inference rules. The additional rules concern tree initial states, that work in analogy to initial states, and also apply rules that works similarly as internal rules. The new inference rules for accessible products $A \times S$ are given in Figure 17 . As before we define $scl_S(A) = \Pi_A(A \times S)$. The rules for schema-based determinization $det_S(A)$ are extended in Figure 18. The complexity upper bound, however, now becomes quadratic even with fixed alphabet:

Proposition 12. *If A and S are dSHAs then the accessible product $A \times S$ and the schema-based cleaning $scl_S(A)$ can be computed in expected amortized time $O(|\mathcal{Q}^{A \times S}|^2 + |\mathcal{Q}^{A \times S}| |\Sigma| + |A| + |S|)$.*

Theorem 2 (Correctness). *$det_S(A) = scl_S(det(A))$ for any SHA A and dSHA S with the same alphabet.*

Proposition 13. *The schema-based determinization $det_S(A)$ of a SHA A with respect to a dSHA S can be computed in expected amortized time $O(|\mathcal{Q}^{det(A) \times S}|^2 + |\mathcal{Q}^{det(A) \times S}| |\Sigma| + |\mathcal{Q}^{det_S(A)}|^2 |\Delta^A| + |A| + |S|)$.*

The proof of Theorem 2 extends on that for NFAs (Theorem 1) in a direct manner. Proposition 13 follows the result in Proposition 8 with an additional quadratic factor in the size of states of the product $det(A) \times S$ and the states of the schema-based determinized automaton. This is always due to the apply rules of type \mathcal{Q}^3 . By Propositions 11 and 12, computing $scl_S(det(A))$ by schema-based cleaning after accessible determinization needs time in $O(|\mathcal{Q}^{det(A) \times S}|^2 + |\mathcal{Q}^{det(A) \times S}| |\Sigma| + |\mathcal{Q}^{det(A)}|^2 |\Delta^A| + |A| + |S|)$. This complexity bound is similar to that of schema-based determinization from Proposition 13. Since $\mathcal{Q}^{det_S(A)} \subseteq \mathcal{Q}^{det(A)}$, Proposition 13 shows that the worst case time complexity of schema-based determinization is never worse than for schema-based cleaning after determinization.

8 Experiments

In this section, we present an experimental evaluation of the sizes of the automata produced by the different determinization methods. For this, we consider a scalable family of SHAs that is compiled from the following scalable family of XPATH queries where n and m are natural numbers.

```
(Qn.m)  /*[self::a0 or ... or self::an]
        [descendant::*[self::b0 or ... or self::bm]]
```

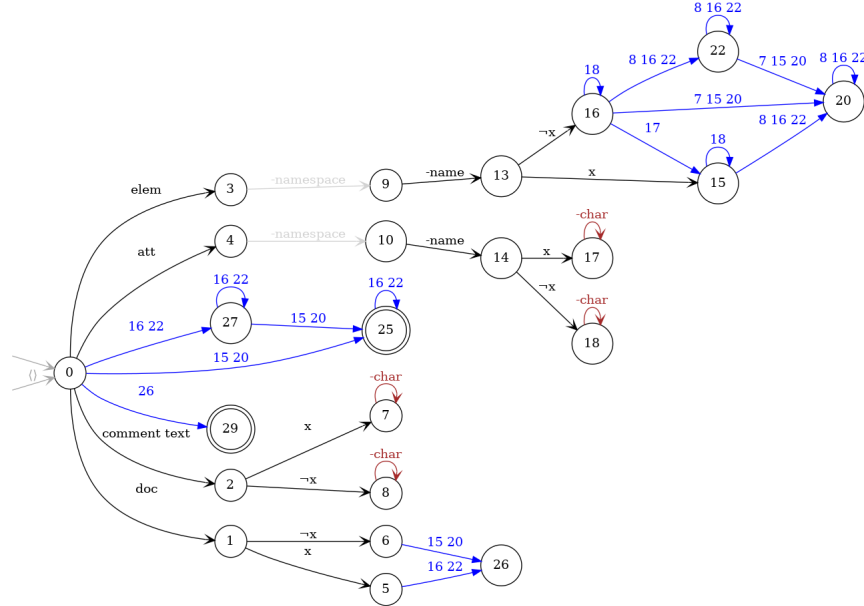


Figure 19: A schema for the intersection of *XML* data model with one^x .

Query $Q_{n,m}$ selects all elements of an *XML* document, that are named by either of a_0, \dots, a_n and have some descendant element named by either of b_1, \dots, b_m . We compile those *XPATH* queries to *SHAs* based on the compiler from [19]. As schema S , we chose the product of the *dSHA* one^x with a *dSHA* for the *XML* data model given in Figure 19. Beside the concepts presented above, this *SHA* also has typed else rules. Actually, we use a richer class of *SHAs* in the experiments, which is converted back into the class of the paper when showing the results (except for else rules and typed else rules).

The results of our experiments are summarized in Table 20. For each automaton we present two numbers, $size(\#states)$, its size and the number of its states. Unless specified otherwise, we use a timeout of 1000 seconds whenever calling some determinization algorithm. Fields of the table are left blank if an exception was raised. This happens when the determinization algorithm reached the timeout, the memory was filled, or the stack overflowed. We conducted all the experiments on a Dell laptop with the following specs: Intel® Core™ i7-10875H CPU @ 2.30 GHz, 16 cores, and 32 GB of RAM.

The first column A of Table 20 reports on the *SHAs* obtained from the queries $Q_{n,m}$, by the compiler from [19] that is written in *XSLT*. The second column $det(A)$ is obtained from *SHA* A by accessible determinization. The blank cell in column $det(A)$ for query $Q_{4,4}$ was raised by a timeout of the determinization algorithm. As one can see, this happens for all larger pairs (n,m) . Furthermore, it appears that the sizes of the automata $det(A)$ grow exponentially with $n+m$.

In the third column $det(A \times S)$, the determinization of the product is presented. It yields much smaller automata than with $det(A)$. For $Q_{4,3}$ for instance, $det(A)$ has size 53550 (2161) while $det(A \times S)$ has size 5412 (438). The computation continues successfully until $Q_{6,4}$. For the larger queries $Q_{6,5}$ and $Q_{6,6}$, our determinizer runs out of memory. The fourth column $det_S(A)$ reports on schema-based determinization. For $Q_{4,3}$ for instance we obtain 3534 (329). Here and in all given examples, both measures are always smaller for $det_S(A)$ than for $det(A \times S)$. While this may not always be the case, but both approaches yield decent results generally. The numbers for the $det_S(A)$ for $Q_{6,6}$ are marked in gray, since its computation took around one hour, so we obtain it only when ignoring the timeout. In contrast to $det(A \times S)$, however, the computation of $det_S(A)$ did not run out of memory though. The fifth

	A	det(A)	det(A × S)	det _S (A)	scl _S (det(A))	mini(det(A × S))	mini(det _S (A))
Q2.1	166 (67)	1380 (101)	540 (92)	284 (53)	284 (53)	160 (43)	73 (20)
Q2.2	199 (79)	3635 (214)	1488 (167)	830 (106)		162 (43)	75 (20)
Q2.3	232 (91)	9574 (471)	4174 (334)	2424 (227)		164 (43)	77 (20)
Q2.4	265 (103)	24813 (1052)	11502 (713)	6826 (504)		166 (43)	79 (20)
Q4.1	240 (95)	8020 (435)	710 (116)	418 (75)		164 (43)	77 (20)
Q4.2	287 (111)	20945 (968)	1944 (215)	1220 (152)		166 (43)	79 (20)
Q4.3	334 (127)	53550 (2161)	5412 (438)	3534 (329)		168 (43)	81 (20)
Q4.4	381 (143)		14794 (945)	9856 (734)		170 (43)	83 (20)
Q6.1	314 (123)	48212 (2113)	880 (140)	552 (97)		168 (43)	81 (20)
Q6.2	375 (143)		2400 (263)	1610 (198)		170 (43)	83 (20)
Q6.3	436 (163)		6650 (542)	4644 (431)		172 (43)	85 (20)
Q6.4	497 (183)		18086 (1177)	12886 (964)			87 (20)
Q6.5	558 (203)			34376 (2169)			
Q6.6	619 (223)			88666 (4862)			

Figure 20: Statistics of automata for XPATH queries: size(#states)

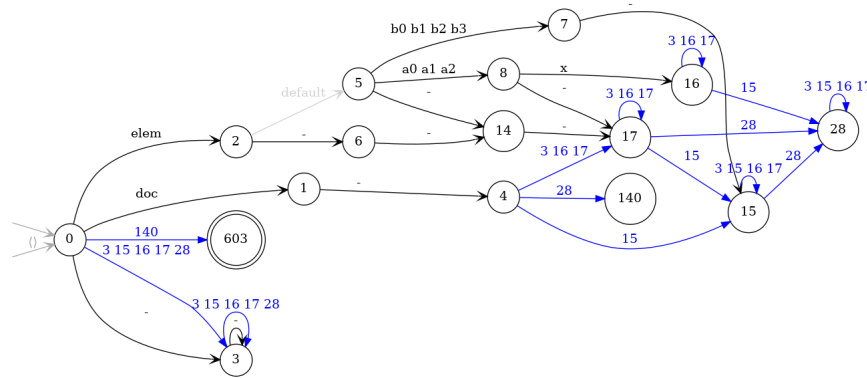


Figure 21: The automaton $mini(det_S(A))$ of the query Q3.4.

column $scl_S(det(A))$ contains the schema-based cleaning of $det(A)$. This automaton is equal to $det_S(A)$ by Correctness Theorem 2. Nevertheless, this cell is left blank in all but the smallest case $Q2.1$, since our datalog implementation of schema-based cleaning quickly runs out of memory for automata with many states. The time in seconds that for determinization in $det(A \times S)$ and $det_S(A)$ grows in dependence of the size of the output from 0.9 seconds until passing over the timeout.

In the last two columns for $mini(det(A \times S))$ and $mini(det_S(A))$ we report the sizes of the minimization of $det(A \times S)$ and $det_S(A)$. It turns out that $mini(det_S(A))$ is always smaller than $mini(det(A \times S))$, if both can be computed successfully. An example of $mini(det_S(Q3.4))$ is shown in Figure 21.

Conclusion and Future Work

We presented an algorithm for schema-based determinization for SHAs and proved that it always produces the same results as determinization followed by schema-based cleaning. We argued why schema-based determinization is often way more efficient than standard determinization, and why it is close in efficiency to the determinization of the schema-product. The statements are supported by upper complexity bounds and experimental evidence. The experimental results of the present paper are enhanced by follow up work [1]. They show that one can indeed obtain small deterministic automata based on schema-based determinization of stepwise hedge automata for all regular XPATH queries in practice. We hope that these automata are useful in the future for experiments with query answering.

References

- [1] Antonio Al Serhali & Joachim Niehren (2022): *A Benchmark Collection of Deterministic Automata for XPath Queries*. In: *XML Prague 2022*, Prague, Czech Republic. Available at <https://hal.inria.fr/hal-03527888>.
- [2] Rajeev Alur (2007): *Marrying Words and Trees*. In: *26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ACM-Press, pp. 233–242. Available at <http://dx.doi.org/10.1145/1265530.1265564>.
- [3] Rajeev Alur & P. Madhusudan (2004): *Visibly pushdown languages*. In László Babai, editor: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, ACM, pp. 202–211, doi:10.1145/1007352.1007390.
- [4] Rajeev Alur & P. Madhusudan (2009): *Adding nesting structure to words*. *Journal of the ACM* 56(3), pp. 1–43. Available at <http://doi.acm.org/10.1145/1516512.1516518>.
- [5] Mikolaj Bojanczyk & Igor Walukiewicz (2008): *Forest algebras*. In Jörg Flum, Erich Grädel & Thomas Wilke, editors: *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, *Texts in Logic and Games 2*, Amsterdam University Press, pp. 107–132.
- [6] Burchard von Braunmühl & Rutger Verbeek (1985): *Input Driven Languages are Recognized in log n Space*. In Marek Karplinski & Jan van Leeuwen, editors: *Topics in the Theory of Computation, North-Holland Mathematics Studies 102*, North-Holland, pp. 1 – 19, doi:10.1016/S0304-0208(08)73072-X.
- [7] J. Richard Büchi (1960): *Weak Second-Order Arithmetic and Finite Automata*. *Mathematical Logic Quarterly* 6(1-6), pp. 66–92, doi:10.1002/malq.19600060105. arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/malq.19600060105>.
- [8] Julien Carme, Joachim Niehren & Marc Tommasi (2004): *Querying Unranked Trees with Stepwise Tree Automata*. In Vincent van Oostrom, editor: *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings, Lecture Notes in Computer Science 3091*, Springer, pp. 105–118, doi:10.1007/978-3-540-25979-4_8.
- [9] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison & Marc Tommasi (2007): *Tree Automata Techniques and Applications*. Available online since 1997: <http://tata.gforge.inria.fr>.
- [10] Denis Debarbieux, Olivier Gauwin, Joachim Niehren, Tom Sebastian & Mohamed Zergaoui (2015): *Early nested word automata for XPath query answering on XML streams*. *Theor. Comput. Sci.* 578, pp. 100–125, doi:10.1016/j.tcs.2015.01.017.
- [11] Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert & Robert Endre Tarjan (1994): *Dynamic Perfect Hashing: Upper and Lower Bounds*. *SIAM J. Comput.* 23(4), pp. 738–761, doi:10.1137/S0097539791194094.
- [12] Massimo Franceschet: *XPathMark Performance Test*. <https://users.dimi.uniud.it/~massimo.franceschet/xpathmark/PTbench.html>. Accessed: 2020-10-25.
- [13] Olivier Gauwin, Joachim Niehren & Yves Roos (2008): *Streaming Tree Automata*. *Information Processing Letters* 109(1), pp. 13–17, doi:10.1016/j.ipl.2008.08.002.
- [14] Olivier Gauwin, Joachim Niehren & Sophie Tison (2009): *Earliest Query Answering for Deterministic Nested Word Automata*. In: *17th International Symposium on Fundamentals of Computer Theory, Lecture Notes in Computer Science 5699*, Springer Verlag, pp. 121–132, doi:10.1007/978-3-642-03409-1_12.
- [15] Anthony Lick & Schmitz Sylvain (Last visited April 13th 2022): *XPath Benchmark*. Available at <https://archive.softwareheritage.org/browse/directory/1ea68cf5bb3f9f3f2fe8c7995f1802ebadf17fb5>.
- [16] Kurt Mehlhorn (1980): *Pebbling Mountain Ranges and its Application of DCFL-Recognition*. In J. W. de Bakker & Jan van Leeuwen, editors: *Automata, Languages and Programming, 7th Colloquium*, No-

- ordweijkerhout, *The Netherlands, July 14-18, 1980, Proceedings, Lecture Notes in Computer Science* 85, Springer, pp. 422–435, doi:10.1007/3-540-10003-2_89.
- [17] Barzan Mozafari, Kai Zeng & Carlo Zaniolo (2012): *High-performance complex event processing over XML streams*. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, Ariel Fuxman, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano & Ariel Fuxman, editors: *SIGMOD Conference*, ACM, pp. 253–264, doi:10.1145/2213836.2213866.
- [18] Andreas Neumann & Helmut Seidl (1998): *Locating Matches of Tree Patterns in Forests*. In: *Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science* 1530, Springer Verlag, pp. 134–145, doi:10.1007/978-3-642-03409-1_12.
- [19] Joachim Niehren & Momar Sakho (2021): *Determinization and Minimization of Automata for Nested Words Revisited*. *Algorithms* 14(3), p. 68, doi:10.3390/a14030068.
- [20] Joachim Niehren, Momar Sakho & Antonio Al Serhali (2022): *Schema-Based Automata Determinization*. In: *Gandalf*. Available at <https://hal.inria.fr/hal-03536045>.
- [21] Alexander Okhotin & Kai Salomaa (2014): *Complexity of input-driven pushdown automata*. *SIGACT News* 45(2), pp. 47–67, doi:10.1145/2636805.2636821.
- [22] Markus L. Schmid & Nicole Schweikardt (2021): *A Purely Regular Approach to Non-Regular Core Spanners*. In Ke Yi & Zhewei Wei, editors: *24th International Conference on Database Theory (ICDT 2021), Leibniz International Proceedings in Informatics (LIPIcs)* 186, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 4:1–4:19, doi:10.4230/LIPIcs.ICDT.2021.4.
- [23] H. Straubing (1994): *Finite Automata, Formal Logic, and Circuit Complexity*. Progress in Computer Science and Applied Series, Birkhäuser, doi:10.1007/978-1-4612-0289-9.
- [24] J. W. Thatcher (1967): *Characterizing derivation trees of context-free grammars through a generalization of automata theory*. *Journal of Computer and System Science* 1, pp. 317–322, doi:10.1016/S0022-0000(67)80022-9.

Generating Tokenizers with Flat Automata

Hans de Nivelles

School of Engineering and Digital Sciences,
Nazarbayev University, Nur-Sultan City, Kazakhstan
hans.denivelle@nu.edu.kz

Dina Muktubayeva

School of Engineering and Digital Sciences,
Nazarbayev University, Nur-Sultan City, Kazakhstan
dina.muktubayeva@nu.edu.kz

We introduce flat automata for automatic generation of tokenizers. Flat automata are a simple representation of standard finite automata. Using the flat representation, automata can be easily constructed, combined and printed. Due to the use of border functions, flat automata are more compact than standard automata in the case where intervals of characters are attached to transitions, and the standard algorithms on automata are simpler. We give the standard algorithms for tokenizer construction with automata, namely construction using regular operations, determinization, and minimization. We prove their correctness. The algorithms work with intervals of characters, but are not more complicated than their counterparts on single characters. It is easy to generate C++ code from the final deterministic automaton. All procedures have been implemented in C++ and are publicly available. The implementation has been used in applications and in teaching.

1 Introduction

This paper is part of a project to obtain a programming language for the implementation of logical algorithms. Logic is special because its algorithms operate on trees that have many different forms with different subtypes. Algorithms need to distinguish the form of the tree, and take different actions dependent on this form. Intended applications of our language are parts of theorem provers, or interactive verification systems. For the interested reader, we refer to ([11]). Part of this project is to obtain a working compiler. We have looked at existing tools for the generation of the parser and the tokenizer, but none of them fulfilled our needs. In particular, there was no bottom-up parser generation tool available that supports modern C++, and existing tokenizer generation tools are not flexible enough. Existing tokenizer generators like LEX ([9]) and RE2C ([3]) generate the complete tokenizer, which makes them unsuitable for our language. Our language uses Python-style indentation, which requires that the tokenizer must generate a token when the indentation level changes. Detecting a change of indentation level is quite complicated, and it cannot be represented by regular expressions. Lack of flexibility is a general problem, for example C and C++ require that the tokenizer has access to type information, so that different tokens can be generated for identifiers that represent a type name or a template name. C++-11 allows use of >> to close two template arguments at once (for example in `std::vector<std::pair<int,int>>`). In that case, >> must be tokenized as two separate >. In order to do this correctly, the tokenizer needs to know if the parser is currently parsing a template argument.

In order to obtain the required flexibility, we created a new implementation that does not generate the complete tokenizer, but which only cuts the input in small chunks, and classifies them by type. We discuss details of our implementation in Section 8. In this paper, we concentrate on the representation of finite automata used by our implementation. We use so-called border functions to represent interval-based transitions. Instead of storing transitions of form $([\sigma_1, \sigma_2], q)$, (for characters between σ_1 and σ_2 , go to state q) we store only the points where the behavior of the transition changes, i.e. the borders, so instead we store $(\sigma_1, q), (\sigma + 2, \#)$, with # denoting 'getting stuck'. For every character, the transition

is determined by the greatest border that is not greater than the character itself. When implementing transformations on automata, border functions are much easier to deal with than intervals, because there is no need to distinguish between the beginning and the end of an interval. All that needs to be looked at, are the borders.

In addition to the use of border functions, we store the automata in an array (vector) using relative state references. This removes the need to represent automata as graphs, and the combinations that correspond to regular operators become trivial. In most cases, the automata can be just concatenated with the addition of a few ε transitions.

These two modifications result in a representation that is easy to explain and implement, and whose automata are easy to read. This is useful both for teaching and for debugging. Big automata representing complete tokenizers tend to be local, and our transformations preserve this locality.

In general, our automaton representation is somewhat more complicated than the standard representation, some of the correctness proofs become a bit more complicated, but the operations themselves are equally complicated. The extra effort in defining the automata and proving the operations correct pays off when the automata are applied: The standard representation must be further adapted in order to make it work in practice, while ours works without further adaptation. We have implemented flat automata in C++, and the implementation is available from [12].

In the next section, we will define alphabets and border functions. In Section 3, we define acceptors, which are automata that can only accept or reject. In Section 4 we explain how to obtain acceptors by means of regular operations. We do not define regular expressions as separate entities, instead we directly construct the automata. In Section 5, we define classifiers, which are obtained by pairing acceptors with token names. In Section 6 we adapt the standard determinization procedure to automata with border functions. The border functions make it possible to keep the algorithm simple. In Section 7 we adapt state minimization to our representation of automata. The algorithm can be kept simple (as simple as for single characters) because of the border functions. We use Hopcroft's algorithm ([7]) with an adaptation of a filter from [10]. In Section 8 we draw some conclusions, and sketch possibilities for future work.

2 Preliminaries

We will assume that alphabets are well-ordered sets. In the usual case where the alphabet is finite, it is sufficient that there exists a total order on the alphabet.

Definition 2.1 *An alphabet is a pair $(\Sigma, <)$, s.t. Σ is a non-empty set, and $<$ is a well-order on Σ . We define $c_{\perp} = \min(\Sigma)$. If $\{c' \in \Sigma \mid c < c'\}$ is non-empty, then we write c^{+1} for $\min\{c' \in \Sigma \mid c < c'\}$.*

As far as we know, all alphabets in use, including ASCII and Unicode ([5]) satisfy the requirements of Definition 2.1 or can be adapted in such a way that they do.

Our aim is to define automata by means of intervals, because in practice, many tokens (like for example numbers or identifiers) use intervals in their definitions. Another advantage of use of intervals is that it becomes possible to use large alphabets, like Unicode.

Dealing with intervals becomes easier if one removes the distinction between start and end of interval. This can be done by storing only the points where a new value starts, and creating a special value # denoting 'not in any interval'. For example, when defining identifiers, one may want to define a transition to some state q , for $\sigma \in \{A, \dots, Z\} \cup \{a, \dots, z\}$, because all letters usually behave the same. This can be represented as $\{(A, q), (Z + 1, \#), (a, q), (z + 1, \#)\}$. Here $A, Z + 1, a, z + 1$ are the borders where the behavior changes. In order to determine the transition for a given symbol one needs to find the largest

border that is not greater than the symbol at hand. We will call a function, that is defined in this way, a border function.

Definition 2.2 Let $(\Sigma, <)$ be an alphabet, let D be an arbitrary, non-empty set. A border function ϕ on $(\Sigma, <)$ is a partial function from Σ to D , defined for a finite subset of Σ , but at least for c_\perp . We write $\text{dom}(\phi)$ for the set of symbols for which ϕ is defined. We call the set D the range of ϕ . We will write border functions as sets of ordered pairs, whenever it is convenient.

Definition 2.3 For a given $\sigma \in \Sigma$, we first define $\sigma^\leq = \max\{\sigma' \in \text{dom}(\phi) \mid \sigma' < \sigma \text{ or } \sigma' = \sigma\}$. After that, we define $\phi^\leq(\sigma) = \phi(\sigma^\leq)$.

It can be easily checked that $\phi^\leq(\sigma)$ always exists and is uniquely defined, because ϕ is finite and has c_\perp in its domain.

Definition 2.4 Let ϕ_1 and ϕ_2 be two border functions on the same alphabet $(\Sigma, <)$. We say that ϕ_1 and ϕ_2 are equivalent if for all $\sigma \in \Sigma$, $\phi_1^\leq(\sigma) = \phi_2^\leq(\sigma)$.

We call ϕ minimal if there exists no equivalent $\phi' \subset \phi$. We define the minimization of ϕ as the \subseteq -minimal border function that is equivalent to ϕ .

Definition 2.4 uses the fact that border functions can be viewed as sets of ordered pairs. It can be easily checked that border functions can be minimized. If $\phi(\sigma_1) = \phi(\sigma_2)$, and there is no σ' with $\sigma_1 < \sigma' < \sigma_2$ in the domain of ϕ , then $\phi(\sigma_2)$ can be removed from ϕ .

If for example both $\phi(1) = \phi(3) = 4$, and 2 is not in the domain of ϕ , then removing 3 from the domain will not have effect on ϕ^\leq . Assume that $\Sigma = \{-100, -99, \dots, 99, 100\}$. Assume that $\phi(-100) = -1$, $\phi(-4) = 3$, $\phi(2) = 8$, and $\phi(6) = 4$, then $\phi^\leq(-100) = \phi^\leq(-3) = -1$, $\phi^\leq(-4) = \phi^\leq(1) = 3$, $\phi^\leq(2) = \phi^\leq(5) = 8$, and $\phi^\leq(6) = \phi^\leq(100) = 4$.

Definition 2.5 Let ϕ_1 and ϕ_2 be two border functions over the same alphabet $(\Sigma, <)$. Let D_1 be the range of ϕ_1 and let D_2 be the range of ϕ_2 . We define the product $\phi_1 \times \phi_2$ as the border function

$$\{ (\sigma, (\phi_1^\leq(\sigma), \phi_2^\leq(\sigma))) \mid \sigma \in \text{dom}(\phi_1) \cup \text{dom}(\phi_2) \}.$$

The range of $\phi_1 \times \phi_2$ is $D_1 \times D_2$.

Definition 2.6 Let ϕ be a border function over alphabet $(\Sigma, <)$. Let D be the range of ϕ . Let f be a function from D to some set D' . We define the application of f on ϕ as the minimization of

$$\{ (\sigma, f(\phi(\sigma))) \mid \sigma \in \text{dom}(\phi) \}.$$

We will write $f(\phi)$ for the application of f on Φ .

3 Acceptors

We distinguish two types of automata which we call *acceptor* and *classifier*. An acceptor can only accept or reject a word, while a classifier is able to classify words. A complete tokenizer is a classifier, while single tokens are defined by acceptors. A classifier is obtained by associating acceptors with token classes.

Although acceptors can be directly defined in code through initializers, it is inconvenient to do this, and we will construct them from regular expressions. We do not view regular expressions as independently existing objects. Instead we view regular operators as operators that work directly on acceptors. We have no data structure for regular expressions.

Acceptors are standard finite automata. We represent them in such a way that the regular operations are easy to present and to implement. In order to obtain this, we use a flat, linear representation which we will introduce shortly. In the literature, finite automata are traditionally represented by graphs whose vertices are states and whose edges are labeled with symbols. (See for example [1, 14]). This is implementable, but we believe that our representation is simpler. There is no problem of memory management, and printing automata is easy. When we print an automaton, the states are printed absolute instead of relative.

Definition 3.1 *Let $(\Sigma, <)$ be an alphabet. An acceptor \mathcal{A} over Σ is a finite sequence*

$$\mathcal{A} = (\Lambda_1, \phi_1), \dots, (\Lambda_n, \phi_n) \quad (n \geq 0),$$

where each $\Lambda_i \subseteq \mathcal{Z}$, and each ϕ_i is a border function from Σ to $\mathcal{Z} \cup \{\#\}$.

Each Λ_i denotes the set of epsilon transitions from state i , while each ϕ_i represents the set of non epsilon transitions from state i .

We call \mathcal{A} deterministic if all Λ_i are empty. We often write $\|\mathcal{A}\|$ instead of n for the size of \mathcal{A} .

We use the following conventions:

- # means that no transition is possible. Note that $\phi(\sigma) = \#$ should not be confused with ' $\phi(\sigma)$ is undefined'. Due to the use of border functions, one has to explicitly state that $\phi(\sigma)$ has no transition, because otherwise $\phi^{\leq}(\sigma)$ would 'inherit' a transition from a $\sigma' < \sigma$.
- The initial state is always 1, and the accepting state is always $n + 1$, just outside of the acceptor.
- State references in a Λ_i or ϕ_i are always relative to i . That means that i itself is represented by 0, $i + 1$ is represented by 1, while $i - 1$ is represented by -1 , etc.
- There are no transitions to states < 2 or states $> n + 1$.

Note that the last condition stipulates that the acceptor cannot return to the first state during a run. Most of the constructions for combining acceptors become simpler with this condition. Forbidding transitions to the initial state of an automaton is common in the literature, see for example [8]. An automaton with this property is usually called *committing*.

In addition, it is usually required that there is exactly one accepting state, and that there are no transitions going out of the accepting state. These conditions are automatically fulfilled by our representation. Acceptors can be non-deterministic, but all non-determinism must be inside the Λ_i , i.e. in the form of ε -transitions. All acceptors constructed by the regular operations of Section 4 have this form. If one wants to represent a general non-deterministic automaton, one has to remove transitions from the same state with overlapping intervals. For example, a state can have transitions to different states for the intervals $[a, \dots, z]$, and $[a, \dots, d]$. In this case, the original state can be split into two states connected by an ε -transition. During this process, the number of states and ε -transitions can increase, but it will not become more than the total number of borders in the step functions of the original automaton.

We will now formally define when \mathcal{A} accepts a word w .

Definition 3.2 *Let \mathcal{A} be an acceptor over alphabet Σ . We define a configuration of \mathcal{A} as a pair (z, w) , with $1 \leq z \leq \|\mathcal{A}\|$ and $w \in \Sigma^*$.*

We define the transition relation \vdash between configurations as follows:

- If $j \in \Lambda_i$ and $w \in \Sigma^*$, then $(i, w) \vdash (i + j, w)$.
- If $w \in \Sigma^*$, $\sigma \in \Sigma$, and $\phi_i^{\leq}(\sigma) = j$ with $j \neq \#$, then $(i, w) \vdash (i + j, w\sigma)$, where $\phi_i^{\leq}(\sigma)$ is the border function of state i applied on σ .

We define \vdash^i and \vdash^* between configurations as usual.

We say that \mathcal{A} accepts $w \in \Sigma^*$ if $(1, \varepsilon) \vdash^* (\|\mathcal{A}\| + 1, w)$.

We write $\mathcal{L}(\mathcal{A})$ for the language $\{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}$.

Example 3.3 We give an acceptor that accepts standard identifiers (starting with a letter, followed by zero or more letters, digits, or underscores). The first column, which numbers the states, is not part of the automaton.

$$\begin{array}{l} 1: \quad \{ \} \quad \{ (c_{\perp}, \#), (A, 1), (Z^{+1}, \#), (a, 1), (z^{+1}, \#) \} \\ 2: \quad \{ 1 \} \quad \{ (c_{\perp}, \#), (0, 0), (9^{+1}, \#), (A, 0), (Z^{+1}, \#), (-, 0), (-^{+1}, \#), (a, 0), (z^{+1}, \#) \} \end{array}$$

The initial state is 1. From state 2, there is one epsilon transition to state $2 + 1 = 3$, which is the accepting state. If $\sigma \in \{a, \dots, z\} \cup \{A, \dots, Z\} \cup \{-\}$, there is a transition from state 2 to state $2 + 0 = 2$.

Example 3.4 The following acceptor accepts the reserved word "while". The accepting state is 6.

$$\begin{array}{l} 1: \quad \{ \} \quad \{ (c_{\perp}, \#), (w, 1), (w^{+1}, \#) \} \\ 2: \quad \{ \} \quad \{ (c_{\perp}, \#), (h, 1), (h^{+1}, \#) \} \\ 3: \quad \{ \} \quad \{ (c_{\perp}, \#), (i, 1), (i^{+1}, \#) \} \\ 4: \quad \{ \} \quad \{ (c_{\perp}, \#), (l, 1), (l^{+1}, \#) \} \\ 5: \quad \{ \} \quad \{ (c_{\perp}, \#), (e, 1), (e^{+1}, \#) \} \end{array}$$

It may seem from Examples 3.3 and 3.4 that acceptors can be easily written by hand, but unfortunately that is not the case in general, because one needs to know the order of the alphabet. One must remember that upper case letters come before lower case letters in ASCII, and the relative positions of special symbols. We initially thought that it would be doable, but it turned out impossible to write non-trivial acceptors by hand. Despite this, automata are easily readable if one prints the states in transitions as absolute, and uses the following printing convention: In the transition function, pairs of form $(\sigma, \#)$ where σ is the successor of a symbol τ , are printed in the form $(\tau^{+1}, \#)$. Without this convention, for example $(A, 0), (Z^{+1}, \#)$ would be printed as $(A, 0), ([, \#)$, which is a bit hard to read.

4 Obtaining Acceptors by Regular Operations

As explained below Example 3.4, writing down acceptors directly by hand is unpractical. The standard approach in the literature and in existing systems, is to obtain automata by means of regular expressions ([1, 14]). We follow this approach, but we will not view regular expressions as independently existing objects. Rather we define a set of regular operators on automata that construct acceptors at once.

Definition 4.1 Let $(\Sigma, <)$ be an alphabet. In the current definition, we will construct border functions with range $\{\mathbf{f}, \mathbf{t}\}$. We define $\phi_{\emptyset} = \{(\sigma_{\perp}, \mathbf{f})\}$, and $\phi_{\Sigma} = \{(\sigma_{\perp}, \mathbf{t})\}$. We define

$$\phi_{\geq \sigma} = \text{if } (\sigma = \sigma_{\perp}) \text{ then } \{(\sigma_{\perp}, \mathbf{t})\} \text{ else } \{(\sigma_{\perp}, \mathbf{f}), (\sigma, \mathbf{t})\}.$$

For $\sigma \in \Sigma$, let $C_{>\sigma} = \{\sigma' \in \Sigma \mid \sigma' > \sigma\}$, the set of symbols greater than σ . We define

$$\phi_{\leq \sigma} = \text{if } (C_{>\sigma} = \emptyset) \text{ then } \{(\sigma_{\perp}, \mathbf{t})\} \text{ else } \{(\sigma_{\perp}, \mathbf{t}), (\min(C_{>\sigma}), \mathbf{f})\}$$

We define $\phi_1 \cap \phi_2 = I(\phi_1 \times \phi_2)$, with $I((d_1, d_2)) = \text{if } (d_1 = \mathbf{t} \text{ and } d_2 = \mathbf{t}) \text{ then } \mathbf{t} \text{ else } \mathbf{f}$, and we define $\neg\phi = N(\phi)$, with $N(d) = \text{if } (d = \mathbf{t}) \text{ then } \mathbf{f} \text{ else } \mathbf{t}$. Other Boolean combinations, like $\phi_1 \cup \phi_2$, and $\phi_1 \setminus \phi_2$ can be defined analogously.

Definition 4.2 We define the following ways of constructing acceptors over Σ :

- The acceptor \mathcal{A}_ε , which accepts exactly the empty word, is defined as $()$.
- Let $f_\#$ be the function defined from $f_\#(\mathbf{f}) = \#$, and $f_\#(\mathbf{t}) = 1$. Then, if ϕ is a border function with range $\{\mathbf{f}, \mathbf{t}\}$, we define $\mathcal{A}[\phi]$ as the acceptor $((\{\}, f_\#(\phi))$. (We are using Definition 2.6.)

$\mathcal{A}[\phi]$ accepts exactly the symbols (as words) for which $\phi \leq$ returns \mathbf{t} . Using $\mathcal{A}[\phi]$, it is easy to construct acceptors for Boolean combinations of intervals. For example, an acceptor that accepts exactly letters can be defined as $\mathcal{A}[(\phi_{\geq a} \cap \phi_{\leq z}) \cup (\phi_{\geq A} \cap \phi_{\leq Z})]$. An acceptor that accepts all letters except X can be defined as $\mathcal{A}[\phi_\Sigma \cap \neg(\phi_{\geq X} \cap \phi_{\leq X})]$. The acceptor that accepts nothing can be defined as $\mathcal{A}_\emptyset = \mathcal{A}[\phi_\emptyset]$.

Definition 4.3 Let $\mathcal{A} = (\Lambda_1, \Phi_1), \dots, (\Lambda_n, \Phi_n)$ and $\mathcal{A}' = (\Lambda'_1, \Phi'_1), \dots, (\Lambda'_{n'}, \Phi'_{n'})$ be acceptors. We define the concatenation $\mathcal{A} \circ \mathcal{A}'$ as $(\Lambda_1, \Phi_1), \dots, (\Lambda_n, \Phi_n), (\Lambda'_1, \Phi'_1), \dots, (\Lambda'_{n'}, \Phi'_{n'})$.

Operation \circ simply concatenates acceptors.

Theorem 4.4 Let \mathcal{A}_1 and \mathcal{A}_2 be acceptors. $\mathcal{L}(\mathcal{A}_1 \circ \mathcal{A}_2) = \{w_1 w_2 \mid w_1 \in \mathcal{L}(\mathcal{A}_1) \text{ and } w_2 \in \mathcal{L}(\mathcal{A}_2)\}$.

Proof

Throughout the proof, we define $n_1 = \|\mathcal{A}_1\|$ and $n_2 = \|\mathcal{A}_2\|$.

Let $w \in \mathcal{L}(\mathcal{A}_1 \circ \mathcal{A}_2)$. By definition, $(1, \varepsilon) \vdash^* (n_1 + n_2 + 1, w)$. There exists at least one prefix w' of w , s.t. $(1, \varepsilon) \vdash^* (n', w') \vdash^* (n_1 + n_2 + 1, w)$ having $n' > n_1$ because w itself satisfies this condition. Let w_1 be the smallest such prefix. By the last condition of Definition 3.1, n' must be equal to $n_1 + 1$, hence $w_1 \in \mathcal{L}(\mathcal{A}_1)$. Let w_2 be the rest of w , so we have $w = w_1 w_2$. Because $(n_1 + 1, w_1) \vdash^* (n_1 + n_2 + 1, w)$, it follows that $(n_1 + 1, \varepsilon) \vdash^* (n_1 + n_2 + 1, w_2)$. Note that this sequence still uses $\mathcal{A}_1 \circ \mathcal{A}_2$. Since \mathcal{A}_2 has no transitions to states < 2 , and all transitions originate from \mathcal{A}_2 , the configurations (n'', w'') in the sequence $(n_1 + 1, \varepsilon) \vdash^* (n_1 + n_2 + 1, w_2)$ must have $n'' \geq n_1 + 1$. Since transitions are relative, we have $(1, \varepsilon) \vdash^* (n_2 + 1, w_2)$ in \mathcal{A}_2 .

Now assume that $w_1 \in \mathcal{L}(\mathcal{A}_1)$ and $w_2 \in \mathcal{L}(\mathcal{A}_2)$. We have $(1, \varepsilon) \vdash^* (n_1, w_1)$ in \mathcal{A}_1 , and $(1, \varepsilon) \vdash^* (n_2, w_2)$ in \mathcal{A}_2 . The second sequence can be easily modified into $(n_1 + 1, \varepsilon) \vdash^* (n_1 + n_2 + 1, w_2)$ in $\mathcal{A}_1 \circ \mathcal{A}_2$, which in turn can be modified into $(n_1 + 1, w_1) \vdash^* (n_1 + n_2 + 1, w_1 w_2)$ in $\mathcal{A}_1 \circ \mathcal{A}_2$.

Definition 4.5 We first define an operation that adds ε transitions to an acceptor. Let $\mathcal{A} = (\Lambda_1, \Phi_1), \dots, (\Lambda_n, \Phi_n)$ be an acceptor. We define $\mathcal{A}\{i \rightarrow^\varepsilon j\}$ as $(\Lambda_1, \Phi_1), \dots, (\Lambda_i \cup \{j - i\}, \Phi_i), \dots, (\Lambda_n, \Phi_n)$. We add $j - i$ instead of just j to Λ_i because transitions are relative.

The union $\mathcal{A}_1 \mid \mathcal{A}_2$ of \mathcal{A}_1 and \mathcal{A}_2 is defined as

$$(\mathcal{A}_1 \circ \mathcal{A}_\emptyset \circ \mathcal{A}_2)\{1 \rightarrow^\varepsilon \|\mathcal{A}_1\| + 2, \|\mathcal{A}_1\| + 1 \rightarrow^\varepsilon \|\mathcal{A}_1\| + \|\mathcal{A}_2\| + 2\}.$$

In this definition, we use \mathcal{A}_\emptyset as defined below Definition 4.2, namely $\mathcal{A}_\emptyset = (\{\}, \{(c_\perp, \#)\})$. We prove that union behaves as expected:

Theorem 4.6 For every two acceptors \mathcal{A}_1 and \mathcal{A}_2 , we have $\mathcal{L}(\mathcal{A}_1 \mid \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

Proof

As before, we use $n_1 = \|\mathcal{A}_1\|$ and $n_2 = \|\mathcal{A}_2\|$. Assume that $w \in \mathcal{L}(\mathcal{A}_1 \mid \mathcal{A}_2)$. By definition, $(1, \varepsilon) \vdash^* (n_1 + n_2 + 2, w)$. If state $n_1 + 2$ does not occur in this sequence, it must be the case that the state $n_1 + 1$ occurs in the sequence, because the accepting state is reachable only from $n_1 + 1$ or from states $\geq n_1 + 2$. This implies that $w \in \mathcal{L}(\mathcal{A}_1)$. Similarly, if state $n_1 + 2$ occurs in the sequence, then we note that $n_1 + 2$ originates from the initial state of \mathcal{A}_2 . It follows that $w \in \mathcal{L}(\mathcal{A}_2)$. As a consequence, we have $\mathcal{L}(\mathcal{A}_1 \mid \mathcal{A}_2) \subseteq \mathcal{L}(\mathcal{A}_1 \cup \mathcal{A}_2)$.

Now assume that $w \in \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. If $w \in \mathcal{L}(\mathcal{A}_1)$, we have $(1, \varepsilon) \vdash^* (n_1 + 1, w)$ in \mathcal{A}_1 . In $\mathcal{A}_1 | \mathcal{A}_2$, this sequence can be extended to $(1, \varepsilon) \vdash^* (n_1 + 1, w) \vdash (n_1 + n_2 + 2, w)$.

If $w \in \mathcal{L}(\mathcal{A}_2)$, we have $(1, \varepsilon) \vdash^* (n_1 + 1, w)$ in \mathcal{A}_2 . In $\mathcal{A}_1 | \mathcal{A}_2$, this sequence becomes $(1, \varepsilon) \vdash (n_1 + 2, \varepsilon) \vdash^* (n_1 + n_2 + 2, w)$. This implies that $\mathcal{L}(\mathcal{A}_1 \cup \mathcal{A}_2) \subseteq \mathcal{L}(\mathcal{A}_1 | \mathcal{A}_2)$.

Definition 4.7 *The Kleene star \mathcal{A}^* of \mathcal{A} is defined as*

$$(\mathcal{A}_0 \circ \mathcal{A} \circ \mathcal{A}_0) \{ 1 \xrightarrow{\varepsilon} 2, 2 \xrightarrow{\varepsilon} \|\mathcal{A}\| + 3, \|\mathcal{A}\| + 2 \xrightarrow{\varepsilon} 2 \}.$$

Theorem 4.8 *For every acceptor \mathcal{A} , the following holds:*

$$w \in \mathcal{L}(\mathcal{A}^*) \text{ iff there exist } w_1, \dots, w_k \text{ (} k \geq 0 \text{), s.t. } w = w_1 w_2 \cdots w_k \text{ and each } w_i \in \mathcal{L}(\mathcal{A}).$$

Proof

In this proof, let $n = \|\mathcal{A}\|$. First assume that $(1, \varepsilon) \vdash^* (n + 3, w)$. By separating out the visits of state 2, we can write this sequence in the following form:

$$(1, \varepsilon) \vdash (2, \varepsilon) \vdash^* (2, v_1) \vdash^* (2, v_2) \vdash^* \cdots \vdash^* (2, v_{k-1}) \vdash^* (2, v_k) \vdash^* (n + 3, w),$$

where each subsequence \vdash^* contains no visits to state 2. For simplicity, set $v_0 = \varepsilon$. Then, for i with $1 \leq i \leq k$, the word v_{i-1} is a prefix of v_i . For $1 \leq i \leq k$, define the difference w_i such that $v_{i-1} w_i = v_i$.

By construction of \mathcal{A}^* , state 2 originates from the original acceptor \mathcal{A} . Hence $(2, v_{i-1}) \vdash^* (2, v_i)$ implies that $(2, v_{i-1}) \vdash^* (2 + n, v_i) \vdash (2, v_i)$. Since the sequence $(2, v_{i-1}) \vdash^* (2 + n, v_i)$ must be completely within \mathcal{A} , it follows that $w_i \in \mathcal{L}(\mathcal{A})$. For the final sequence $(2, v_k) \vdash^* (n + 3, w)$, it can be easily checked that the only transition to $n + 3$ is an ε -transition from state 2. Hence, we have $(2, v_k) \vdash (n + 3, w)$ and $v_k = w$. Since we have $w = w_1 \cdots w_k$, this completes one direction of the proof.

For the other direction, assume we have w_1, \dots, w_k , s.t each $w_i \in \mathcal{L}(\mathcal{A})$ for some $k \geq 0$. By definition, we have $(1, \varepsilon) \vdash^* (n + 1, w_i)$ in \mathcal{A} , which implies that for every word $v' \in \Sigma^*$, we have $(1, v') \vdash^* (n + 1, v' w_i)$ in \mathcal{A} .

In \mathcal{A}^* , we have $(2, v') \vdash^* (n + 2, v' w_i)$. By combining and properly instantiating the v' , we obtain

$$(1, \varepsilon) \vdash (2, \varepsilon) \vdash^* (2, w_1) \vdash^* (2, w_1 w_2) \vdash^* \cdots \vdash^* (2, w_1 w_2 \cdots w_k) \vdash (n + 3, w_1 w_2 \cdots w_k),$$

which completes the proof.

At this point, we can define all other common regular operations. For example \mathcal{A}^+ can be defined as $\mathcal{A} \circ \mathcal{A}^*$, and $\mathcal{A}^?$ can be defined as $\mathcal{A} | \mathcal{A}_\varepsilon$. Since direct construction results in slightly smaller acceptors, we still give the following definitions:

Definition 4.9 *Let \mathcal{A} be an acceptor. We define the non-empty repetition \mathcal{A}^+ as*

$$(\mathcal{A}_0 \circ \mathcal{A} \circ \mathcal{A}_0) \{ 1 \xrightarrow{\varepsilon} 2, \|\mathcal{A}\| + 2 \xrightarrow{\varepsilon} 2, \|\mathcal{A}\| + 2 \xrightarrow{\varepsilon} \|\mathcal{A}\| + 3 \}.$$

We define the optional expression $\mathcal{A}^?$ as $\mathcal{A} \{ 1 \xrightarrow{\varepsilon} \|\mathcal{A}\| + 1 \}$.

The construction of $\mathcal{A}^?$ relies on the fact that \mathcal{A} is committing.

Theorem 4.10 *For every acceptor \mathcal{A} , the following holds:*

$$w \in \mathcal{L}(\mathcal{A}^+) \text{ iff there exist } w_1, \dots, w_k \text{ (} k \geq 1 \text{), s.t. } w = w_1 w_2 \cdots w_k \text{ and each } w_i \in \mathcal{L}(\mathcal{A}).$$

$$\mathcal{L}(\mathcal{A}^?) = \mathcal{L}(\mathcal{A}) \cup \{\varepsilon\}.$$

Instead of the automaton in example 3.3, we can now write:

$$\mathcal{A} [(\phi_{\geq a} \cap \phi_{\leq z}) \cup (\phi_{\geq A} \cap \phi_{\leq Z})] \circ \mathcal{A} [(\phi_{\geq a} \cap \phi_{\leq z}) \cup (\phi_{\geq A} \cap \phi_{\leq Z}) \cup (\phi_{\geq 0} \cap \phi_{\leq 9}) \cup (\phi_{\geq -} \cap \phi_{\leq -})]^*.$$

5 Classifiers

In order to obtain a complete tokenizer, it is not sufficient to accept or reject a given input. Instead one must classify input into different groups. We call an automata that can classify a *classifier*. Contrary to standard text books, like for example [14], we define determinization and minimization on classifiers, not on acceptors.

Definition 5.1 *Let $(\Sigma, <)$ be an alphabet. Let T be a non-empty set of token classes. A classifier over Σ into T is a non-empty, finite sequence*

$$\mathcal{C} = (\Lambda_1, \phi_1, t_1), \dots, (\Lambda_n, \phi_n, t_n) \quad (n \geq 1),$$

where each $\Lambda_i \subseteq \mathcal{L}$, each ϕ_i is a border function from Σ to $\mathcal{L} \cup \{\#\}$, and each $t_i \in T$. We will often write $\|\mathcal{C}\|$ for the size of \mathcal{C} . We call \mathcal{C} deterministic if all Λ_i are empty.

For representing transitions, we use the same conventions as for acceptors, namely that transitions are stored relative, and $\phi_i(\sigma) = \#$ means that no transition is possible. In contrast to acceptors, we allow transitions to state 1, and we forbid transitions to state $n+1$. Intuitively, a classifier is a non-deterministic automaton, which looks for the longest run possible, and classifies as t_i when it gets stuck in state i . We will make this more precise soon.

In order to obtain a classifier, we start with a trivial classifier that classifies every input as error (actually, this classifier defines what is an error), and add pairs of acceptors and token classes.

We always assume that state 1 defines the error class. This is a reasonable choice, because no classifier can classify ε as a meaningful token.

Definition 5.2 *Let T be a token class. Let $e, t \in T$ and let $\mathcal{A} = (\Lambda_1, \phi_1), \dots, (\Lambda_n, \phi_n)$ be an acceptor. We define $\mathcal{A}[e, t]$ as the classifier $(\Lambda_1, \phi_1, e), \dots, (\Lambda_n, \phi_n, e), (\{\ }, \{(\sigma_\perp, \#)\}, t)$, i.e. as the classifier that classifies words accepted by \mathcal{A} as t , and all other words as e .*

Let $e \in T$. We define $\mathcal{C}_e = (\{\ }, \{(\sigma_\perp, 0)\}, e)$, i.e. as the classifier that classifies every word as e .

For a classifier \mathcal{C} with first classification t_1 , acceptor \mathcal{A} , and $t \in T$, we define $\mathcal{C}[t : \mathcal{A}]$ as

$$\mathcal{C}\{1 \xrightarrow{\varepsilon} \|\mathcal{C}\| + 1\} \circ \mathcal{A}[t_1, t].$$

Here \circ denotes concatenation of acceptors.

The construction of $\mathcal{C}[t : \mathcal{A}]$ appends \mathcal{A} to \mathcal{C} in such a way that words accepted by \mathcal{A} will be classified as t . Since acceptors accept by falling out of the automaton, we need to add an additional state without outgoing transitions, which will classify words that are able to reach it as t . We also add an ε transition from the first state to the added acceptor. Words that cannot reach an accepting state of any of the acceptors will be classified as t_1 , because the classification of the first state is used as error classification.

Example 5.3 *Assume that we want to construct a classifier that classifies identifiers as I with the exception of ‘while’, which should be classified as W . Using the acceptors of Examples 3.3 and 3.4, we can*

construct $\mathcal{C}_E[I : \mathcal{A}_{\text{id}}, W : \mathcal{A}_{\text{while}}]$ as

1:	$\{1, 4\}$	$\{(c_{\perp}, 0)\}$	E
2:	\emptyset	$\{(c_{\perp}, \#), (A, 1), (Z^{+1}, \#), (a, 1), (z^{+1}, \#)\}$	E
3:	$\{1\}$	$\{(c_{\perp}, \#), (0, 0), (9^{+1}, \#), (A, 0), (Z^{+1}, \#),$ $(-, 0), (-^{+1}, \#), (a, 0), (z^{+1}, \#)\}$	E
4:	\emptyset	$\{(c_{\perp}, \#)\}$	I
5:	\emptyset	$\{(c_{\perp}, \#), (w, 1), (w^{+1}, \#)\}$	E
6:	\emptyset	$\{(c_{\perp}, \#), (h, 1), (h^{+1}, \#)\}$	E
7:	\emptyset	$\{(c_{\perp}, \#), (i, 1), (i^{+1}, \#)\}$	E
8:	\emptyset	$\{(c_{\perp}, \#), (l, 1), (l^{+1}, \#)\}$	E
9:	\emptyset	$\{(c_{\perp}, \#), (e, 1), (e^{+1}, \#)\}$	E
10:	\emptyset	$\{(c_{\perp}, \#)\}$	W

Without further restrictions, the classifier above can classify ‘while’ either as I or as W. In order to avoid such ambiguity, we always take the classification of the maximal (using $<$ on natural numbers) reachable state that is not an error state. In the current case, after reading ‘while’ the reachable states are 1, 3, 4 and 10. Since 10 is the maximal state and its label is not $t_1 = E$, the classifier classifies as W.

Other solutions for solving ambiguity do not work well. In particular using an order $<$ on T is unpleasant. If T is an enumeration type, it is difficult to control how T is ordered. If T is a string type, its order is determined by the lexicographic order, and it is tedious to override it.

Before we can make classification precise, we need to introduce one technical condition. By default, the first state defines the error state t_1 . If from the first state it is possible to reach a state i with $t_i \neq t_1$, we could possibly classify the word as non-error. Whenever we encounter such a situation in real, it is due to a mistake, mostly due to writing \mathcal{A}^* where \mathcal{A}^+ would have been required. Hence, we will forbid such automata.

Definition 5.4 A classifier \mathcal{C} is well-formed if it does not allow a sequence $(1, \varepsilon) \vdash^* (i, \varepsilon)$ with $t_i \neq t_1$.

The automaton in Example 5.3 is well-formed. Changing t_2 into $t_2 = I$ would make it ill-formed.

The following definition makes classification precise:

Definition 5.5 For classifiers, we define configurations as in Definition 3.2. We also define \vdash and \vdash^* in the same way.

We define classification: Classifying a word $w \in \Sigma^*$ means obtaining a maximal prefix w' of w that is not classified as error (t_1), together with the preferred classification of w' . Let \mathcal{C} be a classifier, let $w \in \Sigma^*$. Let w' be a maximal prefix of w , s.t. there exists a state i of \mathcal{C} with $(1, \varepsilon) \vdash^* (i, w')$ and $t_i \neq t_1$.

If no such state i exists, then the classification of w equals (ε, t_1) .

If such a state exists, assume that i is the largest state for which $(1, \varepsilon) \vdash^* (i, w')$ and $t_i \neq t_1$. In this case, the classification equals (w', t_i) .

6 Determinization

It is possible to run a non-deterministic classifier directly, but it is inefficient in the long run when many input words need to be classified. As with standard automata, a non-deterministic classifier can be transformed into an equivalent, deterministic classifier. The construction is almost standard (See for

example [1, 8, 14]), but there are a few differences: We perform the construction on classifiers instead of acceptors, because that is what will be used in applications, and we get generalization to character intervals for free, because of the use of border functions. The advantage of border functions is that there is no need to distinguish between starts and ends of intervals. The only points that need to be looked at are the borders. As a result the construction is only slightly more complicated than the standard approach, while at the same time working in practice without adaptation. The following definition is completely standard:

Definition 6.1 Let \mathcal{C} be a classifier. Let S be a subset of its states. We define the closure of S , written as $\text{CLOS}_{\mathcal{C}}(S)$ as the smallest set of states S' with $S \subseteq S'$, and whenever $i \in S'$ and $j \in \Lambda_i$, we have $i + j \in S'$.

As said before, during determinization one only needs to consider the borders:

Definition 6.2 Let \mathcal{C} be a classifier defined over alphabet $(\Sigma, <)$. Let S be a non-empty set of states of \mathcal{C} . We define

$$\text{BORD}_{\mathcal{C}}(S) = \{ \sigma \in \Sigma \mid \sigma \text{ is in the domain of a } \phi_i \text{ with } i \in S \}.$$

$\text{BORD}_{\mathcal{C}}(S)$ is the set of symbols where the border function of one of the states in S has a border. These are the points where 'something happens', and which have to be checked when constructing the deterministic classifier. In the classifier of Example 5.3, we have $\text{CLOS}_{\mathcal{C}}(\{1\}) = \{1, 2, 5\}$ and

$$\text{BORD}_{\mathcal{C}}(\{1, 2, 5\}) = \{c_{\perp}, A, Z^{+1}, a, w, w^{+1}, z^{+1}\}.$$

Before we describe the determinization procedure, we need a way of extracting classifications from sets of states:

Definition 6.3 Let \mathcal{C} be a classifier. Let S be a subset of its states. We define $\text{CLASS}_{\mathcal{C}}(S)$ as follows: If for all $i \in S$, one has $t_i = t_1$, then $\text{CLASS}_{\mathcal{C}}(S) = t_1$. Otherwise, let i be the maximal element in S for which $t_i \neq t_1$. We define $\text{CLASS}_{\mathcal{C}}(S) = t_i$.

In example 5.3, $\text{CLASS}_{\mathcal{C}}(\emptyset) = \text{CLASS}_{\mathcal{C}}(\{1, 2, 3, 5, 6, 7, 8, 9\}) = E$, $\text{CLASS}_{\mathcal{C}}(\{4, 6, 7\}) = I$, and $\text{CLASS}_{\mathcal{C}}(\{3, 4, 10\}) = W$.

Now we are ready to define the determinization procedure. It constructs a deterministic classifier \mathcal{C}_{det} from \mathcal{C} .

Definition 6.4 The determinization procedure maintains a map H that maps subsets of states of \mathcal{C} that we have discovered into natural numbers. It also maintains a map S_i that is the inverse of H , so we always have $S_{H(S)} = S$.

1. Start by setting $H(\text{CLOS}_{\mathcal{C}}(\{1\})) = 1$, and by setting $S_1 = \text{CLOS}_{\mathcal{C}}(\{1\})$.
2. Set $\mathcal{C}_{\text{det}} = ()$.
3. As long as $\|\mathcal{C}_{\text{det}}\| < \|H\|$, repeat the following steps:
4. Let $i = \|\mathcal{C}_{\text{det}}\| + 1$. Append $(\{\}, \{\}, \text{CLASS}_{\mathcal{C}}(S_i))$ to \mathcal{C}_{det} .
5. For every $\sigma \in \text{BORD}_{\mathcal{C}}(S_i)$, do the following:
 - Let $S' = \{s + \phi_s^{\leq}(\sigma) \mid s \in S \text{ and } \phi_s^{\leq}(\sigma) \neq \#\}$. (ϕ_s is the border function of state s .)
 - If $S' = \emptyset$, then extend ϕ_i by setting $\phi_i(\sigma) = \#$. Skip the remaining steps.
 - Set $S'' = \text{CLOS}_{\mathcal{C}}(S')$.
 - If S'' is not in the domain of H , then add $H(S'') = \|H\| + 1$ to H , and set $S_{\|H\|+1} = S''$.
 - At this point, we are sure that $H(S'')$ is defined. Extend ϕ_i by setting $\phi_i(\sigma) = H(S'')$.

As usual, H and S can be discarded when the construction of \mathcal{C}_{det} is complete. It is easily checked that \mathcal{C} is deterministic, because all its Λ_i are empty.

Theorem 6.5 *Let \mathcal{C} be a classifier that is well-formed, and \mathcal{C}_{det} be the classifier constructed from \mathcal{C} by using the determinization procedure of Definition 6.4. For every word $w \in \Sigma^*$, if \mathcal{C} classifies w as (w', t') , and \mathcal{C}_{det} classifies w as (w'', t'') , then $w' = w''$ and $t' = t''$.*

Proof

The proof is mostly standard, and we sketch only the points where it differs from the standard proof. Because \mathcal{C} is well-formed, we have $t_1 = t_{\text{det},1}$, which means that both classifiers will use the same token class as error class.

For every word $w \in \Sigma^*$, define the set $R_w = \{r \in \{1, \dots, \|\mathcal{C}\|\} \mid (1, \varepsilon) \vdash^* (w, r)\}$. These are the set of states that classifier \mathcal{C} can reach while reading w .

Also define the relation $\delta_{\text{det}}(w, i)$ as $(\varepsilon, 1) \vdash^* (w, i)$. (Classifier \mathcal{C}_{det} reaches state i while reading w .)

It can be proven by induction, that

1. if $R_w \neq \emptyset$, then $\delta_{\text{det}}(w, i)$ implies $i = H(R_w)$. If $R_w = \emptyset$, then there is no i , s.t. $\delta_{\text{det}}(w, i)$.
2. if $R_w \neq \emptyset$, then $\delta_{\text{det}}(w, i)$ implies $t_{\text{det},i} = \text{CLASS}(R_w)$.

Now we can look at the classification of an arbitrary word $w \in \Sigma^*$. If for all prefixes w' of w , we have $\text{CLASS}(R_{w'}) = t_1$, then \mathcal{C} will classify w as (\emptyset, t_1) . If for some prefix there exists an i' , s.t. $\delta_{\text{det}}(w', i')$ holds, we have $t_{\text{det},i'} = \text{CLASS}(R_{w'}) = t_1$ by (2), so that $\text{CLASS}(R_{w'}) = t_{\text{det},1}$. It follows that \mathcal{C}_{det} also classifies w as (\emptyset, t_1) .

If there exists a prefix w' of w for which $\text{CLASS}(R_{w'}) \neq t_1$, then let w' be the largest such prefix. There exists exactly one i' , s.t. $\delta_{\text{det}}(w', i')$ holds, and by (2) again, we have $t_{\text{det},i'} = \text{CLASS}(R_{w'})$, which is not equal to $t_{\text{det},1}$.

Because w' was chosen maximal, it follows that for all words $w'' \neq w'$ s.t. w' is a prefix of w'' and w'' is a prefix of w , either we have $R_{w''} = \emptyset$ or $\text{CLASS}(R_{w''}) = t_1$. In both cases, there is no i'' , s.t. $(1, \varepsilon) \vdash (i'', w'')$ and $t_{\text{det},i''}$ in classifier \mathcal{C}_{det} . In the former case, no i'' exists at all, and in the latter case, $\delta_{\text{det}}(w'', i'')$ holds, and we have $t_{\text{det},i''} = \text{CLASS}(R_{w''}) = t_1$.

As a consequence, both \mathcal{C} and \mathcal{C}_{det} will classify w as $(w', \text{CLASS}(R_{w'}))$.

7 State Minimization

It is well-known that for every regular language there exists a unique deterministic automaton with minimal number of states (See [1] Section 3.9, or [8] Section 4.4.3). The minimal automaton can be obtained in time $O(n \cdot \log(n))$ from any deterministic automaton by means of Hopcroft's algorithm ([7]).

Although it probably has minimal impact on performance, minimization has a suprising effect on the size of the classifier. It turns out that on classifiers obtained from realistic programming languages, the number of states decreases by 30/40%.

It is straightforward to adapt Hopcroft's algorithm to classifiers. We sketch the implementation below. The algorithm takes a deterministic classifier \mathcal{C} as input, and constructs the smallest (in terms of equivalence classes) partition on the states of \mathcal{C} , s.t. $i \equiv j$ implies $t_i = t_j$ and for every $\sigma \in \Sigma$, $i + \phi_i^{\leq}(\sigma) \equiv j + \phi_j^{\leq}(\sigma)$. (We are implicitly assuming that $\# \equiv \#$ and $\# \neq i$.) Once one has the partition, the automaton can be minimized by selecting one state from each partition.

Definition 7.1 *We use an array (P_1, \dots, P_p) for storing the current state partition. We have $\bigcup_{1 \leq i \leq p} P_i = \{1, \dots, \|\mathcal{C}\|\}$ and $i \neq j \Rightarrow P_i \cap P_j = \emptyset$.*

In addition to the partition (P_1, \dots, P_p) , we use an index map I that maps states to their partition, i.e. for every state i ($1 \leq i \leq \|\mathcal{C}\|$), we have $i \in P_{I(i)}$.

The initial partition is obtained from a function f with domain $\{1, \dots, \|\mathcal{C}\|\}$ and arbitrary range. States i and j are put in the same class iff $f(i) = f(j)$.

We tried two initialization strategies: The first strategy is simply taking $f(r) = t_r$, which means that two states will be equivalent if they have the same classification. The second is an adaptation of a heuristic in [10] that takes paths to possible future classifications into account. We discuss it in more detail shortly.

Due to the use of border functions instead of intervals, Hopcroft's algorithm needs only minor adaptation for classifiers in our representation. We give the algorithm:

Definition 7.2 First create an array B of back transitions. For every state i with $1 \leq i \leq \|\mathcal{C}\|$, $B(i)$ is the set of states that have a transition into i , i.e.

$$B(i) = \{j \mid 1 \leq j \leq \|\mathcal{C}\| \text{ s.t. there exists a } \sigma \in \Sigma^*, \text{ s.t. } \phi_j(\sigma) \neq \# \text{ and } j + \phi_j(\sigma) = i\}.$$

Construct the initial partition $P = (P_1, \dots, P_p)$ from the chosen initialization function f . Initialize the index array I from P . Create a stack $U = (1, \dots, p)$ of indices. The variable name U stands for unchecked.

1. As long as U is non-empty, pop an element from U , call it u , and do the following:
2. Construct $S = \bigcup_{i \in P_u} B(i)$. This is the set of states that have a transition into a state $i \in P_u$.
3. For every $\sigma \in \text{BORD}_{\mathcal{C}}(S)$ do: Construct $F_{\sigma} = \{i \in S \mid \phi_i^{\leq}(\sigma) \neq \# \text{ and } i + \phi_i^{\leq}(\sigma) \in P_u\}$. Refine P, I, U with F_{σ} .

(F_{σ} is the set of states whose σ -transition goes into a state in P_u)

The refinement operation is defined as follows: Assume that we want to refine P, I, U by a set of states F . For every P_i , s.t. $P_i \cap F \neq \emptyset$ and $P_i \not\subseteq F$, do the following:

1. Construct $N = P_i \setminus F$ and replace P_i by $P_i \cap F$.
2. If this results in $\|P_i\| < \|N\|$, then exchange N and P_i .
3. Append N to $(P_1, \dots, P_i, \dots, P_p)$, and assign $I(i) = p + 1$, for $i \in N$. Add $(p + 1)$ to U .

The intuition of refinement is the fact that if some P_i partially lies inside F and partially outside F , then P_i needs to be split.

When the final partition (P_1, \dots, P_p) has been obtained, it is trivial to construct the quotient classifier $\mathcal{Q} = \mathcal{C} / (P_1, \dots, P_p)$.

It is essential that $1 \in P_1$ because Definition 5.1 and Definition 5.5 treat t_1 as the error state. This can be easily obtained by sorting (P_1, \dots, P_p) by their minimal element before constructing the quotient classifier. An additional advantage of sorting is that it improves readability, because it preserves more of the structure of the original classifier.

Both [2] and [13] agree that U should be implemented as stack, as opposed to a queue.

Although Hopcroft's algorithm is theoretically optimal, it can be improved by a preprocessing stage. In the early stage of the algorithm, all states that classify as error will be in a single equivalence class. This equivalence class is gradually refined into smaller classes dependent on possible computations originating from these classes. Although the number of steps is limited by the number of states in the class, it may still be costly because the initial class is big.

The initial refinements can be removed by using a preprocessing stage. In [10], a filter for simple, deterministic automata is proposed that marks states with the shortest distance towards an accepting state. This can be done in linear time. In order to adapt this approach to classifiers, one has to include the accepted token in the markings.

Definition 7.3 Let \mathcal{C} be a classifier from alphabet $(\Sigma, <)$ into token set T . A reachability function ρ is a total function from $\{1, \dots, \|\mathcal{C}\|\}$ to partial functions from T to \mathcal{N} .

Intuitively, $\rho(i)(t) = n$ means that there exists a path of length n from i to a state j with $t_j = t$.

Although theoretically, the total size of ρ could be quadratic in the size of \mathcal{C} , in all cases that we encountered, all states except for the initial state, can reach only a few token classes.

Our goal is to compute the optimal reachability function and use it to initialize the first partition. This can be done with Dijkstra's algorithm.

Definition 7.4 Start by setting $\rho(i) = \{(t_i, 0)\}$, for every state i that has $t_i \neq t_1$. (Every state can reach its own classification in 0 steps.) Set $\rho(i) = \{\}$ for the remaining states (that classify as error). Create a stack $U = (1, \dots, \|\mathcal{C}\|)$ of unchecked states.

- While U is not empty, pick and remove a state from U , call it u , and do the following:
 - For every $i \in B(u)$, for every $(t, n) \in \rho(u)$ do the following: If $\rho(i)(t)$ is undefined, insert $(t, n+1)$ to $\rho(i)$. Otherwise, if $\rho(i)(t) = n'$, set $\rho(i)(t) = \min(n', n+1)$.
- If this results in a change of $\rho(i)$, then add i to U .

Using ρ to initialize the partition in Definition 7.2 works well in practice. In most cases, the first partition is also the final partition. We end the section with an example of a reachability function for a simple classifier that classifies identifiers and the reserved word 'for':

Example 7.5 Consider the following deterministic classifier that classifies identifiers (for simplicity only lower case and digits), and the reserved word 'for':

1:	\emptyset	$\{(c_{\perp}, \#), (a, 1), (f, 2), (g, 1), (z^{+1}, \#)\}$	E
2:	\emptyset	$\{(c_{\perp}, \#), (0, 2), (9^{+1}, \#), (a, 3), (z^{+1}, \#)\}$	I
3:	\emptyset	$\{(c_{\perp}, \#), (0, 1), (9^{+1}, \#), (a, 2), (o, 3), (p, 2), (z^{+1}, \#)\}$	I
4:	\emptyset	$\{(c_{\perp}, \#), (0, 0), (9^{+1}, \#), (a, 1), (z^{+1}, \#)\}$	I
5:	\emptyset	$\{(c_{\perp}, \#), (0, -1), (9^{+1}, \#), (a, 0), (z^{+1}, \#)\}$	I
6:	\emptyset	$\{(c_{\perp}, \#), (0, -2), (9^{+1}, \#), (a, -1), (r, 1), (s, -1), (z^{+1}, \#)\}$	I
7:	\emptyset	$\{(c_{\perp}, \#), (0, -3), (9^{+1}, \#), (a, -2), (z^{+1}, \#)\}$	F

This classifier was constructed by the determinization procedure. If one initializes the partition with $f(i) = t_i$, the initial partition will be $(\{1\}, \{2, 3, 4, 5, 6\}, \{7\})$. The optimal reachability function has

$$\begin{aligned}
 \rho(1) &= \{(I, 1), (F, 3)\} \\
 \rho(2) &= \rho(4) = \rho(5) = \{(I, 0)\} \\
 \rho(3) &= \{(I, 1), (F, 2)\} \\
 \rho(6) &= \{(I, 1), (F, 1)\} \\
 \rho(7) &= \{(I, 1), (F, 0)\}
 \end{aligned}$$

The minimal classifier has 5 states, so the initial partition based on ρ is already the final partition.

8 Conclusions and Future Work

We have introduced a way of representing finite automata which uses relative state references and border functions. Border functions make it possible to concisely represent interval-based transition functions. Our representation is more complicated than the standard representation in text books (like [1, 14])

and the proofs are slightly harder, but the algorithms are not, and the representation can be used in practice without further adaptation. We have implemented our representation and used it in practice. We gave a presentation about it, together with our parser generation tool, at the C⁺⁺ Now conference. The implementation is available from [12].

On the practical level, we make the threshold for using our automated tools as low as possible. In the simplest case, one compiles the library, defines a classifier in code by means of regular expressions, and calls a default function for classification. Constructing classifiers in code has the advantage that the user does not need to learn a dedicated syntax, and that construction of classifiers has full flexibility.

Our implementation does not construct a complete tokenizer. This is important, because in our experience this is the obstacle that stopped us from using an existing tokenizer generator tool. There is always something in the language that cannot be handled by an automatically generated tokenizer. Therefore, in our implementation, we automated only the classification process, and leave all remaining implementation to the user. In practice, not much additional code needs to be written. If one needs efficiency, one can create an executable classifier in C⁺⁺. Both the default classifier and the C⁺⁺ classifier can be compiled with any input source which satisfies a small set of interface requirements.

In the future, we plan to look into full Boolean operations (extend regular expressions with intersection and negation), or more advanced matching techniques, as specified by POSIX.

The final point that needs consideration is the use of compile time computation. Compile time computation was introduced in C⁺⁺-11 with the aim of allowing more general functions in declarations, primarily for the computation of the size of a fixed-size array. Since then, the restrictions on compile time computation have gradually been relaxed, and nowadays, it is possible to convert a regular expression represented as an array of characters into a table-based DFA at compile-time. This was implemented in the CTRE library ([6]). We did not try to make our implementation suitable for compile time computation, because it would result in reduced expressivity in the code that constructs the acceptors. In addition, the experiments with RE2C imply that directly coded automata are an order of magnitude faster than table-based automata ([4]).

9 Acknowledgements

This work gained from comments by Witold Charatonik and Cláudia Nalon. We thank Nazarbayev University for supporting this research through the Faculty Development Competitive Research Grant Program (FDCRGP) number 021220FD1651.

References

- [1] Alfred V. Aho, Monica S. Lam, Ravi Sethi & Jeffrey D. Ullman (2007): *Compilers (Principles, Techniques and Tools)*. Pearson, Addison Wesley.
- [2] Manuel Baclet & Claire Pagetti (2006): *Around Hopcroft's Algorithm*. In Oscar Ibarra & Hsu-Chun Yen, editors: *Implementation and Application of Automata*, LNCS, Springer Verlag, pp. 114–125, doi:10.1007/11812128_12.
- [3] Markus Boerger, Peter Bumbulis, Dan Nuffer, Ulya Trofimovich & Brian Young (2003-2021): *re2c System*. <https://re2c.org/>.
- [4] Klaus Brouwer, Wolfgang Gellerich & Erhard Ploederer (1998): *Myths and Facts about the Efficient Implementation of Finite Automata and Lexical Analysis*. In K. Koskimies, editor: *Compiler Construction (CC 1998)*, LNCS 1383, Springer, pp. 1–15, doi:10.1007/BFb0026419.

- [5] The Unicode Consortium: *Unicode*. <https://home.unicode.org/>.
- [6] Hana Dusíková (2019-): *CTRE (Compile-Time Regular Expressions) Library*. <https://compile-time.re/>.
- [7] John E. Hopcroft (1971): *An $n \cdot \log(n)$ algorithm for minimizing the states in a finite automaton*. *The theory of machines and computations* 43, pp. 189–196, doi:10.1016/B978-0-12-417750-5.50022-1.
- [8] John E. Hopcroft, Rajeev Motwani & Jeffrey D. Ullman (2006): *Introduction to Automata Theory, Languages, and Computation*, 3d edition. Pearson, Addison Wesley.
- [9] Michael E Lesk & Eric Schmidt (1975): *Lex: A lexical analyzer generator*.
- [10] Desheng Liu, Zhiping Huang, Yimeng Zhang, Xiaojun Guo & Shaojing Su (2016): *Efficient Deterministic Finite Automata Minimization Based on Backward Depth Information*. *PLOS ONE* 11(11), pp. 59–78, doi:10.1371/journal.pone.0165864.
- [11] Hans de Nivelles (2021): *A Recursive Inclusion Checker for Recursively Defined Subtypes*. *Modeling and Analysis of Information Systems* 28(4), pp. 414–433, doi:10.18255/1818-1015-2021-4-414-433. Available at <https://www.mais-journal.ru/jour/article/view/1568>.
- [12] Hans de Nivelles & Dina Muktubayeva (2021): *Tokenizer Generation*. <http://www.compiler-tools.eu/>.
- [13] Andrei Păun, Mihaela Păun & Alfonso Rodríguez-Páton (2009): *On the Hopcroft’s minimization technique for DFA and DFCA*. *theoretical computer science*, pp. 2424–2430, doi:10.1016/j.tcs.2009.02.034.
- [14] Michael Sipser (2013): *Introduction to the Theory of Computation (Third Edition)*. CENGAGE Learning.

Analyzing Robustness of Angluin’s L^* Algorithm in Presence of Noise

Igor Khmelnitsky
Université Paris-Saclay
CNRS, ENS Paris-Saclay
INRIA, LMF, France
igor.khme@gmail.com

Serge Haddad
Université Paris-Saclay
CNRS, ENS Paris-Saclay
INRIA, LMF, France
haddad@lsv.fr

Lina Ye
Université Paris-Saclay
CNRS, ENS Paris-Saclay
CentraleSupélec, LMF, France
lina.ye@centralesupelec.fr

Benoît Barbot
Université Paris-Est Créteil
France
benoit.barbot@u-pec.fr

Benedikt Bollig
Université Paris-Saclay
CNRS, ENS Paris-Saclay, LMF, France
bollig@lsv.fr

Martin Leucker
Institute for Software Engineering and
Programming Languages
Universität zu Lübeck, Germany
leucker@isp.uni-luebeck.de

Daniel Neider
Carl von Ossietzky
University of Oldenburg
Germany
daniel.neider@uol.de

Rajarshi Roy
Max Planck Institute
for Software Systems
Kaiserslautern, Germany
rajarshi@mpi-sws.org

Angluin’s L^* algorithm learns the minimal (complete) deterministic finite automaton (DFA) of a regular language using membership and equivalence queries. Its probabilistic approximately correct (PAC) version substitutes an equivalence query by a large enough set of random membership queries to get a high level confidence to the answer. Thus it can be applied to any kind of (also non-regular) device and may be viewed as an algorithm for synthesizing an automaton abstracting the behavior of the device based on observations. Here we are interested on how Angluin’s PAC learning algorithm behaves for devices which are obtained from a DFA by introducing some noise. More precisely we study whether Angluin’s algorithm reduces the noise and produces a DFA closer to the original one than the noisy device. We propose several ways to introduce the noise: (1) the noisy device inverts the classification of words w.r.t. the DFA with a small probability, (2) the noisy device modifies with a small probability the letters of the word before asking its classification w.r.t. the DFA, and (3) the noisy device combines the classification of a word w.r.t. the DFA and its classification w.r.t. a counter automaton. Our experiments were performed on several hundred DFAs.

Our main contributions, bluntly stated, consist in showing that: (1) Angluin’s algorithm behaves well whenever the noisy device is produced by a random process, (2) but poorly with a structured noise, and, that (3) almost surely randomness yields systems with non-recursively enumerable languages.

1 Introduction

Discrete-event systems and their languages. Discrete-event systems [5] form a large class of dynamic systems that, given some internal state, evolve from one state to another one due to the occurrence of an event. For instance, discrete-event systems can represent a cyber-physical process whose events are triggered by a controller or the environment, or, a business process whose events are triggered by human activities or software executions. Often, the behaviors of such systems are classified as safe (aka correct, representative, etc.) or unsafe. Since a behavior may be identified by its sequence of occurred events, this leads to the notion of a language.

Analysis versus synthesis. There are numerous formalisms to specify (languages of) discrete-event systems. From a designer's perspective, the simpler it is the better its analysis will be. So finite automata and their languages (regular languages) are good candidates for the specification. However, even when the system is specified by an automaton, its implementation may slightly differ due to several reasons (bugs, unplanned human activities, unpredictable environment, etc.). Thus, one generally checks whether the implementation conforms to the specification. However, in many contexts, the system has already been implemented and the original specification (if any) is lost, as for instance in the framework of process mining [1]. Thus, by observing and interacting with the system, one aims to recover a specification close to the system at hand but that is robust with respect to its pathologic behaviors.

Language learning. The problem of learning a language from finite samples of strings by discovering the corresponding grammar is known as grammatical inference. Its significance was initially stated in [11] and an overview of very first results can be found in [4]. As it may not always be possible to infer a grammar that exactly identifies a language, approximate language learning was introduced in [13], where a grammar is selected from a solution space whose language approximates the target language with a specified degree of accuracy. To provide a deeper insight into language learning, the problem of identifying a (minimal) deterministic finite automaton (DFA) that is consistent with a given sample has attracted substantial attention in the literature since several decades [6, 2, 12]. An understanding of regular language learning is very valuable for a generalization to other more complex classes of languages.

Angluin's L^* algorithm. Angluin's L^* algorithm learns the minimal DFA of a regular language using membership and equivalence queries. Thus, one could try to adapt it to the synthesis task described above. However, for most black box systems, it is almost impossible to implement the equivalence query. Thus, its probabilistic approximatively correct (PAC) version substitutes an equivalence query with a large enough set of random membership queries. However, one needs to define and evaluate the accuracy of such an approach. Thus, here we are interested in how PAC Angluin's algorithm behaves for devices which are obtained from a DFA by introducing some noise.

Noisy learning. Most learning algorithms in the literature assume the correctness of the training data, including the example data such as attributes as well as classification results. However, sometimes noise-free datasets are not available. [10] carried out an experimental study of the noise effects on the learning performance. The results showed that generally the classification noise had more negative impact than the attribute one, i.e., errors in the values of attributes. [3] studied how to compensate for randomly introduced noise and discovered a theorem giving a bound on the sample size that is sufficient for PAC-identification in the presence of classification noise when the concept classes are finite. Michael Kearns formalized another related learning model from statistical queries by extending Valiant's learning model [8]. One main result shows that any class of functions learnable from this statistical query model is also learnable with classification noise in Valiant's model.

Our contribution. In this paper, we study against which kinds of noise Angluin's algorithm¹ is *robust*. To the best of our knowledge, this is the very first attempt of noise analysis in the automata learning setting. More precisely, we consider the following setting (cf. Figure 1): Assume that a regular device \mathcal{A} is given, typically as a black box. Due to some noise \mathcal{N} , the system \mathcal{A} is perturbed resulting in a

¹In this work by "Angluin's algorithm" we refer to the optimized version from [9].

not necessarily regular system $\mathcal{M}_{\mathcal{N}}$. This one is consulted by the PAC version of L^* to obtain a regular system \mathcal{A}_E . The question studied in this paper is whether \mathcal{A}_E is closer to \mathcal{A} than $\mathcal{M}_{\mathcal{N}}$, or, in other words, to which extent learning via L^* is robust against the noise \mathcal{N} . To this end, we introduce three kinds of

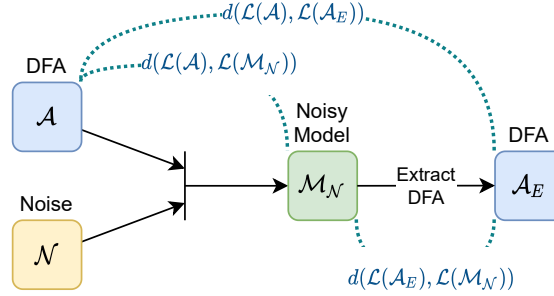


Figure 1: The experimental setup and the studied distances

noisy devices obtained from the DFA \mathcal{A} : (1) the noisy device is obtained by a random process from a given DFA by inverting the classification of words with a small probability, which corresponds to the classification noise in the classical learning setting, (2) the noisy device is obtained by a random process that, with a small probability, replaces each letter of a word by one chosen uniformly from the alphabet and then determines its classification based on the DFA, which corresponds to the attribute noise in the classical setting, and (3) the noisy DFA combines the classification of a word w.r.t. the DFA and its status w.r.t. a counter automaton. Our studies are based on the distribution over words that is used for generating words associated with membership queries and defining (and statistically measuring) the *distance* between two devices as the probability that they differ on word acceptance. We have performed experiments over several hundreds random DFA. We have pursued several goals along our experiments, expressed by the following questions:

- What is the threshold (in terms of distance) between perturbing the DFA or producing a device that is no more “similar to” the DFA?
- What is the impact of the nature of noise on the robustness of Angluin’s algorithm?
- What is the impact of the words distribution on the robustness of Angluin’s algorithm?

Due to the approximating nature of the PAC version of L^* , we had to consider the question of how to choose the accuracy of the approximate equivalence query to get a good trade-off between accuracy and efficiency. Moreover, since in most cases, Angluin’s algorithm may perform a huge number of refinement rounds before a possible termination, we considered what a “good” number of rounds to stop the algorithm avoiding underfitting and overfitting is.

We experimentally show that w.r.t. the random noise, i.e., the noise introduced with a small probability in different ways, Angluin’s algorithm behaves quite well, i.e., the learned DFA (\mathcal{A}_E) is very often closer to the original one (\mathcal{A}) than the noisy random device ($\mathcal{M}_{\mathcal{N}}$). When the noise is obtained using the counter automaton, Angluin’s algorithm is not robust. Instead, the device \mathcal{A}_E is closer to the noisy device $\mathcal{M}_{\mathcal{N}}$.

Moreover, we establish that the expectation of the length of a random word should be large enough to cover a relevant part of the set of words in order for Angluin's algorithms to be robust.

In order to understand why Angluin's algorithm is robust w.r.t. random noise we have undertaken a theoretical study establishing that almost surely the language of the noisy device ($\mathcal{M}_{\mathcal{N}}$) for case (1) and, with a further weak assumption, also for case (2) is not recursively enumerable. Considering non-recursively enumerable languages as unstructured, this means that due to the noise, the (regular) structure of \mathcal{A} vanishes. This is not the case for the counter automaton setting. Altogether, to put it bluntly: the less structure the noisy device has, the better Angluin's algorithm works.

Organization. In Section 2, we introduce the technical background required for the robustness analysis. In Section 3, we detail the goals and the settings of our analysis. In Section 4, we provide and discuss the experimental results. In Section 5, we discuss randomness versus structure. Finally in Section 6, we draw our the conclusions and identify future work.

2 Preliminaries

Here we provide the technical background required for the robustness analysis.

Languages. Let Σ be an alphabet, i.e., a nonempty finite set, whose elements are called *letters*. A *word* w over Σ is a finite sequence over Σ , whose length is denoted by $|w|$. The unique word of length 0 is called the *empty word* and denoted by λ . As usual, Σ^* is the set of all words over Σ , and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ is the set of words of positive length. A *language* (over Σ) is any set $L \subseteq \Sigma^*$. The symmetric difference of languages $L_1, L_2 \subseteq \Sigma^*$ is defined as $L_1 \Delta L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$.

Words distribution and measure of a language. A distribution D over Σ^* is defined by a mapping \Pr_D from Σ^* to $[0, 1]$ such that $\sum_{w \in \Sigma^*} \Pr_D(w) = 1$. Let L be a language. Its probabilistic measure w.r.t. D , $\Pr_D(L)$ is defined by $\Pr_D(L) = \sum_{w \in L} \Pr_D(w)$.

Our analysis requires that we are able to efficiently sample a word according to some D . Thus we only consider distributions D_μ with $\mu \in]0, 1[$, that are defined for a word $w = a_1 \dots a_n \in \Sigma^*$ by

$$\Pr_{D_\mu}(w) = \mu \left(\frac{1 - \mu}{|\Sigma|} \right)^n.$$

To sample a random word according to D_μ in practice, we start with the empty word and iteratively we flip a biased coin with probability $1 - \mu$ to add a letter (and μ to return the current word) and then uniformly select the letter in Σ .

Language distance. Given languages L_1 and L_2 , their distance w.r.t. a distribution D , $d_D(L_1, L_2)$, is defined by $d_D(L_1, L_2) = \Pr_D(L_1 \Delta L_2)$. Computing the distance between languages is in most of the cases impossible. Fortunately whenever the membership problem for L_1 and L_2 is decidable, then using Chernoff-Hoeffding bounds [7], this distance can be statistically approximated as follows. Let $\alpha, \gamma > 0$ be an error parameter and a confidence level, respectively. Let S be a set of words sampled independently according to D , called a sampling, such that $|S| \geq \frac{\log(2/\gamma)}{2\alpha^2}$. Let $dist = \frac{|S \cap (L_1 \Delta L_2)|}{|S|}$. Then, we have

$$\Pr_D(|d_D(L_1, L_2) - dist| > \alpha) < \gamma.$$

Since we will not simultaneously discuss about multiple distributions, we omit the subscript D almost everywhere.

Finite Automata. A (complete) deterministic finite automaton (DFA) over Σ is a tuple $\mathcal{A} = (Q, \sigma, q_0, F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\sigma : Q \times \Sigma \rightarrow Q$ is the transition function. The transition function is inductively extended over words by $\sigma(q, \lambda) = q$ and $\sigma(q, wa) = \sigma(\sigma(q, w), a)$. The language of \mathcal{A} is defined as $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \sigma(q_0, w) \in F\}$. A language $L \subseteq \Sigma^*$ is called *regular* if $L = \mathcal{L}(\mathcal{A})$ for some DFA \mathcal{A} .

PAC version of Angluin's L* algorithm. Given a regular language L , Angluin's L* algorithm learns the unique minimal DFA \mathcal{A} such that $\mathcal{L}(\mathcal{A}) = L$ using only membership queries 'Does w belong to L ?' and equivalence queries 'Does $\mathcal{L}(\mathcal{A}_E) = L$? and if not provide a word $w \in L \Delta \mathcal{L}(\mathcal{A}_E)$ '. An abstract version of this algorithm is depicted by Algorithm 1. The main features of this algorithm are: a data structure *Data* from which `Synthesize(Data)` returns an automaton \mathcal{A}_E and such that given a word $w \in L \Delta \mathcal{L}(\mathcal{A}_E)$, `Update(Data, w)` updates *Data*. The number of states of \mathcal{A}_E is incremented by one after each round and so the algorithm terminates after its number of states is equal to the (unknown) number of states of \mathcal{A} .

The Probably Approximately Correct (PAC) version of Angluin's L* algorithm takes as input an error parameter ε and a confidence level δ , and replaces the equivalence query by a number of membership queries ' $w \in L \Delta \mathcal{L}(\mathcal{A})$?' where the words are sampled from some distribution D unknown to the algorithm. Thus this algorithm can stop too early when all answers are negative while $L \neq \mathcal{L}(\mathcal{A})$. However due to the number of such queries which depends on the current round r (i.e., $\lceil \frac{\log(1/\delta) + (r+1)\log(2)}{\varepsilon} \rceil$) this algorithm ensures that

$$\Pr_D(d_D(L, \mathcal{L}(\mathcal{A})) > \varepsilon) < \delta.$$

A key observation is that this algorithm could be used for every language L for which the membership problem is decidable. However since L is not necessarily a regular language, the algorithm might never stop and thus our adaptation includes a parameter *maxround* that ensures termination.

3 Robustness Analysis

3.1 Principle and goals of the analysis

Principle of the analysis. Figure 1 illustrates the whole process of our analysis. First we set the qualitative and quantitative nature of the noise (\mathcal{N}). Then we generate a set of random DFA (\mathcal{A}). Combining \mathcal{A} and \mathcal{N} , one gets a noisy model $\mathcal{M}_{\mathcal{N}}$. More precisely, depending on whether the noise is random or not, $\mathcal{M}_{\mathcal{N}}$ is either generated off-line (deterministic noise) or on-line (random noise) when a membership query is asked during Angluin's L* algorithm. Finally we compare (1) the distances between \mathcal{A} and $\mathcal{M}_{\mathcal{N}}$, and (2) between \mathcal{A} and \mathcal{A}_E , the automaton returned by the algorithm. The aim of this comparison is to establish whether \mathcal{A}_E is closer to \mathcal{A} than $\mathcal{M}_{\mathcal{N}}$. In order to get a quantitative measure, we define the *information gain* as:

$$\text{Information gain} = \frac{d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{M}_{\mathcal{N}}))}{d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_E))}$$

Algorithm 1: Angluin's L^* algorithm

Input: L , a language unknown to the algorithm
Input: an integer $maxround$ ensuring termination

Angluin()
Data: an integer r , a boolean b and a data structure $Data$
Output: a DFA \mathcal{A}_E

Initialize($Data$)
 $r \leftarrow 0$
// The control of $maxround$ is unnecessary when L is regular
while $r < maxround$ **do**
 $\mathcal{A}_E \leftarrow Synthesize(Data)$
 $(b, w) \leftarrow IsEquivalent(\mathcal{A}_E)$
 if b **then return** \mathcal{A}_E
 Update($Data, w$)
 $r \leftarrow r + 1$
end
return Synthesize($Data$)

We consider a **low** information gain to be in $[0, 0.9)$, a **medium** information gain to be in $[0.9, 1.5)$, and a **high** information gain to be in $[1.5, \infty)$. To make high information gain more evident, we set its threshold value as 1.5.

In addition, we also evaluate the distance between \mathcal{A}_E and \mathcal{M}_N in order to study in which cases the algorithm learns in fact the noisy device instead of the original DFA.

Goals of the analysis.

- **Quantitative analysis.** The information gain highly depends on the ‘quantity’ of the noise, i.e., error rate. So we analyze the information gain depending of the distance between the original DFA and the noisy device and want to identify a threshold (if any) where the information gain starts to significantly increase.
- **Qualitative analysis.** Another important criterion of the information gain is the ‘nature’ of the noise. So we analyze the information gain w.r.t. the three noisy devices that we have introduced.
- **Impact of word distribution.** Finally, the robustness of the L^* algorithm with respect to word distribution is also analyzed.

In order to perform relevant experiments, one needs to tune two critical parameters of Angluin's L^* algorithm. Since the running time of the algorithm quadratically depends on the number of rounds (i.e. iterations of the loop), selecting an appropriate *maximal number of rounds* is a critical issue. We vary this maximal number of rounds and analyze how the information gain decreases w.r.t. this number. As an equivalence query is replaced with a set of membership queries whose number depends on the current round and the pair (ϵ, δ) , it is thus interesting to study (1) what is the effect of *accuracy of the approximate equivalence queries*, i.e., the values of (ϵ, δ) on the ratio of executions that reach the maximal number of rounds and (2) compare the information gain for executions that stop before reaching this maximal number and the same execution when letting it run up to this maximal number.

3.2 Noise

A *random language* $R \subseteq \Sigma^*$ is determined by a random process: for each $w \in \Sigma^*$, membership $w \in R$ is determined independently at random, *once and for all*, according to some probability $\Pr(w \in R) \in [0, 1]$. The probability $\Pr(w \in R)$ may depend on some parameters such as w itself and a given DFA.

We now describe the three kinds of noise that we analyze in this paper. Each type adds noise to a given DFA \mathcal{A} in form of a random language R . For the first two types, *noise with output* and *noise with input*, the probability $\Pr(w \in R)$ of including $w \in \Sigma^*$ in R depends on w itself, $\mathcal{L}(\mathcal{A})$, and some parameter $0 < p < 1$. The third kind of noise, *counter DFA*, is actually *deterministic*, i.e., $\Pr(w \in R) \in \{0, 1\}$ for all $w \in \Sigma^*$. In that case, the given DFA \mathcal{A} determines a unique “noisy” language. Let us be more precise:

DFA with noisy output. Given a DFA \mathcal{A} over the alphabet Σ and $0 < p < 1$, the random language $\mathcal{L}(\mathcal{A}^{\rightarrow p})$ flips the classification of words w.r.t. $\mathcal{L}(\mathcal{A})$ with probability p . More formally, for all $w \in \Sigma^*$,

$$\Pr(w \in \mathcal{L}(\mathcal{A}^{\rightarrow p})) = (1 - p)\mathbf{1}_{w \in \mathcal{L}(\mathcal{A})} + p\mathbf{1}_{w \notin \mathcal{L}(\mathcal{A})}$$

where $\mathbf{1}_C$ is 1 if condition C holds, and 0 otherwise. Observe that the expected value of the distance $d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}^{\rightarrow p}))$ is p . Moreover, in our experiments, we observe that $\left| \frac{d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}^{\rightarrow p})) - p}{p} \right| < 5 \cdot 10^{-2}$ for all the generated languages.

DFA with noisy input. Given a DFA \mathcal{A} over the alphabet Σ (with $|\Sigma| > 1$) and $0 < p < 1$, the random language $\mathcal{L}(\mathcal{A}^{\leftarrow p})$ changes every letter of the word with probability p uniformly to another letter and then returns the classification of the new word w.r.t. $\mathcal{L}(\mathcal{A})$. More formally, for $w = a_1 \dots a_n \in \Sigma^*$,

$$\Pr(w \in \mathcal{L}(\mathcal{A}^{\leftarrow p})) = \sum_{\substack{w' = b_1 \dots b_n \in \mathcal{L}(\mathcal{A}) \\ \text{s.t. } |w| = |w'|}} \prod_{1 \leq i \leq n} \left((1 - p)\mathbf{1}_{a_i = b_i} + \frac{p}{|\Sigma| - 1} \mathbf{1}_{a_i \neq b_i} \right).$$

Counter DFA. Let \mathcal{A} be a DFA over the alphabet Σ and $c : \Sigma \cup \{\lambda\} \rightarrow \mathbb{Z}$ be a function. We inductively define the function $\bar{c} : \Sigma^* \rightarrow \mathbb{Z}$ by

$$\bar{c}(\lambda) = c(\lambda) \text{ and } \bar{c}(wa) = \bar{c}(w) + c(a).$$

The counter language $\mathcal{L}(\mathcal{A}_c)$ is now given as

$$\mathcal{L}(\mathcal{A}_c) = \mathcal{L}(\mathcal{A}) \cup \{w \in \Sigma^* \mid \bar{c}(w) \leq 0\}.$$

4 Experimental Evaluation

In order to empirically evaluate our ideas, we have implemented a prototype and benchmarks in Python, using the NumPy library. They are available on GitHub². All evaluations were performed on a computer equipped by Intel i5-8250U CPU with 4 cores, 16GB of memory and Ubuntu Linux 18.03.

²https://github.com/LeaRNNify/Noisy_Learning

4.1 Generating DFAs

We now describe the settings of the experiments we made with three different types of noises. We choose $\mu = 10^{-2}$ for the parameter of the word distribution so that the average length of a random word is 99. All the statistic distances were computed using the Chernoff-Hoeffding bound [7] with $\alpha = 5 \cdot 10^{-4}$ as error parameter and $\gamma = 10^{-3}$ as confidence level.

The benchmarks were performed on DFA randomly generated using the following procedure. Let $M_q = 50$ and $M_a = 20$ be two parameters, which impose upper bounds on the number of states and of the alphabet, that could be tuned in future experiments. The DFA $\mathcal{A} = (Q, \sigma, q_0, F)$ on Σ is generated as follows:

- Uniformly choose $n_q \in [10, M_q]$ and $n_a \in [3, M_a]$;
- Set $Q = [0, n_q]$ and $\Sigma = [0, n_a]$;
- Uniformly choose $n_f \in [0, n_q - 1]$ and let $F = [0, n_f]$;
- Uniformly choose q_0 in Q ;
- For all $(q, a) \in Q \times \Sigma$, choose the target state $\sigma(q, a)$ uniformly among all states.

The choice of M_q and M_a was inspired by observing that these values often occur when modeling realistic processes like in business process management.

4.2 Tunings

Before launching our experiments, we first tune two key parameters for both efficiency and accuracy purposes: the maximal number of rounds of the algorithm and the accuracy of the approximate equivalence query. This tuning is based on experiments over the DFA with the noisy output since the expected distance between the DFA and the noisy device is known (p), thus simplifying the tuning.

Maximal number of rounds. In order to specify a maximal number of rounds that lead to the good performances of the Angluin's Algorithm, we took a DFA with noisy output $\mathcal{A}^{\rightarrow p}$ for $p \in \{0.005, 0.0025, 0.0015, 0.001\}$. We ran the learning algorithm, stopping every 20 rounds to estimate the distance between the current DFA \mathcal{A}_E to the original DFA \mathcal{A} . Figure 2 shows the evolution graphs of $d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_E))$ w.r.t. the number of rounds according to the different values of p each of them summarizing five runs on five different DFAs. The vertical axis corresponds to the distance to original DFA \mathcal{A} , and the horizontal axis corresponds to the number of rounds. The red line is the distance with $\mathcal{A}^{\rightarrow p}$, and the blue line is the distance with \mathcal{A}_E . We observe that after about 250 rounds $d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_E))$ is stabilizing except some rare peaks, which are worth further investigation. Therefore, from now on all the experiments are made with a maximum of 250 rounds. Of course this number depends on the size of \mathcal{A} but for the variable size that we have chosen (between 10 and 50 states) it seems to be a good choice.

Accuracy of the approximate equivalence query. We have generated thirty-five DFA and for each of them we generated five $\mathcal{A}^{\rightarrow p}$ with different values of p . Table 1 summarizes our results with different ε and δ for the approximate equivalence query. The rows correspond to the value of the noise p , the columns correspond to the values of ε and δ (where we always choose $\varepsilon = \delta$) and each cell shows the average information gain. Looking at this table, $\varepsilon = \delta = 0.01$ and $\varepsilon = \delta = 0.005$ seem to be optimal values. We decided to fix $\varepsilon = \delta = 0.005$ for all our experiments.

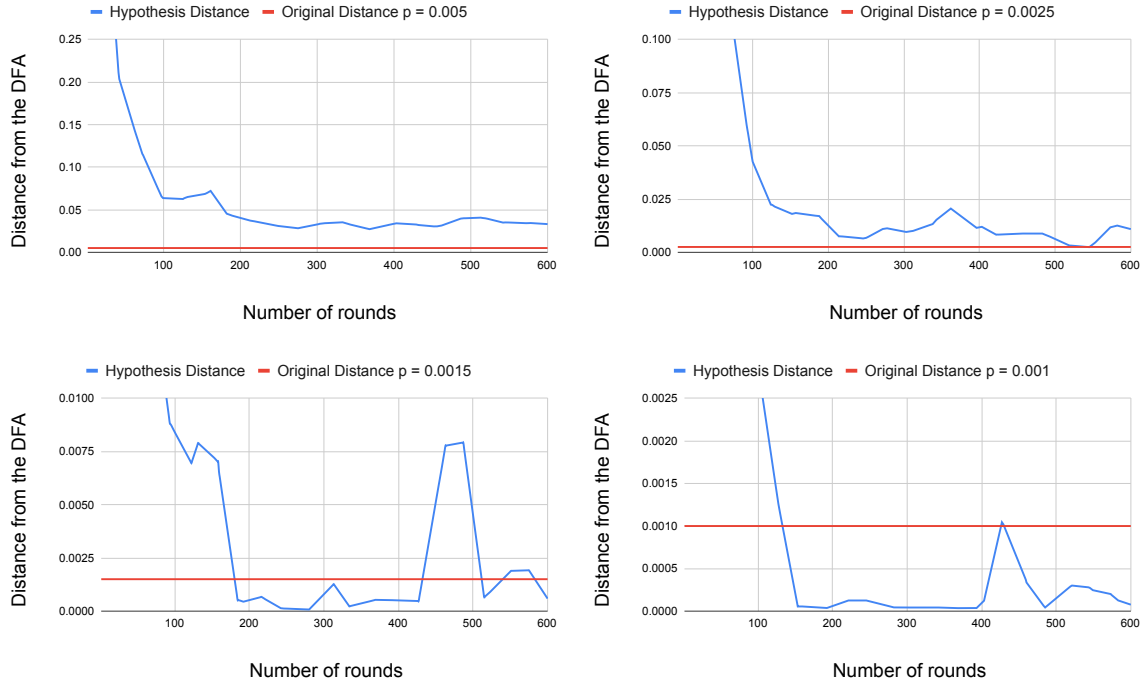


Figure 2: Number of rounds analysis

4.3 Qualitative and Quantitative analysis

For the three types of noise we have generated numerous DFA (as described shortly above), and for each DFA we have generated several noisy devices depending on the ‘quantity’ of noise. By computing the (average) information gain for all these experiments, we have been able to get conclusions about the effect of the nature and the quantity of the noise on the performance of Angluin’s algorithm.

When Angluin’s algorithm is applied to a noisy device, a corresponding random language is generated on-the-fly: once membership of a word in the target language has been determined (e.g., through a membership query), the corresponding truth value is stored and not changed anymore.

DFA with noisy output. We have generated fifty DFA, and for each such DFA \mathcal{A} , we have generated random languages with noisy output $\mathcal{L}(\mathcal{A} \rightarrow P)$ with five values for p between 0.01 and 0.001. Table 2

$\varepsilon = \delta$	0.05	0.01	0.005	0.001	0.0005
0.01	0.081	0.054	0.047	0.048	0.050
0.005	0.086	0.087	0.072	0.070	0.094
0.0025	0.867	0.292	0.591	0.321	0.748
0.0015	1.401	2.933	3.082	0.980	0.710
0.001	5.334	4.524	3.594	1.811	6.440

Table 1: Evaluation of the impact of ε and δ .

summarizes the results. Recall that the expected value of $d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}^{\rightarrow p}))$ is p . We have identified a threshold for p between 0.0015 and 0.0025: if the noise is above 0.0025 the resulting DFA \mathcal{A}_E has a bigger distance to the original one \mathcal{A} than $\mathcal{A}^{\rightarrow p}$, and smaller if the noise is under 0.0015. Moreover, once we cross the threshold the robustness of the algorithm increases very quickly. We have also included a column that represents the standard deviation of the random variable $d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_E))$ to assess that our conclusions are robust w.r.t. the probabilistic feature.

p	$d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_E))$	$d(\mathcal{L}(\mathcal{A}^{\rightarrow p}), \mathcal{L}(\mathcal{A}_E))$	gain	standard deviation
0.01	0.12625	0.13320	0.07432	0.04102
0.005	0.04420	0.04827	0.11312	0.03366
0.0025	0.00333	0.00568	0.75031	0.00523
0.0015	0.00027	0.00174	5.52999	0.00047
0.001	0.00006	0.00103	15.75817	0.00007

Table 2: Evaluation of the algorithm w.r.t. the noisy output.

DFA with noisy input. We have generated forty-five random DFA, and for each such DFA \mathcal{A} , we have generated random languages $\mathcal{L}(\mathcal{A}^{\leftarrow p})$ with $p \in \{10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}\}$. Contrary to the case of noisy output, p does not correspond to the expected value of $d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}^{\leftarrow p}))$. Thus we have evaluated this distance for every pair of the experiments and we have gathered the pairs whose distances belong to intervals that are described in the first column of Table 3. The second column of this table reports the number of pairs in the interval while the third one reports the average value of this distance for these pairs. Again we identify a threshold for $d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}^{\leftarrow p}))$ between 0.001 and 0.005 and once we cross the threshold the robustness of the algorithm increases very quickly.

Range	#	$d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}^{\leftarrow p}))$	$d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_E))$	$d(\mathcal{L}(\mathcal{A}^{\leftarrow p}), \mathcal{L}(\mathcal{A}_E))$	gain	standard deviation
[0.025, 1]	36	0.04027	0.21513	0.22658	0.18	0.05279
[0.005, 0.025]	53	0.00924	0.05416	0.06077	0.17	0.04172
[0.002, 0.005]	33	0.00378	0.01260	0.01611	0.30	0.01783
[0.001, 0.002]	11	0.00123	0.00030	0.00154	4.1	0.00058
[0.0005, 0.001]	25	0.00079	0.00002	0.00082	39.5	0.00007

Table 3: Evaluation of the algorithm w.r.t. the noisy input.

Counter DFA. We have randomly generated the counter function as follows: We have uniformly chosen $c(\lambda)$ in $[0, |\Sigma|]$. Then, for all $a \in \Sigma$, $\Pr(c(a) = -1) = \frac{1}{4}$ and for all $0 \leq i \leq 6$, $\Pr(c(a) = i) = \frac{3}{28}$.

We have generated 160 DFA. For each of them, we have generated a counter automaton (as described before). The results of our experiments are given in Table 4. Here whatever the quantity of noise the Angluin’s algorithm is unable to get closer to the original DFA. Moreover the extracted DFA \mathcal{A}_E is very often closer to the counter automaton \mathcal{A}_c than the original DFA \mathcal{A} .

Thus we conjecture that when the noise is ‘unstructured’ and the quantity is small enough such that the word noise is still meaningful, then Angluin’s algorithm is robust. On the contrary when the noise is structured, then Angluin’s algorithm ‘tries to learn’ the noisy device whatever the quantity of noise. In

Range	#	$d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_c))$	$d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_E))$	$d(\mathcal{L}(\mathcal{A}_c), \mathcal{L}(\mathcal{A}_E))$	gain	standard deviation
[0.005, 0.025]	14	0.01238	0.02586	0.02053	0.47886	0.01898
[0.002, 0.005]	57	0.00245	0.00396	0.00262	0.61765	0.00298
[0.001, 0.002]	22	0.00143	0.00209	0.00121	0.68156	0.00126
[0.0005, 0.001]	20	0.00079	0.00108	0.00064	0.72481	0.00065
[0.0001, 0.0005]	44	0.00025	0.00035	0.00021	0.71054	0.00021

Table 4: Evaluation of the algorithm w.r.t. the ‘noisy’ counter.

Section 5, we will strengthen this conjecture establishing that in some sense noise produced by random process implies unstructured noise.

4.4 Words distribution

We now discuss the impact of word distribution on the robustness of the Angluin algorithm. The parameter μ determines the average length of a random word ($\frac{1}{\mu} - 1$). Table 5 summarizes experimental results with values of μ indicated on the first row. The other rows correspond to different values of the noise p for $\mathcal{A} \rightarrow p$. The cells (at the intersection of a pair (p, μ)) contain the (average) information gain where experiments have been done over twenty-two DFA always eliminating the worst and best cases to avoid that the pathological cases perturb the average values. For values of p that matter (i.e., when the gain is greater than 1), there is clear tendency for the gain to first increase w.r.t. μ , reaching a maximum about $\mu = 0.01$ the value that we have chosen and then decrease. A possible explanation would be the following: too short words (i.e., big μ) does not help to discriminate between languages while too long words (i.e., small μ) lead to overfitting and does not reduce the noise.

$p \backslash \mu$	0.001	0.005	0.01	0.05	0.1
0.01	0.059	0.067	0.078	0.184	0.317
0.005	0.078	0.130	0.134	0.559	0.966
0.0025	0.165	0.298	0.398	1.246	0.823
0.0015	0.465	0.671	2.267	2.074	1.651
0.001	1.801	10.94	8.907	3.753	2.341

Table 5: Analysis of different distributions on Σ^*

5 Random languages versus structured languages

Recall that in the precedent section, from the experimental results, we conjecture that Angluin’s algorithm is robust, when the noise is random, i.e., unstructured, and its quantity is small enough, such as for DFA with noisy output and with noisy input. This is however not the case for structured counter DFA, for which Angluin’s algorithm learns the noisy device itself instead of the original one whatever the quantity of noise.

In this section, we want to theoretically establish that the main factor of the robustness of the Angluin’s L^* algorithm w.r.t. random noise is that almost surely randomness, in most cases, yields the perturbed

language that is unstructured. We consider a language as structured if it can be produced by some general device. Thus we identify the family of structured languages with the family of recursively enumerable languages. More precisely, we show that almost surely DFA with noisy output leads to a language that is not recursively enumerable. We then demonstrate further that with a mild condition, almost surely DFA with noisy input yields also non-recursively enumerable language. As for the counter DFA, by definition, it is clearly recursively enumerable, thus not being studied further.

The following lemma gives a simple means to establish that almost surely a random language is not recursively enumerable.

Lemma 1 *Let R be a random language over Σ . Let $(w_n)_{n \in \mathbb{N}}$ be a sequence of words of Σ^* . Let $W_n = \{w_i\}_{i < n}$ and $\rho_n = \max_{W \subseteq W_n} \Pr(R \cap W_n = W)$. Assume that $\lim_{n \rightarrow \infty} \rho_n = 0$. Then, for all countable families of languages \mathcal{F} , almost surely $R \notin \mathcal{F}$. In particular, almost surely R is not a recursively enumerable language.*

Proof Let us consider an arbitrary language L . Then, for all n , $\Pr(R = L) \leq \Pr(R \cap W_n = L \cap W_n) \leq \rho_n$. Thus, $\Pr(R = L) = 0$ and $\Pr(R \in \mathcal{F}) = \sum_{L \in \mathcal{F}} \Pr(R = L) = 0$. ■

From Lemma 1, we immediately obtain that almost surely the noisy output perturbation of any language is not recursively enumerable. The proofs of the two next theorems use the same notations as those given in Lemma 1.

Theorem 1 *Let L be a language and $0 < p < 1$. Then almost surely $L^{\rightarrow p}$ is not a recursively enumerable language.*

Proof Consider any enumeration $(w_n)_{n \in \mathbb{N}}$ of Σ^* and any $W \subseteq W_n$. The probability that $L^{\rightarrow p} \cap W_n$ is equal to W is bounded by $\max(p, 1 - p)^n$. Thus, $\rho_n \leq \max(p, 1 - p)^n$ and $\lim_{n \rightarrow \infty} \rho_n = 0$. ■

We cannot get a similar result for the noisy input perturbation. Indeed consider the language Σ^* , whatever the kind of noise brought to the input, the obtained language is still Σ^* . With the kind of input noise that we study, consider the language that accepts words of odd length (see the automaton \mathcal{A}' of Figure 3). Then the perturbed language is unchanged.

However given a DFA \mathcal{A} , we establish a mild condition on \mathcal{A} ensuring that almost surely the random language $\mathcal{L}(\mathcal{A}^{\leftarrow p})$ is not recursively enumerable. We abbreviate bottom strongly connected components (of \mathcal{A} viewed as a graph) by BSCC.

Definition 1 *Let $\mathcal{A} = (Q, F, \sigma, q_0)$ be a DFA. We call \mathcal{A} equal-length-distinguishing if there exist (possibly identical) BSCC $\mathcal{C}, \mathcal{C}'$ of \mathcal{A} , $q_1 \in \mathcal{C} \cap F$, $q'_1 \in \mathcal{C}' \setminus F$, and $w, w' \in \Sigma^*$ such that we have $q_1 = \sigma(q_0, w)$, $q'_1 = \sigma(q_0, w')$, and $|w| = |w'|$.*

Theorem 2 *Let Σ be an alphabet with $|\Sigma| > 1$. Let $\mathcal{A} = (Q, F, \sigma, q_0)$ be a DFA over Σ , $0 < p < 1$ and $\mathcal{C}, \mathcal{C}'$ some BSCC of \mathcal{A} (possibly equal). Assume that \mathcal{A} is equal-length-distinguishing. Then almost surely $\mathcal{L}(\mathcal{A}^{\leftarrow p})$ is not a recursively enumerable language.*

Proof Let us denote $\ell = |w|$ and m (resp. m') the periodicity of \mathcal{C} (resp. \mathcal{C}'). Moreover, let $a \in \Sigma$. We build a Markov chain \mathcal{M} from \mathcal{C} as follows: every transition $q \xrightarrow{a} q'$ has probability $1 - p$ and for all $b \neq a$, every transition $q \xrightarrow{b} q'$ has probability $\frac{p}{|\Sigma| - 1}$. We proceed similarly from \mathcal{C}' to build \mathcal{M}' .

Let us denote α_n (resp. α'_n) the probability in \mathcal{M} (resp. \mathcal{M}') that starting from q_1 (resp. q'_1), the current state at time n is q_1 (resp. q'_1). Since \mathcal{M} and \mathcal{M}' are irreducible, $\lim_{n \rightarrow \infty} \alpha_{mn}$ (resp. $\lim_{n \rightarrow \infty} \alpha'_{m'n}$) exists and is positive. Let us denote α (resp. α') this limit. There exists n_0 such that for all $n \geq n_0$, $\alpha_{mn} \geq \frac{\alpha}{2}$ and $\alpha'_{m'n} \geq \frac{\alpha'}{2}$.

Define $w_n = wa^{mm'(n+n_0)}$ for all $n \in \mathbb{N}$. The probability that w_n is accepted by $\mathcal{L}(\mathcal{A})^{\leftarrow p}$ is lower bounded by the probability that the prefix w is unchanged (thus reaching q_1) and that after $mm'(n+n_0)$ steps the current state in \mathcal{M} is q_1 . So a lower bound is: $\min(p, 1-p)^\ell \frac{\alpha}{2}$.

The probability that w_n is rejected by $\mathcal{L}(\mathcal{A})^{\leftarrow p}$ is lower bounded by the probability that the prefix w is changed into w' (thus reaching q'_1) and that after $mm'(n+n_0)$ steps the current state in \mathcal{M}' is q'_1 . So a lower bound is: $\min(p, 1-p)^\ell \frac{\alpha'}{2}$.

Let $W \subseteq W_n$. The probability that $L^{\leftarrow p} \cap W_n$ is equal to W is upper bounded by:

$$\left(1 - \min(p, 1-p)^\ell \frac{\min(\alpha, \alpha')}{2}\right)^n$$

Thus $\rho_n \leq \left(1 - \min(p, 1-p)^\ell \frac{\min(\alpha, \alpha')}{2}\right)^n$ and $\lim_{n \rightarrow \infty} \rho_n = 0$. ■

The DFA \mathcal{A} of Figure 3 that represents the formula ‘ a Until b ’ of temporal logic LTL is equal-length-distinguishing. The corresponding pair of states consists of the accepting state and the leftmost one. Checking the hypotheses of this theorem can be done in quadratic time by first building a graph whose set of vertices is $Q \times Q$ and there is an edge $(q_1, q_2) \rightarrow (q'_1, q'_2)$ if there are some transitions $q_1 \xrightarrow{a_1} q'_1$ and $q_2 \xrightarrow{a_2} q'_2$ and then looking for a vertex (q_1, q_2) in some BSCC with $q_1 \in F$ and $q_2 \notin F$ reachable from (q_0, q_0) .

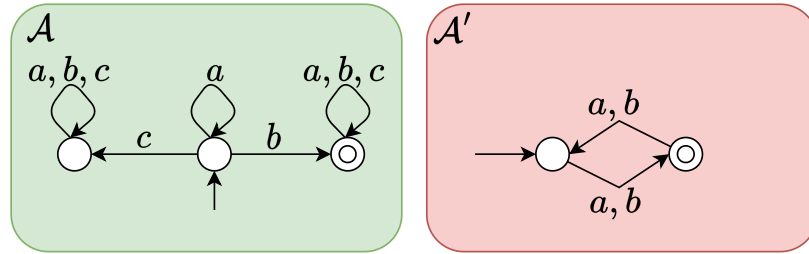


Figure 3: Two DFA

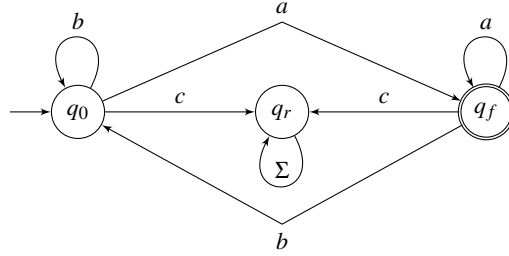
We have that the property of being equal-length-distinguishing is a sufficient condition for ensuring that almost surely $\mathcal{L}(\mathcal{A}^{\leftarrow p})$ is not a recursively enumerable language. So we want to investigate whether it is a necessary condition. The next proposition shows a particular case when this condition is necessary.

Proposition 1 *Let Σ be an alphabet with $|\Sigma| > 1$. Let $\mathcal{A} = (Q, F, \sigma, q_0)$ be a DFA that is not equal-length-distinguishing and such that every circuit of \mathcal{A} belongs to a BSCC. Then, for every sampling L' of $\mathcal{L}(\mathcal{A}^{\leftarrow p})$, L' is regular.*

Proof Pick some $n_0 \in \mathbb{N}$ such that for all w with $|w| \geq n_0$ and $q_0 \xrightarrow{w} q$ implies that q belongs to some BSCC. Observe now that, since \mathcal{A} is not equal-length-distinguishing, for words w, w' with $|w| = |w'| \geq n_0$, $w \in L$ iff $w' \in L$. Thus, for every sampling L' of $\mathcal{L}(\mathcal{A}^{\leftarrow p})$, $L' = (L' \cap \Sigma^{< n_0}) \cup (L \cap \Sigma^{\geq n_0})$ implying that L' is regular. ■

Observe that we establish the next proposition using a generalization of Lemma 1.

Proposition 2 *Let \mathcal{A} be the DFA of Figure 4. Then, \mathcal{A} is not equal-length-distinguishing. Moreover, almost surely $\mathcal{L}(\mathcal{A}^{\leftarrow \frac{2}{3}})$ is not recursively enumerable.*

Figure 4: A DFA \mathcal{A} with $\mathcal{L}(\mathcal{A}) = (a + b)^*a$

Proof There is a single BSCC with a single state $\{q_r\}$. So \mathcal{A} is not equal-length-distinguishing. Let $w \neq \lambda$ be a word with $|w| = n$ and denote \tilde{w} the random word obtained by the noisy perturbation. Observe that every letter of \tilde{w} is uniformly distributed over Σ . So the probability that \tilde{w} does not contain a c is $(\frac{2}{3})^n$ and the conditional probability that \tilde{w} belongs to $\mathcal{L}(\mathcal{A}^{\leftarrow \frac{2}{3}})$ knowing that it does not contain a c is $\frac{1}{2}$.

Fix some $0 < \rho < 1$. The probability that for all words $w \in \Sigma^n$, \tilde{w} contains a c is equal to $(1 - (\frac{2}{3})^n)^{3^n} \leq e^{-2^n}$. Pick an increasing sequence $(n_k)_{k \in \mathbb{N}}$ such $\sum_{k \in \mathbb{N}} e^{-2^{n_k}} \leq 1 - \rho$. Then with probability at least ρ , for all k , there is a word $w_k \in \Sigma^{n_k}$ such that \tilde{w}_k does not contain a c . Letting ρ go to 1, almost surely there is an infinite number of words w such that $\tilde{w} \in (a + b)^+$.

Let us consider an arbitrary language L' and $(w_n)_{n \in \mathbb{N}}$ be an enumeration of Σ^+ . Then almost surely there is an infinite number of w_n such that \tilde{w}_n belong to $(a + b)^+$. Recall that for such a word, the probability that it belongs to $\mathcal{L}(\mathcal{A}^{\leftarrow \frac{2}{3}})$ is equal to $\frac{1}{2}$. Let W_n be the random set of the first n^{th} such words. Then for all n , $\Pr(L' = \mathcal{L}(\mathcal{A}^{\leftarrow \frac{2}{3}})) \leq \Pr(L' \cap W_n = \mathcal{L}(\mathcal{A}^{\leftarrow \frac{2}{3}}) \cap W_n) = 2^{-n}$.

Thus $\Pr(L' = \mathcal{L}(\mathcal{A}^{\leftarrow \frac{2}{3}})) = 0$ and $\Pr(\mathcal{L}(\mathcal{A}^{\leftarrow \frac{2}{3}}) \in \mathcal{F}) = \sum_{L' \in \mathcal{F}} \Pr(L' = \mathcal{L}(\mathcal{A}^{\leftarrow \frac{2}{3}})) = 0$ for \mathcal{F} a countable family of languages. ■

To show the soundness of the structural criterion in Theorem 2 with experiments and comparisons, we have refined our experiments on DFA with noisy inputs partitioning the randomly generated DFA depending on whether they are equal-length-distinguishing.

We have chosen $|\Sigma| = 3$ since with greater size, it was difficult to generate DFAs that do not satisfy the hypotheses. Tables 6 and 7 summarize these experiments. The last rows of the tables (where the information gain is greater than one) confirm our conjecture.

Range	#	$d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}^{\leftarrow p}))$	$d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_E))$	$d(\mathcal{L}(\mathcal{A}^{\leftarrow p}), \mathcal{L}(\mathcal{A}_E))$	gain
[0.005, 0.025]	85	0.01114	0.03604	0.04345	0.30902
[0.002, 0.005]	81	0.00338	0.00421	0.00747	0.80443
[0.001, 0.002]	25	0.00142	0.00035	0.00174	4.09784
[0.0005, 0.001]	16	0.00071	0.00006	0.00077	11.08439

Table 6: Experiments on equal-length-distinguishing DFA

Range	#	$d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}^{\leftarrow p}))$	$d(\mathcal{L}(\mathcal{A}), \mathcal{L}(\mathcal{A}_E))$	$d(\mathcal{L}(\mathcal{A}^{\leftarrow p}), \mathcal{L}(\mathcal{A}_E))$	gain
[0.005, 0.025]	36	0.01089	0.02598	0.03410	0.41905
[0.002, 0.005]	49	0.00308	0.00387	0.00646	0.79628
[0.001, 0.002]	35	0.00136	0.00057	0.00182	2.39863
[0.0005, 0.001]	36	0.00075	0.00063	0.00130	1.18583

Table 7: Experiments on non equal-length-distinguishing DFA

6 Conclusion

We have studied how the PAC-version of Angluin’s algorithm behaves for devices which are obtained from a DFA by introducing noise. More precisely, we have investigated whether Angluin’s algorithm reduces the noise producing a DFA closer to the original one than the noisy device. We have considered three kinds of noise belonging either to random noise or to structured noise. We have shown that, on average, Angluin’s algorithm behaves well for random noise but not for structured noise. We have completed our study by establishing that almost surely the random noisy devices produce a non recursively enumerable language confirming the relevance of the structural criterion for robustness of Angluin’s algorithm.

There are several directions for future work. First the algorithm could be tuned in a more precise way. In addition to stop when the maximal number of rounds is reached or the current automaton is declared equivalent, we could add early stopping when after some stage with distance decreasing the distance stabilizes. This would produce smaller DFA possibly closer to the original DFA. At longer term, Angluin’s algorithm has no information about the original DFA. It would be interesting to introduce a priori knowledge and design more efficient algorithms. For instance, the algorithm could take as input the maximal size of the original DFA or a regular language that is a superset of the original language. In our setting the noise resulted in a noisy device which, once obtained, answers membership queries deterministically. A completely different form of noise would be that the answer to a query is randomly noisy meaning that for the same repeated query, different answers could occur.

References

- [1] Wil M. P. van der Aalst (2012): *Process mining*. *CACM* 55(8), pp. 76–83, doi:10.1145/2240236.2240257.
- [2] Dana Angluin (1987): *Learning Regular Sets from Queries and Counterexamples*. *Inf. Comput.* 75(2), pp. 87–106, doi:10.1016/0890-5401(87)90052-6.
- [3] Dana Angluin & Philip D. Laird (1987): *Learning From Noisy Examples*. *Mach. Learn.* 2(4), pp. 343–370, doi:10.1023/A:1022873112823.
- [4] Alan W. Biermann & Jerome A. Feldman (1972): *A survey of results in grammatical inference*. In S. Watanabe, editor: *Frontiers of Pattern Recognition*, Academic Press, New York, pp. 31–54, doi:10.1016/B978-0-12-737140-5.50007-5.
- [5] Christos G. Cassandras & Stephane Lafortune (2010): *Introduction to Discrete Event Systems*. Springer Publishing Company, Incorporated, doi:10.1007/978-0-387-68612-7.
- [6] E Mark Gold (1978): *Complexity of automaton identification from given data*. *Information and Control* 37(3), pp. 302 – 320, doi:10.1016/S0019-9958(78)90562-4.
- [7] Wassily Hoeffding (1963): *Probability Inequalities for Sums of Bounded Random Variables*. *Journal of the American Statistical Association* 58(301), pp. 13–30, doi:10.2307/2282952.

- [8] Michael J. Kearns (1998): *Efficient Noise-Tolerant Learning from Statistical Queries*. *J. ACM* 45(6), pp. 983–1006, doi:10.1145/293347.293351.
- [9] Michael J. Kearns & Umesh V. Vazirani (1994): *An Introduction to Computational Learning Theory*. MIT Press, doi:10.7551/mitpress/3897.001.0001.
- [10] J. R. Quinlan (1986): *The Effect of Noise on Concept Learning*. In: *Machine Learning, An Artificial Intelligence Approach Volume II*, chapter 6, Morgan Kaufmann, pp. 149–166.
- [11] Ray J. Solomonoff (1964): *A Formal Theory of Inductive Inference*. *Inf. Control*. 7(1, 2), pp. 1–22, 224–254, doi:10.1016/S0019-9958(64)90223-2.
- [12] Leslie G. Valiant (1984): *A Theory of the Learnable*. *Commun. ACM* 27(11), pp. 1134–1142, doi:10.1145/1968.1972.
- [13] R. M. Wharton (1974): *Approximate language identification*. *Information and Control* 26(3), pp. 236 – 255, doi:10.1016/S0019-9958(74)91369-2.

Parametric Interval Temporal Logic over Infinite Words

Laura Bozzelli

University of Napoli “Federico II”, Napoli, Italy

laura.bozzelli@unina.it

Adriano Peron

University of Napoli “Federico II”, Napoli, Italy

adrperon@unina.it

Model checking 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. Here, we focus on the *trace-based semantics*, where the infinite execution paths (traces) of the given (finite) Kripke structure are the main semantic entities. In this setting, each finite infix of a trace is interpreted as an interval, and a proposition holds over an interval if and only if it holds over each component state (*homogeneity assumption*). In this paper, we introduce a quantitative extension of HS over traces, called *parametric HS* (PHS). The novel logic allows to express parametric timing constraints on the duration (length) of the intervals. We show that checking the existence of a parameter valuation for which a Kripke structure satisfies a PHS formula (model checking), or a PHS formula admits a trace as a model under the homogeneity assumption (satisfiability) is decidable. Moreover, we identify a fragment of PHS which subsumes parametric LTL and for which model checking and satisfiability are shown to be EXPSpace-complete.

1 Introduction

Interval temporal logic HS. *Point-based* Temporal Logics (PTLs), such as the linear-time temporal logic LTL [33] and the branching-time temporal logics CTL and CTL* [16] provide a standard framework for the specification of the dynamic behavior of reactive systems that makes it possible to describe how a system evolves state-by-state (“point-wise” view). PTLs have been successfully employed in model checking (MC) [15, 35] for the automatic verification of complex finite-state systems modeled as finite propositional Kripke structures. *Interval Temporal Logics* (ITLs) provide an alternative setting for reasoning about time [20, 32, 38]. They 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. ITLs find applications in a variety of computer science fields, including artificial intelligence (reasoning about action and change, qualitative reasoning, planning, and natural language processing), theoretical computer science (specification and verification of programs), and temporal and spatio-temporal databases (see, e.g., [32, 25, 34]).

The most prominent example of ITLs is *Halpern and Shoham’s modal logic of time intervals* (HS) [20] which features one modality for each of the 13 possible ordering relations between pairs of intervals (the so-called Allen’s relations [1]), apart from equality. The satisfiability problem for HS turns out to be highly undecidable for all interesting (classes of) linear orders [20]. The same happens with most of its fragments [13, 24, 28] with some meaningful exceptions like the logic of temporal neighbourhood \overline{AA} , over all relevant (classes of) linear orders [14], and the logic of sub-intervals D, over the class of dense linear orders [31].

Model checking of (finite) Kripke structures against HS has been investigated only recently [25, 26, 27, 29, 30, 6, 7, 4, 9]. The idea is to interpret each finite path of a Kripke structure as an interval, whose labelling is defined on the basis of the labelling of the component states, that is, a proposition letter holds

over an interval if and only if it holds over each component state (*homogeneity assumption* [36]). Most of the results have been obtained by adopting the so-called *state-based semantics* [29]: 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 [5], while the only known upper bound is non-elementary [29]. 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. These complexity bounds easily transfer to finite satisfiability, that is, satisfiability over finite linear orders, of BE under the homogeneity assumption. Whether or not these problems can be solved elementarily is a difficult open question. On the other hand, 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 $\mathbf{co-NP}$ to $\mathbf{P}^{\mathbf{NP}}$, PSPACE, and beyond, has been determined in a series of papers [30, 6, 8, 10, 12, 9].

The expressiveness of HS with the state-based semantics has been studied in [7], together with other two decidable variants: the *computation-tree-based semantics* variant and the *traces-based* one. For the first variant, past is linear: each interval may have several possible futures, 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. The state-based variant is more expressive than the computation-tree-based variant and expressively incomparable with both LTL and CTL*. To the best of our knowledge, complexity issues about MC and the satisfiability problem of HS and its syntactic fragments under the trace-based semantics have not been investigated so far.

Parametric extensions of point-based temporal logics. Traditional PTLs such as standard LTL [33] allow only to express *qualitative* requirements on the temporal ordering of events. For example, in expressing a typical request-response temporal requirement, it is not possible to specify a bound on the amount of time for which a request is granted. A simple way to overcome this drawback is to consider quantitative extensions of PTLs where temporal modalities are equipped with timing constraints for allowing the specification of *constant* bounds on the delays among events. A well-known representative of such logics is Metric Temporal Logic (MTL) [22]. However this approach is not practical in the first stages of a design, when not much is known about the system under development, and is useful for designers to use parameters instead of specific constants. Parametric extensions of traditional PTLs, where time bounds can be expressed by means of parameters, have been investigated in many papers. Relevant examples include parametric LTL [2], Prompt LTL [23], and parametric MTL [19].

Our contribution. In this paper we introduce a parametric extension of the interval temporal logic HS under the trace-based semantics, called *parametric HS* (PHS). The extension is obtained by means of inequality constraints on the temporal modalities of HS which allow to specify parametric lower/upper bounds on the duration (length) of the interval selected by the temporal modality. Similarly to parametric LTL [2], we impose that a parameter can be exclusively used either as upper bound or as lower bound in the timing constraints. We address the decision problems of checking the existence of a parameter valuation such that (1) a given PHS formula is satisfiable, and (2) a given Kripke structure satisfies a given PHS formula (MC). By adapting the alternating color technique for Prompt LTL [23] and by exploiting known results on *linear-time hybrid logic HL* [18, 37, 3], we show that the considered problems are decidable. Additionally, we consider the syntactic fragment $P(AB\bar{B})$ of PHS which allows only temporal modalities for the Allen’s relations *meets* \mathcal{R}_A , *started-by* \mathcal{R}_B and its inverse $\mathcal{R}_{\bar{B}}$. We show that $P(AB\bar{B})$

subsumes parametric LTL, and its flat fragment $AB\bar{B}$ is exponentially more succinct than LTL + past. Moreover, we establish that satisfiability and MC of $P(AB\bar{B})$ are EXPSpace-complete, and we provide tight bounds on optimal parameter values for both problems.

2 Preliminaries

We fix the following notation. Let \mathbb{Z} be the set of integers, \mathbb{N} the set of natural numbers, and $\mathbb{N}_+ \stackrel{\text{def}}{=} \mathbb{N} \setminus \{0\}$. Let Σ be an alphabet and w be a non-empty finite or infinite word over Σ . We denote by $|w|$ the length of w ($|w| = \infty$ if w is infinite). For all $i, j \in \mathbb{N}$, with $i \leq j < |w|$, $w(i)$ is the $(i+1)$ -th letter of w , while $w[i, j]$ is the infix of w given by $w(i) \cdots w(j)$.

We fix a finite set AP of atomic propositions. A *trace* is an infinite word over 2^{AP} . For a logic \mathfrak{F} interpreted over traces and a formula $\varphi \in \mathfrak{F}$, $\mathcal{L}(\varphi)$ denotes the set of traces satisfying φ . The *satisfiability problem* for \mathfrak{F} is checking for a given formula $\varphi \in \mathfrak{F}$, whether $\mathcal{L}(\varphi) \neq \emptyset$.

Kripke Structures. In the context of model-checking, finite state systems are usually modelled as finite Kripke structures over a finite set AP of atomic propositions which represent predicates over the states of the system. A (finite) *Kripke structure* over AP is a tuple $\mathcal{K} = (AP, S, E, Lab, s_0)$, where S is a finite set of states, $E \subseteq S \times S$ is a left-total transition relation, $Lab : S \mapsto 2^{AP}$ 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 $Lab(\pi(0))Lab(\pi(1)) \dots$. We denote by $\mathcal{L}(\mathcal{K})$ the set of traces associated with the infinite paths of \mathcal{K} . Given a logic \mathfrak{F} interpreted over traces, the (linear-time) *model checking problem against \mathfrak{F}* is checking for a given Kripke structure \mathcal{K} and a formula $\varphi \in \mathfrak{F}$, whether $\mathcal{L}(\mathcal{K}) \subseteq \mathcal{L}(\varphi)$.

Büchi nondeterministic automata. A *Büchi nondeterministic finite automaton* over infinite words (Büchi NFA for short) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, where Σ is a finite input alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \mapsto 2^Q$ is the transition relation, and $F \subseteq Q$ is a set of accepting states. Given an infinite word w over Σ , a run π of \mathcal{A} over w is an infinite sequence π of states such that $\pi(0) = q_0$ and $\pi(i+1) \in \delta(\pi(i), w(i))$ for all $i \geq 0$. The run is accepting if for infinitely many $i \geq 0$, $\pi(i) \in F$. The language $\mathcal{L}(\mathcal{A})$ accepted by \mathcal{A} is the set of infinite words w over Σ such that there is an accepting run of \mathcal{A} over w .

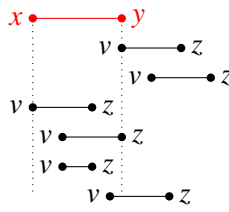
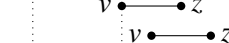
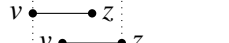
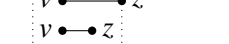
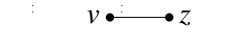
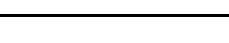
2.1 Allen's relations and Interval Temporal Logic HS

An interval algebra to reason about intervals and their relative orders was proposed by Allen in [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, but equality [20].

Let $\mathbb{U} = (Pt, <)$ be a linear order over the nonempty set $Pt \neq \emptyset$, and \leq be the reflexive closure of $<$. Given two elements $x, y \in Pt$ such that $x \leq y$, we denote by $[x, y]$ the (non-empty closed) *interval* over Pt given by the set of elements $z \in Pt$ such that $x \leq z$ and $z \leq y$. We denote the set of all intervals over \mathbb{U} by $\mathbb{I}(\mathbb{U})$. We now recall the Allen's relations over intervals of the linear order $\mathbb{U} = (Pt, <)$:

1. the *meet* relation \mathcal{R}_A , defined by $[x, y] \mathcal{R}_A [v, z]$ if $y = v$ (i.e., the start-point of the second interval coincides with the end-point of the first interval);
2. the *before* relation \mathcal{R}_L , defined by $[x, y] \mathcal{R}_L [v, z]$ if $y < v$ (i.e., the start-point of the second interval strictly follows the end-point of the first interval);

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	$\langle O \rangle$	$[x, y] \mathcal{R}_O [v, z] \iff x < v < y < z$	

3. the *started-by* relation \mathcal{R}_B , defined by $[x, y] \mathcal{R}_B [v, z]$ if $x = v$ and $z < y$ (i.e., the second interval is a proper prefix of the first interval);
4. the *finished-by* relation \mathcal{R}_E , defined by $[x, y] \mathcal{R}_E [v, z]$ if $y = z$ and $x < v$ (i.e., the second interval is a proper suffix of the first interval);
5. the *contains* relation \mathcal{R}_D , defined by $[x, y] \mathcal{R}_D [v, z]$ if $x < v$ and $z < y$ (i.e., the second interval is contained in the interval of the first interval);
6. the *overlaps* relation \mathcal{R}_O , defined by $[x, y] \mathcal{R}_O [v, z]$ if $x < v < y < z$ (i.e., the second interval overlaps at the right the first interval);
7. for each $X \in \{A, L, B, E, D, O\}$ the relation $\mathcal{R}_{\bar{X}}$, defined as the inverse of \mathcal{R}_X , i.e. $[x, y] \mathcal{R}_{\bar{X}} [v, z]$ if $[v, z] \mathcal{R}_X [x, y]$.

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 together with the corresponding HS (existential) modalities.

Syntax and semantics of HS. HS formulas φ over AP are defined as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle X \rangle \varphi$$

where $p \in AP$ and $\langle X \rangle$ is the existential temporal modality for the (non-trivial) Allen's relation \mathcal{R}_X , where $X \in \{A, L, B, E, D, O, \bar{A}, \bar{L}, \bar{B}, \bar{E}, \bar{D}, \bar{O}\}$. The size $|\varphi|$ of a formula φ is the number of distinct subformulas of φ . We also exploit the standard logical connectives \vee (disjunction) and \rightarrow (implication) as abbreviations, and for any temporal modality $\langle X \rangle$, the dual universal modality $[X]$ defined as: $[X] \psi \stackrel{\text{def}}{=} \neg \langle X \rangle \neg \psi$. Moreover, we will also use the reflexive closure of the Allen's relation $\mathcal{R}_{\bar{B}}$ (resp., $\mathcal{R}_{\bar{E}}$) and the associated temporal modalities $\langle \bar{B}_w \rangle$ and $[\bar{B}_w]$ (resp., $\langle \bar{E}_w \rangle$ and $[\bar{E}_w]$) where $\langle \bar{B}_w \rangle \varphi$ corresponds to $\varphi \vee \langle \bar{B} \rangle \varphi$ and $\langle \bar{E}_w \rangle \varphi$ corresponds to $\varphi \vee \langle \bar{E} \rangle \varphi$. Given any subset of Allen's relations $\{\mathcal{R}_{X_1}, \dots, \mathcal{R}_{X_n}\}$, we denote by $X_1 \cdots X_n$ the HS fragment featuring temporal modalities for $\mathcal{R}_{X_1}, \dots, \mathcal{R}_{X_n}$ only.

The logic HS is interpreted on *interval structures* $\mathcal{S} = (AP, \mathbb{U}, Lab)$, which are linear orders \mathbb{U} equipped with a labelling function $Lab : \mathbb{I}(\mathbb{U}) \rightarrow 2^{AP}$ 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):

$$\begin{aligned} I \models_{\mathcal{S}} p &\iff p \in Lab(I); \\ I \models_{\mathcal{S}} \langle X \rangle \varphi &\iff \text{there is an interval } J \in \mathbb{I}(\mathbb{U}) \text{ such that } I \mathcal{R}_X J \text{ and } J \models_{\mathcal{S}} \varphi. \end{aligned}$$

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

$\langle B \rangle$, $\langle E \rangle$, $\langle \bar{B} \rangle$, and $\langle \bar{E} \rangle$ (see [38]). 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 $\langle \bar{B} \rangle$ as

$$([\bar{E}] \neg \top \wedge (\varphi \vee \langle \bar{B} \rangle \varphi)) \vee \langle E \rangle ([E] \neg \top \wedge (\varphi \vee \langle \bar{B} \rangle \varphi)).$$

Interpretation of HS over traces. In this paper, we focus on interval structures $\mathcal{S} = (AP, (\mathbb{N}, <), Lab)$ 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 $p \in AP$, it holds that $p \in Lab([i, j])$ if and only if $p \in Lab([h, h])$ for every $h \in [i, j]$. Note that homogeneous \mathbb{N} -interval structures over 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 $p \in AP$, $p \in Lab_w([i, j])$ if and only if $p \in w(h)$ for all $h \in [i, j]$. For the given finite set AP of atomic propositions, this mapping from traces to homogeneous \mathbb{N} -interval structures 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 $[0, 0] \models_w \varphi$.

Expressiveness completeness and succinctness of the fragment AB over traces. It is known that HS over traces has the same expressiveness as standard LTL [7], where the latter is expressively complete for standard first-order logic FO over traces [21]. In particular, the fragment AB of HS is sufficient for capturing full LTL [7]: given an LTL formula, one can construct in linear-time an equivalent AB formula [7]. 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. Here, we show that AB is exponentially more succinct than LTL + past. For each $k \geq 1$, we denote by len_k the B formula capturing the intervals of length k : $len_k \stackrel{\text{def}}{=} (\underbrace{\langle B \rangle \dots \langle B \rangle \top}_{k-1 \text{ times}}) \wedge (\underbrace{[\bar{B}] \dots [\bar{B}] \neg \top}_{k \text{ times}})$.

For each $n \geq 1$, let $AP_n = \{p_0, \dots, p_n\}$ and \mathcal{L}_n be the ω -language consisting of the infinite words over 2^{AP_n} such that any two positions that agree on the truth value of propositions p_1, \dots, p_n also agree on the truth value of p_0 . It is known that any Büchi NFA accepting \mathcal{L}_n needs at least 2^{2^n} states [17]. Thus, since any formula φ of LTL + past can be translated into an equivalent Büchi NFA with a single exponential blow-up, it follows that any formula of LTL + past capturing \mathcal{L}_n has size at least single exponential in n . On the other hand, the language \mathcal{L}_n is captured by the following AB formula having size linear in n :

$$[A][A] \left(\bigwedge_{i \in [1, n]} \theta(p_i) \rightarrow \theta(p_0) \right) \quad \theta(p) \stackrel{\text{def}}{=} \langle B \rangle (len_1 \wedge p) \leftrightarrow \langle A \rangle (len_1 \wedge p)$$

Hence, we obtain the following result.

Theorem 1. *AB (over traces) is exponentially more succinct than LTL + past.*

3 Parametric Interval Temporal Logic

In this section, we introduce a parametric extension of the interval temporal logic HS over traces, called *parametric HS* (PHS for short). The extension is obtained by means of inequality constraints on the temporal modalities of HS which allow to compare the length of the interval selected by the temporal modality with an integer parameter. Like parametric LTL [2], the parameterized operators are monotone

(either upward or downward) and a parameter is upward (resp., downward) if it is the subscript of some upward (resp., downward) modality.

Syntax and semantics of PHS Let P_U be a finite set of *upward* parameter variables u and P_L be a finite set of *downward* parameter variables ℓ such that P_U and P_L are disjoint. The syntax of PHS formulas φ over AP and the set $P_U \cup P_L$ of parameter variables is given in positive normal form as follows:

$$\varphi ::= \top \mid p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle X \rangle \varphi \mid \langle X \rangle_{\prec u} \varphi \mid \langle X \rangle_{\succ \ell} \varphi \mid [X] \varphi \mid [X]_{\prec \ell} \varphi \mid [X]_{\succ u} \varphi$$

where $p \in AP$, $X \in \{A, L, B, E, D, O, \bar{A}, \bar{L}, \bar{B}, \bar{B}_w, \bar{E}_w, \bar{D}, \bar{O}\}$, $\prec \in \{<, \leq\}$, $\succ \in \{>, \geq\}$, $u \in P_U$, and $\ell \in P_L$. We denote by PromptHS the fragment of PHS where the unique parameterized temporal modalities are of the form $\langle X \rangle_{\prec u}$. Moreover, given any subset of Allen's relations $\{\mathcal{R}_{X_1}, \dots, \mathcal{R}_{X_n}\}$, we denote by $P(X_1 \dots X_n)$ (resp., Prompt($X_1 \dots X_n$)) the PHS (resp., PromptHS) fragment featuring temporal modalities for $\mathcal{R}_{X_1}, \dots, \mathcal{R}_{X_n}$ only. We will focus on PHS and the fragment $P(\text{ABBB}_w)$.

For an interval $I = [i, j]$ over \mathbb{N} , we denote by $|I|$ the length of I , given by $j - i + 1$. The semantics of a PHS formula φ is inductively defined with respect to a trace w , an interval I over \mathbb{N} , and a *parameter valuation* $\alpha : P_U \cup P_L \mapsto \mathbb{N}_+$ assigning to each parameter variable a positive integer. We write $(I, \alpha) \models_w \varphi$ to mean that φ holds at the interval I of w under the valuation α . The interpretation of all temporal operators of HS and connectives is identical to their HS interpretations. The parameterized operators are interpreted as follows, where $\wp \in P_U \cup P_L$ and $\sim \in \{<, \leq, >, \geq\}$:

$$\begin{aligned} (I, \alpha) \models_w \langle X \rangle_{\sim \wp} \varphi &\Leftrightarrow \text{there is some interval } J \text{ such that } I \mathcal{R}_X J, |J| \sim \alpha(\wp), \text{ and } J, \alpha \models_w \varphi; \\ (I, \alpha) \models_w [X]_{\sim \wp} \varphi &\Leftrightarrow \text{for each interval } J \text{ such that } I \mathcal{R}_X J \text{ and } |J| \sim \alpha(\wp): J, \alpha \models_w \varphi. \end{aligned}$$

We say that the trace w is a model of formula φ under the parameter valuation α , written $(w, \alpha) \models \varphi$, if $([0, 0], \alpha) \models_w \varphi$. For a PHS formula φ and a Kripke structure \mathcal{K} over AP , we consider:

- (i) the set $V(\mathcal{K}, \varphi)$ consisting of the parameter valuations α such that for each trace $w \in \mathcal{L}(\mathcal{K})$ of \mathcal{K} , $(w, \alpha) \models \varphi$, and
- (ii) the set $S(\varphi)$ consisting of the valuations α such that $(w, \alpha) \models \varphi$ for some trace w .

The *(linear-time) model-checking problem against PHS* is checking for a given Kripke structure \mathcal{K} and PHS formula φ whether $V(\mathcal{K}, \varphi) \neq \emptyset$. The *satisfiability problem against PHS* is checking for a given PHS formula φ whether $S(\varphi) \neq \emptyset$.

Given two valuations α and β , we write $\alpha \leq \beta$ to mean that $\alpha(\wp) \leq \beta(\wp)$ for all $\wp \in P_U \cup P_L$. A parameterized operator Θ is *upward-monotone* (resp., *downward-monotone*) if for all formulas φ , valuations α and β such that $\alpha \leq \beta$, $I, \alpha \models_w \Theta \varphi$ entails that $I, \beta \models_w \Theta \varphi$ (resp., $I, \beta \models_w \Theta \varphi$ entails that $I, \alpha \models_w \Theta \varphi$). By construction, all the parameterized operators are monotone. In particular, being P_L and P_U disjoint, by increasing (resp., decreasing) the values of upward (resp., downward) parameters, the satisfaction relation is preserved.

Proposition 1. • *The operators in PHS parameterized by variables in P_U are upward-closed, while those parameterized by variables in P_L are downward-closed.*

- *Let φ be a PHS formula and let α and β be variable valuations satisfying $\beta(u) \geq \alpha(u)$ for every $u \in P_U$ and $\beta(\ell) \leq \alpha(\ell)$ for every $\ell \in P_L$. Then $(w, \alpha) \models \varphi$ entails that $(w, \beta) \models \varphi$.*

Note that if we also allow for all $\ell \in P_L$ and $u \in P_U$, the parameterized modalities $\langle X \rangle_{\succ u}$, $\langle X \rangle_{\prec \ell}$, $[X]_{\succ \ell}$, and $[X]_{\prec u}$, then the modalities $\langle X \rangle_{\sim \wp}$ and $[X]_{\sim \wp}$, for $\wp \in P_U \cup P_L$ and $\sim \in \{<, \leq, >, \geq\}$ are dual and have opposite kind of monotonicity. It easily follows that the logic is indeed closed under negation.

Proposition 2. *Given a PHS formula φ with upward (resp., downward) parameters in P_U (resp., P_L), one can construct in linear time a PHS formula $\bar{\varphi}$ with upward (resp., downward) parameters in P_L (resp.,*

P_U) corresponding to the negation of φ , i.e. such that for each parameter valuation α and trace w over 2^{AP} , $(w, \alpha) \models \varphi$ iff $(w, \alpha) \not\models \bar{\varphi}$.

We now show that parametric LTL (PLTL) [2] can be easily expressed in P(AB). Recall that PLTL formulas φ over AP and the set of parameters $P_U \cup P_L$ are defined as:

$$\varphi ::= \top \mid p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid X\varphi \mid \varphi U \varphi \mid G\varphi \mid F_{\leq u}\varphi \mid G_{\leq \ell}\varphi$$

where $p \in AP$, $u \in P_U$, $\ell \in P_L$, X , U , and G are the standard next, until, and always modalities, respectively, and $F_{\leq u}$ and $G_{\leq \ell}$ are parameterized versions of the always and eventually modalities. Other parameterized modalities such as $F_{> \ell}$ or $G_{> u}$ can be easily expressed in the considered logic [2]. For a PLTL formula φ , a trace w , a parameter valuation α , and a position $i \geq 0$, the satisfaction relation $(w, i, \alpha) \models \varphi$ is defined by induction as follows (we omit the semantics of LTL constructs which is standard):

$$\begin{aligned} (w, i, \alpha) \models F_{\leq u}\varphi &\Leftrightarrow \text{there is some } k \leq \alpha(u) \text{ such that } (w, i+k, \alpha) \models \varphi; \\ (w, i, \alpha) \models G_{\leq \ell}\varphi &\Leftrightarrow \text{for each } k \leq \alpha(\ell): (w, i+k, \alpha) \models \varphi. \end{aligned}$$

Proposition 3. *For a PLTL formula φ , one can build in linear time a P(AB) formula $f(\varphi)$ such that for all traces w , $i \geq 0$, and parameter valuations α , $(w, i, \alpha) \models \varphi$ iff $([i, i], \alpha) \models_w f(\varphi)$.*

Proof. The mapping $f : \text{PLTL} \mapsto \text{P(AB)}$, homomorphic with respect to atomic propositions and Boolean connectives, is defined as follows:

$$\begin{aligned} f(X\varphi) &\stackrel{\text{def}}{=} \langle A \rangle (\text{len}_2 \wedge \langle A \rangle (\text{len}_1 \wedge \varphi)); \\ f(\varphi U \varphi_2) &\stackrel{\text{def}}{=} \langle A \rangle (\langle A \rangle (\text{len}_1 \wedge f(\varphi_2)) \wedge [B] \langle A \rangle (\text{len}_1 \wedge f(\varphi_1))); \\ f(G\varphi) &\stackrel{\text{def}}{=} [A] \langle A \rangle (\text{len}_1 \wedge \varphi); \\ f(F_{\leq u}\varphi) &\stackrel{\text{def}}{=} \langle A \rangle_{\leq u} \langle A \rangle (\text{len}_1 \wedge \varphi); \\ f(G_{\leq \ell}\varphi) &\stackrel{\text{def}}{=} [A]_{\leq \ell} \langle A \rangle (\text{len}_1 \wedge \varphi). \end{aligned}$$

□

Note that by Proposition 3 and the results in [2], the relaxation of the assumption $P_U \cap P_L = \emptyset$ or the adding of parameterized operators of the form $\langle X \rangle_{= \varphi}$ would lead to an undecidable model-checking problem already for the parameterized extension of AB by just one parameter.

Expressively complete fragments. Two PHS formulas φ and ψ are *strongly equivalent*, denoted by $\varphi \equiv \psi$, if for all traces, intervals I over \mathbb{N} , and parameter valuations α , we have that $(I, \alpha) \models_w \varphi$ iff $(I, \alpha) \models_w \psi$. We show that the fragment consisting of $\text{P}(\overline{\text{B}}\overline{\text{B}}_w\overline{\text{E}}\overline{\text{E}}_w)$ formulas with no occurrences of parameterized operators $[X]_{>u}$ is sufficient to capture the full logic PHS.

Proposition 4. *Given a PHS formula φ , one can build in linear time a strongly equivalent $\text{P}(\overline{\text{B}}\overline{\text{B}}_w\overline{\text{E}}\overline{\text{E}}_w)$ formula ψ with no occurrences of the parameterized operators $[X]_{>u}$.*

Proof. We first show that the fragment $\text{P}(\overline{\text{B}}\overline{\text{B}}_w\overline{\text{E}}\overline{\text{E}}_w)$ is expressively complete for PHS. The strong equivalences exploited for expressing all the HS modalities in terms of the modalities in the fragment $\overline{\text{B}}\overline{\text{E}}\overline{\text{E}}$ can be trivially adapted to the parameterized setting. Here, we illustrate the equivalences for the

existential parameterized operators where $\sim \in \{<, \leq, >, \geq\}$ and $\wp \in P_U \cup P_L$:

$$\begin{aligned}
\langle A \rangle_{\sim \wp} \varphi &\equiv (\text{len}_1 \wedge \langle \overline{B} \rangle_{\sim \wp} \varphi) \vee \langle E \rangle (\text{len}_1 \wedge \langle \overline{B} \rangle_{\sim \wp} \varphi); \\
\langle \overline{A} \rangle_{\sim \wp} \varphi &\equiv (\text{len}_1 \wedge \langle \overline{E} \rangle_{\sim \wp} \varphi) \vee \langle B \rangle (\text{len}_1 \wedge \langle \overline{E} \rangle_{\sim \wp} \varphi); \\
\langle L \rangle_{\sim \wp} \varphi &\equiv \langle \overline{B} \rangle \langle E \rangle (\text{len}_1 \wedge \langle \overline{B} \rangle_{\sim \wp} \varphi); \\
\langle \overline{L} \rangle_{\sim \wp} \varphi &\equiv \langle \overline{E} \rangle \langle B \rangle (\text{len}_1 \wedge \langle \overline{E} \rangle_{\sim \wp} \varphi); \\
\langle D \rangle_{\sim \wp} \varphi &\equiv \langle B \rangle \langle E \rangle_{\sim \wp} \varphi; \\
\langle \overline{D} \rangle_{\sim \wp} \varphi &\equiv \langle \overline{B} \rangle \langle \overline{E} \rangle_{\sim \wp} \varphi; \\
\langle O \rangle_{\sim \wp} \varphi &\equiv \langle E \rangle (\neg \text{len}_1 \wedge \langle \overline{B} \rangle_{\sim \wp} \varphi); \\
\langle \overline{O} \rangle_{\sim \wp} \varphi &\equiv \langle B \rangle (\neg \text{len}_1 \wedge \langle \overline{E} \rangle_{\sim \wp} \varphi).
\end{aligned}$$

It remains to show that for the fragment $P(\overline{B}\overline{B}\overline{E}\overline{E}\overline{E}_w)$, the universal upward parameterized operators can be expressed in terms of the other modalities. One can easily show that the following strong equivalences hold, where \prec is $<$ (resp., \prec is \leq) and \succ is \geq (resp., \succ is $>$). Hence, the result follows.

$$\begin{aligned}
[B]_{\succ u} \varphi &\equiv [B](\varphi \vee \langle \overline{B}_w \rangle_{\prec u} \top); \\
[E]_{\succ u} \varphi &\equiv [E](\varphi \vee \langle \overline{E}_w \rangle_{\prec u} \top); \\
\langle \overline{B} \rangle_{\succ u} \varphi &\equiv \langle \overline{B} \rangle_{\prec u} (\langle \overline{B} \rangle \varphi) \vee \langle \overline{B} \rangle \varphi; \\
\langle \overline{E} \rangle_{\succ u} \varphi &\equiv \langle \overline{E} \rangle_{\prec u} (\langle \overline{E} \rangle \varphi) \vee \langle \overline{E} \rangle \varphi; \\
\langle \overline{B}_w \rangle_{\succ u} \varphi &\equiv \langle \overline{B}_w \rangle_{\prec u} (\langle \overline{B} \rangle \varphi) \vee \langle \overline{B}_w \rangle \varphi; \\
\langle \overline{E}_w \rangle_{\succ u} \varphi &\equiv \langle \overline{E}_w \rangle_{\prec u} (\langle \overline{E} \rangle \varphi) \vee \langle \overline{E}_w \rangle \varphi.
\end{aligned}$$

□

For the logic $P(\overline{A}\overline{B}\overline{B}_w)$, we obtain a similar result.

Proposition 5. *Given a $P(\overline{A}\overline{B}\overline{B}_w)$ formula φ , one can build in linear time a strongly equivalent $P(\overline{A}\overline{B}\overline{B}_w)$ formula ψ with no occurrences of the parameterized operators $[X]_{\succ u}$.*

Proof. The result directly follows from the strong equivalences provided in the proof of Proposition 4 and the following one, where \prec is $<$ (resp., \prec is \leq) and \succ is \geq (resp., \succ is $>$): $[A]_{\succ u} \varphi \equiv \langle A \rangle_{\prec u} \langle \overline{B} \rangle \varphi$. □

By the monotonicity of the parameterized modalities and Propositions 4 and 5, we can eliminate all the parameterized modalities, but the existential upward ones, for solving the model-checking and satisfiability problems against PHS (resp., $P(\overline{A}\overline{B}\overline{B}_w)$).

Lemma 1. *Model checking PHS (resp., $P(\overline{A}\overline{B}\overline{B}_w)$) can be reduced in linear time to model checking PromptHS (resp., Prompt($\overline{A}\overline{B}\overline{B}_w$)). Similarly, satisfiability of PHS (resp., $P(\overline{A}\overline{B}\overline{B}_w)$) can be reduced in linear time to satisfiability of PromptHS (resp., Prompt($\overline{A}\overline{B}\overline{B}_w$)).*

Proof. Let φ be a PHS (resp., $P(\overline{A}\overline{B}\overline{B}_w)$) formula. By Propositions 4 and 5, we can assume that φ does not contain occurrences of parameterized operators of the form $[X]_{\succ u}$. Let $f(\varphi)$ be the PromptHS (resp., Prompt($\overline{A}\overline{B}\overline{B}_w$)) formula intuitively obtained from φ by replacing each occurrence of a downward parameter ℓ with the constant 1. Formally, $f(\varphi)$ is homomorphic w.r.t. all the constructs but the downward parameterized modalities and:

- $f(\langle X \rangle_{\geq \ell} \varphi) \stackrel{\text{def}}{=} \langle X \rangle f(\varphi)$;
- $f(\langle X \rangle_{> \ell} \varphi) \stackrel{\text{def}}{=} \langle X \rangle (\neg \text{len}_1 \wedge f(\varphi))$;
- $f([X]_{\leq \ell} \varphi) \stackrel{\text{def}}{=} [X] (\neg \text{len}_1 \vee f(\varphi))$;
- $f([X]_{< \ell} \varphi) \stackrel{\text{def}}{=} \top$.

As for the model checking problem, we show that $V(\mathcal{K}, \varphi) \neq \emptyset$ iff $V(\mathcal{K}, f(\varphi)) \neq \emptyset$ for each Kripke structure \mathcal{K} . Let α_1 be a parameter valuation such that $\alpha_1(\ell) = 1$ for each downward parameter $\ell \in L$. By construction, for all traces w , $(w, \alpha_1) \models \varphi$ iff $(w, \alpha_1) \models f(\varphi)$. Hence, $V(\mathcal{K}, f(\varphi)) \neq \emptyset$ implies that $V(\mathcal{K}, \varphi) \neq \emptyset$. On the other hand, if $V(\mathcal{K}, \varphi) \neq \emptyset$, there is a parameter valuation α such that for each trace w of \mathcal{K} , $(w, \alpha) \models \varphi$. Let α_1 be defined as: $\alpha_1(u) = \alpha(u)$ for each $u \in U$, and $\alpha_1(\ell) = 1$ for each $\ell \in L$. By Proposition 1, it follows that for each trace w of \mathcal{K} , $(w, \alpha_1) \models \varphi$. Thus, we obtain that $V(\mathcal{K}, f(\varphi)) \neq \emptyset$ as well, and the result for the model-checking problem follows. The result for the satisfiability problem is similar. \square

4 Decision procedures for PHS

In this section, we first provide a translation of HS formulas into equivalent Büchi NFA (asymptotically optimal for $AB\overline{B}\overline{B}_w$ formulas), by exploiting as an intermediate step a translation of HS formulas into equivalent formulas of *linear-time hybrid logic HL* [18, 37, 3] (Subsection 4.1). Then, in Subsection 4.2, we apply the results of Subsection 4.1 and the alternating color technique for Prompt LTL [23] in order to solve satisfiability and model checking against PHS and $P(AB\overline{B}\overline{B}_w)$. In particular, for the logic $P(AB\overline{B}\overline{B}_w)$, we show that the considered problems are EXPSPACE-complete.

4.1 Translation of HS in linear-time Hybrid Logic

In this section, we recall the linear-time hybrid logic HL [18, 37, 3], which extends standard LTL + past by first-order concepts. We show that while HS can be translated into the two-variable fragment of HL, for the logic $AB\overline{B}\overline{B}_w$, it suffices to consider the one-variable fragment HL_1 of HL. Thus, by exploiting known results on HL_1 [37, 3], we obtain an asymptotically optimal automata-theoretic approach for $AB\overline{B}\overline{B}_w$ of elementary complexity.

Syntax and semantics of HL. Given a set X of (position) variables, the set of HL formulas φ over AP and X is defined by the following syntax:

$$\varphi \stackrel{\text{def}}{=} \top \mid p \mid x \mid \neg \varphi \mid \varphi \wedge \varphi \mid X\varphi \mid Y\varphi \mid F\varphi \mid P\varphi \mid \downarrow x.\varphi$$

$p \in AP$, $x \in X$, Y and P are the past counterparts of the next modality X and the eventually modality F , respectively, and $\downarrow x$ is the *downarrow binder* operator which assigns the variable name x to the current position. We denote by HL_1 (resp., HL_2) the one-variable (resp., two-variable) fragment of HL. An HL *sentence* is a formula where each variable x is not free (i.e., occurs in the scope of a binder modality $\downarrow x$). The size $|\varphi|$ of an HL formula φ is the number of distinct subformulas of φ .

HL is interpreted over traces w . A *valuation* g is a mapping assigning to each variable a position $i \geq 0$. The satisfaction relation $(w, i, g) \models \varphi$, meaning that φ holds at position i along w w.r.t. the valuation g , is inductively defined as follows (we omit the semantics of LTL constructs which is standard):

$$\begin{aligned} (w, i, g) \models x &\iff i = g(x) \\ (w, i, g) \models \downarrow x.\varphi &\iff (w, i, g[x \mapsto i]) \models \varphi \end{aligned}$$

where $g[x \mapsto i](x) = i$ and $g[x \mapsto i](y) = g(y)$ for $y \neq x$. Thus, $\downarrow x$ binds the variable x to the current position. Note that the satisfaction relation depends only on the values assigned to the variables occurring free in the given formula φ . 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 $(w, 0) \models \varphi$. Note that HL formulas can be trivially translated into

equivalent formulas of first-order logic FO over traces and LTL formulas can be trivially translated into equivalent HL formulas. Thus, by the first-order expressiveness completeness of LTL, HL and LTL have the same expressiveness [18].

Translation of HS into HL. We establish the following result.

Proposition 6. *Given an HS (resp., $AB\overline{B}\overline{B}_w$) formula φ , one can construct in linear-time a two-variable (resp., one-variable) sentence HL φ' such that $\mathcal{L}(\varphi) = \mathcal{L}(\varphi')$.*

Proof. We first consider full HS. We can restrict ourselves to consider the fragment $B\overline{B}\overline{E}\overline{E}$ of HS since all temporal modalities in HS can be expressed in $B\overline{B}\overline{E}\overline{E}$ by a linear-time translation. Fix two distinct variables x_L and x_R . We define a mapping $f : B\overline{B}\overline{E}\overline{E} \mapsto HL_2$ assigning to each $B\overline{B}\overline{E}\overline{E}$ formula φ a HL_2 formula $f(\varphi)$ with variables x_L and x_R which occur free in $f(\varphi)$. Intuitively, in the translation, x_L and x_R refer to the left and right endpoints of the current interval in \mathbb{N} , while the current position corresponds to the left endpoint of the current interval. Formally, the mapping f is homomorphic w.r.t. the Boolean connectives and is inductively defined as follows:

- $f(p) \stackrel{\text{def}}{=} G(Fx_r \rightarrow p)$;
- $f(\langle B \rangle \varphi) \stackrel{\text{def}}{=} F(XFx_R \wedge \downarrow_{x_R}. P(x_L \wedge f(\varphi)))$;
- $f(\langle \overline{B} \rangle \varphi) \stackrel{\text{def}}{=} F(x_R \wedge XF\downarrow_{x_R}. P(x_L \wedge f(\varphi)))$;
- $f(\langle E \rangle \varphi) \stackrel{\text{def}}{=} XF(Fx_R \wedge \downarrow_{x_L}. f(\varphi))$;
- $f(\langle \overline{E} \rangle \varphi) \stackrel{\text{def}}{=} XP\downarrow_{x_L}. f(\varphi)$.

By a straightforward induction on φ , we obtain that given a trace w , an interval $[i, j]$, a valuation g such that $g(x_L) = i$ and $g(x_R) = j$, it holds that $[i, j] \models_w \varphi$ if and only if $(w, i, g) \models f(\varphi)$. The desired HL_2 sentence φ' equivalent to φ is then defined as follows: $\varphi' \stackrel{\text{def}}{=} \downarrow_{x_L}. \downarrow_{x_R}. f(\varphi)$.

We now consider the logic $AB\overline{B}\overline{B}_w$. We can restrict ourselves to consider the fragment $AB\overline{B}$ of HS since the modality $\langle \overline{B}_w \rangle$ can be trivially expressed in terms of $\langle \overline{B} \rangle$. Fix a variable x . We define a mapping $h : AB\overline{B} \mapsto HL_1$ assigning to each $AB\overline{B}$ formula φ an HL_1 formula $h(\varphi)$ with one variable x , which occurs free in $h(\varphi)$. Intuitively, in the translation, x refers to the left endpoint of the current interval in \mathbb{N} , while the current position corresponds to the right endpoint of the current interval. Formally, the mapping h is homomorphic w.r.t. the Boolean connectives and is inductively defined as follows:

- $h(p) \stackrel{\text{def}}{=} \neg P(Px \wedge \neg p)$;
- $h(\langle A \rangle) \stackrel{\text{def}}{=} \downarrow_x. Fh(\varphi)$;
- $h(\langle B \rangle \varphi) \stackrel{\text{def}}{=} YP(h(\varphi) \wedge Px)$;
- $h(\langle \overline{B} \rangle \varphi) \stackrel{\text{def}}{=} XFh(\varphi)$.

By a straightforward induction on φ , we can prove 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 $(w, j, g) \models h(\varphi)$. The desired HL_1 sentence φ' equivalent to φ is then defined as follows: $\varphi' \stackrel{\text{def}}{=} \downarrow_x. h(\varphi)$. \square

It is known that HL_2 is already non-elementarily decidable [37] and for an HL formula φ , one can construct a Büchi NFA accepting $\mathcal{L}(\varphi)$ whose size is a tower of exponentials having height equal to the nesting depth of the binder modality plus one [3]. For the one-variable fragment HL_1 of HL, one can do much better [3]: the size of the Büchi NFA equivalent to a HL_1 formula φ has size doubly exponential in the size of φ . Hence, by Proposition 6, we obtain the following result.

Proposition 7. *Given an HS formula φ , one can build a Büchi NFA \mathcal{A}_φ accepting $\mathcal{L}(\varphi)$. Moreover, if φ is a $AB\overline{B}\overline{B}_w$ formula, then \mathcal{A}_φ has size doubly exponential in the size of φ .*

Note that by [3], the Büchi NFA equivalent to a HL_1 formula can be built on the fly. Recall that non-emptiness of Büchi NFA is NLOGSPACE-complete, and the standard model checking algorithm consists in checking emptiness of the Büchi NFA resulting from the synchronous product of the given finite Kripke structure with the Büchi NFA associated with the negation of the fixed formula. Thus, by Proposition 7, we obtain algorithms for satisfiability and model-checking of $AB\overline{B}\overline{B}_w$ which run in non-deterministic single exponential space. In [11], it is shown that satisfiability and model checking of AB over *finite words* is already EXPSPACE-hard. The EXPSPACE-hardness proof in [11] can be trivially adapted to handle AB over infinite words. Thus, since $EXPSPACE = NEXPSPACE$, we obtain the following result.

Corollary 1. *Model checking and satisfiability problems for $AB\overline{B}\overline{B}_w$ are both EXPSPACE-complete.*

4.2 Solving satisfiability and model checking of PHS

In this section, we provide an automata-theoretic approach for solving satisfiability and model checking of PromptHS and $\text{Prompt}(AB\overline{B}\overline{B}_w)$ based on Proposition 7 and the alternating color technique for Prompt LTL [23]. By Lemma 1, we devise algorithms for solving satisfiability and model checking against PHS and $P(AB\overline{B}\overline{B}_w)$ as well, which for the case of $P(AB\overline{B}\overline{B}_w)$ are asymptotically optimal.

Alternating color technique [23]. We fix a fresh proposition $c \notin AP$. Let us consider a trace w . A c -coloring of w is a trace w' over $AP \cup \{c\}$ such that w and w' agree at every position on all the truth values of the propositions in AP , i.e. $w'(i) \cap AP = w(i)$ for all $i \geq 0$. A position $i \geq 0$ is a c -change point in w' if either $i = 0$, or the colors of i and $i - 1$ are different, i.e. $c \in w'(i)$ iff $c \notin w'(i - 1)$. A c -block of w' is a maximal interval $[i, j]$ which has exactly one c -change point in w' , and this change point is at the first position i of $[i, j]$. Given $k \geq 1$, we say that w' is k -bounded if each c -block of w' has length at most k , which implies that w' has infinitely many c -change points. Dually, we say that w' is k -spaced if w' has infinitely many c -change points and every c -block has length at least k .

We apply the alternating color technique [23] for replacing a parameterized modality $\langle X \rangle_{\prec u} \psi$ in PromptHS with a non-parameterized one requiring that the selected interval where ψ holds has at most one c -change point. Formally, let $rel_c : \text{PromptHS} \mapsto \text{HS}$ be the mapping associating to each PromptHS formula a HS formula, homomorphic w.r.t. propositions, connectives, and non-parameterized modalities, and defined as follows on parameterized formulas $\langle X \rangle_{\prec u} \psi$:

$$rel_c(\langle X \rangle_{\prec u} \psi) \stackrel{\text{def}}{=} \langle X \rangle (rel_c(\psi) \wedge (\theta_c \vee \theta_{\neg c})).$$

where for each $d \in \{c, \neg c\}$, θ_d is an AB formula requiring that the current interval has at most one c -change point and the right endpoint is a d -colored position:

$$\theta_d \stackrel{\text{def}}{=} \langle A \rangle (\text{len}_1 \wedge d) \wedge [B] (\langle A \rangle (\text{len}_1 \wedge \neg d) \rightarrow [B] \langle A \rangle (\text{len}_1 \wedge \neg d)).$$

For a PromptHS formula φ , let $c(\varphi)$ be the HS formula defined as follows:

$$c(\varphi) \stackrel{\text{def}}{=} rel_c(\varphi) \wedge alt_c \quad alt_c \stackrel{\text{def}}{=} [A] \langle A \rangle \langle A \rangle (\text{len}_1 \wedge c) \wedge [A] \langle A \rangle \langle A \rangle (\text{len}_1 \wedge \neg c)$$

Note that $c(\varphi)$ is a $AB\overline{B}\overline{B}_w$ formula if φ is a $\text{Prompt}(AB\overline{B}\overline{B}_w)$ formula. Moreover, the AB formula alt_c requires that there are infinitely many c -change points. Thus, $c(\varphi)$ forces a c -coloring of the given trace w to be partitioned into infinitely many blocks such that each parameterized modality selects an interval with at most one c -change point. Like Prompt LTL [23], there is a weak equivalence between φ and $c(\varphi)$ on k -bounded and k -spaced c -coloring of w . The following lemma rephrases Lemma 2.1 in [23] and can be proved in a similar way.

Lemma 2. *Let φ be a PromptHS formula and w be a trace.*

1. *If $(w, \alpha) \models \varphi$, then $w' \models c(\varphi)$ for each k -spaced c -coloring w' of w with $k = \max_{u \in P_U} \alpha(u)$.*
2. *Let $k \geq 1$. If w' is a k -bounded c -coloring of w with $w' \models c(\varphi)$, then $(w, \alpha) \models \varphi$, where $\alpha(u) = 2k$ for each $u \in P_U$.*

Solving satisfiability. Let φ be a PromptHS formula and \mathcal{A}_c be the Büchi NFA of Proposition 7 accepting the models of the HS formula $c(\varphi)$. By Lemma 2, we deduce that $S(\varphi) \neq \emptyset$ if and only if there is $k \geq 1$ and some k -bounded c -coloring w' accepted by \mathcal{A}_c . Indeed, if $S(\varphi) \neq \emptyset$, then there is a parameter valuation α and a trace w such that $(w, \alpha) \models \varphi$. Let $k = \max_{u \in P_U} \alpha(u)$ and w' be the c -coloring of w whose c -blocks have length exactly k . Note that w' is both k -spaced and k -bounded. By Lemma 2(1), $w' \models c(\varphi)$, hence, w' is accepted by \mathcal{A}_c . Vice versa, if there is a trace w and a k -bounded c -coloring w' of w accepted by \mathcal{A}_c , then, by Lemma 2(2), $(w, \alpha) \models \varphi$, where $\alpha(u) = 2k$ for each $u \in P_U$. Hence, $S(\varphi) \neq \emptyset$.

Let N_c be the number of \mathcal{A}_c states. Assume that there is k -bounded c -coloring w' accepted by \mathcal{A}_c for some $k \geq 1$. We claim that there is also a $2N_c + 1$ -bounded c -coloring accepted by \mathcal{A}_c . If $k \leq 2N_c + 1$, the result is obvious. Otherwise, let π be an accepting run of \mathcal{A}_c over w' , and let us consider the infixes ν of π associated with the c -blocks of w' greater than $2N_c + 1$. We replace ν with an infix of length at most $2N_c + 1$ as follows:

- If ν does not visit accepting states, we remove from ν the maximal cycles (but the first states of such cycles) by obtaining a finite path of length at most N_c .
- If ν visits some accepting state, then ν can be written in the form $\nu = \nu_1 \cdot q_a \cdot \nu_2$, where q_a is an accepting state. We remove the maximal cycles from ν_1 and ν_2 (but the first states of such cycles) by obtaining a finite path of length at most $2N_c + 1$.

In this way, we obtain an accepting run of \mathcal{A}_c over a $2N_c + 1$ -bounded c -coloring, and the result follows. Then, starting from \mathcal{A}_c , one can easily construct in time polynomial in the size of \mathcal{A}_c , a Büchi NFA \mathcal{A}'_c accepting the $2N_c + 1$ -bounded colorings which are accepted by \mathcal{A}_c . \mathcal{A}'_c keeps track in its state of the current state of \mathcal{A}_c and the binary encoding of the value of a counter modulo $2N_c + 1$, where the latter is reset whenever a c -change point occurs. Note that if φ is a Prompt(\overline{ABBB}_w) formula, then $c(\varphi)$ is a \overline{ABBB}_w formula, and by Proposition 7, the size of \mathcal{A}_c is doubly exponential in the size of φ . By the previous observations, it follows that if $S(\varphi) \neq \emptyset$, then there is a parameter valuation $\alpha \in S(\varphi)$ which is bounded doubly exponentially in $|\varphi|$. Thus, since non-emptiness of Büchi NFA is NLOGSPACE-complete, by Lemma 1 and Proposition 7, we obtain the following result.

Theorem 2. *Satisfiability of PHS is decidable. Moreover, satisfiability of $P(\overline{ABBB}_w)$ is EXPSPACE-complete and given a $P(\overline{ABBB}_w)$ formula φ , in case $S(\varphi) \neq \emptyset$, there is a parameter valuation in $S(\varphi)$ which is bounded doubly exponentially in $|\varphi|$.*

We now show that the double exponential upper bound on the values of the parameters in Theorem 2 for satisfiable $P(\overline{ABBB}_w)$ formulas cannot be in general improved. Indeed, we provide a matching lower bound by defining for each $n \geq 1$, a $P(\overline{AB})$ formula of size polynomial in n which encodes a yardstick of length $(n + 1) * 2^n * 2^{2^n}$. This is done by using a 2^n -bit counter for expressing integers in the range $[0, 2^{2^n} - 1]$ and an n -bit counter for keeping track of the position (index) $i \in [0, 2^n - 1]$ of the $(i + 1)^{th}$ -bit of each valuation ν of the 2^n -bit counter. In particular, such a valuation $\nu \in [0, 2^{2^n} - 1]$ is encoded by a sequence, called n -block, of 2^n sub-blocks of length $n + 1$ where for each $i \in [0, 2^n - 1]$, the $(i + 1)^{th}$ sub-block encodes both the value and the index of the $(i + 1)^{th}$ -bit in the binary representation of ν .

Formally, let $AP \stackrel{\text{def}}{=} \{\#, \#, \$, 0, 1\}$. Fix $n \geq 1$. An n -sub-block is a finite word ν over 2^{AP} of length $n + 1$ of the form $\nu = \{\#, p, bit\} \{bit_1\}, \dots, \{bit_n\}$ where $bit, bit_1, \dots, bit_n \in \{0, 1\}$ and $p \in \{\#, \#\}$. If

$p = \#_2$, we say that v is marked. The *content* of v is *bit*, and the *index* of v is the number in $[0, 2^n - 1]$ whose binary code is bit_1, \dots, bit_n . An *n-block* is a finite word v of length $(n + 1) * 2^n$ of the form $v = v_0 \dots v_{2^n - 1}$, where v_0 is a marked *n-sub-block* of index 0, and for each $i \in [1, 2^n - 1]$, v_i is an unmarked *n-sub-block* having index i . The index of i is the natural number in $[0, 2^{2^n} - 1]$ whose binary code is $bit_0, \dots, bit_{2^n - 1}$, where bit_i is the content of the sub-block v_i for each $i \in [0, 2^n - 1]$. The yardstick of length $(n + 1) * 2^n * 2^{2^n}$ is then encoded by the trace, called *n-trace*, given by $bl_0 \cdot \dots \cdot bl_{2^{2^n} - 1} \cdot \{\$\}^\omega$ where bl_i is the *n-block* having index i for each $i \in [0, 2^{2^n} - 1]$. We first show the following result.

Lemma 3. *For each $n \geq 1$, one can construct in time polynomial in n a satisfiable AB formula ψ_n whose unique model is the *n-trace*.*

Proof. Fix $n \geq 1$. The AB formula ψ_n is defined as $\psi_n \stackrel{\text{def}}{=} \psi_{bl} \wedge \psi_{inc}$. The conjunct ψ_{bl} captures the traces w over $AP = \{\#_1, \#_2, \$, 0, 1\}$ having the form $bl_0 \cdot \dots \cdot bl_k \cdot \{\$\}^\omega$, for some $k \geq 1$, such that the following conditions are satisfied:

- bl_0, \dots, bl_k are *n-blocks*;
- bl_0 is the *n-block* of index 0 (i.e., each *n-sub-block* of bl_0 has content 0);
- bl_k is the *n-block* of index $2^{2^n} - 1$ (i.e., each *n-sub-block* of bl_k has content 1).

One can easily construct an LTL formula of size polynomial in n characterizing the traces satisfying the previous requirements. Thus, since an LTL formula can be translated in linear time into an equivalent AB formula [7], we omit the details of the construction of the AB formula ψ_{bl} .

The conjunct ψ_{inc} additionally ensures that $k = 2^{2^n} - 1$ and for each $i \in [1, 2^{2^n} - 2]$, bl_i is the *n-block* of index i . To this purpose, it suffices to guarantee that in moving from a non-last *n-block* bl to the next one bl' , the 2^n -counter is incremented. This is equivalent to require that there is an *n-sub-block* sbl_0 of bl whose content is 0 such that for each *n-sub-block* sbl of bl , denoted by sbl' the *n-sub-block* of bl' having the same index as sbl , the following holds: (i) if sbl precedes sbl_0 , then the content of sbl (resp., sbl') is 1 (resp., 0), (ii) if sbl corresponds to sbl_0 , then the content of sbl' is 1, and (iii) if sbl follows sbl_0 , then there is $b \in \{0, 1\}$ such that the content of both sbl and sbl' is b . In order to express these conditions, we define auxiliary AB formulas. Recall that proposition $\#_1$ marks the first position of an *n-sub-block*, while $\#_2$ marks the first position of an *n-block* bl (corresponding to the first position of the first *n-sub-block* of bl). For each AB formula φ , the AB formula $right(\varphi)$ requires that φ holds at the singleton interval corresponding to the right endpoint of the current interval:

$$right(\varphi) \stackrel{\text{def}}{=} \langle A \rangle (\text{len}_1 \wedge \varphi)$$

The AB formula $\psi_{one}(\#_2)$ ensures that proposition $\#_2$ occurs exactly once in the current interval, while $\psi_{not}(\#_2)$ ensures that $\#_2$ does not occur in the current interval. We focus on the definition of $\psi_{one}(\#_2)$ (the definition of $\psi_{not}(\#_2)$ being similar).

$$\psi_{one}(\#_2) \stackrel{\text{def}}{=} [right(\#_2) \vee \langle B \rangle right(\#_2)] \wedge \neg \langle B \rangle [right(\#_2) \wedge \langle B \rangle right(\#_2)] \wedge \neg [right(\#_2) \wedge \langle B \rangle right(\#_2)]$$

Moreover, we define the AB formulas $\psi_{\pm}(b, b')$ where $b, b' \in \{0, 1\}$. Formula $\psi_{\pm}(b, b')$ holds at a singleton interval $[h, h]$ (along the given trace) iff whenever h corresponds to the beginning of a *n-sub-block* sbl of an *n-block* bl , then (i) the content of sbl is b , (ii) the *n-block* bl is followed by an *n-block* bl' , and (iii) the *n-sub-block* of bl' having the same index as sbl has content b' .

$$\psi_{\pm}(b, b') \stackrel{\text{def}}{=} \#_1 \rightarrow [b \wedge \langle A \rangle (\text{len}_2 \wedge \langle A \rangle (\psi_{one}(\#_2) \wedge \theta_{\pm} \wedge right(b' \wedge \#_1)))]$$

$$\theta_{\pm} \stackrel{\text{def}}{=} \bigwedge_{h=1}^n \bigvee_{b \in \{0, 1\}} [\langle B \rangle (\text{len}_h \wedge right(b)) \wedge \langle A \rangle (\text{len}_{h+1} \wedge right(b))]$$

Finally, the conjunct ψ_{inc} in the definition of ψ_n is given by

$$\begin{aligned}\psi_{inc} &\stackrel{\text{def}}{=} [A]([\text{right}(\#_2) \wedge \langle A \rangle (\neg \text{len}_1 \wedge \text{right}(\#_2))] \longrightarrow \langle A \rangle [\psi_{one}(\#_2) \wedge \text{right}(\#_1 \wedge \psi_{=}(0, 1)) \wedge \psi_L \wedge \psi_R]) \\ \psi_L &\stackrel{\text{def}}{=} [B](\text{right}(\#_1) \rightarrow \text{right}(\psi_{=}(1, 0))) \\ \psi_R &\stackrel{\text{def}}{=} \langle A \rangle (\text{len}_2 \wedge [A]([\psi_{not}(\#_2) \wedge \text{right}(\#_1)] \rightarrow \bigvee_{b \in \{0,1\}} \text{right}(\psi_{=}(b, b))))\end{aligned}$$

This concludes the proof of Lemma 3. \square

Fix $n \geq 1$ and let ψ_n be the AB formula in Lemma 3. We consider the P(AB) formula φ_n with just one parameter given by $\varphi_n \stackrel{\text{def}}{=} \psi_n \wedge \langle A \rangle_{\leq u} \langle A \rangle (\text{len}_1 \wedge \$)$. By Lemma 3, the smallest value for parameter u for which φ_n has a model is greater than $(n+1) * 2^n * 2^{2^n}$. Hence, we obtain the following result.

Proposition 8. *There is a finite set AP of atomic propositions and a family $\{\varphi_n\}_{n \geq 1}$ of satisfiable P(AB) formulas over AP with just one parameter such that for each $n \geq 1$, φ_n has size polynomial in n and the smallest parameter valuation in $S(\varphi_n)$ is doubly exponential in n .*

Solving model checking. A fair Kripke structure \mathcal{K}_f is a Kripke structure equipped with a set S_f of accepting states. An infinite path of \mathcal{K}_f is *fair* if it visits infinitely many times states in S_f . Assume that \mathcal{K}_f is over the set of atomic propositions given by $AP \cup \{c\}$, and let Lab_c be the associated propositional labeling. A *c-pumpable fair path* of \mathcal{K}_f is a fair infinite path π of \mathcal{K}_f such that each infix of π associated to a c -block of the trace $Lab_c(\pi)$ visits some state at least twice. Let $\mathcal{K} = (AP, S, E, Lab, s_0)$ be a Kripke structure over AP, φ a PromptHS formula, and $\mathcal{A}_{-c} = (2^{AP \cup \{c\}}, Q, q_0, \delta, F)$ be the Büchi NFA of Proposition 7 accepting the models of the HS formula $\neg rel_c(\varphi) \wedge alt_c$ (note that we consider the negation of $rel_c(\varphi)$). We define the fair Kripke structure

$$\mathcal{K} \times \mathcal{A}_{-c} = (AP \cup \{c\}, S \times Q \times 2^{\{c\}}, (s_0, q_0, \emptyset), E_c, Lab_c, S \times F \times 2^{\{c\}})$$

where (i) $((s, q, C), (s', q', C')) \in E_c$ iff $(s, s') \in E$ and $q' \in \delta(q, C \cup Lab(s))$, and (ii) $Lab_c(s, q, C) = Lab(s) \cup C$. By construction, the traces associated to the fair infinite paths of $\mathcal{K} \times \mathcal{A}_{-c}$ correspond to the c -colorings w' of the traces of \mathcal{K} which are accepted by \mathcal{A}_{-c} such that $c \notin w'(0)$. The following lemma is similar to Lemma 4.2 in [23] and provides a characterization of emptiness of the set $V(\mathcal{K}, \varphi)$ of parameter valuations.

Lemma 4. *\mathcal{K} does not satisfy φ (i.e., $V(\mathcal{K}, \varphi) = \emptyset$) iff $\mathcal{K} \times \mathcal{A}_{-c}$ has a c -pumpable fair path.*

Proof. For the right implication, assume that $V(\mathcal{K}, \varphi) = \emptyset$. We need to show that $\mathcal{K} \times \mathcal{A}_{-c}$ has a c -pumpable fair path. Let $k = |Q||S| + 1$ and α be the parameter valuation defined by $\alpha(u) = 2k$ for each $u \in U$. Since $V(\mathcal{K}, \varphi) = \emptyset$, there is a trace w of \mathcal{K} such that $(w, \alpha) \not\models \varphi$. Let w' be the k -bounded c -coloring of w such that each c -block of w' has length exactly k and $c \notin w'(0)$. Since $w' \models alt_c$ and $c(\varphi) = rel_c(\varphi) \wedge alt_c$, by Lemma 2(2), it follows that $w' \models \neg rel_c(\varphi) \wedge alt_c$. Hence, w' is accepted by the Büchi automaton \mathcal{A}_{-c} , and by construction there is a fair path π of $\mathcal{K} \times \mathcal{A}_{-c}$ whose trace is w' . Now, each infix of π associated to a c -block of w' has length $k = |Q||S| + 1$. Moreover, by construction, the third component C of the states $(s, q, C) \in S \times Q \times 2^c$ along such an infix does not change. It follows that such an infix visits one state at least twice. Thus, π is a c -pumpable fair path of $\mathcal{K} \times \mathcal{A}_{-c}$.

For the left implication, assume that $\mathcal{K} \times \mathcal{A}_{-c}$ has a c -pumpable fair path ρ . Let α be an arbitrary parameter valuation and $k = \max_{u \in U} \alpha(u)$. We need to show that there is a trace w of \mathcal{K} such that

$(w, \alpha) \not\models \varphi$. Since ρ is a c -pumpable fair path, each infix of ρ associated to a c -block of the trace $Lab_c(\rho)$ visits some state at least twice. The corresponding cycle in the infix can be pumped k -times. It follows that there is a c -pumpable fair path ρ' of $\mathcal{K} \times \mathcal{A}_{-c}$ such that the c -blocks of the associated trace w' have length at least k . By construction, w' is the c -coloring of some trace w of \mathcal{K} and w' is accepted by \mathcal{A}_{-c} , i.e. $w' \models \neg rel_c(\varphi) \wedge alt_c$. Hence, w' is a k -spaced coloring of w and $w' \not\models c(\varphi)$. By Lemma 2(1), it follows that $(w, \alpha) \not\models \varphi$, and the result follows. \square

By Lemma 4, we deduce that if $V(\mathcal{K}, \varphi) \neq \emptyset$, then for the parameter valuation α such that $\alpha(u) = 2(|Q||S| + 1)$ for each $u \in P_U$, it holds that $\alpha \in V(\mathcal{K}, \varphi)$. Indeed if $\alpha \notin V(\mathcal{K}, \varphi)$, by the first part of the proof of Lemma 4, there is a c -pumpable fair path of $\mathcal{K} \times \mathcal{A}_{-c}$, which leads to the contradiction $V(\mathcal{K}, \varphi) = \emptyset$. It is known that checking the existence of a c -pumpable fair path in a fair Kripke structure is NLOGSPACE-complete [23]. Recall that if φ is a Prompt($AB\overline{B}\overline{B}_w$) formula, then $c(\varphi)$ is a $AB\overline{B}\overline{B}_w$ formula, and by Proposition 7, the size of \mathcal{A}_{-c} is doubly exponential in the size of φ . Thus, since both \mathcal{A}_{-c} and $\mathcal{K} \times \mathcal{A}_{-c}$ can be built on the fly, by Lemma 1, Proposition 7, and Lemma 4, we obtain the following result.

Theorem 3. *Model checking against PHS is decidable. Moreover, model checking a Kripke structure \mathcal{K} against a $P(AB\overline{B}\overline{B}_w)$ formula φ is EXPSPACE-complete and, in case $V(\mathcal{K}, \varphi) \neq \emptyset$, there is a parameter valuation in $V(\mathcal{K}, \varphi)$ which is bounded doubly exponentially in $|\varphi|$ and linearly in the number of \mathcal{K} -states.*

Similarly to the satisfiability problem for $P(AB\overline{B}\overline{B}_w)$, for each $n \geq 1$, we provide a lower bound of 2^{2^n} on the minimal parameter valuation for which a fixed Kripke structure satisfies a $P(AB)$ formula by using a $P(AB)$ formula of size polynomial in n . For each $n \geq 1$, let ψ_n be the AB formula over $AP = \{\#_1, \#_2, \$, 0, 1\}$ in Lemma 3 whose unique model is the n -trace. One can trivially define a Kripke structure \mathcal{K} over AP whose set of traces consists of the traces whose first position has label $\{\#_1, \#_2, 0\}$. Let us consider the $P(AB)$ formula φ_n with just one parameter given by $\varphi_n \stackrel{\text{def}}{=} \psi_n \rightarrow \langle A \rangle_{\leq u} \langle A \rangle (\text{len}_1 \wedge \$)$. Evidently, by Lemma 3, $V(\mathcal{K}, \varphi_n)$ is not empty and the minimal parameter valuation in $V(\mathcal{K}, \varphi_n)$ is doubly exponential in n . Hence, we obtain the following result.

Proposition 9. *There is a Kripke structure \mathcal{K} over a set AP of atomic propositions and a family $\{\varphi_n\}_{n \geq 1}$ of $P(AB)$ formulas over AP with just one parameter such that for each $n \geq 1$, φ_n has size polynomial in n , $V(\mathcal{K}, \varphi_n) \neq \emptyset$, and the smallest parameter valuation in $V(\mathcal{K}, \varphi_n)$ is doubly exponential in n .*

5 Conclusion

We have introduced parametric HS (PHS), a parametric extension of the interval temporal logic HS under the trace-based semantics. The novel logic allows to express parametric timing constraints on the duration of the intervals. We have shown that the satisfiability and model checking problems for the whole logic are decidable, and for the fragment $P(AB\overline{B})$ of PHS, the problems are EXPSPACE-complete. Moreover, for the fragment $P(AB\overline{B})$, we gave tight bounds on optimal parameter values for the considered problems. An intriguing open question is the expressiveness of $P(AB\overline{B})$ (or more in general PHS) versus parametric LTL (PLTL). We have shown that $P(AB\overline{B})$ subsumes PLTL. In particular, given a PLTL formula φ , it is possible to construct in linear time a $P(AB\overline{B})$ on the same set of parameters which is equivalent to φ for each parameter valuation. Is $P(AB\overline{B})$ more expressive than PLTL? Another problem left open is whether PromptHS is strictly less expressive than full PHS.

References

- [1] J.F. Allen (1983): *Maintaining Knowledge about Temporal Intervals*. *Communications of the ACM* 26(11), pp. 832–843, doi:10.1145/182.358434.
- [2] R. Alur, K. Etessami, S. La Torre & D.A. Peled (2001): *Parametric temporal logic for "model measuring"*. *ACM Trans. Comput. Log.* 2(3), pp. 388–407, doi:10.1145/377978.377990.
- [3] L. Bozzelli & R. Lanotte (2010): *Complexity and succinctness issues for linear-time hybrid logics*. *Theor. Comput. Sci.* 411(2), pp. 454–469, doi:10.1016/j.tcs.2009.08.009.
- [4] L. Bozzelli, A. Molinari, A. Montanari & A. Peron (2020): *Model checking interval temporal logics with regular expressions*. *Information and Computation* 272, p. 104498, doi:10.1016/j.ic.2019.104498.
- [5] L. Bozzelli, A. Molinari, A. Montanari, A. Peron & P. Sala (2016): *Interval Temporal Logic Model Checking: the Border Between Good and Bad HS Fragments*. In: *Proc. 8th IJCAR*, LNAI 9706, Springer, pp. 389–405, doi:10.1007/978-3-319-40229-1_27.
- [6] L. Bozzelli, A. Molinari, A. Montanari, A. Peron & P. Sala (2018): *Model checking for fragments of the interval temporal logic HS at the low levels of the polynomial time hierarchy*. *Information and Computation* 262(Part), pp. 241–264, doi:10.1016/j.ic.2018.09.006.
- [7] L. Bozzelli, A. Molinari, A. Montanari, A. Peron & P. Sala (2019): *Interval vs. Point Temporal Logic Model Checking: An Expressiveness Comparison*. *ACM Trans. Comput. Log.* 20(1), pp. 4:1–4:31, doi:10.1145/3281028.
- [8] L. Bozzelli, A. Molinari, A. Montanari, A. Peron & P. Sala (2019): *Which fragments of the interval temporal logic HS are tractable in model checking?* *Theor. Comput. Sci.* 764, pp. 125–144, doi:10.1016/j.tcs.2018.04.011.
- [9] L. Bozzelli, A. Molinari, A. Montanari, A. Peron & P. Sala (2022): *Satisfiability and Model Checking for the Logic of Sub-Intervals under the Homogeneity Assumption*. *Log. Methods Comput. Sci.* 18(1), doi:10.46298/lmcs-18(1:24)2022.
- [10] L. Bozzelli, A. Montanari & A. Peron (2021): *Complexity analysis of a unifying algorithm for model checking interval temporal logic*. *Inf. Comput.* 280, p. 104640, doi:10.1016/j.ic.2020.104640.
- [11] L. Bozzelli, A. Montanari, A. Peron & P. Sala (2021): *Adding the Relation Meets to the Temporal Logic of Prefixes and Infixes makes it EXPSpace-Complete*. In: *Proc. 12th GandALF, EPTCS* 346, pp. 179–194, doi:10.4204/EPTCS.346.12.
- [12] L. Bozzelli, A. Montanari, A. Peron & P. Sala (2021): *Pspace-Completeness of the Temporal Logic of Sub-Intervals and Suffixes*. In: *Proc. 28th TIME, LIPIcs* 206, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 9:1–9:19, doi:10.4230/LIPIcs.TIME.2021.9.
- [13] D. Bresolin, D. Della Monica, V. Goranko, A. Montanari & G. Sciavicco (2014): *The dark side of interval temporal logic: marking the undecidability border*. *Annals of Mathematics and Artificial Intelligence* 71(1-3), pp. 41–83, doi:10.1007/s10472-013-9376-4.
- [14] D. Bresolin, A. Montanari, P. Sala & G. Sciavicco (2011): *Optimal Tableau Systems for Propositional Neighborhood Logic over All, Dense, and Discrete Linear Orders*. In: *Proc. 20th TABLEAUX*, LNCS 6973, Springer, pp. 73–87, doi:10.1007/978-3-642-22119-4_8.
- [15] E.M. Clarke & E.A. Emerson (1981): *Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic*. In: *Proc. Logics of Programs*, LNCS 131, pp. 52–71, doi:10.1007/BFb0025774.
- [16] E. A. Emerson & J. Y. Halpern (1986): *"Sometimes" and "not never" revisited: on branching versus linear time temporal logic*. *Journal of the ACM* 33(1), pp. 151–178, doi:10.1145/4904.4999.
- [17] K. Etessami, M.Y. Vardi & T. Wilke (2002): *First-Order Logic with Two Variables and Unary Temporal Logic*. *Inf. Comput.* 179(2), pp. 279–295, doi:10.1006/inco.2001.2953.

- [18] M. Franceschet, M. de Rijke & B.H. Schlingloff (2003): *Hybrid Logics on Linear Structures: Expressivity and Complexity*. In: *Proc. 10th TIME-ICTL*, IEEE Computer Society, pp. 166–173, doi:10.1109/TIME.2003.1214893.
- [19] B. Di Giampaolo, S. La Torre & M. Napoli (2010): *Parametric Metric Interval Temporal Logic*. In: *Proc. 4th LATA*, LNCS 6031, Springer, pp. 249–260, doi:10.1007/978-3-642-13089-2_21.
- [20] J.Y. Halpern & Y. Shoham (1991): *A Propositional Modal Logic of Time Intervals*. *Journal of the ACM* 38(4), pp. 935–962, doi:10.1145/115234.115351.
- [21] J.A.W. Kamp (1968): *Tense logic and the theory of linear order*. University of California, Los Angeles.
- [22] R. Koymans (1990): *Specifying Real-Time Properties with Metric Temporal Logic*. *Real Time Syst.* 2(4), pp. 255–299, doi:10.1007/BF01995674.
- [23] O. Kupferman, N. Piterman & M.Y. Vardi (2009): *From liveness to promptness*. *Formal Methods Syst. Des.* 34(2), pp. 83–103, doi:10.1007/s10703-009-0067-z.
- [24] K. Lodaya (2000): *Sharpening the Undecidability of Interval Temporal Logic*. In: *Proc. 6th ASIAN*, LNCS 1961, Springer, pp. 290–298, doi:10.1007/3-540-44464-5_21.
- [25] A. Lomuscio & J. Michaliszyn (2013): *An Epistemic Halpern-Shoham Logic*. In: *Proc. 23rd IJCAI, IJCAI/AAAI*, pp. 1010–1016.
- [26] A. Lomuscio & J. Michaliszyn (2014): *Decidability of model checking multi-agent systems against a class of EHS specifications*. In: *Proc. 21st ECAI*, IOS Press, pp. 543–548, doi:10.3233/978-1-61499-419-0-543.
- [27] A. Lomuscio & J. Michaliszyn (2016): *Model Checking Multi-Agent Systems against Epistemic HS Specifications with Regular Expressions*. In: *Proc. 15th KR*, AAAI Press, pp. 298–308. Available at <http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12823>.
- [28] J. Marcinkowski & J. Michaliszyn (2014): *The Undecidability of the Logic of Subintervals*. *Fundamenta Informaticae* 131(2), pp. 217–240, doi:10.3233/FI-2014-1011.
- [29] A. Molinari, A. Montanari, A. Murano, G. Perelli & A. Peron (2016): *Checking interval properties of computations*. *Acta Informatica* 53(6-8), pp. 587–619, doi:10.1007/s00236-015-0250-1.
- [30] Alberto Molinari, Angelo Montanari, Adriano Peron & Pietro Sala (2016): *Model Checking Well-Behaved Fragments of HS: The (Almost) Final Picture*. In Chitta Baral, James P. Delgrande & Frank Wolter, editors: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, AAAI Press, pp. 473–483. Available at <http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12792>.
- [31] A. Montanari, G. Puppis & P. Sala (2015): *A decidable weakening of Compass Logic based on cone-shaped cardinal directions*. *Logical Methods in Computer Science* 11(4), doi:10.2168/LMCS-11(4:7)2015.
- [32] B. Moszkowski (1983): *Reasoning About Digital Circuits*. Ph.D. thesis, Dept. of Computer Science, Stanford University, Stanford, CA.
- [33] A. Pnueli (1977): *The temporal logic of programs*. In: *Proc. 18th FOCS*, IEEE Computer Society, pp. 46–57, doi:10.1109/SFCS.1977.32.
- [34] I. Pratt-Hartmann (2005): *Temporal propositions and their logic*. *Artificial Intelligence* 166(1-2), pp. 1–36, doi:10.1016/j.artint.2005.04.003.
- [35] J.P. Queille & J. Sifakis (1982): *Specification and verification of concurrent programs in CESAR*. In: *Proc. 5th SP*, LNCS 137, Springer, pp. 337–351, doi:10.1007/3-540-11494-7_22.
- [36] P. Roeper (1980): *Intervals and Tenses*. *Journal of Philosophical Logic* 9, pp. 451–469.
- [37] T. Schwentick & V. Weber (2007): *Bounded-Variable Fragments of Hybrid Logics*. In: *Proc. 24th STACS*, LNCS 4393 4393, Springer, pp. 561–572, doi:10.1007/978-3-540-70918-3_48.
- [38] Y. Venema (1990): *Expressiveness and Completeness of an Interval Tense Logic*. *Notre Dame Journal of Formal Logic* 31(4), pp. 529–547, doi:10.1305/ndjfl/1093635589.

Realizable and Context-Free Hyperlanguages

Hadar Frenkel

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

hadar.frenkel@cispa.de

Sarai Sheinvald

Department of Software Engineering, Braude College of Engineering, Karmiel, Israel

sarai@braude.ac.il

Hyperproperties lift conventional trace-based languages from a set of execution traces to a set of sets of executions. From a formal-language perspective, these are sets of sets of words, namely *hyperlanguages*. *Hyperautomata* are based on classical automata models that are lifted to handle hyperlanguages. *Finite hyperautomata* (NFH) have been suggested to express regular hyperproperties. We study the *realizability* problem for regular hyperlanguages: given a set of languages, can it be precisely described by an NFH? We show that the problem is complex already for singleton hyperlanguages. We then go beyond regular hyperlanguages, and study *context-free* hyperlanguages. We show that the natural extension to context-free hypergrammars is highly undecidable. We then suggest a refined model, namely *synchronous hypergrammars*, which enables describing interesting non-regular hyperproperties, while retaining many decidable properties of context-free grammars.

1 Introduction

Hyperproperties [10] generalize traditional trace properties [1] to *system properties*, i.e., from sets of traces to sets of sets of traces. A hyperproperty dictates how a system should behave in its entirety and not just based on its individual executions. Hyperproperties have been shown to be a powerful tool for expressing and reasoning about information-flow security policies [10] and important properties of cyber-physical systems [23] such as *sensitivity* and *robustness*, as well as consistency conditions in distributed computing such as *linearizability* [4]. Different types of logics, such as HyperLTL, HyperCTL* [9], HyperQPTL [19] and HyperQCTL* [11] have been suggested for expressing hyperproperties.

In the *automata-theoretic approach* both the system and the specification are modeled as automata whose language is the sets of execution traces of the system, and the set of executions that satisfy the specification [21, 20]. Then, problems such as model-checking [8] (“Does the system satisfy the property?”) and satisfiability (“Is there a system that satisfies the property?”) are reduced to decision problems for automata, such as containment (“Is the language of an automaton A contained in the language of automaton B ?”) and nonemptiness (“Is there a word that the automaton accepts?”). Finite-word and ω -regular automata are used for modeling trace specifications [22]. *Hyperautomata*, introduced in [5], generalize word automata to automata that run on sets of words. Just as hyperproperties describe a system in its entirety, the *hyperlanguages* of hyperautomata describe the language in its entirety.

The work in [5] focuses on *nondeterministic finite-word hyperautomata* (NFH), that are able to model *regular hyperlanguages*. An NFH \mathcal{A} uses *word variables* that are assigned words from a language \mathcal{L} , as well as a *quantification condition* over the variables, which describes the existential (\exists) and global (\forall) requirements from \mathcal{L} . The *underlying NFA* of \mathcal{A} runs on the set of words assigned to the word variables, all at the same time. The hyperlanguage of \mathcal{A} is then the set of all languages that satisfy the

Type of language	Quantification Type	Realizability
Finite	\exists^*, \forall^* $\forall\exists$	unrealizable (T.5) polynomial (T.6)
Infinite	$\exists^*, \forall^*, \exists^*\forall^*$	unrealizable (T.5)
Ordered	$\exists\forall\exists$	polynomial (T. 8)
partially Ordered	$\exists^*\forall\exists^*$	exponential (T.10)
Prefix-Closed Regular	$\exists\forall\exists^*$	polynomial (T.11)
Regular	$\exists^*\forall\exists^*$	doubly exponential (T.12) ¹

Table 1: Summary of realizability results for singleton hyperlanguages.

quantification condition. The decidability of the different decision problems for NFH heavily depends on the quantification condition. For example, nonemptiness of NFH is decidable for the conditions \forall^* (a sequence of \forall -quantifiers), \exists^* and $\exists^*\forall^*$, but is undecidable for $\forall\exists$. In [15] NFH are used to specify *multi-properties*, which express the behaviour of several models that run in parallel.

A natural problem for a model M for languages is *realizability*: given a language \mathcal{L} , can it be described by M ? For finite-word automata, for example, the answer relies on the number of equivalence classes of the Myhill-Nerode relation for \mathcal{L} . For hyperlanguages, we ask whether we can formulate a hyperproperty that precisely describes a given set of languages. We study this problem for NFH: given a set \mathcal{L} of languages, can we construct an NFH whose hyperlanguage is \mathcal{L} ? We can ask the question generally, or for specific quantification conditions. We focus on a simple case of this problem, where \mathcal{L} consists of a single language, (that is, $\mathcal{L} = \{\mathcal{L}\}$ for some language \mathcal{L}), which turns out to be non-trivial. In [12], the authors present automata constructions for safety regular hyperproperties. As such, they are strictly restricted only to \forall^* -conditions.

We show that for the simplest quantification conditions, \exists^* and \forall^* , no singleton hyperlanguage is realizable, and that single alternation does not suffice for a singleton hyperlanguage consisting of an infinite language. We show that when \mathcal{L} is finite, then $\{\mathcal{L}\}$ is realizable with a $\forall\exists$ quantification condition.

We then define *ordered* languages. These are languages that can be enumerated by a function f that can be described by an automaton that reads pairs of words: a word w , and $f(w)$. We show that for an ordered language \mathcal{L} , the hyperlanguage $\{\mathcal{L}\}$ is realizable with a quantification condition of $\exists\forall\exists$. We then generalize this notion to *partially ordered* languages, which are enumerated by a relation rather than a function. We show that for this case, $\{\mathcal{L}\}$ is realizable with a quantification condition of $\exists^*\forall\exists^*$.

Finally, we use ordered languages to realize singleton hyperlanguages consisting of regular languages: we show that when \mathcal{L} is a prefix-closed regular language, then it is partially ordered, and that an NFH construction for $\{\mathcal{L}\}$ is polynomial in the size of a finite automaton for \mathcal{L} . We then show that every regular language is partially ordered. Therefore, when \mathcal{L} is regular, then $\{\mathcal{L}\}$ is realizable with a quantification condition of $\exists^*\forall\exists^*$. The summary of our results is listed in Table 1.

In the second part of the paper, we go beyond regular hyperlanguages, and study *context-free hyperlanguages*. To model this class, we generalize context-free grammars (CFG) to *context-free hypergrammars* (CFHG), similarly to the generalization of finite-word automata to NFH: we use an *underlying CFG* that derives the sets of words that are assigned to the word variables in the CFHG G . The quantification condition of G defines the existential and global requirements from these assignments.

The motivation for context-free hyperlanguages is clear: they allow expressing more interesting hyperproperties. As a simple example, consider a robot which we want to return to its charging area before its battery is empty. This can be easily expressed with a CFHG with a \forall -condition, as we demonstrate in

¹See remark 2.

Problem		syncCFHGs	General CFHGs
Emptiness	\exists^*	polynomial (T.15)	polynomial (T.15)
	$\forall^*, \exists\forall^*$	polynomial (T.23)	undecidable (T.18)
	$\exists^*\forall^*$	undecidable (T.25)	undecidable (T.18)
Finite Membership	*	exponential (R.4)	exponential (T.17)
Regular Membership	\exists^*	exponential (R.5)	exponential (T.16)
	\forall^*	undecidable (T.24)	undecidable (T.24)

Table 2: Summary of decidability results for synchronous and general CFHGs.

Section 4, Example 3. Note that since the underlying property – *charging time is larger than action time* – is non-regular, NFH cannot capture this specification. Extending this example, using an $\exists\forall$ -condition we can express the property that all such executions of the robot are bounded, so that the robot cannot charge and act unboundedly.

Some aspects regarding context-free languages in the context of model-checking have been studied. In [13, 18] the authors explore model-checking of HyperLTL properties with respect to context-free models. There, the systems are context-free, but not the specifications. The work of [6] studies the verification of non-regular temporal properties, and [14] studies the synthesis problem of context-free specifications. These do not handle context-free *hyperproperties*.

While most natural decision problems are decidable for regular languages, this is not the case for CFG. For example, the universality (“Does the CFG derive all possible words?”) and containment problems for context-free languages are undecidable. The same therefore holds also for CFHG. However, the nonemptiness and membership (“Does the CFG derive the word w ?”) problems are decidable for CFG.

We study the various decision problems for CFHG. Specifically, We explore the nonemptiness problem (“Is there a language that the CFHG derives?”); and the membership problem (“Does the CFHG derive the language \mathcal{L} ?”). These problems correspond to the satisfiability and model-checking problems, respectively. We show that for general CFHG, most of these problems soon become undecidable (see Table 2). Some of the undecidability results are inherent to CFG. Some, however, are due to the *asynchronous* nature of CFHG: when the underlying CFG of a CFHG derives a set of words that are assigned to the word variables, it does not necessarily do so synchronously. For example, in one derivation step one word in the set may be added 2 letters, and another 1 letter. NFH read one letter at a time from every word in the set, and are hence naturally synchronous. In [3], the authors study asynchronous hyperLTL, which suffers from the same phenomenon.

We therefore define *synchronous context-free hyperlanguages*, which require synchronous reading of the set of words assigned to the word variables. We also define *synchronous CFHG* (syncCFHG), a fragment of CFHG in which the structure of the underlying CFG is limited in a way that ensures synchronous behavior. We prove that syncCFHG precisely captures the class of synchronous context-free hyperlanguages. Further, we show that some of the undecidable problems for CFHG, such as the nonemptiness problem for the \forall^* - and $\exists\forall^*$ -fragments, become decidable for syncCFHG.

2 Preliminaries

Hyperautomata We assume that the reader is familiar with the definitions of deterministic finite automata (DFA) and non-deterministic finite automata (NFA).

Definition 1. Let Σ be an alphabet. A hyperlanguage \mathcal{L} over Σ is a set of languages over Σ , that is, $\mathcal{L} \in 2^{2^{\Sigma^*}}$. A nondeterministic finite-word hyperautomaton (NFH) is a tuple $\mathcal{A} = \langle \Sigma, X, Q, Q_0, F, \delta, \alpha \rangle$,

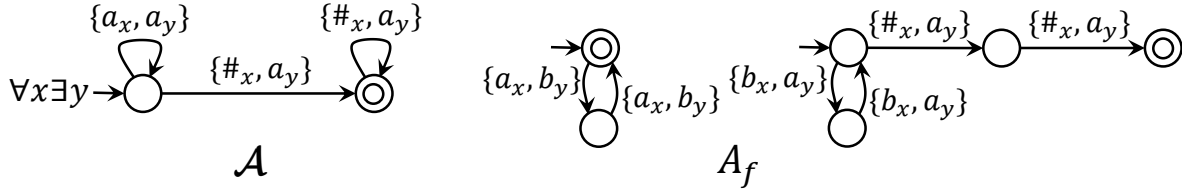


Figure 1: The NFH \mathcal{A} (left), whose hyperlanguage is the set of infinite languages over $\{a\}$, and the NFA A_f that computes f (right).

where X is a finite set of word variables, and $\alpha = \mathbb{Q}_1 x_1 \cdots \mathbb{Q}_k x_k$ is a quantification condition, where $\mathbb{Q}_i \in \{\exists, \forall\}$ for every $i \in [1, k]$, and $\langle \hat{\Sigma}, Q, Q_0, F, \delta \rangle$ forms an underlying NFA over $\hat{\Sigma} = (\Sigma \cup \{\#\})^X$.

Let \mathcal{L} be a language. We represent an assignment $v : X \rightarrow \mathcal{L}$ as a *word assignment* w_v , which is a word over the alphabet $(\Sigma \cup \{\#\})^X$ (that is, assignments from X to $(\Sigma \cup \{\#\})^*$), where the i 'th letter of w_v represents the k i 'th letters of the words $v(x_1), \dots, v(x_k)$ (in case that the words are not of equal length, we “pad” the end of the shorter words with $\#$ -symbols). We represent these k i 'th letters as an assignment denoted $\{\sigma_{1x_1}, \sigma_{2x_2}, \dots, \sigma_{kx_k}\}$, where x_j is assigned σ_j . For example, the assignment $v(x_1) = aa$ and $v(x_2) = abb$ is represented by the word assignment $w_v = \{a_{x_1}, a_{x_2}\} \{a_{x_1}, b_{x_2}\} \{\#_{x_1}, b_{x_2}\}$.

The acceptance condition for NFH is defined with respect to a language \mathcal{L} , the underlying NFA \hat{A} , the quantification condition α , and an assignment $v : X \rightarrow \mathcal{L}$.

- For $\alpha = \varepsilon$, define $\mathcal{L} \vdash_v (\alpha, \hat{A})$ if $w_v \in \mathcal{L}(\hat{A})$.
- For $\alpha = \exists x. \alpha'$, define $\mathcal{L} \vdash_v (\alpha, \hat{A})$ if there exists $w \in \mathcal{L}$ s.t. $\mathcal{L} \vdash_{v[x \rightarrow w]} (\alpha', \hat{A})$.
- For $\alpha = \forall x. \alpha'$, define $\mathcal{L} \vdash_v (\alpha, \hat{A})$ if $\mathcal{L} \vdash_{v[x \rightarrow w]} (\alpha', \hat{A})$ for every $w \in \mathcal{L}$.²

When α includes all of X , then membership is independent of the assignment, and we say that \mathcal{A} *accepts* \mathcal{L} , and denote $\mathcal{L} \in \mathfrak{L}(\mathcal{A})$.

Definition 2. Let \mathcal{A} be an NFH. The hyperlanguage of \mathcal{A} , denoted $\mathfrak{L}(\mathcal{A})$, is the set of all languages that \mathcal{A} accepts. When the quantification condition α of an NFH \mathcal{A} is $\mathbb{Q}_1 x_1. \mathbb{Q}_2 x_2 \cdots \mathbb{Q}_k x_k$, we denote \mathcal{A} as being a $\mathbb{Q}_1 \mathbb{Q}_2 \dots \mathbb{Q}_k$ -NFH (or, sometimes, as an α -NFH).

Example 1. Consider the NFH \mathcal{A} depicted in Figure 1, over the alphabet $\{a\}$. The quantification condition $\forall x. \exists y$ requires that in a language \mathcal{L} accepted by \mathcal{A} , for every word u_1 that is assigned to x , there exists a word u_2 that is assigned to y such that the joint run of u_1, u_2 is accepted by the underlying NFA \hat{A} of \mathcal{A} . The NFA \hat{A} requires that the word assigned to y is longer than the word assigned to x : once the word assigned to x ends (and the padding $\#$ begins), the word assigned to y must still read at least one more a . Therefore, \mathcal{A} requests that for every word in \mathcal{L} , there exists a longer word in \mathcal{L} . This holds iff \mathcal{L} is infinite. Therefore, the hyperlanguage of \mathcal{A} is the set of all infinite languages over $\{a\}$.

Context-Free Grammars

Definition 3. A context-free grammar (CFG) is a tuple $G = \langle \Sigma, V, V_0, P \rangle$, where Σ is an alphabet, V is a set of grammar variables, $V_0 \in V$ is an initial variable, and $P \subseteq V \times (V \cup \Sigma)^*$ is a set of grammar rules.

We say that w is a *terminal word* if $w \in \Sigma^*$. Let $v \in V$ and $\alpha, \beta \in (V \cup \Sigma)^*$.

²In case that α begins with \forall , membership holds vacuously with the empty language. We restrict the discussion to satisfaction by nonempty languages.

- We say that v derives α if $(v, \alpha) \in P$. We then denote $v \rightarrow \alpha$.
- We denote $\alpha \Rightarrow \beta$ if there exist $v \in V$ and $\alpha_1, \alpha_2, \beta' \in (V \cup \Sigma)^*$ such that $v \rightarrow \beta'$, $\alpha = \alpha_1 v \alpha_2$ and $\beta = \alpha_1 \beta' \alpha_2$.
- We say that α derives β , or that β is derived by α , if there exists $n \in \mathbf{N}$ and $\alpha_1, \dots, \alpha_n \in (V \cup \Sigma)^*$ such that $\alpha_1 = \alpha$, $\alpha_n = \beta$ and $\forall 1 \leq i < n : \alpha_i \Rightarrow \alpha_{i+1}$. We then denote $\alpha \Rightarrow^* \beta$.

The language of a CFG G is the set of all terminal words that are derived by the initial variable. That is, $\mathcal{L}(G) = \{w \in \Sigma^* \mid V_0 \Rightarrow^* w\}$.

3 Realizability of Regular Hyperlanguages

Every NFH \mathcal{A} defines a set of languages \mathcal{L} . In the *realizability problem* for NFH, we are given a hyperlanguage \mathcal{L} , and ask whether there exists an NFH \mathcal{A} such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. The answer may depend on the quantification condition that we allow using. In this section we study the realizability problem for singleton hyperlanguages, which turns out to be non-trivial. We show that while for finite languages we can construct a $\forall\exists$ -NFH, such a condition cannot suffice for infinite languages. Further, we show that a general regular language requires a complex construction and quantification condition.

Definition 4. Let \mathcal{L} be a hyperlanguage. For a sequence of quantifiers $\alpha = \mathbb{Q}_1 \dots \mathbb{Q}_k$, we say that \mathcal{L} is α -realizable if there exists an NFH \mathcal{A} with a quantification condition $\mathbb{Q}_1 x_1 \dots \mathbb{Q}_k x_k$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}$.

We first define some operations and notations on the underlying NFA of NFH we use in our proofs.

For a word w , the NFA A_w is an NFA for $\{w\}$.

Let A_1, A_2 be NFA, and let $A_1 \uparrow^\# = \langle \Sigma, Q, q_0, \delta_1, F_1 \rangle$ and $A_2 \uparrow^\# = \langle \Sigma, P, p_0, \delta_2, F_2 \rangle$, where $A_i \uparrow^\#$ is an NFA for the language $\mathcal{L}(A_i) \cdot \#^*$. We define the *composition* $A_1 \otimes A_2$ of A_1 and A_2 to be an NFA over $\Sigma_2 = (\Sigma \cup \{\#\})^{\{x,y\}}$, defined as $A_1 \otimes A_2 = \langle \Sigma_2, Q \times P, (q_0, p_0), \delta, F_1 \times F_2 \rangle$, where for every $(q, \sigma, q') \in \delta_1$, $(p, \tau, p') \in \delta_2$, we have $((q, p), \{\sigma_x, \tau_y\}, (q', p')) \in \delta$. That is, $A_1 \otimes A_2$ is the composition of A_1 and A_2 , which follows both automata simultaneously on two words (adding padding by $\#$ when necessary). A word assignment $\{x \mapsto w_1, y \mapsto w_2\}$ is accepted by $A_1 \otimes A_2$ iff $w_1 \in \mathcal{L}(A_1)$ and $w_2 \in \mathcal{L}(A_2)$ (excluding the $\#$ -padding).

We also define $A_1 \oplus A_2$ of A_1 and A_2 in a similar way, but the transitions are restricted to equally labeled letters. That is, for every $(q, \sigma, q') \in \delta_1$, $(p, \sigma, p') \in \delta_2$, we have $((q, p), \{\sigma_x, \sigma_y\}, (q', p')) \in \delta$. A run of the restricted composition then describes the run of A_1 and A_2 on the same word.

We generalize the definition of both types of compositions to a sequence of k NFA A_1, A_2, \dots, A_k , forming NFA $\otimes_{i=1}^k A_i$ and $\oplus_{i=1}^k A_i$ over $(\Sigma \cup \{\#\})^{\{x_1, x_2, \dots, x_k\}}$, in the natural way. When all NFA are equal to A , we denote this composition by $A^{\otimes k}$ (or $A^{\oplus k}$). When we want to explicitly name the variables x_1, x_2, \dots, x_k in the compositions, we denote $\otimes_{i=1}^k A_i[x_1, \dots, x_k]$ (or $\oplus_{i=1}^k A_i[x_1, \dots, x_k]$).

We also generalize the notion of composition to NFA over $(\Sigma \cup \{\#\})^X$ in the natural way. That is, for NFA A_1 and A_2 with sets of variables X and Y , respectively, the NFA $A_1 \otimes A_2$ is over $X \cup Y$, and follows both NFA simultaneously on both assignments (if $X \cap Y \neq \emptyset$, we rename the variables).

We study the realizability problem for the case of singleton hyperlanguages, that is, hyperlanguages of the type $\{\mathcal{L}\}$. We begin with a few observations on unrealizability of this problem, and show that general singleton hyperlanguages cannot be realized using simple quantification conditions.

3.1 Unrealizability

For the homogeneous quantification conditions, we have that a \forall -NFH \mathcal{A} accepts a language \mathcal{L} iff \mathcal{A} accepts every $\mathcal{L}' \subseteq \mathcal{L}$. Therefore, a hyperlanguage $\{\mathcal{L}\}$ is not \forall -realizable for every \mathcal{L} that is not a

singleton. The same holds for every \forall^* -NFH.

An \exists -NFH \mathcal{A} accepts a language \mathcal{L} iff \mathcal{L} contains some word that is accepted by $\hat{\mathcal{A}}$. Thus if \mathcal{A} is nonempty, its hyperlanguage is infinite, and clearly not a singleton. The same holds for every \exists^* -NFH.

Now, consider an $\exists^k \forall^m$ -NFH \mathcal{A} . As shown in [5], \mathcal{A} is nonempty iff it accepts a language whose size is at most k . Therefore, $\{\mathcal{L}\}$ is not $\exists^k \forall^m$ -realizable for every \mathcal{L} such that $|\mathcal{L}| > k$.

As we show in Theorem 6, if \mathcal{L} is finite then $\{\mathcal{L}\}$ is $\forall\exists$ -realizable. We now show that if \mathcal{L} is infinite, then $\{\mathcal{L}\}$ is not $\forall\exists$ -realizable. Assume otherwise by contradiction, and let \mathcal{A} be a $\forall x.\exists y$ -NFH that accepts $\{\mathcal{L}\}$. Then for every $w \in \mathcal{L}$ there exists $u \in \mathcal{L}$ such that $w_{[x \rightarrow w][y \rightarrow u]} \in \mathcal{L}(\hat{\mathcal{A}})$. Let w_1 be some word in \mathcal{L} . We construct an infinite sequence w_1, w_2, \dots of words in \mathcal{L} , as follows. For every w_i , let w_{i+1} be a word in \mathcal{L} such that $w_{[x \rightarrow w_i][y \rightarrow w_{i+1}]} \in \mathcal{L}(\hat{\mathcal{A}})$. If $w_i = w_j$ for some $i < j$, then the language $\{w_i, w_{i+1}, \dots, w_j\}$ is accepted by \mathcal{A} , and so \mathcal{L} is not the only language that \mathcal{A} accepts. Otherwise, all words in the sequence are distinct. Then, the language $\mathcal{L}_i = \{w_i, w_{i+1}, \dots\}$ is accepted by \mathcal{A} for every $i > 1$, and $\mathcal{L}_i \subset \mathcal{L}$. In both cases, \mathcal{L} is not the only language that \mathcal{A} accepts, and so $\{\mathcal{L}\}$ is not $\forall\exists$ -realizable.

To conclude, we have the following.

Theorem 5. *If \mathcal{L} contains more than one word, then $\{\mathcal{L}\}$ is not \forall^* -realizable and not \exists^* -realizable. If \mathcal{L} contains more than k words then $\{\mathcal{L}\}$ is not $\exists^k \forall^*$ -realizable. If \mathcal{L} is infinite then $\{\mathcal{L}\}$ is not $\exists^* \forall^*$ -realizable and not $\forall\exists$ -realizable.*

For positive realizability results, we first consider a simple case of a hyperlanguage consisting of a single finite language.

Theorem 6. *Let \mathcal{L} be a finite language. Then $\{\mathcal{L}\}$ is $\forall\exists$ -realizable.*

Proof. Let $\mathcal{L} = \{w_1, w_2, \dots, w_k\}$. We construct a $\forall\exists$ -NFH \mathcal{A} for \mathcal{L} , whose underlying NFA $\hat{\mathcal{A}}$ is the union of all NFA $A_{w_i} \otimes A_{w_{i+1} \pmod k}$. Let $\mathcal{L}' \in \mathcal{L}(\mathcal{A})$. Since $\hat{\mathcal{A}}$ can only accept words in \mathcal{L} , we have that $\mathcal{L}' \subseteq \mathcal{L}$. Since \mathcal{A} requires, for every $w_i \in \mathcal{L}'$, the existence of $w_{i+1 \pmod k}$, and since $\mathcal{L}' \neq \emptyset$, we have that by induction, $w_i \in \mathcal{L}'$ implies $w_{i+j \pmod k} \in \mathcal{L}'$ for every $1 \leq j \leq k$. Therefore, $\mathcal{L} \subseteq \mathcal{L}'$. \square

3.2 Realizability of Ordered Languages

Since every language is countable, we can always order its words. We show that for an ordering of a language \mathcal{L} that is *regular*, that is, can be computed by an NFA, $\{\mathcal{L}\}$ can be realized by an $\exists\forall\exists$ -NFH.

Definition 7. *Let \mathcal{L} be a language. We say that a function $f : \mathcal{L} \rightarrow \mathcal{L}$ is \mathcal{L} -regular if there exists an NFA A_f over $(\Sigma \cup \{\#\})^{\{x,y\}}$ such that for every $w \in \mathcal{L}$, it holds that $f(w) = u$ iff $w_{[x \rightarrow w][y \rightarrow u]} \in \mathcal{L}(A_f)$. We then say that A_f computes f .*

We say that a language \mathcal{L} is ordered if the words in \mathcal{L} can be arranged in a sequence w_1, w_2, \dots such that there exists an \mathcal{L} -regular function such that $f(w_i) = w_{i+1}$ for every $i \geq 0$.

Example 2. Consider the language $\{a^{2i}, b^{2i} \mid i \in \mathbb{N}\}$, and a function $\forall i \in \mathbb{N} : f(a^{2i}) = b^{2i}, f(b^{2i}) = a^{2i+2}$, which matches the sequence $\varepsilon, a^2, b^2, a^4, b^4, \dots$. The function f can be computed by the NFA A_f depicted in Figure 1, which has two components: one that reads a^{2i} on x and b^{2i} on y , and one that reads b^{2i} on x and a^{2i+2} on y .

Theorem 8. *Let \mathcal{L} be an ordered language. Then $\{\mathcal{L}\}$ is $\exists\forall\exists$ -realizable.*

Proof. Let $\mathcal{L} = \{w_1, w_2, \dots\}$ be an ordered language via a regular function f , and let A_f be an NFA that computes f . We construct an $\exists x_1 \forall x_2 \exists x_3$ -NFH \mathcal{A} for $\{\mathcal{L}\}$ by setting its underlying NFA to be $\hat{\mathcal{A}} = A_{w_1} \otimes A_f[x_1, x_2, x_3]$. Intuitively, \mathcal{A} creates a ‘‘chain-reaction’’: A_{w_1} requires the existence of w_1 , and A_f requires the existence of w_{i+1} for every w_i . By the definition of f , only words in \mathcal{L} may be assigned to x_2 . Therefore, $\mathcal{L}(\mathcal{A}) = \{\mathcal{L}\}$. \square

We now generalize the definition of ordered languages, by allowing several minimal words instead of one, and allowing each word to have several successors. The computation of such a language then matches a *relation* over the words, rather than a function.

Definition 9. We say that a language \mathcal{L} is m, k -ordered, if there exists a relation $R \subseteq \mathcal{L} \times \mathcal{L}$ such that:

- There exist exactly m words $w \in \mathcal{L}$ such that $(u, w) \notin R$ for every $u \neq w \in \mathcal{L}$ (that is, there are m minimal words).
- $R \subseteq S$ for a total order S of \mathcal{L} with a minimal element.
- For every $w \in \mathcal{L}$ there exist $1 \leq i \leq k$ successor words: words u such that $(w, u) \in R$.
- There exists an NFA A_R over $(\Sigma \cup \{\#\})^{\{x, y\}}$ such that for every $u, v \in \mathcal{L}$, it holds that $R(u, v)$ iff $w_{[x \mapsto u][y \mapsto v]} \in \mathcal{L}(A_R)$.

We then say that A_R computes \mathcal{L} . We call \mathcal{L} partially ordered if there exist m, k such that \mathcal{L} is m, k -ordered.

Theorem 10. Let \mathcal{L} be m, k -ordered. Then $\{\mathcal{L}\}$ is $\exists^m \forall \exists^k$ -realizable.

Proof. Let $A_R = \langle (\Sigma \cup \{\#\})^{\{x, y\}}, Q, q_0, \delta, F \rangle$ be a DFA that computes \mathcal{L} . We construct an $\exists^m \forall \exists^k$ -NFH \mathcal{A} for \mathcal{L} , as follows. The quantification condition of \mathcal{A} is $\exists x_1 \dots \exists x_m \forall z \exists y_1 \dots \exists y_k$. The x -variables are to be assigned u_1, \dots, u_m , the m minimal words of R . We set $A_U = \bigotimes_{i=1}^m A_{u_i}[x_1, \dots, x_m]$.

The underlying NFA $\hat{\mathcal{A}}$ of \mathcal{A} comprises of an NFA A_i for every $1 \leq i \leq k$. Let \mathcal{L}_i be the set of words in \mathcal{L} that have exactly i successors. Intuitively, A_i requires, for every word $w \in \mathcal{L}_i$ that is assigned to z , the existence of the i successors of w .

To do so, we construct an NFA B_i over z, y_1, \dots, y_i that accepts $w_{[z \mapsto w][y_1 \mapsto w_1] \dots [y_i \mapsto w_i]}$, for every word w and its successors w_1, \dots, w_i . The construction of B_i requires that: (1) all assignment to y -variables are successors of the assignment to z , by basing B_i on a composition of A_R , and (2) y_1, \dots, y_i are all assigned different words. This is done by keeping track of the pairs of assignments to y -variables that at some point read different letters. The run may accept only once all pairs are listed. We finally set $A_i = A_U \otimes B_i$, and require $y_{i+1} \dots y_k$ to be equally assigned to z .

Since $R \subseteq S$ and R has minimal elements, for every word $w \in \mathcal{L}$ there exists a sequence w_0, w_1, \dots, w_t such that $w_t = w$, and w_0 is minimal, and $R(w_j, w_{j+1})$ for every $j \in [0, t-1]$. The NFH \mathcal{A} then requires w_0 (via A_u), and for every w_j , requires the existence of all of its successors, according to their number, and in particular, the existence of w_{j+1} (via A_i). Therefore, \mathcal{A} requires the existence of w_t . On the other hand, by construction, every word that is assigned to z is in \mathcal{L} . Therefore, $\mathcal{L}(\mathcal{A}) = \{\mathcal{L}\}$.

The size of A_U is linear in $|U|$, and the size of A_i is exponential in k and in $|A|$. □

3.3 Realizability of Regular Languages

We now show that every regular language \mathcal{L} is partially ordered. To present the idea more clearly, we begin with a simpler case of prefix-closed regular languages, and then proceed to general regular languages. A prefix-closed regular language \mathcal{L} has a DFA A in which every accepting state is only reachable by accepting states. We use the structure of A to define a relation that partially orders \mathcal{L} .

Theorem 11. Let \mathcal{L} be a non-empty prefix-closed regular language. Then \mathcal{L} is partially ordered.

Proof. Let $A = \langle \Sigma, Q, q_0, \delta, F \rangle$ be a DFA for \mathcal{L} . Then for every $p, q \in Q$, if $q \in F$ and q is reachable from p , then $p \in F$. Let k be the maximal number of transitions from a state $q \in F$ to its neighboring accepting states. We show that \mathcal{L} is $1, k$ -ordered. We define a relation R as follows. Since \mathcal{L} is prefix-closed, we have that $\varepsilon \in \mathcal{L}$. We set it to be the minimal element in R .

Let $q \in Q$, and let $\{(q, \sigma_1, p_1), \dots, (q, \sigma_m, p_m)\}$ be the set of transitions in δ from q to accepting states. For every word $w \in \mathcal{L}$ that reaches q , we set $(w, w\sigma_1), \dots, (w, w\sigma_m) \in R$. For every word $w \in \mathcal{L}$ that reaches a state q from which there are no transitions to accepting states, we set $(w, w) \in R$.

It holds that the number of successors for every $w \in \mathcal{L}$ is between 1 and k . Further, $R \subseteq S$ for the length-lexicographic order S of \mathcal{L} .

We construct an NFA A_R for R by replacing every transition labeled σ with $\{\sigma_x, \sigma_y\}$ and adding a state p' , which is the only accepting state. For every $q \in F$, we add a transition $(q, \{\#_x, \sigma_y\}, p')$ for every $(q, \sigma, p) \in \delta$ such that $p \in F$. If q has no transitions to accepting states in A , then we add $(q, \{\#_x, \#_y\}, p')$. A_R then runs on word assignments $w_{[x \rightarrow w][y \rightarrow u]}$ such that $u = w\sigma$ for some σ , such that $w, u \in \mathcal{L}$, or $w_{[x \rightarrow w][y \rightarrow w]}$ if w cannot be extended to a longer word in \mathcal{L} . Therefore, A_R computes \mathcal{L} . \square

Remark 1. The construction in the proof of Theorem 10 is exponential, due to the composition of several automata. In the case of prefix-closed languages, the successors of a word $w \in \mathcal{L}$ are all of the type $w\sigma$. Therefore, it suffices to extend every transition in A to $\{\sigma_{x_1}, \sigma_{x_2}, \dots, \sigma_{x_k}\}$, and to add a transition from every $q \in F$ to a new accepting state with all letters leading from q to an accepting state. Composed with a single-state DFA for ε , we get an $\exists \forall \exists^k$ -NFH for $\{\mathcal{L}\}$, whose size is polynomial in $|A|$.

We now turn to prove the realizability of $\{\mathcal{L}\}$ for every regular language \mathcal{L} . The proof relies on a similar technique to that of Theorem 11: a relation that computes $\{\mathcal{L}\}$ requires, for every word $w \in \mathcal{L}$, the existence of a longer word $w' \in \mathcal{L}$. Here, w' is not simply the extension of w by a single letter, but a pumping of w by a single cycle in a DFA for \mathcal{L} .

Theorem 12. *Let \mathcal{L} be a regular language. Then $\{\mathcal{L}\}$ is partially ordered.*

Proof. Let $A = \langle \Sigma, Q, q_0, \delta, F \rangle$ be a DFA for \mathcal{L} . We mark by P the set of words that reach accepting states from q_0 along a simple path. For a state $q \in Q$, we mark by C_q the set of words that reach q from q along a simple cycle. Note that P and C_q are finite for every $q \in Q$. Let $n = |P|$, and let $m = \sum_{q \in Q} |C_q|$. We show that \mathcal{L} is n, m -ordered, by defining an appropriate relation R .

The set of minimal words in R is P . The successors of a word $w \in \mathcal{L}$ are w itself (that is, R is reflexive), and every possible pumping of w by a single simple cycle that precedes all other cycles within the run of A on w . That is, for a state q that is reached by a prefix u of w along a simple path, and for a word c read along a simple cycle from q to itself, the word ucv is a successor of w in R , where $w = uv$.

To see that the only minimal words in R are P , let $w = \sigma_1 \sigma_2 \dots \sigma_k \in \mathcal{L}$, and let $r = (q_0, q_1, \dots, q_k)$ be the accepting run of A on w . If all states in r are unique, then $w \in P$. Otherwise, we set $w_t = w$, and repeatedly remove simple cycles from r : let j be a minimal index for which there exists $j' > j$ such that $q_j = q_{j'}$ and such that $q_{j+1}, \dots, q_{j'}$ are unique. We define $w_{t-1} = w_1 \dots w_j w_{j'+1} \dots w_k$. We repeat this process until we reach a run in which all states are unique, which matches a word $w_0 \in P$. The sequence of words w_t, w_{t-1}, \dots, w_0 we obtain is such that $(w_i, w_{i+1}) \in R$ for every $i \in [0, t-1]$.

It is easy to see that $R \subseteq S$ for the length-lexicographical order S of \mathcal{L} . Additionally, every $w \in \mathcal{L}$ has between 1 and m successors. We now construct an NFA A_R for R .

A_R is the union of several components, described next. Let A_q be the DFA obtained from A by setting its only accepting state to be q . For every $p \in Q$ and for every $c \in C_p$, we construct an NFA $B_{c,q}$, which pumps a word read along a run that reaches q and traverses p , by c . The NFA $B_{c,q}$ comprises two copies A_1, A_2 of A , where the copy q_2 of q in A_2 is the only accepting state. The word c is read between A_1 to A_2 , from p_1 and p_2 .

We construct an NFA $A_{c,q}$ by composing $B_{c,q}$ and A_q , and making sure that $B_{c,q}$ reads the same word as A_q , pumped by c . That is, if A_q reads a word uv , where u reaches p , then $B_{c,q}$ reads ucv . To this end, while $B_{c,q}$ is in A_1 , the DFA A_q and $B_{c,q}$ both advance on the same letters. When $B_{c,q}$ leaves A_1 to read c

followed by the suffix v in A_2 , the composition remembers, via states, the previous (up to) $|c|$ letters read by A_q , to make sure that once $B_{c,q}$ finishes reading uc , it reads the same suffix v as A_q did. The NFA A_R is then the union of $A_{c,q}$ for every $q \in Q, c \in \bigcup_{p \in Q} C_p$. To accept the reflexive pairs as well, we union all the components with an additional component $A \oplus A$.

The size of every $A_{c,q}$ is exponential in c , due to the need to remember the previous c letters. There are exponentially many simple paths and cycles in A . Therefore, we have that the size of A_R is exponential in $|A|$. Combined with the exponential blow-up involved in the proof of Theorem 10, we have that an NFH for $\{\mathcal{L}\}$ is doubly-exponential in the $|A|$. \square

Remark 2. Using automatic structures [17] and relying on the length-lexicographical order S , one can prove the existence of an $\exists\forall\exists$ -NFH \mathcal{A} for $\{\mathcal{L}\}$, which is smaller and simpler than the one we present in Theorem 12. Indeed, one can phrase the direct successor relation in \mathcal{L} with respect to S using the First Order Logic (FOL) formula $\varphi(x, y) = \mathcal{L}(x) \wedge \mathcal{L}(y) \wedge S(x, y) \wedge \forall(z). (z \neq y) \rightarrow (\neg(S(x, z) \wedge S(z, y)))$. Since S is NFA-realizable, and since every relation expressible by FOL over an automatic structure is regular [17], we have that φ is NFA-realizable. We can then construct \mathcal{A} , requiring the existence of a minimal word in \mathcal{L} with respect to S , together with the requirement of the existence of a successor for every $w \in \mathcal{L}$.

While this construction is polynomial, it does not directly rely on the structure of A . Since in this paper we wish to lay the ground for richer realizable fragments, in which relying on the underlying graph structures may be useful, we present it here.

4 Context-Free Hypergrammars

We now go beyond regular hyperlanguages, and define and study *context-free hyperlanguages*. We begin with a natural definition for context-free hypergrammars (CFHG), based on the definition of NFH, and then identify a more decidable fragment of CFHG, namely *synchronized CFHG*.

Definition 13. A context-free hypergrammar (CFHG) is a tuple $\langle \Sigma, X, V, V_0, P, \alpha \rangle$, where X and α are as in NFH, and where $\hat{G} = \langle \hat{\Sigma}, V, V_0, P \rangle$ is a CFG over the alphabet $\hat{\Sigma} = (\Sigma \cup \{\#\})^X$.

Definition 1 defines word assignments for NFH, where the $\#$ -symbol may only appear at the end of a word. This is naturally enforced by the nature of the underlying NFA. For the most general case of hypergrammars, we consider words in which $\#$ can appear anywhere in the word. In Section 4.1 we allow $\#$ to occur only at the end of the word. For a word $w \in \Sigma^*$ we define the *set of words* $w \uparrow_{\#}$ to be the set of all words that are obtained from w by adding $\#$ -symbols in arbitrary locations in w . For $w \in (\Sigma \cup \{\#\})^*$, we define the *word* $w \downarrow_{\#}$ to be the word obtained from w by removing all occurrences of $\#$.

The acceptance condition for CFHG is defined with respect to a language \mathcal{L} , the underlying CFG \hat{G} , the quantification condition α , and an assignment $u : X \rightarrow \mathcal{L}$.

1. For $\alpha = \varepsilon$, define $\mathcal{L} \vdash_v (\alpha, \hat{G})$ if $w_u \in \mathcal{L}(\hat{G})$.
2. For $\alpha = \exists x. \alpha'$, define $\mathcal{L} \vdash_u (\alpha, \hat{G})$ if there exist $w \in \mathcal{L}$ and $w_{\#} \in w \uparrow_{\#}$ s.t. $\mathcal{L} \vdash_{u[x \rightarrow w_{\#}]} (\alpha', \hat{G})$.
3. For $\alpha = \forall x. \alpha'$, define $\mathcal{L} \vdash_u (\alpha, \hat{G})$ if for every $w \in \mathcal{L}$ there exists $w_{\#} \in w \uparrow_{\#}$ s.t. $\mathcal{L} \vdash_{u[x \rightarrow w_{\#}]} (\alpha', \hat{G})$.

When α includes all of X , we say that G *derives* \mathcal{L} (or that G *accepts* \mathcal{L}), and denote $\mathcal{L} \in \mathfrak{L}(G)$.

Definition 14. Let G be a CFHG. The hyperlanguage of G , denoted $\mathfrak{L}(G)$, is the set of all languages that G derives. We denote G as being a $\mathbb{Q}_1 \mathbb{Q}_2 \dots \mathbb{Q}_k$ -CFHG similarly as with NFH.

Henceforth we assume that (1) the underlying grammar \hat{G} does not contain variables and rules that derive no terminal words (these can be removed); and (2) there are no rules of the form $v \rightarrow \varepsilon$ except for possibly $V_0 \rightarrow \varepsilon$. Every CFG can be converted to a CFG that satisfies these conditions [16].

Example 3. Consider the robot scenario described in Section 1, and the $\forall x$ -CFHG G_1 with the rules

$$\begin{aligned} P_1 &:= V_0 \rightarrow \{c_x\}V_0\{a_x\} \mid \{c_x\}V_1 \\ V_1 &\rightarrow \{c_x\}V_1 \mid \{c_x\} \end{aligned}$$

The letters a and c correspond to *action* and *charge*, respectively. Then, $\mathfrak{L}(G_1)$ is the set of all languages in which the robot has enough battery to act.

Consider now the CFHG $G_2 = \langle \{a, c\}, \{x_1, x_2\}, \{V_0, V_1\}, V_0, P_2, \exists x_1 \forall x_2 \rangle$ where

$$\begin{aligned} P_2 &:= V_0 \rightarrow \{c_{x_1}, c_{x_2}\}V_0\{a_{x_1}, a_{x_2}\} \mid \{c_{x_1}, c_{x_2}\}V_1\{a_{x_1}, \#_{x_2}\} \mid \{c_{x_1}, c_{x_2}\}V_1\{a_{x_1}, a_{x_2}\} \\ V_1 &\rightarrow \{c_{x_1}, \#_{x_2}\}V_1\{a_{x_1}, \#_{x_2}\} \mid \{c_{x_1}, \#_{x_2}\} \mid \{c_{x_1}, c_{x_2}\} \end{aligned}$$

We now require that the robot only has one additional unit of charging (unlike in G_1). In addition, we require an upper bound (assigned to x_1) on the charging and action times. All other words in the language (assigned to x_2) correspond to shorter computations.

We now study the nonemptiness and membership problems for CFHGs. When regarding a CFHG as a specification, these correspond to the model-checking and satisfiability problems.

Theorem 15. *The nonemptiness problem for \exists^* -CFHG is in P.*

Proof. According to the semantics of the \exists -requirement, an \exists^* -CFHG G derives a language \mathcal{L} if \hat{G} accepts a word assignment that corresponds to words in \mathcal{L} . Therefore, it is easy to see that G is nonempty iff \hat{G} is nonempty. Since the nonemptiness of CFG is in P [16], we are done. \square

Theorem 16. *The membership problem for a regular language in an \exists^* -CFHG is in EXPTIME.*

Proof. Let $A = \langle \Sigma, Q, Q_0, \delta, F \rangle$ be an NFA and let G be a CFHG with $\alpha = \exists x_1 \cdots \exists x_k$. In order to check whether $\mathcal{L}(A) \in \mathfrak{L}(G)$, we need to check whether there exists a subset of $\mathcal{L}(A)$ of size k or less, that can be accepted as a word assignment by \hat{G} . Since \hat{G} derives words over $\Sigma \cup \{\#\}$, we first construct the NFA $A \uparrow \#$, that accepts all $\#$ -paddings of words in $\mathcal{L}(A)$. We can do so easily by adding a self-loop labeled $\#$ to every state in A . We then compute $(A \uparrow \#)^{\otimes k}$ to allow different paddings for different words (an exponential construction), intersect the resulting automaton with \hat{G} and test the intersection for nonemptiness. Context-free languages are closed under intersection with regular languages via a polynomial construction. In addition, if the grammar is given in Chomsky Normal Form [7], then checking the emptiness of the intersection is polynomial in the sizes of the grammar and automaton. As the conversion to Chomsky normal form is also polynomial, we get that the entire procedure is exponential, due to the size of $(A \uparrow \#)^{\otimes k}$. \square

Theorem 17. *The membership problem for a finite language in a CFHG is in EXPTIME.*

Proof. Let \mathcal{L} be a finite language and let G be a CFHG with variables $\{x_1, \dots, x_k\}$. Since \mathcal{L} is finite, we can construct every assignment of words in \mathcal{L} to the variables in G , and check if it is accepted by \hat{G} . Similarly to the proof of Theorem 16, to do so, we use an NFA A_w whose language is the set $w \uparrow \#$, for every $w \in \mathcal{L}$. For an assignment $v = [x_1 \mapsto w_1] \dots [x_k \mapsto w_k]$, we construct $\bigotimes_{i=1}^k A_{w_i}$, and check the nonemptiness of its intersection with \hat{G} . As in Theorem 16, this procedure is exponential in the length

of the words in \mathcal{L} and in $|G|$. Since we can finitely enumerate all assignments, we can check whether the quantification condition α of G is satisfied. Enumerating all assignments amounts to traversing the decision tree dictated by α , which is exponential in $|\alpha|$. Therefore, the entire procedure can be done in exponential time in $|G|$ and \mathcal{L} . \square

Theorem 18. *The emptiness problem for \forall^* -CFHG and $\exists\forall$ -CFHG is undecidable.*

Proof. We show reductions from the Post correspondence problem (PCP). A PCP instance is a set of pairs of the form $[a_1, b_1], \dots, [a_n, b_n]$ where $a_i, b_i \in \{a, b\}^*$. The problem is then to decide whether there exists a sequence of indices $i_1 \cdots i_m$, $i_j \in [1, n]$, such that $a_{i_1} a_{i_2} \cdots a_{i_m} = b_{i_1} b_{i_2} \cdots b_{i_m}$. For example, consider the instance $\{[a, baa]_1, [ab, aa]_2, [bba, bb]_3\}$. Then, a solution to the PCP is the sequence 3, 2, 3, 1 since $a_3 a_2 a_3 a_1 = bba \cdot ab \cdot bba \cdot a$ and $b_3 b_2 b_3 b_1 = bb \cdot aa \cdot bb \cdot baa$.

Let $T = \{[a_1, b_1], \dots, [a_n, b_n]\}$ be a PCP instance. Let $G = \langle \{a, b\}, \{x_1, x_2\}, \{V_0\}, V_0, P, \forall x_1 \forall x_2 \rangle$ be a \forall^* -CFHG defined as follows. For every pair $[a_i, b_i] \in T$ we define the words $A_i, B_i \in (\Sigma \cup \{\#\})^*$ obtained from a_i, b_i by padding the shorter of a_i, b_i with $\#$ -symbols so that A_i, B_i are of equal length. We define P as follows.

$$P := V_0 \rightarrow \{A_{1x_1}, B_{1x_2}\}V_0 \mid \cdots \mid \{A_{nx_1}, B_{nx_2}\}V_0 \mid \{A_{1x_1}, B_{1x_2}\} \mid \cdots \mid \{A_{nx_1}, B_{nx_2}\}$$

For a language $\mathcal{L} \in \mathcal{L}(G)$, it must hold that $w_{[x_1 \mapsto u][x_2 \mapsto v]} \in \mathcal{L}(\hat{G})$ for every $u, v \in \mathcal{L}$, due to the $\forall\forall$ -condition. Let $u \in \mathcal{L}$. Then, in particular, $w_{[x_1 \mapsto u][x_2 \mapsto u]} \in \mathcal{L}(\hat{G})$. Notice that in this case, u is a solution to T . In the other direction, a solution to T induces a word $u = a_{i_1} a_{i_2} \cdots a_{i_m}$ such that $\{u\} \in \mathcal{L}(G)$. The same reduction holds also for the case of $\exists\forall$, since according to the \forall requirement, one of the word assignments must assign the same word to both variables. \square

Note that the proof of Theorem 18 compares between two words in order to simulate PCP. For a single \forall -quantifier, the nonemptiness problem is equivalent to that of CFG, and is therefore in P.

The underlying CFG we use in the proof of Theorem 18 is linear, and so the result follows also to asynchronous NFH, that allow $\#$ -symbols arbitrarily. This is in line with the results in [3], which shows that the model-checking problem for asynchronous hyperLTL is undecidable.

4.1 Synchronous Hypergrammars

As we show in Section 4, the asynchronicity of general CFHG leads to undecidability of most decision problems for them, already for simple quantification conditions. We now introduce *ranked CFHG*, a fragment of CFHG that ensures synchronous behavior. We then prove that ranked CFHG capture exactly the set of *synchronous hyperlanguages*. Intuitively, synchronous hyperlanguages are derived from grammars in which $\#$ only appears at the end of the word, similarly to NFH (we say that such a word assignment is *synchronous*). Since CFHG may use non-linear rules, in order to characterize the grammar rules that derive synchronous hyperlanguages, we need to reason about structural properties of the grammar. To this end, we define a *rank* for each variable v , which, intuitively, corresponds to word variables for which v derives $\#$ -symbols.

Remark 3. Before we turn to the definition of ranks of variables and ranked grammars we note on the difference between a definition of grammars which their hyperlanguages are synchronous, as we do in the rest of this section; and the problem of, given some hypergrammar G , finding the hyperlanguage $\mathcal{L}(G_s) \subseteq \mathcal{L}(G)$ that corresponds to the synchronous sub-hyperlanguage of G . Assume that G is over Σ and has k quantifiers. Then, the latter can be done by constructing an NFA A_s over $(\Sigma \cup \{\#\})^k$ that

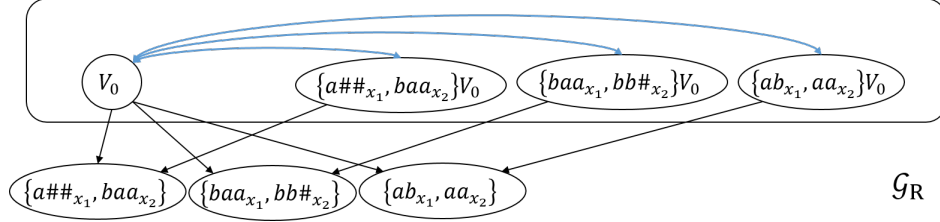


Figure 2: The MSSC graph \mathcal{G}_R for the grammar and PCP instance of Example 4 and the Proof of Theorem 18. Blue edges are bidirectional, and the rectangle represents an MSSC.

accepts all words in which # appears only at the end of words. The intersection of A_s and G results in the grammar G_s , whose language is a subset of that of G . We approach a different problem, namely defining a fragment of grammars that accept exactly the class of synchronous hyperlanguages.

In order to define the ranks of variables, we use the *rule graph* \mathcal{G} , defined as follows. The set of vertices of \mathcal{G} is $V \cup W$, where $W = \{\gamma \in (\hat{\Sigma} \cup V)^* \mid \exists v \in V. v \rightarrow \gamma \in P\}$ is the set of sequences appearing on the right side of one of the grammar rules. The set of edges E of \mathcal{G} is $E = E_L \cup E_R$ where

$$E_L = \{\langle v, w \rangle \mid v \rightarrow w \in P\} \cup \{\langle w, v \rangle \mid w = v\gamma\}$$

$$E_R = \{\langle v, w \rangle \mid v \rightarrow w \in P\} \cup \{\langle w, v \rangle \mid w = \gamma v\}$$

We partition \mathcal{G} into maximal strongly connected components (MSCCs) with respect to each type of edges (E_L and E_R), resulting in two directed a-cyclic graphs \mathcal{G}_L and \mathcal{G}_R . The vertices of \mathcal{G}_d for $d \in \{L, R\}$ are the MSCCs according to E_d , and there is an edge $C_1^d \rightarrow C_2^d$ iff there exist $u, u' \in (V \cup W)$ such that $u \in C_1^d, u' \in C_2^d$ and $\langle u, u' \rangle \in E_d$. Note that every terminal word is a singleton MSCC in both graphs.

Example 4. Figure 2 presents \mathcal{G}_R for G of the proof of Theorem 18, and the PCP instance $\{[a, baa]_1, [ab, aa]_2, [bba, bb]_3\}$, with the concrete derivation rules:

$$V_0 \rightarrow \{a##_{x_1}, baa_{x_2}\}V_0 \mid \{ab_{x_1}, aa_{x_2}\}V_0 \mid \{baa_{x_1}, bb#_{x_2}\}V_0 \mid \\ \{a##_{x_1}, baa_{x_2}\} \mid \{ab_{x_1}, aa_{x_2}\} \mid \{baa_{x_1}, bb#_{x_2}\}$$

We now define the *left ranks* and *right ranks* of synchronous words, variables and sequences.

1. **Ranks of terminal synchronous words.** The rank of a letter $\hat{\sigma} = \{\sigma_{1_{x_1}}, \dots, \sigma_{n_{x_n}}\} \in \hat{\Sigma}$ is $t(\hat{\sigma}) = \{x_i \mid \sigma_{i_{x_i}} = \#\}$. The left rank of \hat{w} is $\mathbf{L}(\hat{w}) = t(\hat{\sigma}_1)$, and its right rank is $\mathbf{R}(\hat{w}) = t(\hat{\sigma}_n)$, where $\hat{\sigma}_1$ and $\hat{\sigma}_n$ are the first and last letters of \hat{w} , respectively.
2. **Inductive definition for variables and sequences.** Let $d \in \{L, R\}$, and let $C_1^d \rightarrow C_2^d$ in \mathcal{G}_d such that $\mathbf{d}(u')$ is defined for every $u' \in C_2^d$ and $\mathbf{d} \in \{\mathbf{L}, \mathbf{R}\}$. Let $\gamma \in (\hat{\Sigma} \cup V)^*$, $\sigma \in \hat{\Sigma}$, and $v \in V$.
 - For $u \in C_1^d \in \mathcal{G}_d$ such that $u = \sigma\gamma$ we define $\mathbf{L}(u) = l(u) = \mathbf{L}(\sigma)$.
 - For $u \in C_1^d \in \mathcal{G}_d$ such that $u = \gamma\sigma$ we define $\mathbf{R}(u) = r(u) = \mathbf{R}(\sigma)$.
 - For $u \in C_1^L \in \mathcal{G}_L$ such that $u = v\gamma$ we define $l(u) = \bigcup_{C_1^L \rightarrow C_2^L} \bigcap_{u' \in C_2^L} \mathbf{L}(u')$.
 - For $u \in C_1^R \in \mathcal{G}_R$ such that $u = \gamma v$ we define $r(u) = \bigcup_{C_1^R \rightarrow C_2^R} \bigcup_{u' \in C_2^R} \mathbf{R}(u')$.

Now, for each $u = v\gamma \in C_1^L$ we define $\mathbf{L}(u) = \bigcap_{u' \in C_1^L} l(u')$, and for each $u = \gamma v \in C_1^R$ we define $\mathbf{R}(u) = \bigcup_{u' \in C_1^R} r(u')$.

Note that this process is guaranteed to terminate, since we traverse both graphs in reverse topological order. Therefore, at the end of the process, $\mathbf{L}(u)$ and $\mathbf{R}(u)$ are defined for every $u \in V \cup W$.

We define *ranked CFGs* to be CFGs in which for every rule $v \rightarrow \gamma_1 \cdots \gamma_n$ for $\gamma_i \in (\hat{\Sigma} \cup V)$, it holds that $\mathbf{R}(\gamma_i) \subseteq \mathbf{L}(\gamma_{i+1})$. Intuitively, this means that γ_i may not produce $\#$ to its right, if γ_{i+1} can produce $\sigma \neq \#$ to its left, leading to unsynchronous derivation. A CFHG G is *ranked* if \hat{G} is ranked.

Example 5. Consider G of Example 4 and \mathcal{G}_R of Figure 2. The graph \mathcal{G}_L is similar to \mathcal{G}_R , with no edges back to V_0 , (and thus without the rectangle MSCC). We compute some of the ranks for G :

$$\begin{aligned} \mathbf{L}(\{a\#\#_{x_1}, baa_{x_2}\}) &= \mathbf{L}(\{baa_{x_1}, bb\#_{x_2}\}) = \mathbf{L}(\{ab_{x_1}, aa_{x_2}\}) = \emptyset & \mathbf{L}(V_0) &= \emptyset \\ \mathbf{R}(\{a\#\#_{x_1}, baa_{x_2}\}) &= \{x_1\} & \mathbf{R}(\{baa_{x_1}, bb\#_{x_2}\}) &= \{x_2\} & \mathbf{R}(\{ab_{x_1}, aa_{x_2}\}) &= \emptyset & \mathbf{R}(V_0) &= \{x_1, x_2\} \end{aligned}$$

G is not ranked, since for the rule $V_0 \rightarrow \{a\#\#_{x_1}, baa_{x_2}\}V_0$, it holds that $\mathbf{R}(\{a\#\#_{x_1}, baa_{x_2}\}) \not\subseteq \mathbf{L}(V_0)$.

Example 6. The following CFHG $G_r = \langle \{a, b\}, \{x_1, x_2\}, \{V_0, V_1\}, V_0, P, \forall x_1 \exists x_2 \rangle$ is ranked, where P is:

$$\begin{aligned} P &:= V_0 \rightarrow V_1 V_2 \\ V_1 &\rightarrow \{a_{x_1}, a_{x_2}\}V_1 \{b_{x_1}, b_{x_2}\} \mid \{ab_{x_1}, ab_{x_2}\} \\ V_2 &\rightarrow V_2 \{\#_{x_1}, b_{x_2}\} \mid \{\#_{x_1}, b_{x_2}\} \end{aligned}$$

G_r accepts all languages in which for every word of the type $a^n b^n$ there exists a word with more b 's, that is, there exists $a^n b^m$ for $m > n$.

The ranks of G_r , as shown below, demonstrate that G_r is indeed ranked.

$$\begin{aligned} \mathbf{L}(\{\#_{x_1}, b_{x_2}\}) &= \mathbf{R}(\{\#_{x_1}, b_{x_2}\}) = \{x_1\} & \mathbf{L}(\{ab_{x_1}, ab_{x_2}\}) &= \mathbf{R}(\{ab_{x_1}, ab_{x_2}\}) = \emptyset \\ \mathbf{L}(\{a_{x_1}, a_{x_2}\}V_1 \{b_{x_1}, b_{x_2}\}) &= \mathbf{R}(\{a_{x_1}, a_{x_2}\}V_1 \{b_{x_1}, b_{x_2}\}) = \mathbf{R}(V_1) = \mathbf{L}(V_1) = \emptyset \\ \mathbf{R}(V_2 \{\#_{x_1}, b_{x_2}\}) &= \mathbf{L}(V_2 \{\#_{x_1}, b_{x_2}\}) = \mathbf{R}(V_2) = \mathbf{L}(V_2) = \{x_1\} \\ \mathbf{R}(V_0) &= \emptyset & \mathbf{L}(V_0) &= \{x_1\} \end{aligned}$$

Definition 19. \mathcal{L} is a synchronous context-free hyperlanguage if there exists a CFHG G for \mathcal{L} in which \hat{G} only derives synchronous word assignments.

Theorem 20. A hyperlanguage \mathcal{L} is derived by a ranked CFHG iff \mathcal{L} is synchronous context-free.

In order to prove Theorem 20, we use the following claims.

Claim 21. Let $G = \langle \Sigma, X, V, V_0, P, \alpha \rangle$ be a ranked CFHG. Then, for every word $\gamma = \gamma_1 \cdots \gamma_n \in (\hat{\Sigma} \cup V)^*$, if there exists $v \in V$ such that $v \Rightarrow^* \gamma$, then $\mathbf{R}(\gamma_i) \subseteq \mathbf{L}(\gamma_{i+1})$ for all $i \in [1, n-1]$.

Claim 22. Let $G = \langle \Sigma, X, V, V_0, P, \alpha \rangle$ be a (possibly not ranked) CFHG with $|X| = k$, and let $v \in V$.

1. For every $j \in [1, k] \setminus \mathbf{L}(v)$ there exists $w \in \hat{\Sigma}^*$ such that $v \Rightarrow^* w$ and $j \notin \mathbf{L}(w)$.
2. For every $j \in \mathbf{R}(v)$ there exists $w \in \hat{\Sigma}^*$ such that $v \Rightarrow^* w$ and $j \in \mathbf{R}(w)$.

Proof of Theorem 20. Let \mathcal{L} be a context-free language that is accepted by a ranked grammar G . According to Claim 21, for every word $w = w_1 \cdots w_n \in \hat{\Sigma}^*$ such that $V_0 \Rightarrow^* w$, it holds that $\mathbf{R}(w_i) \subseteq \mathbf{L}(w_{i+1})$ for $i \in [1, n-1]$. That is, $\#$ is allowed to only appear at the end of words, and so \hat{G} only derives synchronous word assignments.

For the other direction, let \mathcal{L} be a synchronous context-free hyperlanguage, and let G be a CFHG for \mathcal{L} that only derives synchronous word assignments. Assume by way of contradiction that G is not ranked. Then, there exists some rule $v \rightarrow \gamma_1 \cdots \gamma_n \in P$ where $\gamma_i \in (\hat{\Sigma} \cup V)$ such that $\mathbf{R}(\gamma_i) \not\subseteq \mathbf{L}(\gamma_{i+1})$ for some

$i \in [1, n]$. Recall that we assume that all rules are reachable and that every variable can derive a terminal word. Consider a derivation sequence $V_0 \Rightarrow^* \beta v \beta' \Rightarrow \beta \gamma_1 \cdots \gamma_n \beta'$. Then, there exist $w, w_i, w_{i+1}, w' \in \hat{\Sigma}^*$ such that $\gamma_i \Rightarrow^* w_i$, $\gamma_{i+1} \Rightarrow^* w_{i+1}$ and $V_0 \Rightarrow^* ww_i w_{i+1} w'$; and due to claim 22, for some $j \in \mathbf{R}(\gamma_i) \setminus \mathbf{L}(\gamma_{i+1})$, it holds that $j \in \mathbf{R}(w_i) \setminus \mathbf{L}(w_{i+1})$. Hence, w_i ends with # in some location which is followed by a letter in w_{i+1} , and so the word assignment $ww_i w_{i+1} w'$ is not synchronous, a contradiction. \square

We therefore term ranked grammars *syncCFHG*. Given a CFHG G , deciding whether it is ranked amounts to constructing the graph \mathcal{G} and traversing the topological sorting of its MSCC graph in reverse order in order to compute all ranks, and finally checking that all grammar rules of G comply to the rank rules. All these steps can be computed in polynomial time. We now show that syncCFHG is more decidable than CFHG.

Theorem 23. *The nonemptiness problem for \forall^* -syncCFHG and $\exists\forall^*$ -syncCFHG is in P.*

Proof. Let G be a syncCFHG. Since universal quantification is closed under subsets, it holds that if $\mathcal{L} \in \mathfrak{L}(G)$, then $\mathcal{L}' \in \mathfrak{L}(G)$ for every $\mathcal{L}' \subseteq \mathcal{L}$. Therefore, it suffices to check whether there exists a singleton \mathcal{L} such that $\mathcal{L} \in \mathfrak{L}(G)$. Therefore, we consider only word assignments of the form $\mathbf{w} = \mathbf{w}_{[x_1 \mapsto w] \cdots [x_k \mapsto w]}$ for some $w \in (\Sigma \cup \{\#\})^*$. Notice that \mathbf{w} has a single representation, since # may not appear arbitrarily. We construct a syncCFHG G' by restricting \hat{G} to the alphabet $\bigcup_{\sigma \in \Sigma} \{\sigma\}^X$, that is, all variables are assigned the same letter. All rules over other alphabet letters are eliminated. Since elimination of rules cannot induce asynchronization, G' is synchronous.

Now, for a singleton language $\{w\}$, we have $\{w\} \in \mathfrak{L}(G)$ iff $\{w\} \in \mathfrak{L}(G')$. Therefore, it suffices to check the nonemptiness of $\mathfrak{L}(G')$, which amounts to checking the nonemptiness of \hat{G}' .

The proof holds also for the case of $\exists\forall^*$ -syncCFHG. Indeed, an $\exists\forall^*$ -syncCFHG G is nonempty iff it derives a singleton hyperlanguage. This, since in a language derived by G , a word w that is assigned to the variable under \exists must also be assigned to all variables under \forall in one of the word assignments derived by \hat{G} , which in turn fulfills the requirements for deriving $\{w\}$. Since G is synchronous, it suffices to restrict the alphabet to homogeneous letters and check for nonemptiness, as with \forall^* . \square

In the *regular membership problem*, we ask whether a regular language \mathcal{L} can be derived by a syncCFHG G . This problem is decidable for NFH [5]. For $\mathcal{L} = \Sigma^*$ and a \forall -CFHG G , the question amounts to checking the universality of \hat{G} , which is undecidable [2]. Therefore, we have the following.

Theorem 24. *The regular membership problem for \forall^* -syncCFHG grammars is undecidable.*

Remark 4. Membership of a finite language \mathcal{L} in a CFHG with any quantification condition is decidable already for general CFHG (Theorem 17), with exponential complexity. For syncCFHG, we can reduce the complexity by checking membership of word assignments instead. This, since we only need to consider synchronous words, which have a single representation. Since checking membership is polynomial in the size of the word (for a grammar of fixed size) [16], every such test is then polynomial. Since we may still need to traverse all possible word assignment, the complexity is exponential in the length of the quantification condition, but is polynomial in $|G|$ and the size of the words in \mathcal{L} .

Remark 5. For regular languages and \exists^* -CFHG, synchronization does not avoid the composition of automata, and we use a construction similar to the one of Theorem 16.

We now show that synchronicity does not suffice for deciding nonemptiness of $\exists^*\forall^*$ -syncCFHG.

Theorem 25. *The nonemptiness problem for $\exists^*\forall^*$ -syncCFHG is undecidable.*

Proof. We reduce from PCP. Let $T = \{[a_1, b_1], \dots, [a_n, b_n]\}$ be a PCP instance over $\{a, b\}$, and let

$$G = \langle \{a, b, c\} \cup [1, n], \{x_1, x_2, x_3\}, \{V_0, V_1, V_2\}, V_0, P, \exists x_1 \exists x_2 \forall x_3 \rangle$$

be a CFHG where P is defined as follows.

$$P := V_0 \rightarrow V_1 \mid V_2$$

$$V_1 \rightarrow \{a_{i_{x_1}}, c^{|a_i|}_{x_2}, a_{i_{x_3}}\} V_1 \{i_{x_1}, c_{x_2}, i_{x_3}\} \mid \{a_{i_{x_1}}, c^{|a_i|}_{x_2}, a_{i_{x_3}}\} \{i_{x_1}, c_{x_2}, i_{x_3}\} \quad \forall i \in [1, n]$$

$$V_2 \rightarrow \{b_{i_{x_1}}, c^{|b_i|}_{x_2}, c^{|b_i|}_{x_3}\} V_2 \{i_{x_1}, c_{x_2}, c_{x_3}\} \mid \{b_{i_{x_1}}, c^{|b_i|}_{x_2}, c^{|b_i|}_{x_3}\} \{i_{x_1}, c_{x_2}, c_{x_3}\} \quad \forall i \in [1, n]$$

Since none of the rules include the $\#$ -symbol, G is indeed a syncCFHG. Now, if there exists $\mathcal{L} \in \mathcal{L}(G)$, then there exist $w \in \{a, b\}^* \cdot [1, n]^*$ and $w_c \in \{c\}^*$ both in \mathcal{L} , such that for every $w' \in \mathcal{L}$, we have $V_0 \Rightarrow^* \mathbf{w}_{[x_1 \mapsto w][x_1 \mapsto w_c][x_3 \mapsto w']}$. In particular, for $w' = w$, we have $V_0 \Rightarrow^* \mathbf{w}_{[x_1 \mapsto w][x_1 \mapsto w_c][x_3 \mapsto w]}$. Since V_2 only derives words of the form c^k in x_3 , the derivation of $\mathbf{w}_{[x_1 \mapsto w][x_2 \mapsto w_c][x_3 \mapsto w]}$ is of the form $V_0 \Rightarrow V_1 \Rightarrow^* \mathbf{w}_{[x_1 \mapsto w][x_2 \mapsto w_c][x_3 \mapsto w]}$. In addition, all words in $\{c\}^*$ can only be assigned to x_3 if derived from V_2 , thus we have $V_0 \Rightarrow V_2 \Rightarrow^* \mathbf{w}_{[x_1 \mapsto w][x_2 \mapsto w_c][x_3 \mapsto w_c]}$. Denote $w = w_1 w_2$ where $w_1 \in \{a, b\}^*$, $w_2 \in [1, n]$. Then, w encodes a solution to T , where w_1 is the string obtained from a_i (and b_i), and w_2 is the sequence of indices.

For the other direction, a solution to T encoded by a string w_1 and sequence of indices w_2 corresponds to the language $\{w_1 w_2, c^{|w_1 w_2|}\}$ that is accepted by G . \square

The nonemptiness problem for $\forall^* \exists^*$ -NFH is undecidable [5]. Therefore, this is also the case for syncCFHG, and for general CFHG.

5 Discussion and Future Work

We have studied the realizability problem for regular hyperlanguages, focusing on the case of singleton hyperlanguages. We have shown that simple quantification conditions cannot realize this case. We have defined ordered and partially-ordered languages, for which we can construct hyperautomata that enumerate the language by order. We have shown that all regular languages are partially ordered. Since regular hyperlanguages are closed under union [5], the result extends to a finite hyperlanguage containing regular languages. Naturally, there are richer cases one can consider. For an infinite hyperlanguage \mathcal{L} , some characterization on the elements of \mathcal{L} would need to be defined in order to explore its realizability. We plan on pursuing this direction as future work. Another related direction is finding techniques for proving unrealizability for certain quantification conditions, for various types of hyperlanguages.

In the second part of the paper we have studied the natural extension of context-free grammars to handle context-free hyperlanguages. Here, we have shown that beyond the inherent undecidability of some decision problems for hypergrammars, some undecidability properties stem from the asynchronous nature of these hypergrammars. We have then defined a synchronous fragment of context-free hyperlanguages, and defined a fragment of context-free grammars which exactly captures this fragment. The result retains some of the decidability properties of context-free grammars. As a future direction, we plan to study the realizability problem for CFHG and syncCFHG. Due to the limited closure properties of CFG, this is expected to be more challenging than for NFH. Another possible future direction is studying the entire Chomsky hierarchy for hyperlanguages, and finding fragments of the extensions to hyperlanguages that conserve the properties of these models for standard languages.

Acknowledgements. We thank the anonymous reviewer for suggesting the elegant construction mentioned in Remark 2.

References

- [1] B. Alpern & F.B. Schneider (1985): *Defining Liveness*. *Information Processing Letters*, pp. 181–185, doi:[10.1016/0020-0190\(85\)90056-0](https://doi.org/10.1016/0020-0190(85)90056-0).
- [2] Brenda S. Baker & Ronald V. Book (1974): *Reversal-Bounded Multipushdown Machines*. *J. Comput. Syst. Sci.* 8(3), pp. 315–332, doi:[10.1016/S0022-0000\(74\)80027-9](https://doi.org/10.1016/S0022-0000(74)80027-9).
- [3] Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner & César Sánchez (2021): *A Temporal Logic for Asynchronous Hyperproperties*. In Alexandra Silva & K. Rustan M. Leino, editors: *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I, Lecture Notes in Computer Science 12759*, Springer, pp. 694–717, doi:[10.1007/978-3-030-81685-8_33](https://doi.org/10.1007/978-3-030-81685-8_33).
- [4] Borzoo Bonakdarpour, César Sánchez & Gerardo Schneider (2018): *Monitoring Hyperproperties by Combining Static Analysis and Runtime Verification*. In Tiziana Margaria & Bernhard Steffen, editors: *Leveraging Applications of Formal Methods, Verification and Validation. Verification - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part II, Lecture Notes in Computer Science 11245*, Springer, pp. 8–27, doi:[10.1007/978-3-030-03421-4_2](https://doi.org/10.1007/978-3-030-03421-4_2).
- [5] Borzoo Bonakdarpour & Sarai Sheinvald (2021): *Finite-Word Hyperlanguages*. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira & Claudio Zandron, editors: *Language and Automata Theory and Applications - 15th International Conference, LATA 2021, Milan, Italy, March 1-5, 2021, Proceedings, Lecture Notes in Computer Science 12638*, Springer, pp. 173–186, doi:[10.1007/978-3-030-68195-1_17](https://doi.org/10.1007/978-3-030-68195-1_17).
- [6] Ahmed Bouajjani, Rachid Echahed & Riadh Robbana (1994): *Verification of Nonregular Temporal Properties for Context-Free Processes*. In Bengt Jonsson & Joachim Parrow, editors: *CONCUR '94, Concurrency Theory, 5th International Conference, Uppsala, Sweden, August 22-25, 1994, Proceedings, Lecture Notes in Computer Science 836*, Springer, pp. 81–97, doi:[10.1007/978-3-540-48654-1_8](https://doi.org/10.1007/978-3-540-48654-1_8).
- [7] Noam Chomsky (1959): *On Certain Formal Properties of Grammars*. *Inf. Control.* 2(2), pp. 137–167, doi:[10.1016/S0019-9958\(59\)90362-6](https://doi.org/10.1016/S0019-9958(59)90362-6).
- [8] Edmund M. Clarke, Orna Grumberg, Daniel Kroening, Doron A. Peled & Helmut Veith (2018): *Model checking, 2nd Edition*. MIT Press. Available at <https://mitpress.mit.edu/books/model-checking-second-edition>.
- [9] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe & César Sánchez (2014): *Temporal Logics for Hyperproperties*. In Martín Abadi & Steve Kremer, editors: *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings, Lecture Notes in Computer Science 8414*, Springer, pp. 265–284, doi:[10.1007/978-3-642-54792-8_15](https://doi.org/10.1007/978-3-642-54792-8_15).
- [10] Michael R. Clarkson & Fred B. Schneider (2010): *Hyperproperties*. *J. Comput. Secur.* 18(6), pp. 1157–1210, doi:[10.3233/JCS-2009-0393](https://doi.org/10.3233/JCS-2009-0393).
- [11] Norine Coenen, Bernd Finkbeiner, Christopher Hahn & Jana Hofmann (2019): *The Hierarchy of Hyperlogics*. In: *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, IEEE, pp. 1–13, doi:[10.1109/LICS.2019.8785713](https://doi.org/10.1109/LICS.2019.8785713).
- [12] Bernd Finkbeiner, Lennart Haas & Hazem Torfah (2019): *Canonical Representations of k -Safety Hyperproperties*. In: *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, IEEE, pp. 17–31, doi:[10.1109/CSF.2019.00009](https://doi.org/10.1109/CSF.2019.00009).
- [13] Bernd Finkbeiner & Martin Zimmermann (2017): *The First-Order Logic of Hyperproperties*. In Heribert Vollmer & Brigitte Vallée, editors: *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany, LIPIcs 66*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 30:1–30:14, doi:[10.4230/LIPIcs.STACS.2017.30](https://doi.org/10.4230/LIPIcs.STACS.2017.30).
- [14] Wladimir Fridman & Bernd Puchala (2014): *Distributed Synthesis for Regular and Contextfree Specifications*. *Acta Informatica* 51(3-4), pp. 221–260, doi:[10.1007/s00236-014-0194-x](https://doi.org/10.1007/s00236-014-0194-x).

- [15] Ohad Goudsmid, Orna Grumberg & Sarai Sheinvald (2021): *Compositional Model Checking for Multi-properties*. In Fritz Henglein, Sharon Shoham & Yakir Vizel, editors: *Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings, Lecture Notes in Computer Science 12597*, Springer, pp. 55–80, doi:[10.1007/978-3-030-67067-2_4](https://doi.org/10.1007/978-3-030-67067-2_4).
- [16] John E. Hopcroft, Rajeev Motwani & Jeffrey D. Ullman (2001): *Introduction to Automata Theory, Languages, and Computation, 2nd Edition*. Addison-Wesley series in computer science, Addison-Wesley-Longman.
- [17] Bakhadyr Khoussainov & Anil Nerode (1994): *Automatic Presentations of Structures*. In Daniel Leivant, editor: *Logical and Computational Complexity. Selected Papers. Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, Indiana, USA, 13-16 October 1994, Lecture Notes in Computer Science 960*, Springer, pp. 367–392, doi:[10.1007/3-540-60178-3_93](https://doi.org/10.1007/3-540-60178-3_93).
- [18] Adrien Pommellet & Tayssir Touili (2018): *Model-Checking HyperLTL for Pushdown Systems*. In María-del-Mar Gallardo & Pedro Merino, editors: *Model Checking Software - 25th International Symposium, SPIN 2018, Malaga, Spain, June 20-22, 2018, Proceedings, Lecture Notes in Computer Science 10869*, Springer, pp. 133–152, doi:[10.1007/978-3-319-94111-0_8](https://doi.org/10.1007/978-3-319-94111-0_8).
- [19] Markus N. Rabe (2016): *A Temporal Logic Approach to Information-flow Control*. Ph.D. thesis, Saarland University. Available at <http://scidok.sulb.uni-saarland.de/volltexte/2016/6387/>.
- [20] Moshe Y. Vardi (1995): *An Automata-Theoretic Approach to Linear Temporal Logic*. In Faron Moller & Graham M. Birtwistle, editors: *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, Banff, Canada, August 27 - September 3, 1995, Proceedings)*, Lecture Notes in Computer Science 1043, Springer, pp. 238–266, doi:[10.1007/3-540-60915-6_6](https://doi.org/10.1007/3-540-60915-6_6).
- [21] Moshe Y. Vardi & Pierre Wolper (1986): *An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report)*. In: *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, IEEE Computer Society, pp. 332–344.
- [22] Moshe Y. Vardi & Pierre Wolper (1994): *Reasoning About Infinite Computations*. *Inf. Comput.* 115(1), pp. 1–37, doi:[10.1006/inco.1994.1092](https://doi.org/10.1006/inco.1994.1092).
- [23] Yu Wang, Mojtaba Zarei, Borzoo Bonakdarpour & Miroslav Pajic (2019): *Statistical Verification of Hyperproperties for Cyber-Physical Systems*. *ACM Trans. Embed. Comput. Syst.* 18(5s), pp. 92:1–92:23, doi:[10.1145/3358232](https://doi.org/10.1145/3358232).

Controller Synthesis for Timeline-based Games

Renato Acampora
University of Udine, Italy
acampora.renato@spes.uniud.it

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

Luca Geatti Nicola Gigante
Free University of Bozen-Bolzano
{geatti,gigante}@inf.unibz.it

Valentino Picotti
University of Southern Denmark
picotti@imada.sdu.dk

In the timeline-based approach to planning, originally born in the space sector, the evolution over time of a set of state variables (the timelines) is governed by a set of temporal constraints. Traditional timeline-based planning systems excel at the integration of planning with execution by handling *temporal uncertainty*. In order to handle general nondeterminism as well, the concept of *timeline-based games* has been recently introduced. It has been proved that finding whether a winning strategy exists for such games is 2EXPTIME-complete. However, a concrete approach to synthesize controllers implementing such strategies is missing. This paper fills this gap, outlining an approach to controller synthesis for timeline-based games.

1 Introduction

In the timeline-based approach to planning, the world is viewed as a system made of a set of independent but interacting components whose behaviour over time (the timelines) is governed by a set of temporal constraints, called *synchronization rules*. Timeline-based planning has been originally introduced in the space industry [19], with timeline-based planners developed and used by space agencies on both sides of the Atlantic [5, 4, 13, 2, 6], both for short- to long-term mission planning [7] and on-board autonomy [14].

While successful in practice, only recently timeline-based planning has been studied from a theoretical perspective. The formalism has been at first compared with traditional action-based languages *à la* STRIPS, proving that they can be expressed by means of timeline-based languages [16]. Then, the complexity of the timeline-based plan existence problem has been studied: the problem is EXPSPACE-complete [17] over discrete time in the general case, and PSPACE-complete with qualitative constraints [11]. On dense time, the problem goes from being NP-complete to undecidable, depending on the syntactic restrictions applied [3]. The expressiveness of timeline-based languages has also been studied from a logical perspective [10], and an automata-theoretic point of view [9].

Traditional timeline-based planning systems excel at the integration of planning with *execution* by treating explicitly the concept of *temporal uncertainty*: the exact timings of the events under control of the environment need not to be precisely known in advance. However, general nondeterminism, where the environment can also decide *what* to do (instead of only *when* to do it) is usually not handled by these systems. To overcome this limitation, the concept of *timeline-based games* has been recently introduced [18]. In these games, the state variables are partitioned between the controller and the environment, and the latter has the freedom to play arbitrarily as long as a set of *domain* rules, that define the game arena, are satisfied. The controller plays to satisfy his set of *system* rules. A strategy for controller is winning if it allows him/her to win independently from the choices of the environment.

Establishing whether a winning strategy exists for these games has been proved to be 2EXPTIME-complete [18]. However, no concrete way to synthesize a controller implementing such strategies is

known. The proof technique of the aforementioned complexity result involves the construction of a huge (doubly exponential) *concurrent game structure*, which is used to model check some Alternating-time Temporal Logic (ATL) formulas [1]. While this structure is deterministic and can be in principle used as an arena to solve a reachability game and synthesize a controller, its construction is based on theoretical nondeterministic procedures which have no hope to be ever concretely implemented. On the other hand, the automata-theoretic approach by Della Monica *et al.* [9] provides a concrete and effective construction of an automaton that accepts a word if and only if the original planning problem has a solution plan. However, the automaton is *nondeterministic* and already doubly exponential, and the determinization needed to use it as an arena would result into a further blow up and a non-optimal procedure.

In this paper, we provide a concrete and computationally optimal approach to controller synthesis for timeline-based games. We overcome the limitations of both the above-mentioned approaches by devising a direct construction for a *deterministic* finite-state automaton that recognizes solution plans, which is doubly exponential in size (thus not requiring the determinization of a nondeterministic automaton). This automaton is then used as the arena of a reachability game for which plenty of controller synthesis techniques are known in the literature.

The paper is structured as follows. In Section 2 we introduce the needed background on timeline-based planning and timeline-based games. Then, Section 3 provides the core technical contribution of the paper, namely the construction of the deterministic automaton recognizing solution plans. Section 4 uses this automaton as the game arena to solve the controller synthesis problem. Last, Section 5 summarizes the main contributions of the work and discuss future developments.

2 Timeline-based games

In this section, we introduce timeline-based games, as defined in [18].

2.1 State variables, event sequences, synchronization rules

The first basic concept is that of *state variable*.

Definition 1 (State variable). *A state variable is a tuple $x = (V_x, T_x, D_x, \gamma)$, where:*

- V_x is the finite domain of x ;
- $T_x : V_x \rightarrow 2^{V_x}$ is the value transition function of x , which maps each value $v \in V_x$ to the set of values that can immediately follow it;
- $D_x : V_x \rightarrow \mathbb{N} \times \mathbb{N}$ is the duration function of x , mapping each value $v \in V_x$ to a pair $(d_{\min}^{x=v}, d_{\max}^{x=v})$ specifying respectively the minimum and maximum duration of any interval where $x = v$;
- $\gamma : V_x \rightarrow \{c, u\}$ is the controllability tag, that, for each value $v \in V_x$, specifies whether it is controllable ($\gamma(v) = c$) or uncontrollable ($\gamma(v) = u$).

Intuitively, a state variable x takes a value from a finite domain and represents a simple finite-state machine, whose transition function is T_x . The behaviour over time of a set of state variables SV is defined by a set of *timelines*, one for each variable. Instead of reasoning about timelines directly, though, in this paper we follow the approach outlined in [18] and represent the whole execution of a system, modeled by means of a set of state variables, with a single word, called *event sequence*.

Definition 2 (Event sequence [18]). *Let SV be a set of state variables. Let A_{SV} be the set of all the terms, called actions, of the form $\text{start}(x, v)$ or $\text{end}(x, v)$, where $x \in SV$ and $v \in V_x$.*

An event sequence over SV is a sequence $\bar{\mu} = \langle \mu_1, \dots, \mu_n \rangle$ of pairs $\mu_i = (A_i, \delta_i)$, called events, where $A_i \subseteq A_{SV}$ is a set of actions, and $\delta_i \in \mathbb{N}_+$, such that, for any $x \in SV$:

1. for all $1 \leq i \leq n$, if $\text{start}(x, v) \in A_i$, for some $v \in V_x$, then there is no $\text{start}(x, v')$ in any μ_j before the closest μ_k , with $k > i$, such that $\text{end}(x, v) \in A_k$ (if any);
2. for all $1 \leq i \leq n$, if $\text{end}(x, v) \in A_i$, for some $v \in V_x$, then there is no $\text{end}(x, v')$ in any μ_j after the closest μ_k , with $k < i$, such that $\text{start}(x, v) \in A_k$ (if any);
3. for all $1 \leq i < n$, if $\text{end}(x, v) \in A_i$, for some $v \in V_x$, then $\text{start}(x, v') \in A_i$, for some $v' \in V_x$;
4. for all $1 < i \leq n$, if $\text{start}(x, v) \in A_i$, for some $v \in V_x$, then $\text{end}(x, v') \in A_i$, for some $v' \in V_x$.

Intuitively, an event sequence represents the evolution over time of the state variables of the system by representing the *start* and the *end* of *tokens*, i.e., a sequence of adjacent intervals where a given variable takes a given value. An event $\mu_i = (A_i, \delta_i)$ consists of a set A_i of actions describing the start or the end of some tokens, happening δ_i time steps after the previous one. In an *event sequence*, events are collected to describe a whole plan.

Definition 2 intentionally implies that a started token is not required to end before the end of the sequence, and a token can end without the corresponding starting action to have ever appeared before. In this case we say the event sequence is *open* (on the right or on the left, respectively). Otherwise, it is said to be *closed*. An event sequence closed on the left and open on the right is also called a *partial plan*. Note that the empty event sequence is closed on both sides for any variable. Moreover, on closed event sequences, the first event only contains $\text{start}(x, v)$ actions and the last event only contains $\text{end}(x, v)$ actions, one for each variable x . Given an event sequence $\bar{\mu} = \langle \mu_1, \dots, \mu_n \rangle$ over a set of state variables SV , with $\mu_i = (A_i, \delta_i)$, we define $\delta(\bar{\mu}) = \sum_{1 < i \leq n} \delta_i$, that is, $\delta(\bar{\mu})$ is the time elapsed from the start to the end of the sequence (its duration). The amount of time spanning a subsequence, written as $\delta_{i,j}$ when $\bar{\mu}$ is clear from context, is then $\delta(\bar{\mu}_{i,j}) = \sum_{i < k \leq j} \delta_k$. Finally, given an event sequence $\bar{\mu} = \langle \mu_1, \dots, \mu_n \rangle$, for each $1 < i \leq n$, we define $\bar{\mu}_{<i}$ as $\langle \mu_1, \dots, \mu_{i-1} \rangle$.

In timeline-based games, the controller plays to satisfy a set of *synchronization rules*, which describe the desired behavior of the system. Synchronization rules relate tokens, possibly belonging to different timelines, through temporal relations among token endpoints. Let SV be a set of state variables and $N = \{a, b, \dots\}$ be an arbitrary set of *token names*. Moreover, let an *atomic temporal relation*, or simply *atom*, be an expression of the form $\langle \text{term} \rangle \leq_{l,u} \langle \text{term} \rangle$, where $l \in \mathbb{N}$, $u \in \mathbb{N} \cup \{\infty\}$, and a *term* is either $\text{start}(a)$ or $\text{end}(a)$, for some $a \in N$. A synchronisation rule R takes the following form:

$$\begin{aligned} R &:= a_0[x_0 = v_0] \rightarrow E_1 \vee E_2 \vee \dots \vee E_k \quad \text{where} \\ E_i &:= \exists a_1[x_1 = v_1] a_2[x_2 = v_2] \dots a_n[x_n = v_n] \cdot C_i \end{aligned}$$

where $a_0, \dots, a_n \in N$, $x_0, \dots, x_n \in SV$, v_0, \dots, v_n are such that $v_i \in V_{x_i}$, for all $0 \leq i \leq n$, and C_i is a conjunction of atomic temporal relations (a clause). The elements $a_i[x_i = v_i]$ are called *quantifiers* and the quantifier $a_0[v_0 = x_0]$ is called the *trigger*. The disjuncts in the body are called *existential statements*.

We say that a token $\tau = (x, v, d)$ satisfies a quantifier $a_i[x_i = v_i]$ if $x = x_i$ and $v = v_i$. The semantics of a synchronisation rule R states that for every token satisfying the trigger, at least one of the existential statements is satisfied. Each existential statement E_i requires the existence of some tokens, satisfying the quantifiers in its prefix, such that the clause C_i is satisfied. When a token satisfies the trigger of a rule, it is said to *trigger* such a rule.

For space concerns, we do not provide all the details of the semantics of synchronization rules. The reader can find them in [18]. Intuitively, each time there is a token that satisfies the trigger of a rule,

one of its existential statements must be satisfied as well. The existential statements in turn assert the existence of other tokens that satisfy a conjunction of atoms.

If a and b are two token names, then examples of atomic relations are $\text{start}(a) \leq_{3,7} \text{end}(b)$ and $\text{start}(a) \leq_{0,+\infty} \text{start}(b)$. Intuitively, a token name a refers to a specific token, that is, a pair of $\text{start}(x, v)$ and $\text{end}(x, v)$ actions in an event sequence, and $\text{start}(a)$ and $\text{end}(a)$ to its endpoints. Then, an atom such as $\text{start}(a) \leq_{l,u} \text{end}(b)$ constrains a to start before the end of b , with the distance between the two endpoints to be comprised between the lower and upper bounds l and u .

Examples of synchronization rules are the following, where the relations $=$ and \leq are respectively syntactic sugar for $\leq_{0,0}$ and $\leq_{0,+\infty}$:

$$\begin{aligned} a[x_s = \text{Comm}] &\rightarrow \exists b[x_g = \text{Available}] . \text{start}(b) \leq \text{start}(a) \wedge \text{end}(a) \leq \text{end}(b) \\ a[x_s = \text{Science}] &\rightarrow \exists b[x_s = \text{Slewing}]c[x_s = \text{Earth}]d[x_s = \text{Comm}] . \\ &\quad \text{end}(a) = \text{start}(b) \wedge \text{end}(b) = \text{start}(c) \wedge \text{end}(c) = \text{start}(d) \end{aligned}$$

where the variables x_s and x_g represent respectively the state of a spacecraft and the visibility of the communication ground station. The first rule requires the satellite and the ground station to synchronise their communications, so that when the satellite is transmitting the ground station is available for reception. The second rule instructs the system to transmit data back to Earth after every measurement session, interleaved by the required slewing operation. A rule whose trigger is empty (\top), called *triggerless rule*, can be used to state the *goal* of the system. As an example, they allow one to force the spacecraft to perform some scientific measurement at all:

$$\top \rightarrow \exists a[x_s = \text{Science}]$$

Triggerless rules have a trivial universal quantification, which means they only demand the existence of some tokens, as specified by the existential statements. Although triggerless rules are meant to specify the goals of a planning problem, they can be regarded as syntactic sugar on top of the syntax described above. Indeed, triggerless rules can be translated into triggered rules [18], and thus we do not consider them from here onwards.

Finally, even though our focus is on timeline-based games, we conclude the section by formally defining *timeline-based planning problems*.

Definition 3 (Timeline-based planning problem). *A timeline-based planning problem is a pair $P = (SV, S)$, where SV is a set of state variables and S is a set of synchronization rules over SV . An event sequence $\bar{\mu}$ over SV is a solution plan for P if all the rules in S are satisfied by $\bar{\mu}$.*

2.2 The game arena

We are now ready to introduce *timeline-based games*. Their definition is quite involved, as their structure has been designed with the goal of being strictly more general than *timeline-based planning with uncertainty* [8] while being able to capture its semantics precisely. For space concerns, we keep the exposition quite terse, but the reader can refer to [18] for details.

Definition 4 (Timeline-based game). *A timeline-based game is a tuple $G = (SV_C, SV_E, S, D)$, where SV_C and SV_E are the sets of controlled and external variables, respectively, and S and D are the sets of system and domain synchronisation rules, respectively, both involving variables from SV_C and SV_E .*

A partial plan for G is a partial plan over the state variables $SV_C \cup SV_E$. Let Π_G be the set of all possible partial plans for G , simply Π when there is no ambiguity.

Since ε is a closed event sequence and $\delta(\varepsilon) = 0$, the *empty* partial plan ε is a good starting point for the game. Players incrementally build a richer partial plan, starting from ε , by playing actions that specify which tokens to start and/or to end, adding an event that extends the event sequence, or complementing the existing last event of the sequence. We partition all the available actions into those that are playable by either of the two players.

Definition 5 (Partition of player actions). *Let $SV = SV_C \cup SV_E$. The set A of available actions over SV is partitioned into the sets A_C of Charlie's actions and A_E of Eve's actions, which are defined as follows:*

$$A_C = \underbrace{\{\text{start}(x, v) \mid x \in SV_C, v \in V_x\}}_{\text{start tokens on Charlie's timelines}} \cup \underbrace{\{\text{end}(x, v) \mid x \in SV, v \in V_x, \gamma_x(v) = c\}}_{\text{end controllable tokens}} \quad (1)$$

$$A_E = \underbrace{\{\text{start}(x, v) \mid x \in SV_E, v \in V_x\}}_{\text{start tokens on Eve's timelines}} \cup \underbrace{\{\text{end}(x, v) \mid x \in SV, v \in V_x, \gamma_x(v) = u\}}_{\text{end uncontrollable tokens}} \quad (2)$$

Hence, players can start tokens for the variables that they own, and end the tokens that hold values that they control. Actions are combined into *moves* that can start/end multiple tokens at once.

Definition 6 (Moves). *A move μ_C for Charlie is a term of the form $\text{wait}(\delta_C)$ or $\text{play}(A_C)$, where $\delta_C \in \mathbb{N}^+$ and $\emptyset \neq A_C \subseteq A_C$ is either a set of starting actions or a set of ending actions.*

A move μ_E for Eve is a term of the form $\text{play}(A_E)$ or $\text{play}(\delta_E, A_E)$, where $\delta_E \in \mathbb{N}^+$ and $A_E \subseteq A_E$ is either a set of starting actions or a set of ending actions.

We denote by M_C and M_E the set of moves playable by *Charlie* and *Eve*, respectively. Moves such as $\text{play}(A_C)$ and $\text{play}(\delta_E, A_E)$ can play either $\text{start}(x, v)$ actions only or $\text{end}(x, v)$ actions only. A move of the former kind is called a *starting* move, while a move of the latter kind is called an *ending* move. We consider wait moves as *ending* moves. Moreover, Starting and ending moves have to be alternated during the game.

Definition 7 (Round). *A round ρ is a pair $(\mu_C, \mu_E) \in M_C \times M_E$ of moves such that:*

1. μ_C and μ_E are either both starting or both ending moves;
2. either $\rho = (\text{play}(A_C), \text{play}(A_E))$, or $\rho = (\text{wait}(\delta_C), \text{play}(\delta_E, A_E))$, with $\delta_E \leq \delta_C$;

A *starting* (*ending*) round is one made of starting (*ending*) moves. Note that since *Charlie* cannot play empty moves and wait moves are considered ending moves, each round is unambiguously either a starting or an ending round. Also note that since $\text{play}(\delta_E, A_E)$ moves are played only in rounds together with $\text{wait}(\delta_C)$, and $\text{wait}(\delta_C)$ is always an ending move, then any $\text{play}(\delta_E, A_E)$ must be an ending move. We can now define how a round is applied to the current partial plan to obtain the new one. The game always starts with a single *starting round*.

Definition 8 (Outcome of rounds). *Let $\bar{\mu} = \langle \mu_1, \dots, \mu_n \rangle$ be an event sequence, with $\mu_n = (A_n, \delta_n)$ or $\mu_n = (\emptyset, 0)$ if $\bar{\mu} = \varepsilon$. Let $\rho = (\mu_C, \mu_E)$ be a round, let δ_E and δ_C be the time increments of the moves, with $\delta_C = \delta_E = 1$ for $\text{play}(A)$ moves, and let A_E and A_C be the set of actions of the two moves (A_C is empty if μ_C is a wait move).*

The outcome of ρ on $\bar{\mu}$ is the event sequence $\rho(\bar{\mu})$ defined as follows:

1. if ρ is a starting round, then $\rho(\bar{\mu}) = \bar{\mu}_{<n} \mu'_n$, where $\mu'_n = (A_n \cup A_C \cup A_E, \delta_n)$;
2. if ρ is an ending round, then $\rho(\bar{\mu}) = \bar{\mu} \mu'$, where $\mu' = (A_C \cup A_E, \delta_E)$;

We say that ρ is applicable to $\bar{\mu}$ if:

- a) the above construction is well-defined, i.e., $\rho(\bar{\mu})$ is a valid event sequence by Definition 2;

b) ρ is an ending round if and only if $\bar{\mu}$ is open for all variables.

We say that a single move by either player is applicable to $\bar{\mu}$ if there is a move for the other player such that the resulting round is applicable to $\bar{\mu}$.

The game starts from the empty partial plan ε , and players play in turn, composing a round from the move of each one, which is applied to the current partial plan to obtain the new one.

It is now time to define the notion of *strategy* for each player, and of *winning strategy* for Charlie.

Definition 9 (Strategies). A strategy for Charlie is a function $\sigma_C : \Pi \rightarrow M_C$ that maps any given partial plan $\bar{\mu}$ to a move μ_C applicable to $\bar{\mu}$. A strategy for Eve is a function $\sigma_E : \Pi \times M_C \rightarrow M_E$ that maps a partial plan $\bar{\mu}$ and a move $\mu_C \in M_C$ applicable to $\bar{\mu}$, to a μ_E such that $\rho = (\mu_C, \mu_E)$ is applicable to $\bar{\mu}$.

A sequence $\bar{\rho} = \langle \rho_0, \dots, \rho_n \rangle$ of rounds is called a *play* of the game. A play is said to be *played according to* some strategy σ_C for Charlie, if, starting from the initial partial plan $\bar{\mu}_0 = \varepsilon$, it holds that $\rho_i = (\sigma_C(\Pi_{i-1}), \mu_E^i)$, for some μ_E^i , for all $0 < i \leq n$, and to be played according to some strategy σ_E for Eve if $\rho_i = (\mu_C^i, \sigma_E(\Pi_{i-1}, \mu_C^i))$, for all $0 < i \leq n$. It can be seen that for any pair of strategies (σ_C, σ_E) and any $n \geq 0$, there is a unique run $\bar{\rho}_n(\sigma_C, \sigma_E)$ of length n played according both to σ_C and σ_E .

Then, we say that a partial plan $\bar{\mu}$, and the play $\bar{\rho}$ such that $\bar{\mu} = \bar{\rho}(\varepsilon)$, are *admissible*, if the partial plan satisfies the domain rules, and are *successful* if the partial plan satisfies the system rules.

Definition 10 (Admissible strategy for Eve). A strategy σ_E for Eve is admissible if for each strategy σ_C for Charlie, there is $k \geq 0$ such that the play $\bar{\rho}_k(\sigma_C, \sigma_E)$ is admissible.

Charlie wins if, assuming domain rules are respected, he manages to satisfy the system rules no matter how Eve plays.

Definition 11 (Winning strategy for Charlie). Let σ_C be a strategy for Charlie. We say that σ_C is a winning strategy for Charlie if for any admissible strategy σ_E for Eve, there exists $n \geq 0$ such that the play $\bar{\rho}_n(\sigma_C, \sigma_E)$ is successful.

We say that Charlie wins the game G if he has a winning strategy, while Eve wins the game if a winning strategy for Charlie does not exist.

3 A deterministic automaton for timeline-based planning

In this section we encode a timeline-based planning problem into a *deterministic* finite state automaton (DFA) that recognises all and only those event sequences that represent solution plans for such problem. This automaton will form the basis for the game arena solved in the next section. The words accepted by the automaton are *event sequences* representing solution plans.

Let $P = (SV, S)$ be a timeline-based planning problem. To get a finite alphabet, we define $d = \max(L, U) + 1$, where L and U are in turn the *maximum* lower and (finite) upper bounds appearing in any rule of P , and we account only for event sequences such that the distance between two consecutive events is at most d . It can be easily seen that this assumption does not loose generality (for a proof, see Lemma 4.8 in [15]). Hence, the symbols of the alphabet Σ are *events* of the form $\mu = (A, \delta)$, where $A \subseteq A_{SV}$ and $1 \leq \delta \leq d$. Formally, $\Sigma = 2^{A_{SV}} \times [d]$, where $[d] = \{1, \dots, d\}$. Note that the size of Σ is exponential in the size of the problem. Moreover, we define the amount $\text{window}(P)$ as the product of all the non-zero coefficients appearing as upper bounds in rules of P . Intuitively, $\text{window}(P)$ is the maximum amount of time a rule of P can *count* far away from the occurrence of the quantified tokens. For example, consider the following rule:

$$a_0[x_0 = v_0] \rightarrow \exists a_1[x_1 = v_1] a_2[x_2 = v_2] a_3[x_3 = v_3].$$

$$\text{start}(a_1) \leq_{[4,14]} \text{end}(a_0) \wedge \text{end}(a_0) \leq_{[0,+\infty]} \text{end}(a_2) \wedge \text{start}(a_2) \leq_{[0,3]} \text{end}(a_3)$$

	start(a_0)	end(a_0)	start(a_1)	end(a_1)	start(a_2)	end(a_2)	start(a_3)	end(a_3)
start(a_0)								
end(a_0)			-4					
start(a_1)		14						
end(a_1)								
start(a_2)							3	
end(a_2)		0						
start(a_3)								
end(a_3)					0			

Figure 1: DBM of an example synchronization rule. Missing entries are intended to be $+\infty$.

In this case, $\text{window}(P)$ (assuming this is the only rule of the problem), would be $3 \cdot 14 = 42$. This means the rule can precisely account for what happens at most 42 time point from the occurrence of its quantified tokens. For example, if the token a_1 appears at a given distance from a_0 , it has to be at less than 42 time points (less than 14, in particular), and any modification of the plan that changes such distance has the potential to break the satisfaction of the rule. Instead, what happens further away from a_0 only affects the satisfaction of the rule *qualitatively*. Suppose the tokens a_2 and a_3 lie at 100 time points from a_0 (at most 3 time steps from each other). Changing this distance (while maintaining the qualitative order between tokens) cannot ever break the satisfaction of the rule. See [15] for a precise account of the properties of $\text{window}(P)$.

A key observation underlying our construction is that every atomic temporal relation $T \leq_{l,u} T'$ can be rewritten as the conjunction of two inequalities $T' - T \leq u$ and $T - T' \leq -l$, and that the clause C of an existential statement E can be rewritten as a system of difference constraints $v(C)$ of the form $T - T' \leq n$, with $n \in \mathbb{Z}_{+\infty}$. Then, the system $v(C)$ can be conveniently represented by a squared matrix D indexed by terms, where the entry associated with $D[T, T']$ gives the upper bound on $T - T'$. Such matrices, which take the name of *Difference Bound Matrices* (DBMs) [12, 20], can be conveniently updated as the plan evolves to keep track of the satisfaction of the atomic temporal relations among terms. In building a DBM for the system of constraints $v(C)$, we augment the system with constraints of kind $\text{start}(a_i) - \text{end}(a_i) \leq -d_{\min}^{x_i=v_i}$ and $\text{end}(a_i) - \text{start}(a_i) \leq d_{\max}^{x_i=v_i}$, for any quantified token $a_i[x_i = v_i]$ of E. Moreover, if two different bounds $T - T' \leq n$ and $T - T' \leq n'$ with $n' < n$ belong to $v(C)$, we keep only $T - T' \leq n'$. As an example, the DBM for the existential graph of the rule above is the one in Fig. 1. Note that, when the bounds of the temporal relations are translated into a DBM, there is no longer a distinction between *lower* and *upper* bounds. However, for some of the entries we can retrieve their original meaning. Indeed, if $D[T, T'] < 0$, then such entry is the lower bound of a temporal relation $T \leq_{l,u} T'$, whereas, if $D[T, T'] > 0$, it is the upper bound of a relation $T' \leq_{l,u} T$.

On top of DBMs, we define the concept of *matching structure*, a data structure that allows us to manipulate and reason about partially matched existential statements, *i.e.*, existential statements of which only a part of the requests has already been satisfied by the part of the word already read, while the rest can be still potentially matched in the future.

Definition 12 (Matching Structure). *Let $E \equiv \exists a_1[x_1 = v_1] \dots a_m[x_m = v_m]. C$ be the existential statement of a synchronisation rule $R \equiv a_0[x_0 = v_0] \rightarrow E_1 \vee \dots \vee E_k$ over the set of state variables SV.*

The matching structure for E is a tuple $M_E = (V, D, M, t)$ where:

- V is the set of terms $\text{start}(a)$ and $\text{end}(a)$ for $a \in \{a_0, \dots, a_m\}$;
- $D \in \mathbb{Z}_{+\infty}^{|V|^2}$ is a DBM indexed by terms of V where $D[T, T'] = n$ if $(T - T' \leq n) \in v(C)$, $D[T, T'] = 0$ if $T = T'$, and $D[T, T'] = +\infty$ otherwise;
- $M \subseteq V$ and $0 \leq t \leq \text{window}(P)$.

The set M contains the terms of V that the matching structure has correctly matched over the event sequence read so far. With $\overline{M} = V \setminus M$ we denote the actions that we have yet to see. Then, we say that a matching structure M is *closed* if $M = V$, it is *initial* if $M = \emptyset$ and it is *active* if it is not closed and $\text{start}(a_0) \in M$. Intuitively, a matching structure is *active* if its trigger has been matched over the word the automaton is reading. Then, when all the terms have been matched over the word, the matching structure becomes *closed*. The component t is the time elapsed since $\text{start}(a_0)$ has been matched. When time flows, a matching structure can then be updated as follows.

Definition 13 (Time shifting). *Let $\delta > 0$ be a positive amount of time, and $M = (V, D, M, t)$ be a matching structure. The result of shifting M by δ time units, written $M + \delta$, is the matching structure $M' = (V, D', M, t')$ where:*

- for all $T, T' \in V$:

$$D'[T, T'] = \begin{cases} D[T, T'] + \delta & \text{if } T \in M \text{ and } T' \in \overline{M} \\ D[T, T'] - \delta & \text{if } T \in \overline{M} \text{ and } T' \in M \\ D[T, T'] & \text{otherwise} \end{cases}$$

- and

$$t' = \begin{cases} t + \delta & \text{if } M \text{ is active} \\ t & \text{otherwise} \end{cases}$$

Definition 14 (Matching). *Let $M = (V, D, M, t)$ be a matching structure and $I \subseteq \overline{M}$ a set of matched terms. A matching structure $M' = (V, D, M', t)$ is the result of matching the set I , written $M \cup I$, if $M' = M \cup I$.*

Beside updating the reference t to the trigger occurrence of an active matching structure, Definition 13 dictates how to update the entries of the DBM. In particular, the distance bounds between any pair of terms T and T' where one is in M and the other is not are tightened by the elapsing of time: when $T \in M$ and $T' \in \overline{M}$, $D[T, T']$ is a lower bound loosen by adding the elapsed time δ , when $T \in \overline{M}$ and $T' \in M$, $D[T, T']$ is an upper bound tighten by subtracting δ . For example, consider the DBM in Fig. 1 and consider the pair of terms $\text{start}(a_1)$ and $\text{end}(a_0)$. $D[\text{start}(a_1), \text{end}(a_0)] = -4$, meaning that $\text{end}(a_0) - \text{start}(a_1) \leq 4$ must hold. Suppose $\text{start}(a_1) \in M$ (i.e., it has been matched), and $\text{end}(a_0) \notin M$ (it still has to). Now, if 1 time point passes, the entry in the DBM is incremented and updated to $-4 + 1 = -3$, which corresponds to the constraint $\text{end}(a_0) - \text{start}(a_1) \leq 3$. This reflects the fact that to be able to satisfy the constraint, $\text{end}(a_0)$ has now only 3 time steps left before it is too late. Definition 14 tells us how to update the set M of a matching structure.

To correctly match an existential statement while reading an event sequence, a matching structure is updated only as long as no violations of temporal constraints are witnessed. As such, an event is classified from the standpoint of a matching structure as *admissible* or not.

Definition 15 (Admissible Event). *An event $\mu = (A, \delta)$ is admissible for a matching structure $M_E = (V, D, M, t)$ if and only if, for every $T \in M$ and $T' \in \overline{M}$, $\delta \leq D[T', T]$, i.e., the elapsing of δ time units does not exceed the upper bound of some term T' not yet matched by M_E .*

Each admissible event μ read from the word can be matched with a subset of the terms of the matching structure. There are usually more than one way to match events and terms. The following definition makes this choice explicit.

Definition 16 (*I-match Event*). Let $M_E = (V, D, M, t)$ be a matching structure and $I \subseteq \overline{M}$. An *I-match event* is an admissible event $\mu = (A, \delta)$ for M_E such that:

1. for all token names $a \in N$ quantified as $a[x = v]$ in E we have that:
 - (a) if $\text{start}(a) \in I$, then $\text{start}(x, v) \in A$;
 - (b) $\text{end}(a) \in I$ if and only if $\text{start}(a) \in M$ and $\text{end}(x, v) \in A$;
2. and for all $T \in I$ it holds that:
 - (a) for every other term $T' \in V$, if $D[T', T] \leq 0$, then $T' \in M \cup I$;
 - (b) for all $T' \in M$, $\delta \geq -D[T', T]$, i.e., all the lower bounds on T are satisfied;
 - (c) for each other term $T' \in I$, either $D[T', T] = 0$, $D[T, T'] = 0$, or $D[T', T] = D[T, T'] = +\infty$.

Intuitively, an event is an *I-match event* if the actions in the event correctly match the terms in I . Item 1 ensures that each term is correctly matched over an action it represents and, most importantly, that the endpoints of a quantified token correctly identify the endpoints of a token in the event sequence. Item 2 ensures that matching the terms in I does not violate any atomic temporal relation. In particular, Item 2a deals with the qualitative aspect of an “happens before” relation, while Items 2b and 2c deal with the quantitative aspects of the lower bounds of these relations. Note that an \emptyset -event is admitted.

Let \mathbb{M}_P be the set of all the matching structures for a planning problem P . By Definition 16, a single event can represent several *I-match events* for a matching structure, hence a matching structure can evolve into several matching structures, one for each *I-match event*. Such evolution is defined as a ternary relation $S \subseteq \mathbb{M}_P \times \Sigma \times \mathbb{M}_P$ such that $(M, (A, \delta), M') \in S$ if and only if (A, δ) is an *I-match event* for M and $M' = (M + \delta) \cup I$. To deal with the nondeterministic nature of this relation, states of the automaton will comprise sets of matching structures collecting all the possible outcomes of S , so that suitable notation for working with sets of matching structures, denoted by Υ hereafter, is introduced. We define $\Upsilon_t^R \subseteq \Upsilon$ as the set of all the *active* matching structures $M \in \Upsilon$ with timestamp t , associated with an existential statement of R . Intuitively, matching structures in Υ_t^R contribute to the fulfilment of the same triggering event for the rule R (because they have the same timestamp), regardless of the existential statement they represent. We also define $\Upsilon_\perp \subseteq \Upsilon$ as the set of *non active* matching structures of Υ . A set Υ is *closed* if there exists $M \in \Upsilon$ such that M is *closed*. Lastly, a function step_μ extends the relation S to sets of matching structures: $\text{step}_\mu(\Upsilon) = \{M' \mid (M, \mu, M') \in S, \text{ for some } M \in \Upsilon\}$.

We are now ready to define the automaton. If E is an existential statement, let \mathbb{E}_E be the set of all the existential statements of the same rule of E . Let \mathbb{F}_P be the set of functions mapping each existential statement of P to a set of existential statements, and let \mathbb{D}_P be the set of functions mapping each existential statement to a set of matching structures. A simple automaton T_P that checks the transition function and duration functions of the variables is easy to define. Then, given a timeline-based planning problem $P = (SV, S)$, the corresponding automaton is $A_P = T_P \cap S_P$ where S_P , the automaton that checks the satisfaction of the synchronization rules, is defined as $S_P = (Q, \Sigma, q_0, F, \tau)$, where:

1. $Q = 2^{\mathbb{M}} \times \mathbb{D} \times \mathbb{F} \cup \{\perp\}$ is the finite set of states, i.e., states are tuples of the form $\langle \Upsilon, \Delta, \Phi \rangle \in 2^{\mathbb{M}} \times \mathbb{D} \times \mathbb{F}$, plus a sink state \perp ;
2. Σ is the input alphabet defined above;
3. the initial state $q_0 = \langle \Upsilon_0, \Delta_0, \Phi_0 \rangle$ is such that Υ_0 is the set of initial matching structures of the existential statements of P and, for all existential statements E of P , we have $\Delta_0(E) = \emptyset$ and $\Phi_0(E) = \mathbb{E}_E$;

4. $F \subseteq Q$ is the set of final states defined as:

$$F = \left\{ \langle \Upsilon, \Delta, \Phi \rangle \in Q \mid \begin{array}{l} M \text{ is not } \textit{active} \text{ for all } M \in \Upsilon \\ \text{and } \Delta(E) = \emptyset \text{ for all } E \text{ of } P \end{array} \right\}$$

5. $\tau : Q \times \Sigma \rightarrow Q$ is the transition function that given a state $q = \langle \Upsilon, \Delta, \Phi \rangle$ and a symbol $\mu = (A, \delta)$ computes the new state $\tau(q, \mu)$. Let $\text{step}_\mu^E(\Upsilon_t^R) = \{ M_E \mid M_E \in \text{step}_\mu(\Upsilon_t^R) \}$. Moreover, let $\Psi_t^R = \{ E \mid M_E \in \text{step}_\mu(\Upsilon_t^R) \}$. Then, the updated components of the state are based on what follows, where $W = \text{window}(P)$:

$$\begin{aligned} \Upsilon' &= \text{step}_\mu(\Upsilon_\perp) \cup \bigcup \left\{ \text{step}_\mu(\Upsilon_t^R) \mid t < W - \delta \text{ and } \text{step}_\mu(\Upsilon_t^R) \text{ is not } \textit{closed} \right\} \\ \Delta'(E) &= \begin{cases} \text{step}_\mu^E(\Upsilon_t^R) & \text{where } t \text{ is the minimum such that } t \geq W - \delta \text{ and } \text{step}_\mu^E(\Upsilon_t^R) \neq \emptyset \\ \text{step}_\mu(\Delta(E)) & \text{if such } t \text{ does not exist} \end{cases} \\ \Phi'(E) &= \begin{cases} \mathbb{E}_E & \text{if } E \in \Psi(E') \text{ for some } E' \text{ such that } \Delta'(E') \text{ is } \textit{closed} \\ \Phi(E) \setminus \{ E' \mid \exists t \geq W - \delta . E' \in \Psi_t^R \wedge E \notin \Psi_t^R \} & \text{otherwise} \end{cases} \end{aligned}$$

Let $\Delta''(E) = \Delta'(E)$ unless there is an E' with $E \in \Phi'(E')$ such that $\Delta'(E')$ is *closed*, in which case $\Delta''(E) = \emptyset$. Then, $\tau(q, \mu) = \langle \Upsilon', \Delta'', \Phi' \rangle$ if the following holds:

- (a) for every Υ_t^R , $\text{step}_\mu(\Upsilon_t^R) \neq \emptyset$, and
- (b) for every synchronisation rule $R \equiv a_0[x_0 = v_0] \rightarrow E_1 \vee \dots \vee E_n$ in S , if $\text{start}(x_0, v_0) \in A$, then there exists $M_{E_i} = (V, D, M, 0) \in \Upsilon'$, with $i \in \{1 \dots n\}$, such that $\text{start}(a_0) \in M$;

Otherwise, $\tau(q, \mu) = \perp$.

Let us explain what is going on. The first component Υ of an automaton state $q = (\Upsilon, \Delta, \Phi)$ is a set of matching structures that keeps track of what have been tracked so far. Intuitively, the automaton precisely keeps track of what happened to the last $\text{window}(P)$ time points, and only summarizes what happened before that window, which is what allows us to keep the size under control. Any matching structure in Υ has $t < \text{window}(P)$. Matching structures in Υ evolve following the step function, until they are closed or the t component reaches $\text{window}(P)$. Matching structures that reach $\text{window}(P)$ are promoted to a new role. Their new task is to record the pieces of existential statements that still have to be matched in order to satisfy all the trigger events of R that no longer fit into the *recent history* of the event sequence (*i.e.*, the last $\text{window}(P)$ time points). These matching structures are not stored in Υ though, they are summarized by the function Δ that maps each existential statement E of a rule R to the set of matching structures for E with $t = \text{window}(P)$.

When a set Υ_t^R exceeds the bound $\text{window}(P)$, the Δ function must be updated by merging the information of Υ_t^R to the information already present in Δ . Now, it has to be noted that, by closing a set $\Delta(E)$, we can not conclude that every event that triggered R actually satisfies R . Indeed, there can be sets $\Delta(E)$ and $\Delta(E')$ that are in charge of the satisfaction of the same rule R , but for different trigger events, and closing $\Delta(E)$ does not imply that R has been satisfied. The opposite case may also arise, in which $\Delta(E)$ and $\Delta(E')$ contribute to the fulfilment of the same trigger events and closing either set suffices to satisfy R . To overcome the information lost when a set of matching structures gets added to the Δ function, the Φ function (the third component of the automaton states) maps each existential statement E to the set of existential statements E' such that $\Delta(E')$ tracks the fulfilment of the same trigger events of the set $\Delta(E)$. We use Φ as follows: when a set $\Delta(E)$ gets closed, we can discard its matching structures and all the matching structures of the sets $\Delta(E')$, with $E' \in \Phi(E)$.

One can prove the soundness and completeness of our construction.

Theorem 1. (*Soundness and completeness*) *Let $P = (SV, S)$ be a timeline-based planning problem and let A_P be the associated automaton. Then, any event sequence $\bar{\mu}$ is a solution plan for P if and only if $\bar{\mu}$ is accepted by A_P .*

Recall that we assumed the timestamp of each event of event sequences to be bounded, but since events can have an empty set of actions, Theorem 1 can actually deal with arbitrary event sequences, after adding suitable empty events. Now, let us look at the size of the automaton. Let E be the overall number of existential statements in P , which is linear in the size of P . It can be seen that $|\mathbb{D}_P| \in \mathcal{O}((2^{|\mathbb{M}_P|})^E) = \mathcal{O}(2^{E \cdot |\mathbb{M}_P|})$, *i.e.*, the number of Δ functions is doubly exponential in the size of P . Then, observe that $|\mathbb{F}_P| \in \mathcal{O}((2^E)^E) = \mathcal{O}(2^{E^2})$. Then, $|\mathbb{S}_P| \in \mathcal{O}(|\Sigma| \cdot 2^{|\mathbb{M}_P|})$, that is, the size of \mathbb{S}_P is at most exponential in the number of possible matching structures. To bound this number, let N be the largest finite constant appearing in P as bound in any atom or value duration function and let L be the length of the largest existential prefix of an existential statement occurring inside a rule of P . Notice that N is exponential in the size of P , since constants are expressed in binary, while $L \in \mathcal{O}(|P|)$. Then, the entries of a DBM for P , of which there is a number quadratic in L , are constrained to take values within the interval $[-N, N]$ (excluding the infinitary value $+\infty$), whose size is linear in N . By Definition 12, it follows that, for the planning problem P , $|\mathbb{M}_P| \in \mathcal{O}(N^{L^2} \cdot 2^L \cdot \text{window}(P))$, *i.e.*, the number of matching structures is at most exponential in the size of P . Hence, we proved the following:

Theorem 2 (Size of the automaton). *Let $P = (SV, S)$ be a timeline-based planning problem and let A_P be the associated automaton. Then, the size of A_P is at most doubly-exponential in the size of P .*

Note that this is the same size as the automaton built by Della Monica *et al.* [9], but their automaton was *nondeterministic*, while ours is by construction deterministic, essential for its use as a game arena.

4 Controller synthesis

In this section we use the deterministic automaton constructed above to obtain a deterministic arena where we can solve a simple reachability game for checking the existence of (and, in this case, to synthesize) a controller for the corresponding timeline-based game.

4.1 From the automaton to the arena

Let $G = (SV_C, SV_E, S, D)$ be a timeline-based game. We use the construction of the automaton explained in the previous section in order to obtain a game arena. However, the automaton construction considers a planning problem with a single set of synchronization rules, while here we have to account for the roles of both S and D .

To do that, let A_S and A_D be the deterministic automata built over the timeline-based planning problem $P_S = (SV_C \cup SV_E, S)$ and $P_D = (SV_C \cup SV_E, D)$, respectively. We define the automaton A_G as $\overline{A_D} \cup A_S$, *i.e.*, the union of A_S with the complement of A_D . Note that these are all standard automata-theoretic constructions over DFAs. Any accepting run of A_G represents either a plan that violates the domain rules or a plan that satisfies both the domain and the system rules, in conformance with Definition 11. Note that A_G is deterministic and can be built from A_D and A_S with only a polynomial increase in size.

Now, the A_G automaton is still not suitable as a game arena, because the moves of the timeline-based game are not directly visible in the labels of the transitions. In other words, the A_G automaton reads events, while we need an automaton that reads game *moves*. In particular, a single transition in

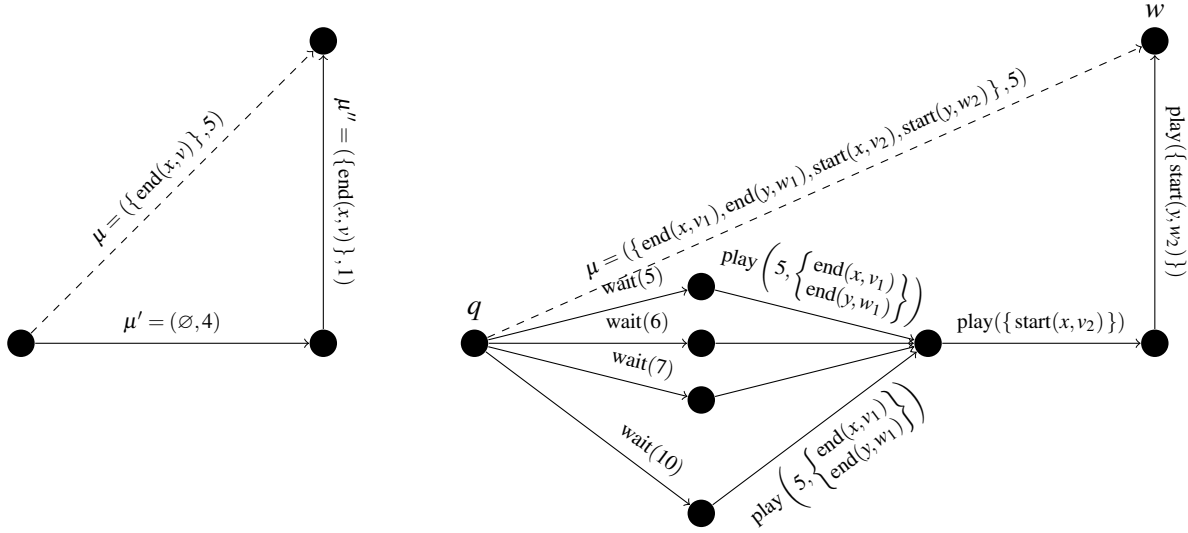


Figure 2: On the left, the removal of transitions $\mu = (A, \delta)$ with $\delta > 1$ and ending actions of controllable tokens in A . On the right, the transformation of a transition of the A_G into a sequence of transitions in A_G^a , with $x \in SV_C$, $y \in SV_E$, and $\gamma_x(v_1) = \gamma_y(w_1) = u$.

the automaton can correspond to different combinations of rounds, since the presence of $\text{wait}(\delta)$ moves is not explicit in the transition. For example, an event $\mu = (A, 5)$ can be the result of a $\text{wait}(5)$ move by *Charlie* followed by a $\text{play}(5, A)$ move by *Eve*, or by any $\text{wait}(\delta)$ move with $\delta > 5$ followed by $\text{play}(5, A)$. Hence, we need to further adapt A_G to obtain a suitable arena.

Let $A_G = (Q, \Sigma, q_0, F, \tau)$ be the automaton built as described before. Let $\mu = (A, \delta)$ be an event. If $\delta > 1$, this transition must have resulted from *Charlie* playing a $\text{wait}(\delta')$ move with $\delta' \geq \delta$. However, if A contains any $\text{end}(x, v)$ action with $x \in SV_C$, this is for sure the result of more than one pair of starting/closing rounds. In order to simplify the construction below, we remove this possibility beforehand. More formally, we define a slightly different automaton $A'_G = (Q, \Sigma, q_0, F, \tau')$ where τ' is now a *partial* transition function (*i.e.*, the automaton becomes *incomplete*) that agrees with τ on everything excepting that transitions $\tau(q, (A, \delta))$ is *undefined* if $\delta > 1$ and A contains any $\text{end}(x, v)$ action with $x \in SV_C$. You can see an example of this operation in Fig. 2, on the left. Note that this removal does not change the plans accepted by the automaton because for each transition $\tau(q, (A, \delta)) = q'$ with $\delta > 1$ there are two transitions $\tau(q, (\emptyset, \delta - 1)) = q''$ and $\tau(q'', (A, 1)) = q'$.

Now we can transform the automaton in order to make the game rounds, and especially $\text{wait}(\delta)$ moves, explicit. Intuively, each transition of the automaton is split into four transitions explicating the four moves of the two rounds. Given the automaton $A'_G = (Q, \Sigma, q_0, F, \tau')$, we define the automaton $A_G^a = (Q^a, \Sigma^a, q_0^a, F^a, \tau^a)$, which will be the arena of our game, as follows:

1. $Q^a = Q \cup \{q_\delta \mid 1 \leq \delta \leq d\} \cup \{q_{\delta, A} \mid 1 \leq \delta \leq d, A \subseteq A\}$ is the set of states;
2. $\Sigma^a = M_C \cup M_E$, *i.e.*, the alphabet is turned into the set of moves of the two players;
3. $q_0^a = q_0$ and $F^a = F$, *i.e.*, initial and final states do not change;
4. the (partial) transition function τ^a is defined as follows. Let $w = \tau(q, \mu)$ with $\mu = (A, \delta)$. We distinguish the case where $\delta = 1$ or $\delta > 1$.

- (a) if $\delta = 1$, let $A_C \subseteq A$ and $A_E \subseteq A$ be the set of actions in A playable by *Charlie* and by *Eve*, respectively. Then:
- i. $\tau(q, \text{play}(A_C^e)) = q_{1, A_C^e}$, where A_C^e is the set of *ending* actions in A_C ;
 - ii. $\tau(q_{1, A_C^e}, \text{play}(A_E^e)) = q_{1, A_C^e \cup A_E^e}$, where A_E^e is the set of *ending* actions in A_E ;
 - iii. $\tau(q_{1, A_C^e \cup A_E^e}, \text{play}(A_C^s)) = q_{1, A_C^e \cup A_E^e \cup A_C^s}$, where A_C^s is the set of *starting* actions in A_C ;
 - iv. $\tau(q_{1, A_C^e \cup A_E^e \cup A_C^s}, \text{play}(A_E^s)) = w$, where A_E^s is the set of *starting* actions in A_E ;
- where the mentioned states are added to Q^a as needed.
- (b) if $\delta > 1$, let $A_C \subseteq A$ and $A_E \subseteq A$ be the set of actions in A playable by *Charlie* and by *Eve*, respectively. Note that by construction, A_C only contains *starting* actions. Then:
- i. $\tau(q, \text{wait}(\delta_C)) = q_{\delta_C}$ for all $\delta \leq \delta_C \leq d$;
 - ii. $\tau(q_{\delta_C}, \text{play}(\delta, A_E^e)) = q_{\delta, A_E^e}$ where A_E^e is the set of *ending* actions in A_E ;
 - iii. $\tau(q_{\delta, A_E^e}, \text{play}(A_C)) = q_{\delta, A_E^e \cup A_C}$;
 - iv. $\tau(q_{\delta, A_E^e \cup A_C}, \text{play}(A_E^s)) = w$ where A_E^s is the set of *starting* actions in A_E ;
- where the mentioned states are added to Q^a as needed.

All the transitions not explicitly defined above are undefined.

A graphical example of the above construction can be seen in Fig. 2, on the right. Note that the structure of the original A_G automaton is preserved by A_G^a . In particular, one can see that for each $q \in Q$ and event $\mu = (A, \delta)$, any sequence of moves whose outcome would append μ to the partial plan (see Definition 8) reach from q the same state w in A_G^a that is reached in A_G by reading μ . Hence, one can consider A_G^a to also being able to *read* event sequences, even though its alphabet is different. We denote as $[\bar{\mu}]$ the state $q \in Q^a$ reached by reading $\bar{\mu}$ in A_G^a .

Moreover, note that, with a minimal abuse of notation, any play \bar{p} for the game G can be seen as a word readable by the automaton A_G^a . Hence, we can prove the following.

Theorem 3. *If G is a timeline-based game, for any play \bar{p} for G , \bar{p} is successful if and only if it is accepted by A_G^a .*

4.2 Computing the Winning Strategy

Once built the arena, we can focus on computing the winning region W_C for *Charlie*, that is, the set of states of the arena from which *Charlie* can force the play to reach a final state of A_G^a , no matter of the strategy of *Eve*. These games are called *reachability games* [21]. If the winning region W_C is not empty, a winning strategy of *Charlie* can be simply derived from W_C . As a consequence of Theorems 1 and 3, the computed winning strategy σ_C for A_G^a respects Definition 11.

As stated in [21, Theorem 4.1], reachability games are determined, and the winning region W_C along with the corresponding positional winning strategy s are computable. Let $A_G^a = (Q^a, \Sigma^a, q_0^a, F^a, \tau^a)$ be the automaton built from G as described in the previous section. Note that, by construction, in any state $q \in Q^a$ only one of the players has available moves. Let $Q_C^a \subseteq Q^a$ be the set of states *belonging to Charlie*, i.e., states from which *Charlie* can move, and let $Q_E^a = Q^a \setminus Q_C^a$. Moreover, let $E = \{(q, q') \in Q^a \times Q^a \mid \exists \mu. \tau^a(q, \mu) = q'\}$, i.e., the set of all the edges of A_G^a .

Now, for each $i \geq 0$, we can compute the i -th attractor of F^a , written $\text{Attr}_C^i(F^a)$, that is, the set of

states from which *Charlie* can win in at most i steps. $Attr_C^i(F^a)$ is defined as follows:

$$\begin{aligned} Attr_C^0(F^a) &= F^a \\ Attr_C^{i+1}(F^a) &= Attr_C^i(F^a) \\ &\cup \{q^a \in Q_C^a \mid \exists r((q^a, r) \in E \wedge r \in Attr_C^i(F^a))\} \\ &\cup \{q^a \in Q_E^a \mid \forall r((q^a, r) \in E \rightarrow r \in Attr_C^i(F^a))\} \end{aligned}$$

As remarked in [21], the sequence $Attr_C^0(F^a) \subseteq Attr_C^1(F^a) \subseteq Attr_C^2(F^a) \subseteq \dots$ becomes stationary for some index $k \leq |Q^a|$. Thus, we define $Attr_C(F^a) = \bigcup_{i=0}^{|Q^a|} Attr_C^i(F^a)$. In order to prove that $W_C = Attr_C(F^a)$, it suffices to use the proof of [21, Theorem 4.1] for showing that $Attr_C(F^a) \subseteq W_C$ and $W_C \subseteq Attr_C(F^a)$.

To compute a winning strategy for *Charlie* in the case that $q_0^a \in W_C$, it is sufficient to define $s(q) = \mu$ for any μ such that $\tau^a(q, \mu) = q'$ with $q, q' \in W_C$ (which is guaranteed to exist by construction of the attractor). Then, the strategy σ_C for *Charlie* in G (see Definition 11) is defined as $\sigma_C(\bar{\mu}) = s([\bar{\mu}])$.

Theorem 4. *Given $A_G^a = (Q^a, \Sigma^a, q_0^a, F^a, \tau^a)$, $q_0^a \in W_C$ if and only if *Charlie* has a winning strategy σ_C for G .*

Proof. We first prove *soundness*, that is, $q_0^a \in W_C$ implies that *Charlie* has a winning strategy σ_C for G . If $q_0^a \in W_C$, then it means that there exists a positional winning strategy s for *Charlie* for the reachability game over the arena A_G^a . By Theorem 3 and by the definition of reachability game, we know that each play generated by s corresponds to a successful play for game G . Let $\sigma_C(\bar{\mu}) = s([\bar{\mu}])$ be the winning strategy for *Charlie* in game G as defined above. By construction of σ_C and by Definition 11, this means that σ_C is a winning strategy of *Charlie* for G .

To prove *completeness* (i.e., if *Charlie* has a winning strategy σ_C for G then $q_0^a \in W_C$), we proceed as follows. From Definition 11 we know that a winning strategy σ_C for *Charlie* is a strategy such that for every admissible strategy σ_E for *Eve*, there exists $n \geq 0$ such that the play $\bar{p}_n(\sigma_C, \sigma_E)$ is successful. From Theorem 3, we know that $\bar{p}_n(\sigma_C, \sigma_E)$ is accepted by A_G^a . Therefore, $\bar{p}_n(\sigma_C, \sigma_E)$ reaches a state in the set F^a starting from q_0^a . By definition of reachability game, this means that $q_0^a \in W_C$. \square

5 Conclusions

In this paper, we completed the picture about timeline-based games by providing an effective procedure for controller synthesis, whereas before only a proof of the complexity of the strategy existence problem was known. Previous approaches either provided a deterministic concurrent game structure which was however not built effectively, or an effectively built automata which was, however, nondeterministic and thus unsuitable for use as a game arena without a costly determinization. Our approach surpasses the limits of both previous ones by providing a deterministic construction, of optimal asymptotic size, suitable to be used as a game arena. Then, we solve the reachability game on the arena with standard methods to effectively compute the winning strategy for the game, if it exists.

This work paves the way to interesting future developments. On the one hand, the effective procedure shown here can be finally implemented, bringing timeline-based games from theory to practice. On the other hand, developing an effective system based on such games requires to answer many interesting questions, from which concrete modeling language to adopt, to which algorithmic improvements are needed to make the approach feasible. For example, it can be foreseen that, to solve the fixpoint computation that leads to the strategy with reasonable performance, the application of *symbolic techniques* would be needed.

Acknowledgements

Nicola Gigante and Luca Geatti acknowledge the support of the Free University of Bozen-Bolzano, Faculty of Computer Science, by means of the projects TOTA (*Temporal Ontologies and Tableaux Algorithms*) and STAGE (*Synthesis of Timeline-based Planning Games*).

References

- [1] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-time temporal logic*. *J. ACM* 49(5), pp. 672–713, doi:10.1145/585265.585270.
- [2] Sara Bernardini & David E. Smith (2007): *Developing Domain-Independent Search Control for Europa2*. In: *Proceedings of the ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning*.
- [3] Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron & Gerhard J. Woeginger (2020): *Timeline-based planning over dense temporal domains*. *Theor. Comput. Sci.* 813, pp. 305–326, doi:10.1016/j.tcs.2019.12.030.
- [4] Amedeo Cesta, Gabriella Cortellessa, Michel Denis, Alessandro Donati, Simone Fratini, Angelo Oddi, Nicola Policella, Erhard Rabenau & Jonathan Schulster (2007): *Mexar2: AI Solves Mission Planner Problems*. *IEEE Intelligent Systems* 22(4), pp. 12–19, doi:10.1109/MIS.2007.75.
- [5] Amedeo Cesta, Gabriella Cortellessa, Simone Fratini, Angelo Oddi & Nicola Policella (2006): *Software Companion: The Mexar2 Support to Space Mission Planners*. In Gerhard Brewka, Silvia Coradeschi, Anna Perini & Paolo Traverso, editors: *Proceedings of the 17th European Conference on Artificial Intelligence, Frontiers in Artificial Intelligence and Applications* 141, IOS Press, pp. 622–626.
- [6] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins & D. Tran (2000): *ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling*. In: *Proceedings of the International Conference on Space Operations*.
- [7] Steve A. Chien, Gregg Rabideau, Daniel Tran, Martina Troesch, Joshua Doubleday, Federico Nespoli, Miguel Perez Ayucar, Marc Costa Sitja, Claire Vallat, Bernhard Geiger, Nico Altobelli, Manuel Fernandez, Fran Vallejo, Rafael Andres & Michael Kueppers (2015): *Activity-Based Scheduling of Science Campaigns for the Rosetta Orbiter*. In Qiang Yang & Michael Wooldridge, editors: *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, AAAI Press, pp. 4416–4422. Available at <http://ijcai.org/Abstract/15/655>.
- [8] Marta Cialdea Mayer, Andrea Orlandini & Alessandro Umbrico (2016): *Planning and execution with flexible timelines: a formal account*. *Acta Informatica* 53(6-8), pp. 649–680, doi:10.1007/s00236-015-0252-z.
- [9] Dario Della Monica, Nicola Gigante, Angelo Montanari & Pietro Sala (2018): *A Novel Automata-Theoretic Approach to Timeline-Based Planning*. In Michael Thielscher, Francesca Toni & Frank Wolter, editors: *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning*, AAAI Press, pp. 541–550. Available at <https://aaai.org/ocs/index.php/KR/KR18/paper/view/18024>.
- [10] Dario Della Monica, Nicola Gigante, Angelo Montanari, Pietro Sala & Guido Sciavicco (2017): *Bounded Timed Propositional Temporal Logic with Past Captures Timeline-based Planning with Bounded Constraints*. In Carles Sierra, editor: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 1008–1014, doi:10.24963/ijcai.2017/140.
- [11] Dario Della Monica, Nicola Gigante, Salvatore La Torre & Angelo Montanari (2020): *Complexity of Qualitative Timeline-Based Planning*. In: *Proceedings of the 27th International Symposium on Temporal Representation and Reasoning, LIPICs* 178, pp. 16:1–16:13, doi:10.4230/LIPICs.TIME.2020.16.
- [12] David L. Dill (1989): *Timing Assumptions and Verification of Finite-State Concurrent Systems*. In Joseph Sifakis, editor: *Proceedings of the International Workshop on Automatic Verification Methods for Finite State*

- Systems, Lecture Notes in Computer Science* 407, Springer, pp. 197–212, doi:10.1007/3-540-52148-8_17.
- [13] Jeremy Frank & Ari K. Jónsson (2003): *Constraint-Based Attribute and Interval Planning*. *Constraints* 8(4), pp. 339–364, doi:10.1023/A:1025842019552.
- [14] Simone Fratini, Amedeo Cesta, Andrea Orlandini, Riccardo Rasconi & Riccardo De Benedictis (2011): *APSI-based Deliberation in Goal Oriented Autonomous Controllers*. In: *ASTRA 2011*, 11, ESA.
- [15] Nicola Gigante (2019): *Timeline-based Planning: Expressiveness and Complexity*. Ph.D. thesis, University of Udine, Italy. Available on *arXiv* at: <https://arxiv.org/abs/1902.06123>.
- [16] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer & Andrea Orlandini (2016): *Timelines Are Expressive Enough to Capture Action-Based Temporal Planning*. In Curtis E. Dyreson, Michael R. Hansen & Luke Hunsberger, editors: *Proceedings of the 23rd International Symposium on Temporal Representation and Reasoning*, IEEE Computer Society, pp. 100–109, doi:10.1109/TIME.2016.18.
- [17] Nicola Gigante, Angelo Montanari, Marta Cialdea Mayer & Andrea Orlandini (2017): *Complexity of Timeline-Based Planning*. In Laura Barbulescu, Jeremy Frank, Mausam & Stephen F. Smith, editors: *Proceedings of the 27th International Conference on Automated Planning and Scheduling*, AAAI Press, pp. 116–124. Available at <https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15758>.
- [18] Nicola Gigante, Angelo Montanari, Andrea Orlandini, Marta Cialdea Mayer & Mark Reynolds (2020): *On timeline-based games and their complexity*. *Theor. Comput. Sci.* 815, pp. 247–269, doi:10.1016/j.tcs.2020.02.011.
- [19] Nicola Muscettola (1994): *HSTS: Integrating Planning and Scheduling*. In Monte Zweben & Mark S. Fox, editors: *Intelligent Scheduling*, chapter 6, Morgan Kaufmann, pp. 169–212.
- [20] Mathias Péron & Nicolas Halbwachs (2007): *An abstract domain extending difference-bound matrices with disequality constraints*. In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*, Springer, pp. 268–282, doi:10.1007/978-3-540-69738-1_20.
- [21] Wolfgang Thomas (2008): *Solution of Church’s Problem: A tutorial*. *New Perspectives on Games and Interaction. Texts on Logic and Games* 5.

CryptoSolve: Towards a Tool for the Symbolic Analysis of Cryptographic Algorithms

Dalton Chichester

University of Mary Washington, Fredericksburg, VA, USA
dchiches@mail.umw.edu

Wei Du

University at Albany–SUNY, Albany, NY, USA^[0000–0002–9149–6229]
wdu2@albany.edu

Raymond Kauffman

University of Mary Washington, Fredericksburg, VA, USA
rkauffma@mail.umw.edu

Hai Lin

Clarkson University, Potsdam, NY, USA^[0000–0001–8658–9634]
hlin@clarkson.edu

Christopher Lynch

Clarkson University, Potsdam, NY, USA^[0000–0003–1141–0665]
clynch@clarkson.edu

Andrew M. Marshall

University of Mary Washington, Fredericksburg, VA, USA^[0000–0002–0522–8384]
amarsha2@umw.edu

Catherine A. Meadows

Naval Research Laboratory, Washington, DC, USA
catherine.meadows@nrl.navy.mil

Paliath Narendran

University at Albany–SUNY, Albany, NY, USA^[0000–0003–4521–5892]
pnarendran@albany.edu

Veena Ravishankar

University of Mary Washington, Fredericksburg, VA, USA^[0000–0003–3498–4039]
vravisha@umw.edu

Luis Rovira

University of Mary Washington, Fredericksburg, VA, USA
lrovira@umw.edu

Brandon Rozek

Rensselaer Polytechnic Institute, Troy NY, USA^[0000–0002–4537–559X]
rozekb@rpi.edu

Recently, interest has been emerging in the application of symbolic techniques to the specification and analysis of cryptosystems. These techniques, when accompanied by suitable proofs of soundness/completeness, can be used both to identify insecure cryptosystems and prove sound ones secure. But although a number of such symbolic algorithms have been developed and implemented, they remain scattered throughout the literature. In this paper, we present a tool, CryptoSolve, which provides a common basis for specification and implementation of these algorithms, CryptoSolve includes libraries that provide the term algebras used to express symbolic cryptographic systems, as well as implementations of useful algorithms, such as unification and variant generation. In its current initial iteration, it features several algorithms for the generation and analysis of cryptographic modes of operation, which allow one to use block ciphers to encrypt messages more than one block long. The goal of our work is to continue expanding the tool in order to consider additional cryptosystems and security questions, as well as extend the symbolic libraries to increase their applicability.

1 Introduction

Although security properties of cryptographic algorithms are generally proved using a computational model in which probabilities of events are explicitly quantified, there are often advantages to using a more easily automated abstract symbolic model. This is particularly the case when one is looking for cryptosystems that obey some non-cryptographic constraints, e.g. parallelizability, or even non-technical constraints, such as absence of intellectual property restrictions. One can use automated both methods to generate a large number of candidate cryptosystems, and to verify the security in the symbolic model. If the symbolic model is computationally sound (that is if the symbolic analogue of a particular security property holds) it is possible to use this technique to identify secure cryptosystems. Even the symbolic model is not computationally sound, but is computationally complete, it is possible to use the technique to weed out insecure constructions. A growing body of work, e.g. [3, 5, 7, 14]], shows how this can be applied to the construction of new cryptosystems. Symbolic methods can also be useful by themselves, even without automatic generation. For example, in [21] Venema and Alpár use symbolic methods to find security flaws in recently proposed attribute-based encryption schemes, in [8] Hollenberg, Rosulek, and Roy and in [16] Meadows respectively find different symbolic criteria guaranteeing the security of blockcipher cryptographic modes of operation under different usage assumptions, and in [15] McQuoid, Swope, and Rosulek develop a polynomial-time algorithm for checking security properties of a class of hash functions.

However, the symbolic problems we encounter often come with constraints tied to the properties of the cryptosystem, such as, requiring that any substitutions be constructible from terms and function symbols available to an adversary, or that the adversary may not be able to perform certain operations, such encryption with a key that is not available to it. Hence specialized algorithms or tools may be necessary.

In this paper we present an overview of an initial version of such a tool, CryptoSolve,¹ that has been designed to generate and analyze specifications of cryptosystems. This in turn allows for the automatic generation and symbolic analysis of certain cryptographic algorithms. The goal of this new tool is broad, to develop not only a usable analysis tool for an extensive family of cryptographic algorithms but to also develop the underlying libraries which could be used in analysis of additional algorithms, properties, and within other symbolic analysis tools.

Our initial version of CryptoSolve provides algorithms for reasoning about the security and functionality of a class of cryptosystems known as *cryptographic modes of operation*. These modes use fixed

¹The current version of the tool can be found here: <https://symcollab.github.io/CryptoSolve/>.

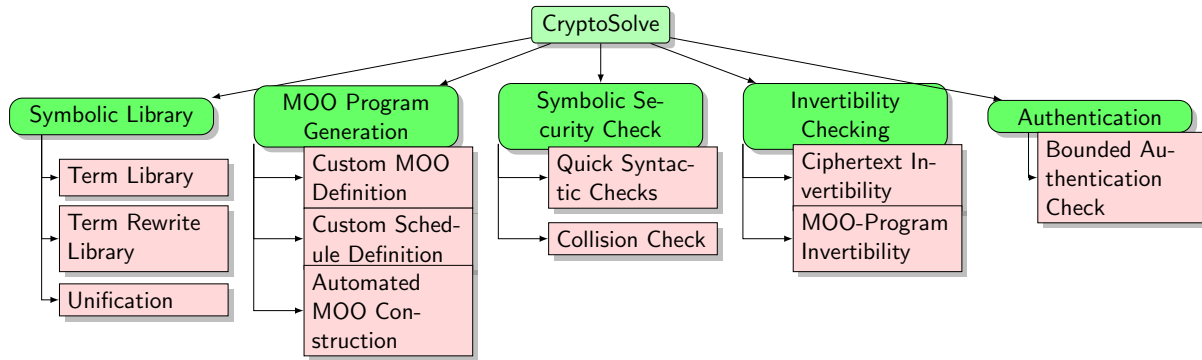


Figure 1: Tool modules and dependencies

length block ciphers to encrypt messages more than one block long. The key property of a mode is to protect the secrecy of the encrypted data, but it may also be used to provide integrity and authentication. In addition, the mode must be invertible by anyone who possesses the decryption key. CryptoSolve supports both the automated and manual generation of modes. It also features algorithms for checking secrecy (in the sense of indistinguishability of ciphertext from random), authenticity, and invertibility.

In the sections that follow we explain each of the above properties and detail how the tool works for every case. The algorithms implemented in CryptoSolve are supported by a set of base libraries for critical symbolic capabilities such as term representation, term rewriting, unification, and more. Currently the modules available in the tool and a simplified representation of their relation to each other is described in Figure 1.

Outline In the remainder of the paper we cover the current state and capabilities of the tool without focusing on the theory behind it, which can be found in [10–12, 16]. Where necessary, we provide a brief theoretical background and indicate aspects on which the tool is based. The rest of the paper is organized as follows. We provide a discussion of related work in Section 2. A brief review of necessary background material is covered in Section 3. An overview of the security modules, their use, capabilities, and pointers to the theory behind these methods are given in sections 4 and 5. The invertibility checking module is covered in Section 6, the bounded authentication checking module is covered in Section 7. We provide preliminary experimental results in Section 8. Finally, the conclusions and future work are discussed in Section 9.

2 Related Work

Publicly available tools for the generation and testing of security property of cryptographic algorithms (e.g. [3,5,7,14]) are the most closely related work to ours. Perhaps the first of these is the work by Barthe et al. [3]. This paper describes a tool, ZooCrypt, designed for the analysis of chosen plaintext and chosen ciphertext security public-key encryption schemes built from trapdoor permutations and hash functions. A ZooCrypt analysis of a cryptosystem consists of two stages. In the first stage a symbolic analysis tool is used to search for attacks on the cryptosystem. If none are found, the analysis enters the second stage, in which an automated theorem prover is used to search for a security proof in the computational model.

Later work looked at applying symbolic techniques and incorporates computational soundness results to prove computational security. For example, Malozemoff et al. [14] provide a symbolic algorithm whose successful termination implies adaptive chosen plaintext security of cryptographic modes of operation using the message-wise schedule. These results are extended by Hoang et al. in [7] to symbolic techniques for proving adaptive chosen ciphertext security of modes. Both papers also include software that implements symbolic algorithms for generating cryptosystems and proving their security. Other work by Carmer et al. [5] gives a symbolic algorithm for deciding security of garbled circuit schemes, and includes a tool, Linisynth, that generates such schemes and verifies their security using the algorithm.

All these tools have one thing in common: they only implement the algorithms described in the paper they accompany, and thus are intended mainly as proofs of concept, not as general tools for the generation and analysis of algorithms. The goal of CryptoSolve, however, is to serve as a tool for designing and experimenting with multiple types of cryptosystems, security properties, and algorithms. Thus, for example, it includes libraries for techniques that may prove useful in application to more than one cryptosystem, such as unification, variant generation, and the automatic generation of recursive functions. It is also extensible, allowing more libraries and algorithms to be added as necessary, and it includes an optional graphical user interface to make interactions with it easier. Currently, it can be applied to three different properties (static equivalence to random, invertibility, and authenticity, using five different algorithms) of cryptographic modes of operation.

There is also a large amount of related research in formulating and proving indistinguishability properties for the symbolic analysis of cryptographic protocols. These properties are analogous to the computational indistinguishability properties used in cryptography. The main differences are that symbolic indistinguishability does not always imply computational security (see, for example Unruh [20]), and 2) the symbolic algorithms are optimized for protocols, not crypto-algorithms, so applying them directly is not always advisable. Even so, the approaches used in symbolic protocol analysis can be helpful. For example an undecidability result in Lin et al. [12] is based on an undecidability result for cryptographic protocols analysis due to Küsters and Truderung [9]. To facilitate this interaction between symbolic protocol analysis and symbolic cryptography, we use a formal model and specification language, due to Baudet et al. [4], that is based on the the concept of *frames* used by the applied pi calculus [1], one of the most popular formal languages used by tools for the formal analysis of cryptographic protocols.

3 Preliminaries

We first need to briefly review some background material both on MOOs and symbolic security, and also on the underlying term rewriting theory used in the tool. We begin with with term rewriting and related concepts. Please note, additional background material on equational theories, rewriting, and unification can be found here [2].

3.1 Terms, Substitutions, and Equational Theories

Given a first-order signature Σ , a countable set N of variables bound by the symbol ν , and a countable set of variables X (s.t. $X \cap N = \emptyset$), the set of terms constructed from X , N , and Σ , is denoted by $T(\Sigma, N \cup X)$. Note that since N is a set of bound variables we can often treat these as constants in the first-order theory and thus won't apply substitutions to these bound variables. A substitution σ is an endomorphism of $T(\Sigma, N \cup X)$ with only finitely many variables not mapped to themselves, denoted by $\sigma = \{x_1 \mapsto t_1, \dots, x_m \mapsto t_m\}$. Application of a substitution σ to a term t is written $t\sigma$.

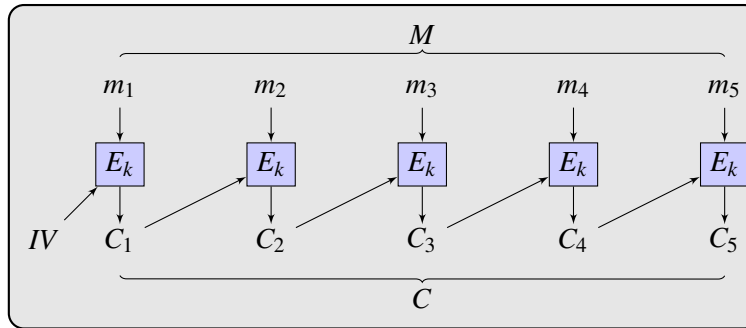


Figure 2: An example block cipher to be modeled by a symbolic history

Given a set E of Σ -axioms (i.e., pairs of Σ -terms, denoted by $l = r$), the *equational theory* $=_E$ is the congruence closure of E under the law of substitutivity. Since $\Sigma \cap N = \emptyset$, the Σ -equalities in E do not contain any bound variables in N . An E -unification problem with bound variables in N is a set of $\Sigma \cup N$ -equations $P = \{s_1 =? t_1, \dots, s_m =? t_m\}$. A solution to P , called an *E -unifier*, is a substitution σ such that $s_i \sigma =_E t_i \sigma$ for all $1 \leq i \leq m$.

The primary equational theory implemented in the tool is the theory of xor, denoted as E_{xor} . This theory can be represented as a combination of a rewrite system, R_{\oplus} , and an associative and commutative (AC) equational theory, E_{\oplus} . $E_{xor} = R_{\oplus} \cup E_{\oplus}$: $R_{\oplus} = \{x \oplus x \rightarrow 0, x \oplus 0 \rightarrow x\}$, $E_{\oplus} = AC(\oplus)$, over the signature $\Sigma_{\oplus} = \{\oplus/2, f/1, 0/0\}$. We will often denote this as the MOO_{\oplus} algebra and modes of operations defined in this algebra as $MOOs_{\oplus}$.

3.2 Modes of Operation and Symbolic Security

Before detailing the features of the tool we need to consider a few critical background notions such as Modes of Operation, Symbolic Security, and Authenticity. We do that in this section.

3.2.1 Modes and Their Security

A cryptographic mode of operation can be described at a high level as follows. The plaintext message M is first broken into fixed sized blocks. Each block m_i is processed using the block encryption function E_k along with some additional operations to produce a ciphertext block C_i . Typically, the previous ciphertext is used in the computation of the current block, and an initialization vector IV is used to add randomness to the first block. The final ciphertext is the sequence of ciphertexts thus produced. Figure 2 illustrates this process for Cipher Block Chaining (CBC) mode.

In order to model these modes so that they can be checked via symbolic methods, we use *symbolic histories* (defined in [16]). These describe interactions between the adversary and the oracle, in which the adversary sends blocks of plaintext to be encrypted, and the oracle sends back blocks of ciphertext according to some fixed *schedule* defined by the mode. E.g., in a block-wise schedule a ciphertext block is sent immediately after it is generated by the mode. In a message-wise schedule, the ciphertext blocks are not sent until after the entire message is encrypted.

The symbolic definition of security we use is based on the computational security property IND $\$$ -CPA introduced by Rogaway in [18]. This is defined in terms of a game in which a challenger first chooses one of two oracles with probability 1/2. The first is an encryption oracle that returns ciphertext when given plaintext, and the second is a random bits oracle that returns a string of random bits that

is as long as the ciphertext would have been. The adversary interacts with the oracle by sending it plaintexts and receiving the oracle's response. At any time it can stop the game and guess which oracle it is interacting with. Its *advantage* is defined to be $|\ .5 - p |$, where p is the probability that the adversary guesses correctly. A mode is IND $\$$ -CPA-secure if its advantage is negligible in some security parameter η , where a function g is said to be negligible if for every polynomial q there is an integer η_q such that $g(\eta) < \frac{1}{q(\eta)}$ for all $\eta > \eta_q$. In the case of modes of operation, the security parameter is the maximum of the block size and the key size. The motivation for a definition of this sort is that if the adversary cannot distinguish the output of the cryptosystem from random noise, then it learns nothing about the plaintext. This form of security, in which the security of a cryptosystem is quantified in terms of the adversary's inability to distinguish between the output of an encryption oracle and the output of an oracle that does not use the content of the plaintext in its calculations, is common in cryptography.

We note that if the adversary can create plaintexts that consistently cause a set of ciphertexts to exclusive-or to zero, then it can distinguish between the real and random case with overwhelming probability. If such an equality holds for the case in which the substitution is the identity, we say that the mode is *degenerate*. In all other cases it is necessary but not sufficient that the adversary must be able to consistently cause at least one given pair of f -rooted terms to be equal, known as a *collision*. We describe the symbolic model below, and then describe the unification problem that is associated with it.

3.2.2 The Symbolic Model and Symbolic Security

The blocks sent between the adversary and the oracle are modeled by terms in the MOO_{\oplus} algebra. These MOO_{\oplus} -terms consist of free variables representing plaintext blocks, bound variables representing a bitstring, and terms built up using these variables and the signature $\Sigma = \{\oplus/2, 0/0, f/1\}$, under the Xor equational theory, where f is the encryption function for some fixed key K , i.e., $enc(K, -) = f(-)$. Note that f is not computable by the adversary.

A *symbolic history* of the adversary's interaction with the oracle is modeled by a list of MOO_{\oplus} -terms of the form $[t_1, t_2, \dots, t_n]$. All MOO_{\oplus} -terms are listed in the order that they are sent. For example, the following symbolic history models the Cipher Block Chaining (CBC) mode of operation with three ciphertexts using the block-wise schedule: $vIV[IV, x_1, f(IV \oplus x_1), x_2, f(x_2 \oplus f(IV \oplus x_1))]$. Here IV is a bound variable representing an initialization vector. Each x_i models a plaintext block sent by the adversary and each f -rooted term is a ciphertext returned by the oracle according to the definition of the mode. For example, in *CBC* the i^{th} ciphertext C_i is modeled by $f(C_{i-1} \oplus x_i)$, where x_i is the i^{th} plaintext.

Each symbolic history models the interleaving of one or more *sessions* between the adversary and oracle, where a session is a history that encrypts a single message consisting of a sequence of plaintext blocks. In this case the initial nonces, the *IVs*, will be fresh for each session.

The notions of *computable substitutions* and *symbolic security* are defined by Lin et al. in [12]. Let P be a symbolic history. A substitution σ is *computable* w.r.t. P if σ maps each variable x_i to a term built up using the operators 0 and \oplus only on terms returned by the oracle prior to receiving x_i in P . A mode of operation M is *symbolically secure* if there is no symbolic history P of M such that there is a non-empty set of terms S returned by the encryptor in P where $\bigoplus_{t \in S} t \sigma =_{\oplus} 0$, and σ is computable substitution w.r.t. P . It is shown in [12] that a mode of operation M is symbolically secure if and only if M is *statically equivalent to random*; static equivalence [1] is a symbolic definition of indistinguishability commonly used in symbolic protocol analysis.

We note that if a mode satisfies IND $\$$ -CPA, then it must be symbolically secure, because if the adversary could make a substitution to the plaintext such that it always satisfies the same equation by the ciphertext, then it could easily distinguish the ciphertext from random with overwhelming probability. A

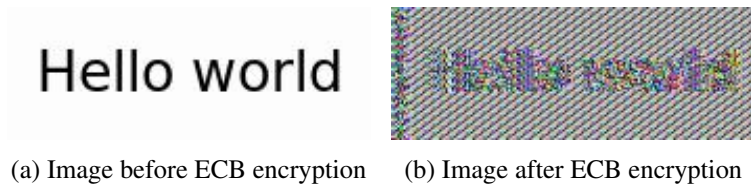


Figure 3: ECB encryption with AES 128 ECB

stronger condition has been shown by Meadows in [16] to imply IND\$-CPA security. It has two parts. The first is non-degeneracy, which requires that symbolic security hold for the trivial case in which the computable substitution σ is the identity. The second is the condition that no two different f -rooted terms have a computable unifier, whether or not it leads to a violation of symbolic security. This does not necessarily mean that symbolically secure modes that fail to satisfy the second condition are not IND\$-CPA secure, simply that more work may be required to prove them so.

3.2.3 Checking Symbolic Security: Examples

Let's consider several examples of symbolic histories and checking for symbolic security. We start with the classic example of an insecure mode: the Electronic Code Book (ECB) mode. In ECB, each block is encrypted separately, so plaintext x_1, \dots, x_n yields ciphertext $f(x_1), \dots, f(x_n)$. Notice that after applying ECB, the image in Figure 3 is still not completely scrambled and some information from the original picture can still be deduced. This is because whenever two plaintext blocks are identical they produce the same ciphertext blocks. Thus, any substitution unifying any two free variables is computable and leads to a violation of symbolic security.

Other MOOs may be symbolically secure or insecure depending on the schedule. For example, consider a symbolic history of CBC with three ciphertext blocks: $P_2 = \nu IV[IV, x_1, f(x_1 \oplus IV), x_2, f(x_2 \oplus f(x_1 \oplus IV))]$. We consider two schedules: the block-wise schedule, where each ciphertext block is returned to the adversary as soon as it can be computed, and the message-wise schedule, where they are returned all together at the end. Note that in the block-wise schedule there is a computable unifier of $f(x_1 \oplus IV)$ and $f(x_2 \oplus f(x_1 \oplus IV))$, namely $\sigma = \{x_1 \mapsto IV, x_2 \mapsto f(0)\}$, but this is not computable in the message-wise schedule, which can be shown to be symbolically secure and IND\$-CPA secure.

Finally, we consider one additional MOO, Output Feedback Mode (OFB). OFB starts with an initialization vector, IV , each consecutive block, C_{i+1} , is computed as $C_{i+1} = T_{i+1} \oplus x_{i+1}$, where x_i is the i^{th} plaintext block, and $T_i = f(T_{i-1})$ (T_0 being IV). For example, the first block would be $C_1 = f(IV) \oplus x_1$, and the second is $C_2 = f(f(IV)) \oplus x_2$. Consider an OFB history with three ciphertext blocks: $P_3 = \nu IV[IV, x_1, f(IV) \oplus x_1, x_2, f(f(IV)) \oplus x_2]$. Note that, in order to unify $f(IV) \oplus x_1$ and $f(f(IV)) \oplus x_2$, the adversary would have to set $\sigma_{x_2} = x_1 \oplus f(IV) \oplus f(f(IV))$, which it cannot do no matter what schedule is used, because it does not learn $f(f(IV))$ until after it has computed x_2 . OFB is also both symbolically secure and IND\$-CPA secure. Notice that when generating the ciphertexts for differing MOOs such as CBC and OFB, the root symbol of the ciphertexts could differ and this will impact the unification algorithm required.

4 MOO Representation

The tool contains a library implementation which allows for the representation and generation of MOO_{\oplus} -Programs. The library currently allows MOO_{\oplus} -Programs that are constructed over the signature $\Sigma = \{\oplus/2, 0/0, f/1\}$ and represented as a simple recursive function. Once a MOO_{\oplus} -Program has been defined, the library can then apply a number of operations on that MOO_{\oplus} -Program, including: generating terms in a run of the MOO_{\oplus} -Program, checking symbolic security of the program, and checking invertibility.

4.1 Standard and Custom MOO_{\oplus} -Programs

Currently there are several well-known cryptosystems implemented to serve as examples for users when they are initially learning the tool. They also provide syntax examples for those wanting to add custom MOOs. For example, the ciphertext chaining cryptosystem is defined below:

Code

```
from symcollab.moe import MOO
@MOO.register('cipher_block_chaining')
def cipher_block_chaining(iteration, nonces, P, C):
    f = Function("f", 1)
    i = iteration - 1
    if i == 0:
        return f(xor(P[0], nonces[0]))
    return f(xor(P[i], C[i-1]))
```

Notice that this provides a relatively simple example of the type of recursive cryptosystems built over an xor-theory that are currently supported. Here the base ciphertext is defined as $f(P_0 \oplus nonces(0))$, where P_0 is the initial plaintext sent by the adversary, and $nonces[0]$ is a bound variable representing the initialization vector. Then the recursive case is $C_i = f(P_i \oplus C_{i-1})$. The underlying libraries have been constructed to allow the encoded version of the system definition to closely match the theoretical one.

Similarly, a user can create their own custom mode of operation by adding the recursive definition to the MOO library.

4.2 User defined schedule

In addition to the block-wise and message-wise schedules (as described in Section 3.2), the user can define their own schedules based on the iteration number. For example, this is a custom schedule that has the oracle only return ciphertexts on even iterations.

Code

```
from symcollab.moe import MOO_Schedule
@MOO_Schedule.register('even')
def even_schedule(iteration: int) -> bool:
    return iteration % 2 == 0
```

MOOs generated via Automatic Generation	
1	$C_0 = IV, C_i = f(f(P[i]) \oplus P[i-1])$
2	$C_0 = IV, C_i = f(f(P[i])) \oplus P[i-1] \oplus r$
3	$C_0 = IV, C_i = f(f(P[i]) \oplus C[i-1]) \oplus C[i-1]$
4	$C_0 = IV, C_i = f(f(P[i]) \oplus C[i-1]) \oplus f(f(P[i]) \oplus f(C[i-1]))$

Table 1: Examples of MOOs generated by the automatic MOO generator

4.3 Automatically Generated Singly Recursive MOO_{\oplus} -Definitions

A user can ask the library to generate a recursive definition of a modes of operation. Currently there is one method in the tool library to automatically generate MOOs. It works by recursively generating MOOs starting with the base components (IV, variables) and building singly recursive definitions using the xor and f function, and recursive references to prior ciphertexts. The current method has some limitations. For example only one nonce is used, the signature is limited to $\Sigma = \{\oplus/2, 0/0, f/1\}$, only single recursion is used, and the base case is fixed to the initialization vector. Thus, the current method won't generate all possible MOO_{\oplus} s. For example, a MOO that uses two nonces in its recursive definition won't be generated. We plan to expand this functionality in future versions of the tool allowing a user to automatically generate more classes of MOOs. Note, this doesn't limit the possible MOOs that a user can analyze by using the custom module. The user can also filter the recursive definitions by properties such as availability of the initialization vector, if it requires chaining, or if the number of calls to the encryption function f is less than a specified bound. A mode of operation has the chaining property if it incorporates a previous ciphertext into its recursive definition.

After creating the recursive MOO_{\oplus} definition, we can then pass it to the class CustomMOO. By default, this creates a MOO_{\oplus} program with the specified recursive definition and a new nonce for each base case.

4.4 Interactions with MOO_{\oplus} -Programs

Once a mode of operation and schedule have been defined, the tool can do several things with the definition. The first and simplest is to generate the terms corresponding to the symbolic representation of the ciphertexts. The user can also ask for the tool to evaluate the symbolic security of the MOO and/or the invertibility. We consider these options in the following sections. Before we move to security let's consider an example.

Examples The example MOOs in Table 1 showcases ones that were generated by the tool using the automatic generation feature. Note that these are just a few examples. In fact, one could allow the automatic generation to run as long as one wanted.

From these examples, the first two MOOs are not symbolically secure, they can be discovered and discarded by the method covered in the next section. The final MOO is symbolically secure, however it is still useless since it doesn't have the invertibility property! This can also be checked via the method detailed below. The third MOO, passes both the security and invertibility check and could be a candidate MOO for some secure application.

5 Checking Symbolic Security Properties

This part of the tool is based on the work developed in [12, 17]. Those papers define a method for checking symbolic security which in turn can be used to synthesize secure cryptographic modes of operation. See Section 3.2 for more background details. We give an overview of each of the components developed for checking symbolic security beginning with the MOO_{\oplus} -Programs.

5.1 Checking Symbolic Security

The tool can check for symbolic security in several ways. The first, and most exhaustive, is via the local unification approach. In this approach ciphertexts of the MOO_{\oplus} -program under consideration are generated and the appropriate local unification algorithm is used to see if any blocks sum to 0, see [16] for the full details of this approach.

The difficulty with this approach is that it can be time consuming in practice. However, a second approach has been developed in [12]. The approach doesn't require the generation of ciphertexts and works directly with the initial MOO_{\oplus} -program definition. This approach is not complete, but it works for many cases and has the advantage of being much more efficient. Therefore, we have implemented it as a first pass symbolic security check for the tool. If the first pass cannot decide symbolic security, then, the full security check requiring block generation will be used.

Examples Continuing With the MOOs from Table 1, let's consider just the first MOO. We can check for security using the MOO check method:

Code

```
moo_check(moo_name = 'table1.1', schedule_name = 'every',
          unif_algo = p_syntactic, length_bound = 10, knows_iv = True,
          invert_check=True)
```

The tool would return the first collision it finds, which violates the symbolic security property, for this example:

Output

```
Here is the problem:
f(xor(f(x1), IV)) = f(xor(f(x2), x2))
```

6 Invertibility and Recovering the Plaintext

A cryptographic algorithm is *invertible* if given a ciphertext and a decryption key, the original plaintext can be retrieved. This is not a given for any MOO_{\oplus} -program, even a secure one. Therefore, in the automatic generated setting we will need methods for checking if the invertibility property holds for any particular MOO_{\oplus} -program. Currently the tool is able to check invertibility for a large class of recursively defined MOOs. This class includes the well known MOOs, such as CBC, ECB, and CFB. More detailed information on theory and method for checking invertibility has been presented in [12].

The invertibility checker is built into the MOO security check functionality in the tool and can be requested simply by setting the “invert_check” flag (which is the last flag) in the *moo_check* function.

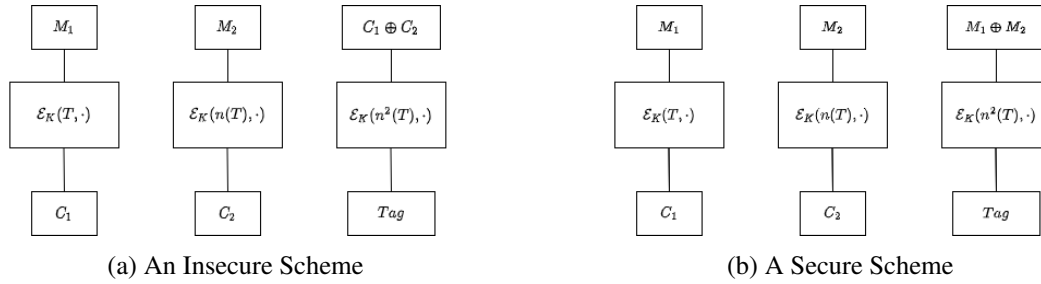


Figure 4: Two Authenticated Encryption Schemes

See Example 1.

Example 1

Code

```
from symcollab.moe.check import moo_check
from symcollab.Unification.p_unif import p_unif
result = moo_check('cipher_block_chaining', "every", p_unif, 2, True, True)
print(result.invert_result) # prints True
```

7 Authentication

An authenticated encryption scheme [6, 19] satisfies the authenticity property if an adversary cannot forge any new valid ciphertext message after observing any number of valid ciphertext messages. In [11], the authors proposed two algorithms for checking authenticity. The first algorithm works for a simplified case, where only messages of fixed length can be handled. The second algorithm works for the general case, where messages of arbitrary length can be handled.

We use M_1, M_2, \dots to denote plaintext blocks, and use C_1, C_2, \dots to denote ciphertext blocks. $\mathcal{E}_K(T, \cdot)$ denotes a tweakable block cipher [13], where K is some key and T is some tweak. The idea is that each key and tweak produce an independent pseudorandom permutation. $\mathcal{D}_K(T, \cdot)$ is the inverse of $\mathcal{E}_K(T, \cdot)$. $n(T)$ produces another tweak, given a tweak T . We use $n^k(T)$ as a shorthand for applying n to T for k times. The idea is that the same key can be used for multiple blocks, as long as the tweaks are different for each different block. In order to achieve authenticity, a tag is attached to each ciphertext message. Each scheme is associated with a *verification condition*, which refers to the ciphertext blocks and the tag. A ciphertext message is *valid*, if the verification condition holds for that ciphertext message.

Figure 4 shows two authenticated encryption schemes, both of which handle messages of two blocks. The verification condition of the scheme in Figure 4a is $\mathcal{E}_K(n^2(T), C_1 \oplus C_2) = \text{Tag}$. The authenticity property is violated. The reason is that if (C_1, C_2, Tag) is a valid ciphertext message, $(C_1 \oplus C_2, 0, \text{Tag})$ is also a valid ciphertext message. The verification condition of the scheme in Figure 4b is $\mathcal{E}_K(n^2(T), \mathcal{D}_K(T, C_1) \oplus \mathcal{D}_K(n(T), C_2)) = \text{Tag}$, the authenticity property is satisfied. The intuition is that the adversary does not know any way of modifying C_1 and C_2 in such a way that $M_1 \oplus M_2$ remains the same.

Here are some other possible verification conditions for authenticated encryption schemes, which handle two message blocks. The first three verification conditions satisfy the authenticity property, and the last two do not.

Secure MOOs Found via Automatic Generation and Testing	
1	$C_0 = IV, C_i = f(f(f(P[i-1]) \oplus r) \oplus C[i-1])$
2	$C_0 = IV, C_i = f(f(f(P[i]) \oplus C[i-1]) \oplus r)$
3	$C_0 = IV, C_i = f(f(P[i]) \oplus C[i-1]) \oplus C[i-1]$
4	$C_0 = IV, C_i = f(f(f(P[i]) \oplus r \oplus C[i-1]))$
5	$C_0 = IV, C_i = f(f(P[i]) \oplus C[i-1]) \oplus f(C[i-1])$

Table 2: Examples of secure MOOs found using the MOO generator

- $\mathcal{E}_K(n^2(T), \mathcal{D}_K(n(T), \mathcal{D}_K(T, C_1) \oplus C_2)) = Tag$
- $\mathcal{E}_K(n^2(T), \mathcal{D}_K(T, \mathcal{D}_K(n(T), C_2) \oplus C_1)) = Tag$
- $\mathcal{E}_K(n^2(T), \mathcal{D}_K(T, C_1) \oplus \mathcal{D}_K(n(T), C_2)) = Tag$
- $\mathcal{E}_K(n^2(T), \mathcal{D}_K(T, C_1)) = Tag$
- $\mathcal{E}_K(n^2(T), \mathcal{D}_K(n(T), C_2)) = Tag$

Given a verification condition of some authenticated encryption scheme, our tool can automatically check if the authenticity property is satisfied.

Code	Output
<pre>from symcollab.Unification.constrained.authenticity import * t = e(n(n(T)), xor(C1, d(n(T), C2))) check_security(t)</pre>	<pre>The authenticity property is satisfied.</pre>

8 Experiments

A benefit of the tool design is that it is easy to integrate the above described functions into a script which can then be used to run experiments. For example, we have included a script, located in the experiments directory of the tool, that allows the user to run longer experiments and can handle restarts. In this script, we generate new candidate MOOs one at a time and test them for security. The output of `moo_check` is the data structure called `MOOCheckResult`. This has the following fields: `collisions` (set of computable substitutions that cause a collision to occur), `invert_result` (whether or not the MOO is invertible), `iterations_needed` (number of iterations before a collision was found), and whether or not the MOO satisfies symbolic security up to the bound checked.

Initial Experimental Results A sample of some of the secure MOOs found during early experiments are listed in Table 2. All of these MOOs were created automatically by the currently implemented recursive `MOOGenerator`. As a future work, we plan to create additional generators that the user can select and allow for user defined generators.

Experiments can also be done without the `MOOGenerator`, where MOOs are generated via hand or a custom script and then checked for security. This is an attractive option because it allows the user to easily customize the type of MOOs they are considering. Table 3 includes some example secure MOOs that were created by hand and then tested for security using the tool. Note, that although all three MOOs

Secure MOOs Found via Custom Generation and Testing	
1	$C_0 = IV, C_i = f(P[i] \oplus f(C[i-1])) \oplus f(C[i-1])$
2	$C_0 = IV, C_i = f(P[i] \oplus f(C[i-1]) \oplus f(P[i]))$
3	$C_0 = IV, C_i = f(f(P[i]) \oplus C[i-1]) \oplus f(f(P[i]) \oplus f(C[i-1]))$

Table 3: Examples of secure MOOs found using a custom generator

are secure only the first MOO can be shown by the tool to be invertible (via the method developed in [12])! Thus, secure but useless MOOs can also be discarded.

Based on the initial experimentation with the tool there are some interesting early questions: Can the set of secure MOOs be closed under some operation such as applying the encryption symbol f on top? Are there cases where we can place a bound on the number of iterations to check security? We're particularly motivated by the second question, due to the complexity of our saturation based decision procedures. For some of the MOOs we tested, it took in the order of days in order for the algorithm to find a collision.

9 Conclusion and Future Work

In this paper we presented a new tool for the symbolic analysis of cryptosystems with the ultimate goal of providing support for multiple types of algorithms and representations. Although at present it only supports modes of operation, the tool provides a widely applicable symbolic foundation based on that of Baudet et al. [4]. Not only can this symbolic foundation represent multiple types of cryptosystems, the symbolic foundation is also amenable to proofs of computational soundness and completeness. The tool also includes libraries that support useful algorithms for checking symbolic security, including unification and variant generation. There are limitations to the currently supported modes of operation. Currently, only modes which can be modeled using the above Xor theory are supported with the needed specialized unification algorithms. For example, modes requiring primitives such as hash functions, successor, or full abelian groups are currently not supported. However, we hope to add, if possible, support for as many of these structures as possible in future versions of the tool.

One avenue of interest to us is to investigate other work on symbolic cryptography to determine whether it can be fit into our framework and its algorithms implemented in our tool. We expect previous work on modes [7, 14] to fit in well, since, although their models are not expressed as symbolic algebras, they are still compatible with the algebra used in CryptoSolve. Other work, such as Linicrypt [5, 15] and Zoocrypt [3] also follow the paradigm of representing cryptographic primitives as function symbols obeying equational theories. In the cases in which soundness and completeness results are provided, we expect them to carry over into the symbolic model. When computational soundness of a symbolic language is not known (e.g. Zoocrypt), it may be possible to ensure it by limiting its expressiveness.

More generally, our tool is intended to be extensible to cryptosystems that may not yet have been studied from the symbolic point of view. Although only a few function symbols have been implemented in CryptoSolve as of now, it is designed to be extensible. The best choice, for this seems to be cryptosystems that can be expressed in terms of combinations of primitives, including randomly chosen bitstrings, each of which has a clearly defined security property. The combinators should be operations that can be characterized in a symbolic way. These include not only finite field and group operations, which are commonly used in cryptography, and can be found in all the work cited in this paper, but concatenation, which is used in [3, 8, 15]. Many cryptosystems are defined using these techniques of building complex

systems from basic components, so we expect the are of application to be wide.

References

- [1] Martín Abadi & Cédric Fournet (2001): *Mobile Values, New Names, and Secure Communication*. In: *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'01*, ACM, New York, NY, USA, pp. 104–115. Available at <http://doi.acm.org/10.1145/360204.360213>.
- [2] Franz Baader & Tobias Nipkow (1998): *Term rewriting and all that*. Cambridge University Press, New York, NY, USA. Available at <https://doi.org/10.1017/CB09781139172752>.
- [3] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt & Santiago Zanella Béguelin (2013): *Fully automated analysis of padding-based encryption in the computational model*. In Ahmad-Reza Sadeghi, Virgil D. Gligor & Moti Yung, editors: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, ACM, pp. 1247–1260. Available at <https://doi.org/10.1145/2508859.2516663>.
- [4] Mathieu Baudet, Véronique Cortier & Steve Kremer (2005): *Computationally sound implementations of equational theories against passive adversaries*. In: *Automata, Languages and Programming*, Springer, pp. 652–663. Available at https://doi.org/10.1007/11523468_53.
- [5] Brent Carmer & Mike Rosulek (2016): *Linicrypt: A Model for Practical Cryptography*. In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, pp. 416–445. Available at http://dx.doi.org/10.1007/978-3-662-53015-3_15.
- [6] Virgil D. Gligor & Pompiliu Donescu (2002): *Fast encryption and authentication: XCBC encryption and XECB authentication modes*. In: *Fast Software Encryption (FSE) 2001*, pp. 92–108, doi:10.1007/3-540-45473-X_8.
- [7] Viet Tung Hoang, Jonathan Katz & Alex J. Malozemoff (2015): *Automated Analysis and Synthesis of Authenticated Encryption Schemes*. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Association for Computing Machinery, New York, NY, USA, pp. 84–95. Available at <https://doi.org/10.1145/2810103.2813636>.
- [8] Tommy Hollenberg, Mike Rosulek & Lawrence Roy (2022): *A Complete Characterization of Security for Linicrypt Block Cipher Modes*. In: *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*, pp. 423–438, doi:10.1109/CSF54842.2022.00028.
- [9] Ralf Küsters & Tomasz Truderung (2007): *On the Automatic Analysis of Recursive Security Protocols with XOR*. In: *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, pp. 646–657. Available at https://doi.org/10.1007/978-3-540-70918-3_55.
- [10] Hai Lin & Christopher Lynch (2020): *Local XOR Unification: Definitions, Algorithms and Application to Cryptography*. *IACR Cryptol. ePrint Arch.* 2020, p. 929. Available at <https://eprint.iacr.org/2020/929>.
- [11] Hai Lin & Christopher Lynch (2021): *Formal Analysis of Symbolic Authenticity*. In Boris Konev & Giles Reger, editors: *Frontiers of Combining Systems - 13th International Symposium, FroCoS 2021, Birmingham, UK, September 8-10, 2021, Proceedings, Lecture Notes in Computer Science 12941*, Springer, pp. 271–286. Available at https://doi.org/10.1007/978-3-030-86205-3_15.
- [12] Hai Lin, Christopher Lynch, Andrew M. Marshall, Catherine A. Meadows, Paliath Narendran, Veena Ravishankar & Brandon Rozek (2021): *Algorithmic Problems in the Symbolic Approach to the Verification of Automatically Synthesized Cryptosystems*. In Boris Konev & Giles Reger, editors: *Frontiers*

- of Combining Systems - 13th International Symposium, FroCoS 2021, Birmingham, UK, September 8-10, 2021, Proceedings, Lecture Notes in Computer Science 12941, Springer, pp. 253–270. Available at https://doi.org/10.1007/978-3-030-86205-3_14.
- [13] Moses Liskov, Ronald L. Rivest & David Wagner (2002): *Tweakable block ciphers*. In: *Advances in Cryptology-Crypto 2002*, pp. 31–46, doi:10.1007/3-540-45708-9_3.
- [14] Alex J. Malozemoff, Jonathan Katz & Matthew D. Green (2014): *Automated Analysis and Synthesis of Block-Cipher Modes of Operation*. In: *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, IEEE Computer Society, pp. 140–152. Available at <https://doi.org/10.1109/CSF.2014.18>.
- [15] Ian McQuoid, Trevor Swope & Mike Rosulek (2019): *Characterizing Collision and Second-Preimage Resistance in LiniCrypt*. In Dennis Hofheinz & Alon Rosen, editors: *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I, Lecture Notes in Computer Science 11891*, Springer, pp. 451–470. Available at https://doi.org/10.1007/978-3-030-36030-6_18.
- [16] Catherine Meadows (2021): *Moving the Bar on Computationally Sound Exclusive-Or*. In Elisa Bertino, Haya Shulman & Michael Waidner, editors: *Computer Security – ESORICS 2021*, Springer International Publishing, Cham, pp. 275–295. Available at https://doi.org/10.1007/978-3-030-88428-4_14.
- [17] Catherine A. Meadows (2020): *Symbolic and Computational Reasoning About Cryptographic Modes of Operation*. *IACR Cryptol. ePrint Arch.* 2020, p. 794. Available at <https://eprint.iacr.org/2020/794>.
- [18] Phillip Rogaway (2004): *Nonce-Based Symmetric Encryption*. In: *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, pp. 348–359. Available at https://doi.org/10.1007/978-3-540-25937-4_22.
- [19] Phillip Rogaway, Mihir Bellare, John Black & Ted Krovetz (2001): *OCB: A block-cipher mode of operation for efficient authenticated encryption*. In: *8th ACM Conference on Computer and Communications Security (CCS)*, pp. 196–205, doi:10.1145/937527.937529.
- [20] Dominique Unruh (2010): *The impossibility of computationally sound XOR*. *IACR Cryptology ePrint Archive* 2010, p. 389. Available at <http://eprint.iacr.org/2010/389>.
- [21] Marloes Venema & Greg Alpar (2021): *A Bunch of Broken Schemes: A Simple yet Powerful Linear Approach to Analyzing Security of Attribute-Based Encryption*. In Kenneth G. Paterson, editor: *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings, Lecture Notes in Computer Science 12704*, Springer, pp. 100–125. Available at https://doi.org/10.1007/978-3-030-75539-3_5.

Adversarial Formal Semantics of Attack Trees and Related Problems

Thomas Brihaye

University of Mons
Mons, Belgium

thomas.brihaye@umons.ac.be

Sophie Pinchinat

Université de Rennes, IRISA
Rennes, France

sophie.pinchinat@irisa.fr

Alexandre Terefenko

Université de Rennes, IRISA
Rennes, France

University of Mons
Mons, Belgium

alexandre.terefenko@irisa.fr

Security is a subject of increasing attention in our actual society in order to protect critical resources from information disclosure, theft or damage. The informal model of attack trees introduced by Schneier, and widespread in the industry, is advocated in the 2008 NATO report to govern the evaluation of the threat in risk analysis. Attack-defense trees have since been the subject of many theoretical works addressing different formal approaches.

In 2017, M. Audinot et al. introduced a path semantics over a transition system for attack trees. Inspired by the latter, we propose a two-player interpretation of the attack-tree formalism. To do so, we replace transition systems by concurrent game arenas and our associated semantics consist of strategies. We then show that the emptiness problem, known to be NP-complete for the path semantics, is now PSPACE-complete. Additionally, we show that the membership problem is CONP-complete for our two-player interpretation while it collapses to P in the path semantics.

1 Introduction

Security is a subject of increasing attention in our actual society in order to protect critical resources from information disclosure, theft or damage. The informal model of attack trees was first introduced by Schneier [15] to schematically model possible threats one could execute against an information system. Attack trees have then been widespread in the industry and are advocated in the 2008 NATO report to govern the evaluation of the threat in risk analysis. The attack tree model is also a subject of increasing attention in the community of formal methods with a lot of different formal approaches [10, 8, 7, 6, 5, 12, 1] (see the survey [16]).

The first formal model of attack trees introduced in [15] aimed at describing a possible attack over a system by refining the main attack goal into sub-goals using either an operator *OR* or an operator *AND* to coordinate those refinements. The analysis conducted over those trees is "static" in the sense that the attacked system does not evolve during the attack. As such, there is no concept of a goal happening before or after another. In [6], the authors introduce a first formal semantics that can be qualified as "dynamic" by allowing a new operator, operator *SAND* (for sequential *AND*), to specify that sub-goals must be attained in a given order. Considering that the *SAND* operator is now commonly accepted, the authors of [1] propose a path semantics for attack trees over a transition system.

In this paper, our goal is to present a new semantics for attack trees in order to be able to model more realistic scenarios: we want our attacker to be able to adapt her actions according to the behavior of the environment – typically, a defender who tries to protect the system. This setting naturally yields a two-player semantics. Our approach is inspired by [1] where we generalize the path semantics to a game-theoretic framework, yielding a strategy semantics for attack trees, without changing their syntax.

While the path semantics from [1] is compositional in a natural way, it turns out that the strategy semantics does not have such a nice property. Indeed, although composition of strategies is possible (see for example [11]), there is no immediate solution to compose two strategies in order to get a strategy that achieves the disjunction of what the formers achieve. This situation makes it difficult to design a compositional strategy semantics for attack trees. We therefore develop a non-trivial strategy semantics that is not compositionally obtained *per se*, but that makes use of the former compositional path semantics.

To our knowledge, our proposal is the first game-theoretic semantics for attack trees, and should not be mixed-up with the multi-player setting induced by the so-called classic model of *attack-defense trees* in the literature (see [9]): attack-defense trees are attack trees equipped with a new operator to express that some defender could use a countermeasure to prevent attacker from achieving her goal. Although well-understood in a "static" framework, we are not aware of any formal semantics of attack-defense trees for the "dynamic" one, i.e., over transition systems. This missing piece of work makes it difficult to conduct a comparison with our contribution, but, from the fact that the defender in our setting is fully formalized, as an opponent in the game arena, while the defender in attack-defense trees takes the form of an abstract entity, it seems that those two formalisms are not expressing the same kind of problems.

Our contribution in this paper is twofold.

We first develop a clean mathematical setting to obtain a formal strategy semantics for attack trees. For pedagogical reasons, we choose to consider a simplified version of the attack trees of [1] where atomic goals (at the leaves of the trees) are reachability goals with no preconditions. However, at the price of tedious definitions, the strategy semantics we propose can be adapted to atomic goals with preconditions. Regarding the design of this semantics, we heavily rely on the older one of [1] based on paths, and we justify our approach by providing evidence that a compositional strategy semantics is hopeless.

Second, we exploit the attack tree semantics to address and study the complexity of two decision problems: the *non-emptiness problem* and the *membership problem*. The former consists in determining if the semantics of an input tree is non-empty, while the latter consists in determining if an input element belongs to the semantics of an input tree. Our results are summarized in Table 1, where we distinguish between the path semantics and the strategy semantics.

Table 1: Complexity results

	Paths semantics	Strategy semantics
Non-Emptiness Problem	NP-complete	PSPACE-complete
Membership Problem	P	CONP-complete

Importantly, both decision problems have a practical counterpart. The non-emptiness of the path semantics of a tree reflects a situation where there exists a favorable scenario for attacker to perform her attack, while the non-emptiness of the strategy semantics reflects an intrinsic vulnerability of an information system. Regarding the membership problem for the path semantics, we are interested in knowing whether a log file of some information system execution makes evidence of an attack, while for the strategy semantics, we wonder if an attack policy is successful.

The paper is organized as follows. We start in Section 2 with an introductory example explaining informally the difference between path semantics and strategy semantics. After some background work in Section 3, we introduce in Section 4 the formal model of attack trees and define the path semantics inspired by [1] as well as our new strategy semantics. We then study the complexity of the Non-Emptiness and the Membership problems in Section 5.

2 Introductory example

Consider a thief (the attacker) who wants to steal some document inside a safe of a building without being seen. The building is composed of two rooms. The first room has two entrance doors (called door 1 and door 2) from the street. There is a guard keeping the entrance doors, but he can only control the bypassing of one of the two entrance doors at a time. The first room has also another door that leads to the second room. This door is locked but the key to unlock it is in the first room. There is also a camera in the first room monitoring the door to the second room. The second room contains a safe with the document that the thief wants to steal. Therefore, in order for the thief to attain his goal, he needs to enter the first room (by either door that is not currently controlled by the guard), then deactivate the camera and collect the key (in whichever order he wants) and finally unlock and go through the door leading to the second room. The thief can be seen by the guard if they appear to be in front of the same door or by the camera if activated when he is in front of the door to room 2. Figure 1a gives a picture of the situation where the thief is still outside of the building and the guard controls the second door.

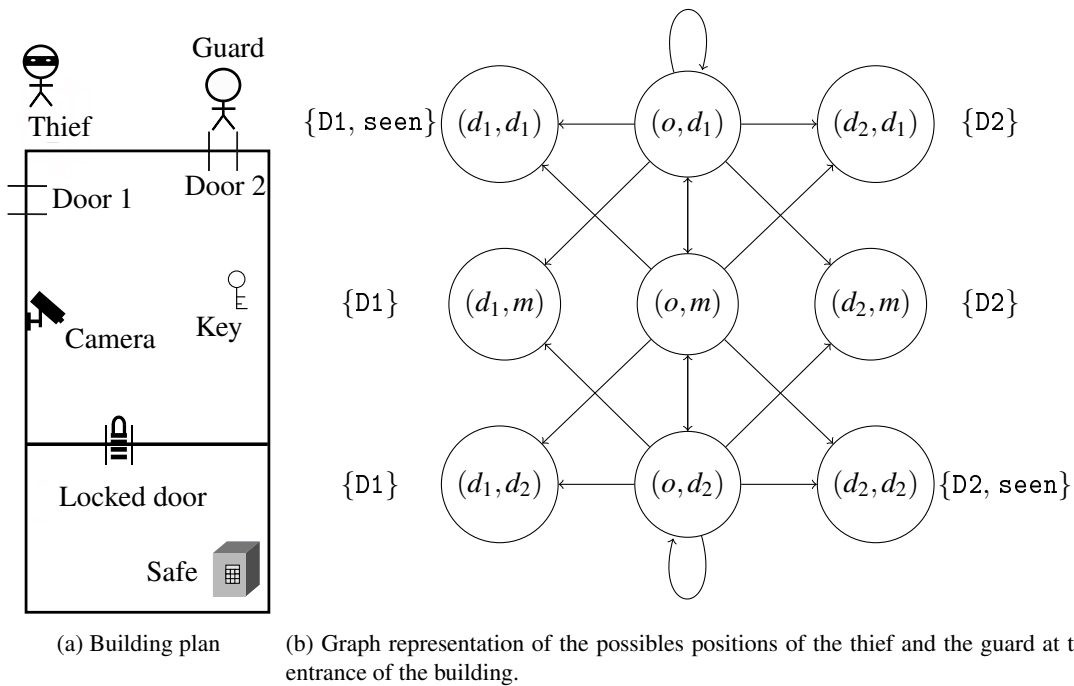


Figure 1: Introductory example building

In security, it is common to use an attack tree to model the goal of the attacker. An attack tree is a tree where each node describes a goal and the children of a node describe a refinement into sub-goals of the parent goal. To model those refinements, it is common to use three operators:

- *OR* operator means that at least one sub-goal needs to be achieved to have the goal accomplished,
- *SAND* operator, read "sequential and", means that the sub-goals need to be achieved in the left-to-right order to have the goal accomplished,
- *AND* operator means that the sub-goals need to be achieved (in whatever order) to have the goal accomplished.

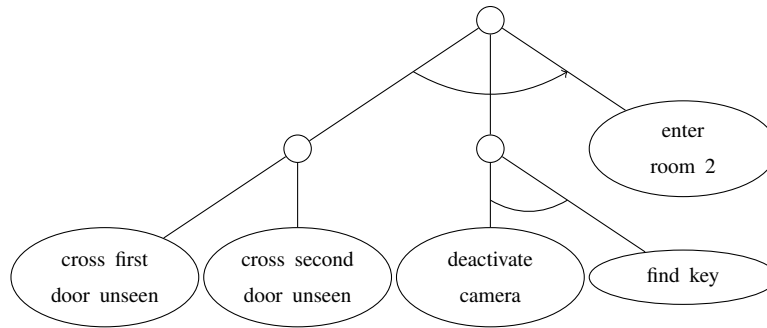


Figure 2: An attack tree to model the goal of our thief.

In Figure 2, we describe the goal of the thief by means of an attack tree. To distinguish the different types of nodes, we draw a curved line below an *AND* operator, a curved arrow below a *SAND* operator and nothing below an *OR* operator.

We focus our attention on the very first part of our problem, that is, the sub-goal of the thief to enter the building. We start by fixing a set of proposition $Prop = \{D1, D2, seen\}$ where $D1$ holds when the thief is in front of the first door, $D2$ holds when the thief is in front of the second door and $seen$ holds when the guard sees the thief because they meet in front of the same door. The goal "crossing the first door unseen" of the thief can be modelled by the formula $D1 \wedge \neg seen$ and the goal "crossing the second door unseen" is modelled by the formula $D2 \wedge \neg seen$. If we assume that the guard can switch door whenever he wants but leaves the two doors unguarded for a brief amount of time during his motion, we can model the situation using the graph of Figure 1b where each state consists of a pair in the set $\{o, d_1, d_2\} \times \{m, d_1, d_2\}$. For a pair (a, b) , element a determines the position of the thief (o means that he is still outside the building, while d_i indicates he is at door i) and element b determines the position of the guard (m means that he is currently in motion between the two doors, leaving them both unguarded). We write next to each state which propositions hold in it.

In a path semantics, a successful attack for goal $D1 \wedge \neg seen$ consists of a sequence of states (i.e., a word) such that the valuation of the last state of this sequence satisfies formula $D1 \wedge \neg seen$. In particular, the sequence $(o, m), (d_1, d_2)$ is a successful attack. Now, if we want to consider a successful attack for the attack tree consisting of the "OR" operator of objective $D1 \wedge \neg seen$ and $D2 \wedge \neg seen$, it is enough to consider the union of the set of all attacks for objective $D1 \wedge \neg seen$ with the set of all attacks for objective $D2 \wedge \neg seen$.

However, in the strategy semantics we introduce in this paper, we consider that an attack is successful if the attacker has a strategy that grants him to reach the states he needs to attain, independently of the environment. In our example, we can see that the thief has no strategy starting at position (o, m) to achieve goal $D1 \wedge \neg seen$. Indeed, if the first move of the guard consists on going to door 1 and he then does not move any more, there is no way for the thief to cross the first door while unseen. Similarly, there is no strategy starting at position (o, m) to achieve goal $D2 \wedge \neg seen$. However, if we consider the "OR" operator of the two goals $D1 \wedge \neg seen$ and $D2 \wedge \neg seen$, the strategy of the thief consisting in waiting for the guard to go to one of the two doors and then going to door 1 if the guard is at door 2 and vice versa is a successful strategy. Later, we will use similar a similar example to show that a compositional definition for a strategy semantics cannot be achieved.

3 Preliminary notions

Formal languages Given an alphabet (i.e., a finite set of symbols) Σ , notation Σ^* represents the set of finite words (i.e., sequences of symbols) over the alphabet Σ , with typical element $w = \ell_1 \dots \ell_n \in \Sigma^*$; the empty word is written ε . On the other hand, the set of infinite words is denoted by Σ^ω . A subset $L \subseteq \Sigma^*$ is a language. Given a word $w = \ell_1 \dots \ell_n \in \Sigma^*$, its set of prefixes is the language $Prefixes(w) = \{\ell_1 \dots \ell_i \mid i \leq n\} \cup \{\varepsilon\}$, and we write $w' \triangleleft w$ whenever $w' \in Prefixes(w)$. A language L is prefix-closed if $w \in L$ implies $w' \in L$ for every $w' \triangleleft w$. The concatenation of a word $w = \ell_1 \dots \ell_n$ with another word $w' = \ell'_1 \dots \ell'_m$ is the word $ww' = \ell_1 \dots \ell_n \ell'_1 \dots \ell'_m$. The concatenation of a word with a language is defined in the usual way: for $w \in \Sigma^*$ and $L \subseteq \Sigma^*$, we let $wL := \{ww' \mid w' \in L\}$.

Game theory and game arena To formalize a strategy semantics for attack trees, we need standard two-player, zero-sum, perfect information games that we recall here.

A *game arena* is a finite graph on which two players play a game of unbounded duration. We choose to consider concurrent games, meaning that, at each round, each player makes an action. To define it properly, we consider two players called Player 1 and Player 2 and a finite set of propositions *Prop*.

Definition 3.1. A *two-player game arena* is a tuple $\mathcal{G} = (Pos, Act, \delta, val)$ where:

- $Pos = \{v, \dots\}$ is a finite set of positions.
- $Act = Act_1 \times Act_2$ is a finite set of actions, as a product of the sets of actions of each player.
- $\delta : Pos \times Act \rightarrow Pos$ is a transition function,
- $val : Pos \rightarrow \mathcal{P}(Prop)$ is a valuation function.

In our introductory example of Section 2, if we consider the set of actions for the thief: {wait, go-to-Door-1, go-to-Door-2} and the following set of actions for the defender: {stay-at-current-door, leave-current-door, go-to-Door-1, go-to-Door-2}, then it is easy to see that the graph drawn in Figure 1b forms a game arena.

For the rest of this paper, we fix a game arena $\mathcal{G} = (Pos, Act, \delta, val)$.

For a game position $v \in Pos$, we define $Post(v) = \{v' \in Pos \mid \delta(v, a) = v' \text{ for some } a \in Act\}$ the set of all positions reachable from v in one step. For convenience, we assume that each player j can play every action a in Act_j at each position of the game arena, so that for each position $v \in Pos$, we have $Post(v) \neq \emptyset$. A play ρ is an infinite sequence of positions of the form $v_0 v_1 v_2 \dots \in Pos^\omega$ such that for each $i \in \mathbb{N}$, there is $a \in Act$ such that $\delta(v_i, a) = v_{i+1}$. For $i \in \mathbb{N}$, we let $\rho_i = v_i$ be the i^{th} position of play ρ . The set of all plays is denoted by $Plays(\mathcal{G})$. Each non-empty prefix h of a play is called a *history* and the set of all histories is denoted by $Hist(\mathcal{G})$. For a history $h \in Hist(\mathcal{G})$, we define $last(h)$ as the last position of h . For $v \in Pos$, we also use the notations $Plays(\mathcal{G}, v)$ and $Hist(\mathcal{G}, v)$ to denote the set of all plays starting from v (i.e., $\rho_0 = v$) and the set of all histories starting from v , respectively.

Winning plays for Player 1 are obtained from a distinguished subset $\Gamma_1 \subseteq Plays(\mathcal{G})$. As we consider zero-sum games, all plays in $\Gamma_1 \setminus Plays(\mathcal{G})$ are winning for Player 2.

Classically, we introduce the notion of *strategy*, as a map prescribing how a player plays depending on the current history: a *strategy* μ^j for the Player j is a map $\mu^j : Hist(\mathcal{G}) \rightarrow Act_j$. The set of all strategies for the Player j is denoted as $Strat^j$.

A history $h = v_0 v_1 \dots v_m$ is *consistent* with a strategy μ^j if for each $1 \leq i \leq m$, there exists $a = (a_1, a_2) \in Act$ such that we have $\delta(v_i, a) = v_{i+1}$ and $\mu^j(v_0 v_1 \dots v_i) = a_j$. We say that a play ρ is consistent with a strategy μ^j if all prefixes of ρ are histories consistent with μ^j . The set of all plays consistent with μ^j is denoted by $Outcomes(\mu^j)$. From a game position v we say that a strategy is *winning* if all

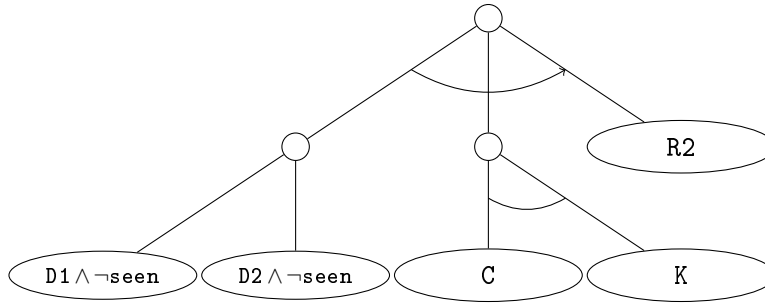


Figure 3: Formal version of the attack tree in Figure 2.

outcomes starting at v are winning. In other words, for a strategy of say Player 1, a strategy μ is winning if $Outcomes(\mu) \cap Plays(\mathcal{G}, v) \subseteq \Gamma_1$.

A classic kind of concurrent games are the *reachability games*. In such games, a player wants to reach some positions while the other player tries to prevent it from happening. These games are clearly zero-sum. More formally, we say that a game is a reachability game for Player 1 if there exists $W_1 \subseteq Pos$ such that $\Gamma_1 = \{\rho \in Plays(\mathcal{G}) \mid \rho_i \in W_1 \text{ for some } i \in \mathbb{N}\}$.

In a game arena, we say that a position v satisfies the formula ϕ if its valuation $val(v)$ satisfies the formula in classic propositional logic, denoted by $v \models \phi$. Note that a Boolean formula ϕ over $Prop$ describes the reachability game (\mathcal{G}, W_1) where $W_1 = \{v \in Pos \mid v \models \phi\}$.

4 Attack trees and their semantics

In this section, we start with the formal definition of attack tree used in this paper. Then we develop two semantics for attack trees: the path semantics and the strategy semantics.

4.1 Syntax of attack trees

To formalize attack trees, we start by fixing a set of propositions $Prop$.

Definition 4.1. An *attack tree* τ over $Prop$ is:

- either a leaf composed of a unique Boolean formula ϕ over $Prop$,
- or an expression $OP(\tau_1, \dots, \tau_n)$ where OP ranges over OR , AND and $SAND$ and τ_1, \dots, τ_n are attack trees.

We define the size of an attack tree τ , noted $|\tau|$, by the number of its nodes.

Example 4.2. We will formalise the example introduced in Section 2. To represent the situation, we use the following set of propositions: $Prop = \{D1, D2, \text{seen}, C, K, R2\}$. We have that $D1$ holds when the thief has crossed the first door, $D2$ holds when the thief has crossed the second door, seen holds when the guard sees the thief, C holds when the camera is on, K holds when the thief has the key and finally $R2$ holds when the thief is in the second room.

We can now propose a formal definition for the attack tree in Figure 2: $\tau = SAND(OR(D1 \wedge \neg \text{seen}, D2 \wedge \neg \text{seen}), AND(C, K), R2)$ to model the objective of the attacker. The graph representation of τ is given by Figure 3.

Let us notice that the attack trees used in [1] are in fact slightly different: the leaves of the attack trees of that paper are of the form $\langle \phi_1, \phi_2 \rangle$ with ϕ_1 and ϕ_2 two Boolean formulas. Formula ϕ_1 describes a precondition for the objective to begin with and formula ϕ_2 describes the postcondition for the objective to be granted. However, in this paper, we never consider preconditions. So a leaf ϕ of our attack trees can be seen as a leaf $\langle true, \phi \rangle$ of attack trees introduced in [1]. Although the semantics defined in this paper could also be defined for attack trees with preconditions, not considering them makes the setting more pedagogical.

The first semantics for attack trees we introduce is a path semantics inspired by [1]. Informally, in the path semantics, we consider all sequences of events that lead to a successful attack. The idea is to determine which scenarios are favourable for the attacker. One could also say that an attack can occur if the attacker is lucky.

The second semantics is our main contribution, and is named the *strategy semantics* for attack trees. In this approach, the attacker should not rely on an opportunity offered by the environment but should be able to find the right sequence of actions whatever the environment does. Otherwise said, an attack is not a favourable scenario anymore but a winning strategy for attacker in some two-player game arena.

4.2 Path semantics for attack trees

To give a path semantics over our trees, we first need to fix a transition system to model which actions/sequences of actions can be executed by the attacker in the system. A transition system is composed of a finite set of states together with a transition relation between pairs of states. We decided not to label every transition with an action as we only consider perfect information here. We also provide a valuation function to our transition system, informing which propositions of *Prop* holds in a state of the transition system.

Definition 4.3. A transition system over *Prop* is a triplet $\mathcal{S} = (S, \delta, val)$ where:

- S is a finite set of states,
- $\delta \subseteq S \times S$ is a relation of transitions,
- $val : S \rightarrow \mathcal{P}(Prop)$ is a valuation function.

The size of \mathcal{S} , noted $|\mathcal{S}|$, is defined by its number of states.

We can see from Definition 4.3 that a transition system is a notion close to a game arena. Indeed, it is easy to associate a transition system with a game arena $\mathcal{G} = (Pos, Act, \delta, val)$, by merging the two players into a single one in the following way: $\mathcal{S}_{\mathcal{G}} = (Pos, \{(v, v') \in Pos \times Pos \mid \text{there exists } a \in Act \text{ such that } \delta(v, a) = v'\}, val)$. Later, we denote $\mathcal{S}_{\mathcal{G}}$ by \mathcal{G} when it is clear from the context.

For the rest of this section, we fix a transition system $\mathcal{S} = (S, \delta, val)$ over *Prop*. A path in a transition system is a finite non-empty sequence of states $\pi = s_0 s_1 \dots s_n$ such that, for each $0 \leq i < n$, $(s_i, s_{i+1}) \in \delta$. The size of a path is its number of states. We denote the set of all paths in \mathcal{S} by $\Pi_{\mathcal{S}}$.

In order to define the path semantics we need to introduce operators over paths.

Definition 4.4. Let $\pi = s_0 s_1 \dots s_n$ and $\pi' = s'_0 s'_1 \dots s'_m$ be two paths in \mathcal{S} with $n, m \geq 0$. The *synchronised concatenation* of π_1 and π_2 is defined only if $s_n = s'_0$ and is given by $\pi \cdot \pi' = s_0 s_1 \dots s_n s'_1 \dots s'_m$.

We lift this operations to sets of paths the following way: if Π_1 and Π_2 are two sets of paths, then $\Pi_1 \cdot \Pi_2 = \{\pi_1 \cdot \pi_2 \mid \pi_1 \in \Pi_1 \text{ and } \pi_2 \in \Pi_2\}$.

The authors of [1] introduce the operator of *parallel composition of paths*. However, our definition of attack trees grants us the possibility to use a simpler operator.

Definition 4.5. Let Π_1, Π_2 be two sets of paths of \mathcal{S} . The *merge* of Π_1 and Π_2 is the set of paths $\Pi_1 \triangle \Pi_2 = \{\pi_1 \in \Pi_1 \mid \text{there exists } \pi_2 \in \Pi_2 \text{ such that } \pi_2 \triangleleft \pi_1\} \cup \{\pi_2 \in \Pi_2 \mid \text{there exists } \pi_1 \in \Pi_1 \text{ such that } \pi_1 \triangleleft \pi_2\}$

Unlike the parallel composition of [1], thanks to the transitivity of the prefix relation, the merge operator is associative.

We can now define our path semantics.

Definition 4.6. Let τ be a attack tree over *Prop*. The path semantics of τ over \mathcal{S} is the set of paths $Paths_{\mathcal{S}}(\tau)$ inductively defined as follow:

- $Paths_{\mathcal{S}}(\phi) = \{s_0 s_1 \dots s_n \in \Pi_{\mathcal{S}} \mid s_n \models \phi\}$
- $Paths_{\mathcal{S}}(OR(\tau_1, \dots, \tau_n)) = Paths_{\mathcal{S}}(\tau_1) \cup \dots \cup Paths_{\mathcal{S}}(\tau_n)$
- $Paths_{\mathcal{S}}(SAND(\tau_1, \dots, \tau_n)) = Paths_{\mathcal{S}}(\tau_1) \cdot \dots \cdot Paths_{\mathcal{S}}(\tau_n)$
- $Paths_{\mathcal{S}}(AND(\tau_1, \dots, \tau_n)) = Paths_{\mathcal{S}}(\tau_1) \triangle \dots \triangle Paths_{\mathcal{S}}(\tau_n)$

It is easy to verify that the semantics of Definition 4.6 is equivalent to the one introduced in [1] if we restrict to the attack trees whose leaves are of the form $\langle true, \phi \rangle$.

Remark that, in our framework, for ϕ_1 and ϕ_2 two formulas over *Prop*, the interpretation of $SAND(\phi_1, \phi_2)$ is that ϕ_1 must hold at some point and ϕ_2 must hold at some point afterwards. This requirement does not prevent ϕ_2 from holding before ϕ_1 .

We also want to point out that our semantics consider that the simultaneity of objectives is always successful: for ϕ_1 and ϕ_2 two formulas over *Prop*, if $\phi_1, \phi_2 \in val(s)$, then $s \in Paths_{\mathcal{S}}(SAND(\phi_1, \phi_2))$ and $s \in Paths_{\mathcal{S}}(AND(\phi_1, \phi_2))$.

Example 4.7. If we consider the game arena \mathcal{G} given in Figure 1b, we have that $(d_1, d_2) \models D1 \wedge \neg seen$, thus the path $(o, m)(d_1, d_2) \in Paths_{\mathcal{G}}(D1 \wedge \neg seen)$. This gives us also $(o, m)(d_1, d_2) \in Paths_{\mathcal{G}}(OR(D1 \wedge \neg seen, D2 \wedge \neg seen))$.

4.3 Strategy semantics for attack trees

We start this section by formally defining *strategic trees* as well as some handful operators over them. We use a definition of a tree really close to the one made from prefix-closed languages (for example in [3, p. 15]) except that we fix a letter to represent the root.

Definition 4.8. A *strategic tree* (written s-tree for short) over an alphabet Σ is a language T of the form ℓL with $\ell \in \Sigma$ and L is a prefix-closed language over Σ .

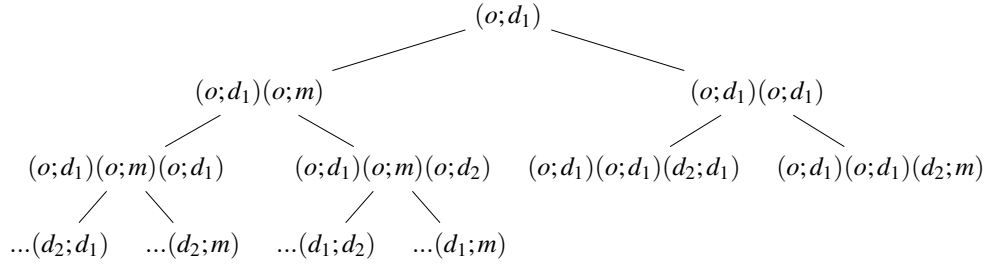
For an s-tree $T = \ell L$, ℓ is called the *root*. For a word $w \in T$, if there exists no $w' \in T$ such that $w \triangleleft w'$ then we call w a *leaf*. The set of all leaves of T is denoted by $Leaves(T)$. For two words $w, w' \in T$ such that $w \triangleleft w'$, if there exist no $w'' \in T$ such that $w \triangleleft w'' \triangleleft w'$, then we says that w is the *parent* of w' and w' is a *child* of w . The set of all children of a word w in a s-tree T is denoted by $Children_T(w)$. The depth of an s-tree is the size of the longest word in it.

Example 4.9. Figure 4 shows an s-tree over the alphabet *Pos*, the set of positions of the game arena of Figure 1b.

As in Example 4.9, for the particular case where alphabet Σ is the set of positions on some game arena, we develop several notions on s-trees and show that strategies can be presented as s-trees.

For the rest of this section we fix a game arena $\mathcal{G} = (Pos, Act, \delta, val)$.

The next lemma asserts that all histories consistent with a strategy and starting from a given position form an s-tree.

Figure 4: strategic tree $T_{(o,d_1)}^\mu$

Lemma 4.10. *Let μ be a strategy for some player and $v \in Pos$ be a game position. The language $T_v^\mu = Prefixes(Outcomes(\mu)) \cap Hist(\mathcal{G}, v)$ is an s-tree over alphabet Pos , and is called the s-tree associated with μ from position v .*

The proof is straightforward from the definition of T_v^μ .

By Lemma 4.10, each branch of T_v^μ is the succession of all prefixes (in terms of words) of a play consistent with μ . Reciprocally, each play consistent with μ and starting from position v is represented by a branch of T_v^μ . Therefore T_v^μ fully describes the strategy μ starting from position v .

Example 4.11. Consider the game arena \mathcal{G} given in Figure 1b and the strategy μ for the thief consisting in waiting one unit of time, then, if the guard is at some door, going to the other door and if the guard is currently in motion, waiting another unit of time before going to the door where the guard will not be. If we call this strategy μ , the strategic tree $T_{(o,d_1)}^\mu$ is given in Figure 4.

As we put the focus on attack trees, we take the convention that, in the game arena, Player 1 is called *Attacker* and Player 2 is called *Defender*. In this setting, Attacker tries to achieve an attack that is described by some attack tree τ , while Defender tries to prevent it from happening. In other words, the winning plays for the Attacker are given as $\Gamma_A = Paths_{\mathcal{G}}(\tau)$. Our strategy semantics consists of the set of winning strategies for this game.

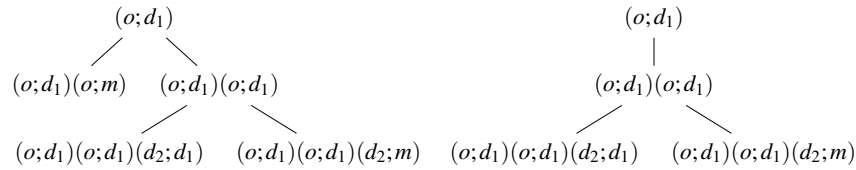
We start by motivating a construction only for a leaf attack tree. The strategy semantics for an attack tree ϕ is the set of all strategies that are winning for the reachability game defined by ϕ . Remark that for the case of reachability games, once a winning position is reached, the continuation of the play does not matter. Therefore, for reachability games, the s-tree corresponding to a winning strategy can be cut as a finite tree: this cut consists in removing all children of a node describing a history ending in a position where ϕ holds. This way of cutting motivates the definition of prefix of s-trees as follows:

Definition 4.12. Let T be an s-tree over Σ^* . An s-tree T' is a *prefix* of T if $root(T') = root(T)$, and $T' \subseteq T$, and for every $w \in T' \setminus Leaves(T')$, we have $Children_{T'}(w) = Children_T(w)$.

Example 4.13. For the s-tree $T_{(o,d_1)}^\mu$ of Figure 4 and the two trees given in Figure 5, we have T_a is a prefix of $T_{(o,d_1)}^\mu$, but T_b is not because $Children_{T_{(o,d_1)}^\mu}(o, d_1) = \{(o, d_1)(o, m), (o, d_1)(o, d_1)\} \neq Children_{T_b}(o, d_1) = \{(o, d_1)(o, d_1)\}$.

With this notion of prefix, it is immediate to characterise attack trees that witness a strategy.

Definition 4.14. Consider a leaf attack tree ϕ , and write $\phi \subseteq Pos$ for the set of positions where ϕ holds. Consider μ a strategy for Attacker in the reachability game (\mathcal{G}, ϕ) and T_v^μ the associated s-tree from position v . A finite s-tree T is a *witness* of μ from position v if T is a finite prefix of T_v^μ , and $Leaves(T) \subseteq Pos^* \phi$.

Figure 5: two s-trees: T_a (left) and T_b (right)

Definition 4.14 can be generalised to an arbitrary reachability condition $W_1 \subseteq Pos$ as follows: T is a witness of μ from position v if T is a finite prefix of T_v^μ and $h \in Leaves(T)$ implies $last(h) \in W_1$.

Example 4.15. In the game arena of Figure 1b, if we consider the reachability condition $W_1 = \{(o, m), (d_2, m), (d_2, d_1)\}$, then the attack tree T_a of Figure 5 is a witness for the s-tree $T_{(o, d_1)}^\mu$ drawn in Figure 4.

Definition 4.14 leads us to the following intuitive lemma.

Lemma 4.16. *Let μ be a strategy for Attacker and W_1 be a winning condition. Then μ is a winning strategy for (\mathcal{G}, W_1) from position $v \in Pos$ if, and only if, there exists a witness T of μ from v .*

The proof relies on the König's Lemma.

Thus, for a leaf ϕ , the strategy semantics is all witnesses that can be constructed from a winning strategy over the reachability game defined by ϕ . Moreover, an s-tree is in the semantics of a leaf attack tree if it is a prefix of some strategy and if all its leaves are in the path semantics of the attack tree. The former condition guarantees that our s-tree has the shape of a strategy, while the latter guarantees that the strategy is winning. As we will see below, those are the two conditions we use to define the strategy semantics of arbitrary attack trees.

For the first condition, we say that an s-tree T is *well-formed* if there exists a strategy μ and a position v such that T is a prefix of T_v^μ . For the second condition, we use the following definition:

Definition 4.17. Let τ be an attack tree. A τ -s-tree is a finite s-tree T over Pos such that $Leaves(T) \subseteq Paths_{\mathcal{G}}(\tau)$.

Since for a leaf attack tree ϕ , we have $Paths_{\mathcal{G}}(\phi) = Pos^* \phi$, a witness T (Definition 4.14) is a ϕ -s-tree. We now have all the material to define the strategy semantics of an attack tree.

Definition 4.18. Let τ be an attack tree. The *strategy semantics* associated with τ , written $Strat_{\mathcal{G}}(\tau)$ is the set of all well-formed τ -s-trees.

In particular, $Strat_{\mathcal{G}}(\phi)$ is the set of all witnesses in the reachability game (\mathcal{G}, ϕ) .

We can see that the idea is far from the one of attack-defence trees in [8]. In attack-defence trees, the countermeasure is a structure similar to an attack tree whose semantics describes paths that prevent an attack from succeeding, and by no means a strategy of the attacker's opponent in the arena.

Now that we defined our semantics, we might want to know if it can be obtained in a compositional manner? Namely, if the semantics of a compound tree can be defined in terms of the semantics of its subtrees: More formally.. can we define $Strat_{\mathcal{G}}(OP(\tau_1, \dots, \tau_n))$ on the basis of $Strat_{\mathcal{G}}(\tau_1), \dots, Strat_{\mathcal{G}}(\tau_n)$? Sadly, the answer is no:

Example 4.19. Consider the game arena defined in Figure 1b. Obviously, our attacker here will be the thief while the guard will do the defender role. We also consider a new proposition: *Start* which only holds at position $\{o, m\}$. We have that the semantics of $SAND(start, D1 \wedge \neg seen)$ is empty. Indeed, the guard can choose to only keep door 1 and thus, the thief will not be able to attain D1 while remaining unseen. Similarly, $Strat_{\mathcal{G}}(SAND(start, D2 \wedge \neg seen))$ is empty. However, the strategy consisting on

waiting one unit of time then going through the door not controlled by the guard is a winning strategy, it is easy to construct a witness for that strategy that attains the objective of $OR(SAND(start, D1 \wedge \neg seen), SAND(start, D2 \wedge \neg seen))$ and thus is in its strategy semantics.

The previous example showcases an empty semantics for τ_1 and τ_2 but a non-empty one for $OR(\tau_1, \tau_2)$. This is because, for ϕ_1 and ϕ_2 two propositional formulas over $Prop$, there are more strategies to achieve $\phi_1 \vee \phi_2$ than strategies only achieving ϕ_1 or only achieving ϕ_2 . We can for example consider a strategy that, depending on the move of the opponent, chooses whether it prefers to attain ϕ_1 or to attain ϕ_2 .

Remark that, using the "merge" operator of [11] provides us a compositional semantics for attack trees with $SAND$ -only operators. However, we have already argued that the OR operator has some problems just as the AND operator for more elaborate examples. Still, it is possible to tune the semantics so that it becomes compositional for the AND operator, at the price of losing clarity, but more regrettably without solving the hopeless case of the OR operator.

5 Decision Problems over attack trees

In this section, we discuss two common decision problems over semantics of attack trees and determine their complexities with respect to the path semantics and the strategy semantics. The first problem we consider is the Non-Emptiness problem. This problem consists of, given an attack tree and a game arena, deciding whether its semantics is not empty:

Definition 5.1. The *Non-Emptiness* problem is the following decision problem for a fixed semantics $\llbracket \cdot \rrbracket_{\mathcal{G}}$ of attack trees:

Input: \mathcal{G} , a game arena, τ , an attack tree.

Output: Yes if $\llbracket \tau \rrbracket_{\mathcal{G}} \neq \emptyset$, No otherwise.

The Non-Emptiness problem for the path semantics is denoted by PNE while the Non-Emptiness problem for the strategy semantics is denoted SNE. A positive instance of PNE tells us that Attacker has a favourable scenario to attack. A positive instance of SNE tells us that Attacker has a strategy (it is possible for him to attack successfully the system independently of the defender/environment component).

We now turn to the Membership problem.

Definition 5.2. The *Membership problem* is the following decision problem for a fixed attack tree semantics $\llbracket \cdot \rrbracket_{\mathcal{G}}$ of of type X :

Input: \mathcal{G} , a game arena, τ , an attack tree and $x \in X$.

Output: Yes if $x \in \llbracket \tau \rrbracket_{\mathcal{G}}$, No otherwise.

The Membership problem for the path semantics is denoted by PM while the Membership problem for the strategy semantics is denoted SM. PM consists of determining whether a path is an attack or not. It can be really useful if we have an attack tree describing an attack goal over an information system and a log file of that system. Determining if the system has been attacked is equivalent to determining whether the path described by the log file is in the path semantics of the attack tree or not. The idea behind SM is different: it is useful to determine whether a strategy is winning or not for a given attack objective. We start to analyse the complexity of PM and take advantage of it for the proofs of the other results. We then consider SNE. After that, PNE is easily determined as a particular case of SNE and we finish by SM whose proof uses similar and simpler constructions than the one for SNE.

If we use attack trees with preconditions, the problem PM is NP-hard; this comes from the fact that the *packed interval covering problem*, which can be easily captured by the parallel composition (see [13]), is NP-complete (see [14]). However, PM becomes simpler if we discard preconditions:

Theorem 5.3. *PM is in P.*

For a polynomial algorithm, we use the fact that a word is in the semantics of an attack tree, then adding an arbitrary prefix to it keeps it in the semantics. As a consequence, we do not need to recompute which sub-goals of the attack tree are satisfied whenever we add a position in front of a path. Thus, the shape of the problem is well-suited for a backward induction over the input path. Moreover, determining if a given input path satisfies an attack tree knowing whether it satisfies the sub-trees can be done in linear time over the size of the attack tree.

We now turn to the complexity of SNE.

Theorem 5.4. *SNE is PSPACE-complete.*

For the membership, we construct an alternating algorithm (see [2]) solving the problem that can be executed in polynomial time. This algorithm consists of synthesizing a history over the game arena and then verifying that this history is an attack (by Theorem 5.3, this verification is doable in polynomial time). To construct this history, we finitely iterate first to make a non-deterministic existential guess for the action of Attacker and then a non-deterministic universal guess for the action of Defender. We then show that the resulting history is in the path semantics of the input attack tree τ if, and only if, the strategy semantics of τ is not empty. We guarantee a polynomial time execution, namely that the resulting history need not be too long with the following lemma.

Lemma 5.5. *Let $\mathcal{G} = (Pos, Act, \delta, val)$ be a game arena and τ be an attack tree with n leaves. If $Strat_{\mathcal{G}}(\tau) \neq \emptyset$, then there exists $T \in Strat_{\mathcal{G}}(\tau)$ of depth $d \leq |Pos| \times n$.*

The basic idea behind to prove Lemma 5.5 is that, memoryless strategies suffice in reachability games (see [4]).

We design Algorithm 1 to solve SNE whose idea is explained above and show that it belongs to PSPACE.

Algorithm 1 $SNE(\mathcal{G}, \tau)$

Input: \mathcal{G} a game arena and τ an attack tree with n leaves

Output: *True* if $Strat_{\mathcal{G}}(\tau) \neq \emptyset$, *False* otherwise.

```

1:  $h \leftarrow$  empty list
2:  $v \leftarrow [\exists]$ guess position in  $Pos$ 
3:  $h.append(v)$ 
4: while  $size(h) < |Pos| \times n$  do
5:    $[\exists]$ guess break or not
6:    $a_1 \leftarrow [\exists]$ guess action in  $Act_A$ 
7:    $a_2 \leftarrow [\forall]$ guess action in  $Act_D$ 
8:    $h.append(\delta(last(h), (a_1, a_2)))$ 
9: end while
10: return  $h \in Paths_{\mathcal{G}}(\tau)$ 

```

Lemma 5.6. *Algorithm 1 is an alternating polynomial-time algorithm and solves SNE.*

Proof. We start by showing the complexity of the algorithm, then we show its correctness.

From the loop at Line 4, it is executed polynomially many times in the size of the input attack tree and of the game arena. We also know (Theorem 5.3) that the condition $h \in \text{Paths}_{\mathcal{G}}(\tau)$ at Line 10 can be evaluated in polynomial, therefore, Algorithm 1 is polynomial-time alternating.

Assume Algorithm 1 returns *True*, then, for each choice made by universal guess, there exists a choice made by existential guess guaranteeing that the obtained history is in $\text{Paths}_{\mathcal{G}}(\tau)$. As a consequence, the choices made by the existential guesses reflect a strategy in the game arena that satisfies τ so, $\text{Strat}_{\mathcal{G}}(\tau) \neq \emptyset$. Conversely, if $\text{Strat}_{\mathcal{G}}(\tau) \neq \emptyset$, then there exists (by Lemma 5.5) an s-tree T of depth $\leq |\text{Pos}| \times n$. Thus the existential guesses can simply follow the strategy given by T and then choose to go out from the main loop by the "break" command at Line 5 of Algorithm 1 whenever the sequence of choices (existential and universal) in the execution is reflected by a full branch of the s-tree T . \square

For the PSPACE-hardness of SNE, our construction is inspired by the one in [1]: the authors reduce (in polynomial time) the SAT problem to the PNE problem with attack trees (using preconditions). In fact, even if in that paper, authors use attack trees with preconditions, we can adapt it without preconditions. We can even cast the approach to QBF that we first recall:

Definition 5.7. The *quantified Boolean formula* (QBF) is the following decision problem:

Input: a formula of the form $Q_1x_1, \dots, Q_nx_n\psi(x_1, \dots, x_n)$ with $Q_i \in \{\exists, \forall\}$ and ψ a Boolean formula in conjunctive normal form over propositions x_1, \dots, x_n .

Output: *Yes* if the input formula is true, *No* otherwise.

Lemma 5.8. *The QBF problem can be reduced to SNE in polynomial time.*

It is easy to understand the reduction principle on an example.

Example 5.9. Consider the formula $\psi = \exists x_1 \forall x_2 \exists x_3, x_1 \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_3)$. Let $C_1 = x_1$, $C_2 = (x_2 \vee x_3)$ and $C_3 = (\neg x_2 \vee x_3)$ be the three clauses in ψ . The game arena \mathcal{G} associated with this formula is drawn in Figure 6: for each position v_i (resp. $\neg v_i$), the proposition p_i holds if $v_i \in C_i$ (resp. $\neg v_i \in C_i$). Remark that this game arena is a special case of game arena called turn-based game arena: only one player makes an action in each position, we say that a position belongs to the player who can play on it. We decide classically which position belongs to each player based on quantifiers of ψ (see the proof of Lemma 5.8 for further explanations). We represent Attacker positions with a circle and Defender positions with a square (position v_3 and position $\neg pos_3$ have only one successor position, therefore, it does not matter which player makes the move; by convention, we say they belong to the attacker). Then, ψ holds if, and only if, $\text{Strat}_{\mathcal{G}}(\text{SAND}(\text{start}, \text{AND}(p_1, p_2, p_3))) \neq \emptyset$.

We now start the proof of lemma 5.8:

Proof. Let $Q_1x_1, \dots, Q_nx_n\psi(x_1, \dots, x_n)$ with $Q_i \in \{\exists, \forall\}$ and with ψ a Boolean formula over variables x_1, \dots, x_n be an instance of the QBF problem. Since ψ is in conjunctive normal form, we can write it as $\psi = \psi_1 \wedge \dots \wedge \psi_k$ with ψ_i denoting disjunctive clauses containing literals of the form x_j or $\neg x_j$ with $x_i \in \{x_1, \dots, x_n\}$.

We consider the set of propositions $\text{Prop} = \{\text{Start}, p_1, \dots, p_k\}$ with the following game arena: $\mathcal{G} = (\text{Pos}, \text{Act}, \delta, \text{val})$, where $\text{Pos} = \{\text{Start}\} \cup \{v_i | 1 \leq i \leq n\} \cup \{\neg v_i | 1 \leq i \leq n\}$, $\text{Act}_A = \text{Act}_D = \{\text{True}, \text{False}\}$. If $Q_0 = \exists$, then position *Start* is an Attacker position, otherwise, it's a defender position. Moreover, playing action *True* at position *start* leads to position v_1 while playing *False* leads to position $\neg pos_1$. Similarly, for each $2 \leq i \leq n$, if $Q_i = \exists$ then v_{i-1} and $\neg v_{i-1}$ are Attacker positions, otherwise, they are Defender positions. Furthermore, playing *True* at position v_{i-1} or $\neg v_{i-1}$ leads to position v_i

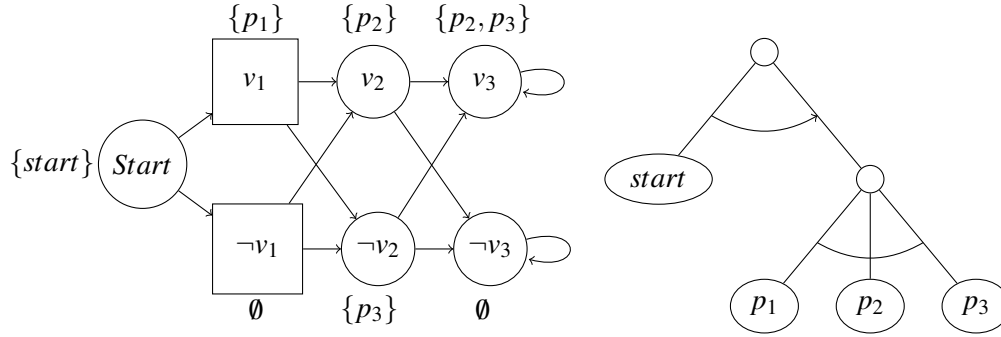


Figure 6: Game arena and attack tree associated to the formula given in Example 5.9

while playing *False* leads to $\neg v_i$. Positions v_n and $\neg v_n$ are Attacker positions, moreover, the transitions over those two positions are self loops.

We define $val(Start) = \{Start\}$ and for each $i \leq i \leq n$, $val(v_i) = \{p_j | x_i \in \psi_j\}$ and $val(\neg v_i) = \{p_j | \neg x_i \in \psi_j\}$. From this definition, if we consider that the attacker tries to satisfy the input QBF formula and the defender tries to prevent it, we have a classic game. We then only need to show that the objective of the attacker can be well described using an attack tree, which is the case by considering $\tau = SAND(Start, AND(p_1, \dots, p_n))$. Indeed, if there exists a strategy to satisfy the input QBF formula, then this strategy satisfies ψ_1, \dots, ψ_k and thus, can be executed in the constructed game arena to achieve $AND(p_1, \dots, p_n)$ while starting at position $Start$, therefore, that strategy is in τ . Conversely, if $Strat_{\mathcal{G}}(\tau) \neq \emptyset$, then one of such strategies assures that we satisfy the input QBF instance. \square

By Lemma 5.8 , SNE is PSPACE-hard, which achieves the proof of Theorem 5.4.

We now turn to PNE.

Theorem 5.10. *PNE is NP-complete.*

For the NP-membership, since our problem is a particular case of the problem discussed in [1], it is at least as easy. For the NP-hardness we reduce SAT: if we apply the same construction as in the proof of Lemma 5.8, since we cannot leave any choice for the defender in a transition system and the path semantics is defined over a transition system and not a game arena, we can reduce formulas of QBF only using \exists operators. In other words, we can reduce SAT. In fact, by doing so, we are doing the exact construction of the proof in [1]. Moreover the attack tree with preconditions $AND(< start, \phi_1 >, \dots, < start, \phi_n >)$ used in that paper is completely equivalent to $SAND(start, AND(\phi_1, \dots, \phi_n))$ in our formalism. Thus the proof in [1] can be well adapted for our problem.

Lastly, we study SM.

Theorem 5.11. *SM is CONP-complete.*

For the membership, we can use the same idea as for the membership of the SNE except that, now, we already know the strategy of the attacker, we thus do not need to use any existential guess for the action of Attacker. In other words, it is equivalent to simply considering Defender choosing a branch of the attack tree and then verifying if it forms an attack or not. Therefore, we use a variant of Algorithm 1 without existential choices, this gives us a CONP algorithm.

For the hardness, we still use the idea of the construction behind the SNE, but now, we consider that only the actions of Defender matter in the progress of the game arena. This way, we can reduce the UNSAT problem, known to be CONP-complete, to SM. The UNSAT problem is nothing less than the sub-problem of the QBF problem where an instance of the problem only uses " \forall " quantifiers.

This concludes the discussion over decision problems; our results are summarised in Table 1.

6 Future work

In this paper, we proposed a strategy semantics for attack trees, useful to tackle some practical questions (SNE and SM) not expressible with standard semantics provided by the literature. The price to pay is to renounce a compositional semantics of attack trees. One way to regain it might be to consider a strategy semantics based on a tree automata: we associate with each attack tree a tree automaton recognising its strategy semantics. This is currently work. Moreover, being able to consider automata recognising the strategy semantics allows us to model attack scenarios with constraints, for example, considering that the attacker cannot perform a given action more than a certain amount of time.

Moreover, we are currently exploring the possibility to expand the path and the strategy semantics to attack-defense trees. The main idea is to consider a counter operator in attack trees. This generalisation could lead to a better understanding of the differences between the strategy semantics and the attack-defence tree formalism.

References

- [1] Maxime Audinot, Sophie Pinchinat & Barbara Kordy (2017): *Is my attack tree correct?* In: *European Symposium on Research in Computer Security*, Springer, pp. 83–102, doi:10.1007/978-3-319-66402-6_7.
- [2] Ashok K Chandra & Larry J Stockmeyer (1976): *Alternation*. In: *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, IEEE, pp. 98–108, doi:10.1109/SFCS.1976.4.
- [3] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison & Marc Tommasi (2008): *Tree automata techniques and applications*.
- [4] Luca De Alfaro, Thomas A Henzinger & Orna Kupferman (2007): *Concurrent reachability games*. *Theoretical computer science* 386(3), pp. 188–217, doi:10.1016/j.tcs.2007.07.008.
- [5] Ross Horne, Sjouke Mauw & Alwen Tiu (2017): *Semantics for specialising attack trees based on linear logic*. *Fundamenta Informaticae* 153(1-2), pp. 57–86, doi:10.3233/FI-2017-1531.
- [6] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Saša Radomirović & Rolando Trujillo-Rasua (2015): *Attack trees with sequential conjunction*. In: *IFIP International Information Security and Privacy Conference*, Springer, pp. 339–353, doi:10.1007/978-3-319-18467-8_23.
- [7] Aivo Jürgenson & Jan Willemson (2008): *Computing exact outcomes of multi-parameter attack trees*. In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, pp. 1036–1051, doi:10.1007/978-3-540-88873-4_8.
- [8] Barbara Kordy, Sjouke Mauw, Saša Radomirović & Patrick Schweitzer (2010): *Foundations of attack-defense trees*. In: *International Workshop on Formal Aspects in Security and Trust*, Springer, pp. 80–95, doi:10.1007/978-3-642-19751-2_6.
- [9] Barbara Kordy, Sjouke Mauw, Saša Radomirović & Patrick Schweitzer (2014): *Attack-defense trees*. *Journal of Logic and Computation* 24(1), pp. 55–87, doi:10.1093/logcom/exs029.
- [10] Sjouke Mauw & Martijn Oostdijk (2005): *Foundations of attack trees*. In: *International Conference on Information Security and Cryptology*, Springer, pp. 186–198, doi:10.1007/11734727_17.
- [11] Soumya Paul, Ramaswamy Ramanujam & Sunil Simon (2015): *Automata and compositional strategies in extensive form games*. In: *Models of Strategic Reasoning*, Springer, pp. 174–201, doi:10.1007/978-3-662-48540-8_6.

- [12] Sophie Pinchinat, Barbara Fila, Florence Wacheux & Yann Thierry-Mieg (2019): *Attack trees: a notion of missing attacks*. In: *International Workshop on Graphical Models for Security*, Springer, pp. 23–49, doi:10.1007/978-3-030-36537-0_3.
- [13] Sophie Pinchinat, François Schwarzentruher & Sébastien Lê Cong (2020): *Library-Based Attack Tree Synthesis*. In: *International Workshop on Graphical Models for Security*, Springer, pp. 24–44, doi:10.1007/978-3-030-62230-5_2.
- [14] Abdallah Saffidine, Sébastien Lê Cong, Sophie Pinchinat & François Schwarzentruher (2019): *The Packed Interval Covering Problem is NP-complete*. *arXiv preprint arXiv:1906.03676*.
- [15] Bruce Schneier (1999): *Attack trees*. *Dr. Dobbs's journal* 24(12), pp. 21–29, doi:10.1002/9781119183631.ch21.
- [16] Wojciech Wideł, Maxime Audinot, Barbara Fila & Sophie Pinchinat (2019): *Beyond 2014: Formal Methods for Attack Tree-based Security Modeling*. *ACM Computing Surveys (CSUR)* 52(4), pp. 1–36, doi:10.1145/3331524.

Avoid One’s Doom: Finding Cliff-Edge Configurations in Petri Nets

Giann Karlo Aguirre-Samboni*¹ Stefan Haar*¹ Loïc Paulevé*²
Stefan Schwoon*¹ Nick Würdemann*³

¹INRIA and LMF, CNRS and ENS Paris-Saclay, Université Paris-Saclay, Gif-sur-Yvette, France
{giann-karlo.aguirre-samboni, stefan.haar, stefan.schwoon}@inria.fr

²Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, Talence, France
loic.pauleve@labri.fr

³Department of Computing Science, University of Oldenburg, Oldenburg, Germany
wuerdemann@informatik.uni-oldenburg.de

A crucial question in analyzing a concurrent system is to determine its long-run behaviour, and in particular, whether there are irreversible choices in its evolution, leading into parts of the reachability space from which there is no return to other parts. Casting this problem in the unifying framework of safe Petri nets, our previous work [3] has provided techniques for identifying *attractors*, i.e. terminal strongly connected components of the reachability space, whose attraction basins we wish to determine. Here, we provide a solution for the case of safe Petri nets. Our algorithm uses net unfoldings and provides a map of all of the system’s configurations (concurrent executions) that act as *cliff-edges*, i.e. any maximal extension for those configurations lies in some basin that is considered fatal. The computation turns out to require only a relatively small prefix of the unfolding, just twice the depth of Esparza’s complete prefix.

1 Introduction

Unfoldings of Petri nets [6], which are essentially event structures in the sense of Winskel et al. [18] with additional information about *states*, are an acyclic representation of the possible sequences of transitions, akin to Mazurkiewicz traces but enriched with branching information.

Many reachability-related verification problems for concurrent systems have been successfully addressed by Petri-net unfolding methods over the past decades, see [15, 7, 6]. However, questions of long-term behaviour and stabilization have received relatively little attention. With the growing interest in formal methods for biology, the key feature of *multistability* of systems [30, 24, 20, 23] comes into focus. It has been studied in other qualitative models such as Boolean and multivalued networks [29, 28, 26]. Multistability characterizes many fundamental biological processes, such as cellular differentiation, cellular reprogramming, and cell-fate decision; in fact, stabilization of a cell regulatory network corresponds to reaching one of the - possibly many - phenotypes of the cell, thus explaining the important role of multistability in cell biology. However, multistability emerges also in many other branches of the life sciences; our own motivation is the qualitative analysis of the fate of *ecosystems*, see [25].

Multistability can be succinctly described as the presence of several *attractors* in the system under study. Attractors characterize the stable behaviours, given as the smallest subsets of states from which the system cannot escape; in other words, they are terminal strongly connected components of the associated

*We gratefully acknowledge the fruitful exchanges with Cédric Gauchere and Franck Pommereau. This work was supported by the *DIGICOSME* grant ESCAPE, *DIGICOSME RD 242-ESCAPE-15203*, and by the French Agence Nationale pour la Recherche (ANR) in the scope of the project “BNeDiction” (grant number ANR-20-CE45-0001).

transition system. In the long run, the system will enter one of its attractors and remain inside; multi-stability arises when there is more than one such attractor. The *basin* of attractor **A** consists of the states from which the system inevitably reaches **A**.

The basin includes the attractor itself, and possibly one or several transient states [14].

We aim at finding the tipping points in which the system switches from an undetermined or *free* state into some basin; while interesting beyond that domain, this is a recurrent question in the analysis of signalling and gene regulatory networks [5, 16]. In [8], the authors provide a method for identifying, in a boolean network model, the states in which one transition leads to losing the reachability of a given attractor (called *bifurcation transitions* there; we prefer to speak of *tipping points* instead). However, enumerating the states in which the identified transitions make the system branch away from the attractor can be highly combinatorial and hinders a fine understanding of the branching. Thus, the challenge resides in identifying the specific contexts and sequences of transitions leading to a strong basin.

Using a bounded unfolding prefix, all reachable attractors [3] can be extracted. Also, we have exhibited ([11]) the particular shape of basins that are visible in a concurrent model.

In the present paper, we build on this previous analyses; the point of view taken here is that all attractors correspond to the *end* of the system's free behaviour, in other words to its *doom*. We will give characterizations of basin boundaries (called *cliff-edges* below), and of those behaviours that remain *free*, in terms of properties of the unfolding, reporting also on practical experiments with an implementation of the algorithms derived. We finally introduce a novel type of quantitative measure, called *protectedness*, to indicate how far away (or close) a system is from doom, in a state that is still free per se. General discussions and outlook will conclude this paper.

2 Petri Nets and Unfoldings

We begin now by recalling the basic definitions needed below. A **Petri net** is a bipartite directed graph whose nodes are either *places* or *transitions*, and places may carry *tokens*. In this paper, we consider only *safe* Petri nets where a place carries either one or no token in any reachable marking. The set of currently active places form the state, or *marking*, of the net.

Note. Some remarks are in order concerning our use of Petri nets versus that of *boolean networks*, which are more widely used in systems biology. Safe (or 1-bounded) Petri nets [17] are close to Boolean and multivalued networks [4], yet enable a more fine-grained specification of the conditions for triggering value changes. Focussing on safe PNs entails no limitation of generality of the model, as two-way behaviour-preserving translations between Boolean and multivalued models exist (see [4] and the appendix of [3] for discussion). We are thus entitled to move between these models without loss of expressiveness; however, Petri nets provide more convenient ways to develop and present the theory and the algorithms here.

Formally, a *net* is a tuple $N = \langle P, T, F \rangle$, where T is a finite set of *transitions*, P a finite set of *places*, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation* whose elements are called *arcs*. In figures, places are represented by circles and the transitions by boxes (each one with a label identifying it).

For any node $x \in P \cup T$, we call *pre-set* of x the set $\bullet x = \{y \in P \cup T \mid \langle y, x \rangle \in F\}$ and *post-set* of x the set $x^\bullet = \{y \in P \cup T \mid \langle x, y \rangle \in F\}$. A subset $M \subseteq P$ of the places is called a *marking*. A *Petri net* is a tuple $\mathcal{N} = \langle P, T, F, M_0 \rangle$, with $M_0 \subseteq P$ an *initial marking*. Markings are represented by dots (or tokens) in the marked places. A transition $t \in T$ is *enabled* at a marking M , denoted $M \xrightarrow{t}$, if and only if $\bullet t \subseteq M$. An enabled transition t can *fire*, leading to the new marking $M' = (M \setminus \bullet t) \cup t^\bullet$;¹ in that case we write

¹This definition does not correspond to the standard semantics of Petri nets, but is equivalent for safe Petri nets, and we

$M \xrightarrow{t} M'$. A *firing sequence* from a marking M'_0 is a (finite or infinite) sequence $w = t_1 t_2 t_3 \dots$ over T such that there exist markings M'_1, M'_2, \dots with $M'_0 \xrightarrow{t_1} M'_1 \xrightarrow{t_2} M'_2 \xrightarrow{t_3} \dots$. If w is finite and of length n , we write $M'_0 \xrightarrow{w} M'_n$, and we say that M'_n is *reachable* from M'_0 , also simply written $M'_0 \rightarrow M'_n$. We denote the set of markings reachable from some marking M in a net N by $\mathbf{R}_N(M)$. A Petri net $\langle N, M_0 \rangle$ is considered *safe* if every marking in $M \in \mathbf{R}_N(M_0)$ and every transition t enabled in M satisfy $(M \cap t^\bullet) \subseteq {}^\bullet t$. In this paper, we assume that all our Petri nets are safe.

From an initial marking of the net, one can recursively derive all possible transitions and reachable markings, resulting in a *marking graph* (Def. 1).

Definition 1 Let $N = \langle P, T, F \rangle$ be a net and \mathcal{M} a set of markings. The marking graph induced by \mathcal{M} is a directed graph $\langle \mathcal{M}, \mathcal{E} \rangle$ such that $\mathcal{E} \subseteq \mathcal{M} \times \mathcal{M}$ contains $\langle M, M' \rangle$ iff $M \xrightarrow{t} M'$ for some $t \in T$; the arc $\langle M, M' \rangle$ is then labeled by t . The reachability graph of a Petri net $\langle N, M_0 \rangle$ is the graph induced by $\mathbf{R}_N(M_0)$.

The reachability graph is always finite for safe Petri nets.

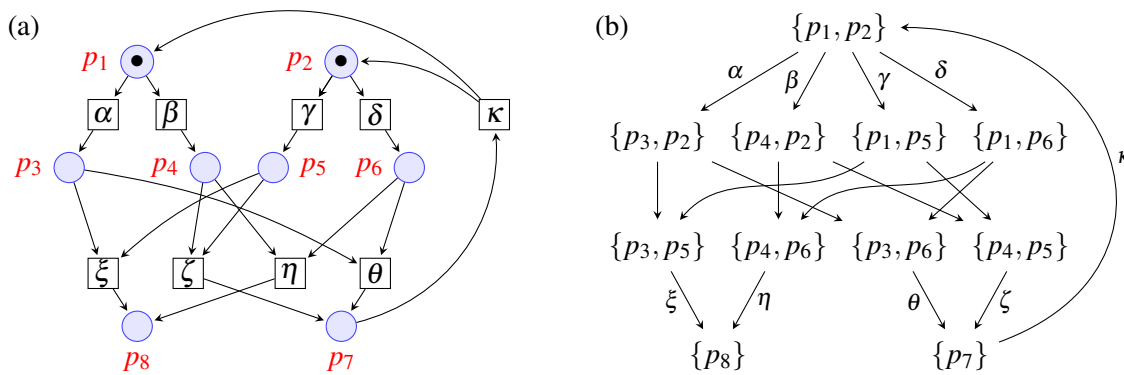


Figure 1: Petri net example from [11] in (a), and its reachability graph in (b).

Figure 1b shows the reachability graph for our running example 1a.

Unfoldings. Roughly speaking, the unfolding of a Petri net \mathcal{N} is an acyclic Petri net (with particular structural properties), \mathcal{U} , that reproduces exactly the same behaviours as \mathcal{N} .

We now give some technical definitions to introduce unfoldings formally. A more extensive treatment can be found, e.g., in [7, 6].

Definition 2 (Causality, conflict, concurrency) Let $N = \langle P, T, F \rangle$ be a net and $x, y \in P \cup T$ two nodes of N . We say that x is a causal predecessor of y , noted $x < y$, if there exists a non-empty path of arcs from x to y . We note $x \leq y$ if $x < y$ or $x = y$. If $x \leq y$ or $y \leq x$, then x and y are said to be causally related. Transitions u and v are in direct conflict, noted $u \#_\delta v$, iff ${}^\bullet u \cap {}^\bullet v \neq \emptyset$; nodes x and y are in conflict, noted $x \# y$, if there exist $u, v \in T$ such that $u \neq v$, $u \leq x$, $v \leq y$, and $u \#_\delta v$. We call x and y concurrent, noted $x \text{ co } y$, if they are neither causally related nor in conflict. A set of concurrent places is called a co-set.

Definition 3 (Occurrence net) Let $\mathcal{O} = \langle B, E, G, c_0 \rangle$ be a Petri net. We say that \mathcal{O} is an occurrence net if it satisfies the following properties:

prefer it for the sake of simplicity.

1. The causality relation $<$ is acyclic;
2. $|\bullet b| \leq 1$ for all places $b \in B$, and $b \in \mathbf{c}_0$ iff $|\bullet b| = 0$;
3. For every transition $e \in E$, $e \# e$ does not hold, and $\{x \mid x \leq e\}$ is finite.

Following the convention in the unfolding literature, we refer to the places of an occurrence net as *conditions* and to its transitions as *events*. Due to the structural constraints, the firing sequences of occurrence nets have special properties: if some condition b is marked during a run, then the token on b was either present initially or produced by one particular event (the single event in $\bullet b$); moreover, once the token on b is consumed, it can never be replaced by another token, due to acyclicity of $<$.

Definition 4 (Configurations, cuts) Let $\mathcal{O} = \langle B, E, G, \mathbf{c}_0 \rangle$ be an occurrence net. A set $C \subseteq E$ is called a configuration of \mathcal{O} if (i) C is causally closed, i.e. $e' < e$ and $e \in C$ imply $e' \in C$; and (ii) C is conflict-free, i.e. if $e, e' \in C$, then $\neg(e \# e')$. In particular, for any $e \in E$, $[e] \triangleq \{e' \in E : e' \leq e\}$ and $\langle e \rangle \triangleq \{e' \in E : e' < e\}$ are configurations, called the cone and stump of e , respectively; any C such that $\exists e \in E : C = [e]$ is called a prime configuration. Denote the set of all configurations of \mathcal{O} as $\mathcal{C}(\mathcal{O})$, and its subset containing all **finite** configurations as $\mathcal{C}^f(\mathcal{O})$, where we drop the reference to \mathcal{O} if no confusion can arise. The cut of a finite C , denoted $\mathbf{cut}(C)$, is the set of conditions $(\mathbf{c}_0 \cup \mathbf{C}^\bullet) \setminus \bullet C$. A run is a maximal element of $\mathcal{C}(\mathcal{O})$ w.r.t. set inclusion; denote the set of \mathcal{O} 's runs as $\Omega = \Omega(\mathcal{O})$, and its elements generically by ω . If $C \in \mathcal{C}^f$, let the crest of C be the set $\mathbf{crest}(C) \triangleq \max_{<}(C)$ of its maximal events. We say that configuration C enables event e , written $C \overset{e}{\rightsquigarrow}$, iff i) $e \notin C$ and ii) $C \cup \{e\}$ is a configuration. Configurations C_1, C_2 are in conflict, written $C_1 \# C_2$, iff $(C_1 \cup C_2) \notin \mathcal{C}$ or, equivalently, iff there exist $e_1 \in C_1$ and $e_2 \in C_2$ such that $e_1 \# e_2$.²

Intuitively, a configuration is a set of events that can fire during a firing sequence of \mathcal{N} , and its cut is the set of conditions marked after that firing sequence. Note that \emptyset is a configuration, that $\mathbf{crest}(\emptyset) = \emptyset$, and that \mathbf{c}_0 is the cut of the configuration \emptyset . The crest of a prime configuration $[e]$ is $\{e\}$. In Figure 2, the

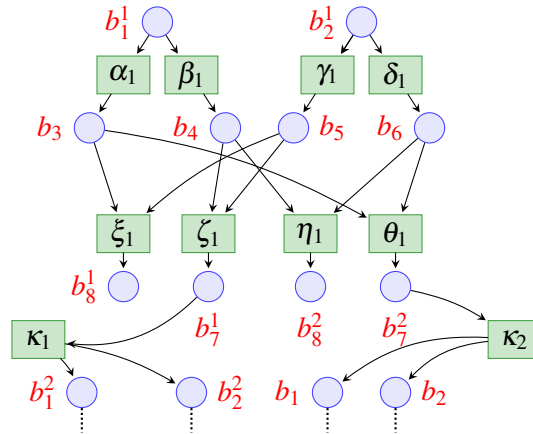


Figure 2: A prefix of the unfolding for the Petri net of Figure 1a.

initial cut is $\mathbf{c}_0 = \{b_1^1, b_2^1\}$; we have prime configurations, e.g., $\{\alpha_1\}$, $\{\beta_1\}$, $\{\xi_1\}$, $\{\zeta_1\}$ etc, and non-prime configurations $\{\alpha_1, \gamma_1\}$, $\{\alpha_1, \delta_1\}$ etc.

²The use of the same symbol $\#$ is motivated by the fact that $C_1 = [e_1]$ and $C_2 = [e_2]$ implies $C_1 \# C_2 \Leftrightarrow e_1 \# e_2$.

Definition of Unfoldings. Let $\mathcal{N} = \langle P, T, F, M_0 \rangle$ be a safe Petri net. The unfolding $\mathcal{U} = \langle B, E, G, \mathbf{c}_0 \rangle$ of \mathcal{N} is an occurrence net (equipped with a mapping π) such that the firing sequences and reachable markings of \mathcal{U} are exactly the firing sequences and reachable markings of \mathcal{N} (modulo π), see below. \mathcal{U} may be infinite; it can be inductively constructed as follows:

1. The condition set B is a subset of $(E \cup \{\perp\}) \times P$. For a condition $b = \langle e, p \rangle$, we will have $e = \perp$ iff $b \in \mathbf{c}_0$; otherwise e is the singleton event in $\bullet b$. Moreover, $\pi(b) = p$. The initial cut \mathbf{c}_0 contains exactly one condition $\langle \perp, p \rangle$ for each initially marked place $p \in M_0$ of \mathcal{N} .
2. The events of E are a subset of $2^B \times T$. More precisely, for every co-set $B' \subseteq B$ such that $\pi(B') = \bullet t$, we have an event $e = \langle B', t \rangle$. In this case, we add edges $\langle b, e \rangle$ for each $b \in B'$ (i.e. $\bullet e = B'$), we set $\pi(e) = t$, and for each $p \in t^\bullet$, we add to B a condition $b = \langle e, p \rangle$ connected by an edge $\langle e, b \rangle$.

Intuitively, a condition $\langle e, p \rangle$ represents the possibility of putting a token onto place p through a particular set of events, while an event $\langle B', e \rangle$ represents a possibility of firing transition e in a particular context.

Configurations and Markings. The following fact from the literature will be used below:

Lemma 1 (see e.g. [7]) Fix a safe Petri net $\mathcal{N} = \langle P, T, F, M_0 \rangle$ and its unfolding $\mathcal{U} = \langle B, E, G, \mathbf{c}_0, \pi \rangle$. Then for any two conditions (events) b, b' (e, e') such that $b \mathbf{co} b'$ ($e \mathbf{co} e'$), one has $\pi(b) \neq \pi(b')$ ($\pi(e) \neq \pi(e')$). Moreover, every finite configuration C of \mathcal{U} represents a possible firing sequence whose resulting marking corresponds, due to the construction of \mathcal{U} , to a reachable marking of \mathcal{N} . This marking is defined as $\text{Mark}(C) \triangleq \{ \pi(b) \mid b \in \mathbf{cut}(C) \}$.

This means, informally speaking, that any configuration of the system can be split into consecutive parts in such a way that each part is itself a configuration obtained by unfolding the Petri net ‘renewed’ with the marking reached by the previous configuration. The following definition formalizes this.

Definition 5 Let $\mathcal{O} = \langle B, E, G, \mathbf{c}_0 \rangle$ be an occurrence net. For any finite configuration $C \in \mathcal{C}^f(\mathcal{O})$, denote by $\mathcal{O}_C \triangleq \mathcal{U}(\langle N, \text{Mark}(C) \rangle)$ the shift of \mathcal{O} by C . C is the concatenation of C_1 and C_2 , written $C = C_1 \oplus C_2$, iff one has

1. $C_1 \in \mathcal{C}^f(\mathcal{O})$ and $C_1 \subseteq C$,
2. $C_2 \in \mathcal{C}^f(\mathcal{O}_{C_1})$ and $C_2 = C \setminus C_1$.

Clearly, the empty configuration \emptyset satisfies $C \oplus \emptyset = \emptyset \oplus C = C$. If $C = C_1 \oplus C_2$, write $C_1 = C \ominus C_2$ and $C_2 = C \oslash C_1$. Moreover, write

$$C = \bigoplus_{i=1}^n C_i \quad \text{iff} \quad C = C_1 \oplus \dots \oplus C_n.$$

In figure 2, setting $C_1 \triangleq \{\beta_1, \gamma_1\}$, $C_2 \triangleq \{\zeta_1, \kappa_1\}$ and $C_3 \triangleq \{\beta_1, \gamma_1, \zeta_1, \kappa_1\}$, one has $C_3 = C_1 \oplus C_2$ and consequently $C_1 = C_3 \ominus C_2$ and $C_2 = C_3 \oslash C_1$.

Complete Prefix. In general, \mathcal{U} is an infinite net, but if \mathcal{N} is safe, then it is possible to compute a finite prefix Π of \mathcal{U} that is ‘‘complete’’ in the sense that every reachable marking of \mathcal{N} has a reachable counterpart in Π , and vice versa.

Definition 6 (complete prefix, see [15, 7, 6]) Let $\mathcal{N} = \langle N, M_0 \rangle$ be a safe Petri net and $\mathcal{U} = \langle B, E, G, \mathbf{c}_0 \rangle$ its unfolding. A finite occurrence net $\Pi = \langle B', E', G', \mathbf{c}_0 \rangle$ is said to be a prefix of \mathcal{U} if $E' \subseteq E$ is causally closed, $B' = \mathbf{c}_0 \cup E'^\bullet$, and G' is the restriction of G to B' and E' . A prefix Π is said to be complete if for every reachable marking M of \mathcal{N} there exists a configuration C of Π such that (i) $\text{Mark}(C) = M$, and (ii) for each transition $t \in T$ enabled in M , there is an event $\langle B'', t \rangle \in E'$ enabled in $\mathbf{cut}(C)$.

We shall write $\Pi_0 = \Pi_0(\mathcal{N})$ to denote an arbitrary complete prefix of the unfolding of \mathcal{N} . It is known ([15, 7]) that the construction of such a complete prefix is indeed possible, and efficient tools such as MOLE ([27]) exist for this purpose. While the precise details of this construction are out of scope for this paper; some ingredients of it will play a role below, so we sketch them here.

Complete prefix scheme. The unfolding is stopped on each branch when some *cutoff event* is added. The criterion for classifying an event e as cutoff is given by Marking equivalence: the marking $\text{Mark}([e])$ that e ‘discovers’ has already been discovered by a *smaller* configuration. Now, the ordering relation \prec to compare two configurations must be an *adequate* order, i.e. $C_1 \subseteq C_2$ must imply $C_1 \prec C_2$, to ensure the completeness of the prefix obtained. As shown in [7], for some choices of \prec , the obtained prefix may be bigger than the reachability graph for some safe nets; however, if \prec is a *total* order, the number of non-cutoff events of the prefix Π_0 thus obtained never exceeds the size of the reachability graph.

We will assume throughout this paper that complete prefixes are computed according to some adequate total order, as is done in particular in the MOLE tool [27]. Below, we will propose a new such order relation that underlies a novel concept of distance between markings.

The nested family $(\Pi_n)_{n \geq 0}$ of finite prefixes. Denote the complete prefix for \mathcal{N} obtained according to definition 6 as Π_0 ; we extend Π_0 to increasing prefixes Π_1, Π_2, \dots as follows. Starting at $n = 0$,

- let $\mathcal{C}^n \triangleq \max(\mathcal{C}(\Pi_n))$,
- set $\mathcal{M}_n \triangleq \{M \in 2^P : \exists C \in \mathcal{C}^n : M = \text{Mark}(C)\}$,
- for all $M \in \mathcal{M}_n$, compute a complete prefix \preceq^M of $\langle N, M \rangle$;
- obtain Π_{n+1} by appending, to every $C \in \mathcal{C}^n$, a copy of $\preceq^{\text{Mark}(C)}$ to every $C \in \mathcal{C}^n$.

3 Doomed configurations, and how to avoid them

3.1 Bad, Free and Doomed Configurations and Markings.

In this section, we present an algorithm that identifies precisely those configurations of a Petri net unfolding from which one can no longer avoid reaching a certain long-term behaviour, its theoretical foundations, and some experimental results. The formal setting here contains and extends the one established in [10], specialized to the 1-safe case. We assume that we are given a set of *bad markings* $\mathcal{Z} \subseteq 2^P$. Since we are interested in long-term behaviours, we assume that \mathcal{Z} is reachability-closed, i.e. $M \in \mathcal{Z}$ and $M \rightarrow M'$ imply $M' \in \mathcal{Z}$.

Define $\mathcal{B} \triangleq \{C \in \mathcal{C}^f : \text{Mark}(C) \in \mathcal{Z}\}$ as the set of *bad configurations*, and let \mathcal{B}_0 be the set of configurations in \mathcal{B} that are contained in Π_0 . $\mathcal{B} \subseteq \mathcal{C}$ is *absorbing* or *upward closed*, that is, for all $C_1 \in \mathcal{B}$ and $C_2 \in \mathcal{C}^f$ such that $C_1 \subseteq C_2$, one must have $C_2 \in \mathcal{B}$.

For any $C \in \mathcal{C}$, let $\Omega_C \triangleq \{\omega \in \Omega : C \subseteq \omega\}$ denote the maximal runs into which C can evolve. We are interested in those finite configurations all of whose maximal extensions are ‘bad’, where we consider infinite configurations as bad if they contain a bad finite configuration. We will call such configurations *doomed*, since from them, the system cannot avoid entering a bad marking sooner or later (and from then on, all reachable markings are bad).

Definition 7 Configuration $C \in \mathcal{C}^f$ is doomed iff

$$\forall \omega \in \Omega_C : \exists C' \in \mathcal{C}^f : \begin{cases} C \subseteq C' \subseteq \omega \\ \wedge \text{Mark}(C') \in \mathcal{Z} \end{cases} \quad (1)$$

The set of doomed configurations is denoted \mathcal{D} ; denote the set of minimal elements in \mathcal{D} by $\check{\mathcal{D}}$. If C is not doomed, it has at least one maximal extension that never reaches bad markings. We call configurations that are not doomed free, and denote the set of free configurations by \mathcal{F} .

All reachable markings are represented by at least one configuration. Moreover, since the future evolution of \mathcal{N} depends only on the current marking, $Mark(C_1) = Mark(C_2)$ for two configurations C_1 and C_2 implies that either both C_1 and C_2 are free, or both are doomed. Therefore, by extension, we call $Mark(C)$ free or doomed whenever C is.

Running Example. In the context of Figures 1a and 2, we consider \mathcal{L} the singleton set containing the marking $M_8 = \{P_8\}$. Clearly, $C_1 \triangleq \{\alpha_1, \gamma_1, \xi_1\}$ and $C_2 = \{\beta_1, \delta_1, \eta_1\}$ satisfy $Mark(C_1) = Mark(C_2) = M_8$ and therefore $C_1, C_2 \in \mathcal{B}$. But note that $C'_1 \triangleq \{\alpha_1, \gamma_1\}$ and $C'_2 = \{\beta_1, \delta_1\}$ produce markings outside \mathcal{L} , but they are doomed since any extension of these configurations leads into \mathcal{L} . Therefore, $C'_1, C'_2 \in \mathcal{B}$. On the other hand, \emptyset is free, as well as $\{\beta_1, \gamma_1\}$, $\{\alpha_1, \delta_1\}$, etc. We note in passing that the Petri net in Fig 1a allows to refine the understanding of the ‘tipping point’ by showing that doom is not brought about by a single transition but rather the combined effect of two independent choices; this fact is obscured, or at least far from obvious, in the state graph shown in Figure 1b.

Identifying free and doomed configurations belongs to the core objectives of this paper. In a first step towards that, Theorem 1 below uses a similar proof idea as Lemma 8 in [12] in the context of fault diagnosis. Let us first recall the notion of *spoilers*, introduced in [12]:

Definition 8 A spoiler of transition t (or event e) is any $t' \in T$ ($e' \in E$) such that $\bullet t' \cap \bullet t \neq \emptyset$ ($\bullet e' \cap \bullet e \neq \emptyset$). We write $\mathbf{spoil}(t)$ ($\mathbf{spoil}(e)$) for the set of t 's (e 's) spoilers.

Note that $t \in \mathbf{spoil}(t)$ for all $t \in T$. The spoilers of t are characterized by the fact that their firing cancels any enabling of t ; that is, by being either in conflict with t , or identical with t .

Theorem 1 A configuration $C \in \mathcal{C}^f$ is **free** iff either a) there exists a finite maximal configuration C' such that $C \subseteq C' \notin \mathcal{B}$, or b) there exist configurations $C_1, C_2 \in \mathcal{C}^f$ such that

1. $C \subseteq C_1 \subseteq C_2 \notin \mathcal{B}$;
2. $Mark(C_1) = Mark(C_2)$;
3. for all events $e \in E$ such that $C_1 \xrightarrow{e}$, one has $\mathbf{spoil}(e) \cap C_2 \neq \emptyset$.

Some comments are in order before giving the proof of Theorem 1. First of all, the requirement to check whether $C_2 \notin \mathcal{B}$ can be met by checking whether $Mark(C_2) \in \check{\mathcal{L}}$. Second, the spoiling condition (3) ensures that the process that takes C_1 to C_2 forms a loop whose iteration yields a run.

Proof: In the following, let $M := Mark(C_1)$. We first prove the right-to-left implication. Case a) is obvious, so assume that b) holds. Let $C_2 = C_1 \oplus \hat{C}$; then by 2., we can append \hat{C} to C_2 , yielding a strictly increasing sequence of configurations $(C_n)_{n \in \mathbb{N}}$ such that $C_{n+1} = C_n \oplus \hat{C}$, and $Mark(C_n) = M$ for all n . By property 3, we know that \hat{C} contains spoilers for all its initially enabled events, hence no transition remains enabled forever, and $\omega \triangleq \bigcup_{n \in \mathbb{N}} C_n$ is a maximal configuration. It remains to show that ω contains no bad configuration: Suppose that there is $C' \subseteq \omega$ with $C' \in \mathcal{B}$. Since C' is finite, we have $C' \subseteq C_n$ for some n . But then, $M = Mark(C_n)$ is reachable from $Mark(C') \in \check{\mathcal{L}}$, contradicting our assumptions. Thus ω never enters a bad state, and C is free.

For the forward implication, assume that C is free. Then there exists $\omega \in \Omega_C$ such that $C' \notin \mathcal{B}$ for all finite $C' \subseteq \omega$. If this ω can be chosen finite, then a) holds and we are done; so assume henceforth that ω must be chosen infinite. Clearly, there must exist a reachable marking M that is visited an infinite number of times by a family of nested finite configurations $(C^n)_{n \in \mathbb{N}}$ such that $\bigcup_{n \in \mathbb{N}} C^n = \omega$. Let $C_1 \triangleq C^1$

and $E' := \{e : C_1 \xrightarrow{e}\}$. Let K be the smallest index such that for all $e \in E'$, one has $\mathbf{spoil}(e) \cap C^K \neq \emptyset$; such a K must exist since ω is maximal. Then C_1 and $C_2 \triangleq C^K$ have the required properties. \square

Notice that the proof could be restructured by observing that case a) of Theorem 1 is indeed a special instance of case b). In fact, taking $C_1 \triangleq C_2 \triangleq C'$ with C' according to case a), conditions 1 and 2 of part b) are obviously satisfied, and condition 3 holds vacuously since no event is enabled in $C_1 = C'$. We note in passing that this observation is helpful in simplifying the implementation used for the experiments below.

The interest of Theorem 1 lies in the following fact:

Lemma 2 For $C \in \mathcal{C}^f$, checking whether C is free can be done using finite prefix Π_1 of $\mathcal{U}(N, \text{Mark}(C))$.

Proof: If C is free, let C_1 and C_2 be the configurations witnessing this fact from Theorem 1, and let $M := \text{Mark}(C_1) = \text{Mark}(C_2)$. If such configurations exist, then C_1 can be chosen from the complete prefix Π_0 , and C_2 can be chosen from Π_1 , notably in the copy of $\Pi_{\text{Mark}(C_1)}$ appended after C_1 .

Checking whether the configuration C_2 thus found is in \mathcal{B} is immediate, since it suffices to check whether its marking is in \mathcal{Z} , using the fact that \mathcal{Z} is reachability-closed. To check the spoiler condition (3) of Theorem 1, it suffices to check whether the conditions of the cut of C_1 that are not consumed by C_2 enable some event. \square

3.2 Finding Minimally Doomed Configurations: Algorithm MINDOO

Shaving and Rubbing. Let us start by observing that \mathcal{B} , an upward closed set by construction, also has some downward closure properties, meaning one can restrict control to act on ‘small’ configurations.

Definition 9 An event e is unchallenged iff there is no e' such that $e \#_\delta e'$, i.e. $(\bullet e)^\bullet = \{e\}$.

Lemma 3 Let $C \in \mathcal{C}^f$ and $e \in \mathbf{crest}(C)$ unchallenged; set $C' \triangleq C \setminus \{e\}$. Then $C' \in \mathcal{C}^f$, and $\Omega_C = \Omega_{C'}$.

Proof: $C' \in \mathcal{C}^f$ holds by construction. Also, $\Omega_C \subseteq \Omega_{C'}$ follows from $C' \subseteq C$; it remains to show the reverse inclusion. Assume there exists $\omega \in \Omega_{C'} \setminus \Omega_C$; then $C' \setminus \omega = \{e\}$, and $\langle e \rangle \subseteq \omega$. By maximality, ω must contain some e' such that $e \# e'$. Then by definition, there are events $u \neq v$, $u \leq e$, $v \leq e'$, and $u \#_\delta v$. In particular, $u \# e'$, and since $\{e'\} \cup \langle e \rangle \subseteq \omega$, this implies $u = e$. But e is unchallenged, so v cannot exist, and neither can ω . \square

Definition 10 A configuration $C \in \mathcal{C}^f$ such that $\mathbf{crest}(C)$ contains no unchallenged event is called shaved.

Clearly, every $C \in \mathcal{C}^f$ contains a unique maximal shaved configuration, which we call $\mathbf{shave}(C)$; it can be obtained from C by recursively ‘shaving away’ any unchallenged $e \in \mathbf{crest}(C)$, and then continuing with the new crest, until no unchallenged events remain.

Example. In the context of Figure 3, for $C_1 = \{x, y, z\}$ and $C_2 = C_1 \cup \{\beta, \gamma, u\}$, one has $\mathbf{shave}(C_1) = \emptyset$ since x , y , and z are unchallenged, and $\mathbf{shave}(C_2) = C_1 \cup \{\beta, \gamma\}$ since u is unchallenged but neither β nor γ are. Note that in the unfolding of the running example shown in Figure 2, the κ -labeled events are the only unchallenged ones.

As a consequence of Lemma 3, any $C \in \mathcal{C}^f$ is in \mathcal{B} iff $\mathbf{shave}(C)$ is. Still, it may be possible that such a $\mathbf{shave}(C)$ can still be reduced further by removing some of its crest events. This would be the case, e.g., if two conflicting events both lead to a bad state. Thus, given a crest event e , we test whether $C \setminus \{e\}$ is free (e.g. because some event in conflict with e may allow to move away from doom) or still doomed.

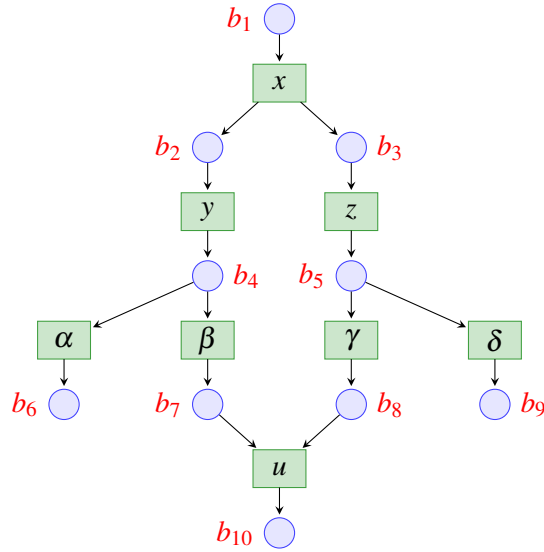


Figure 3: An occurrence net. With $C \triangleq \{x, y, z, \beta, \gamma\}$ and $C' \triangleq C \cup \{u\}$, suppose $\mathcal{Z} = \{\text{Mark}(C')\} = \pi(\{b_{10}\})$. Then $\text{shave}(C') = C$, and C is doomed. Moreover, $C \in \check{\mathcal{D}}$ since both $C_4 \triangleq C \setminus \{\beta\}$ and $C_5 \triangleq C \setminus \{\gamma\}$ are free.

If the latter is the case, then C was not minimally doomed, and analysis continues with $C \setminus \{e\}$ (we say that we ‘rub away’ e). If $C \setminus \{e\}$ is free, we leave e in place and test the remaining events from $\text{crest}(C)$. A configuration that is shaved and from which no event can be rubbed away is minimally doomed.

Algorithm 1 uses a ‘worklist’ set wl of doomed, shaved configurations to be explored; wl is modified when a configuration is replaced by a set of rubbed (and again, shaved) versions of itself, or when a configuration C is identified as minimally doomed, in which case it is removed from wl and added to \mathcal{D} .

Every branch stops when a minimally doomed configuration is reached, i.e., a doomed configuration C such by rubbing off any crest event e from C makes it free, i.e. $C \setminus \{e\}$ is free for all $e \in \text{crest}(C)$. When the worklist is empty, all minimally doomed configurations have been collected in \mathcal{D} . Note that if $\emptyset \in wl$ at any stage during the execution of Algorithm MINDOO, then \emptyset will be added to \mathcal{D} , since MINDOO will not enter the second **foreach**-loop in that case. In fact, if this situation arises, *every* configuration is doomed, and thus \emptyset is the unique minimally doomed configuration.

The configurations produced in the course of the search strictly decrease w.r.t both size and inclusion. Moreover, an upper bound on the prefixes explored at each step is given by \mathcal{B} , itself strictly contained in the complete finite prefix used to find all bad markings. According to [7], this prefix can be chosen of size equal or smaller (typically: considerably smaller) than the reachability graph of \mathcal{N} .

Theorem 2 *For any safe Petri net $\mathcal{N} = \langle N, M_0 \rangle$ and bad states set $\mathcal{Z} \subseteq \mathbf{R}_N(M_0)$, Algorithm MINDOO terminates, with output set \mathcal{D} containing exactly all minimal doomed configurations, i.e. $\mathcal{D} = \check{\mathcal{D}}$.*

Proof: Termination follows from the finiteness of $\min_{\subseteq}(\mathcal{B}_0)$, since in each round of MINDOO there is one configuration C that is either replaced by a set of strict prefixes or removed from wl . Therefore, after a finite number of steps wl is empty. According to Lemma 2, the status (doomed or free) of a given finite configuration can effectively be checked on a fixed finite prefix of \mathcal{U} . Assume that after termination of MINDOO, one has $C \in \mathcal{D}$; we need to show $C \in \check{\mathcal{D}}$. Clearly, when C was added to \mathcal{D} , it had been

Algorithm 1: Algorithm MINDOO

```

Data: Safe Petri Net  $\mathcal{N} = \langle P, T, F, M_0 \rangle$  and  $\mathcal{L} \subseteq 2^P$ 
Result: The set  $\mathcal{D}$  of  $\mathcal{N}$ 's  $\subseteq$ -minimal doomed configurations
 $\mathcal{D} \leftarrow \emptyset$ ;  $wl \leftarrow \emptyset$ ;
foreach  $C \in \min_{\subseteq}(\mathcal{B}_0)$  do
  |  $C' \leftarrow \text{shave}(C)$ ;
  |  $wl \leftarrow wl \cup \{C'\}$ ;
end
while  $wl \neq \emptyset$  do
  | Pick  $C \in wl$ ;  $\text{add} \leftarrow \text{true}$ ;
  | if  $(C \setminus \text{crest}(C))$  is doomed then
  |   |  $\text{add} \leftarrow \text{false}$ ;
  |   |  $C' \leftarrow \text{shave}(C \setminus \text{crest}(C))$ ;
  |   |  $wl \leftarrow wl \cup \{C'\}$ ;
  | else
  |   | foreach  $e \in \text{crest}(C)$  do
  |     | if  $(C \setminus \{e\})$  is doomed then
  |       |  $\text{add} \leftarrow \text{false}$ ;
  |       |  $C' \leftarrow \text{shave}(C \setminus \{e\})$ ;
  |       |  $wl \leftarrow wl \cup \{C'\}$ ;
  |     | end
  |   | end
  | end
  |  $wl \leftarrow wl \setminus \{C\}$ ;
  | if  $\text{add}$  then
  |   |  $\mathcal{D} \leftarrow \mathcal{D} \cup \{C\}$ ;
  | end
end
return  $\mathcal{D}$ 

```

detected as doomed; it remains to show that C is also minimal with this property. Assume that there is $C' \subsetneq C$ that is as doomed as well. But in that case there exists $e \in \text{crest}(C)$ such that $C' \subseteq (C \setminus \{e\}) \subsetneq C$, which implies that this $(C \setminus \{e\})$ is doomed as well. But then add has been set to false in the second **foreach**-loop, before C could have been added to \mathcal{D} .

Conversely, let $C \in \mathcal{D}$. Then $(C \setminus \{e\})$ is free for all $e \in \text{crest}(C)$; the variable add remains thus at the value true because no round of the second **foreach**-loop can flip it. Thus C is added to \mathcal{D} , from which MINDOO never removes any configuration. \square

3.3 Implementation and Experiments.

A prototype implementation of MINDOO is available at [21]. It takes as input a safe Petri net in the PEP format and relies on MOLE [27] for computing the initial finite prefix Π_0 and its extensions. Algorithm 1 is implemented in Python, where the identification of maximal configurations, bad configurations, as well as the verification of doomed status of a configuration is performed in Answer-Set Programming

Table 1: Statistics of Algorithm 1 on Petri net models of biological systems. The size of Π_0 and Π_1 is the number of their events; “# min doomed cfg” is the number of minimally doomed configurations; “# doom checks” is the number of SAT checks for doom status of a configuration. “time” is the total computation time on a 1.8Ghz CPU

Model	size Π_0	size Π_1	# min doomed cfg	# doom checks	time
Lambda switch	126	1,060	10	29	1s
Cell death receptor	791	19,262	57	407	37s
Budding yeast cell cycle	1,413	184,363	114	837	8m3s

(ASP) employing the CLINGO solver [9], a logic programming technology close to SAT solving.

We illustrate in Table 1 the behavior of the implementation on different instances of Petri nets modeling biological processes. In each case, we report the size (number of events) of prefixes Π_0 and Π_1 (including cut-off events), the number of minimally doomed configurations, and the number of configurations which have been tested for being doomed. The purpose of the conducted experiments was to study the tractability of our approach on literature models of biological systems for which the study of doomed configuration was relevant. As exhibited in [3], one of the first potential bottleneck is the tractability of the computation of the finite complete prefix Π_0 and the enumeration of maximal configurations, which is required for computing Π_1 . Then, our experiments have focused on assessing how evolved the number of minimally doomed configurations, the number of candidate configurations screened by Algorithm 1, and the overall computation time, with different sizes of prefixes Π_1 .

We selected 3 models published as Boolean networks, which can be translated as safe Petri nets using the encoding described in [3] implemented in the tool PINT [22]. The “Lambda switch” model [28] comprises 11 places and 41 transitions, and possesses two limit behaviors, one being a deadlock, marked as a bad marking. The “Cell death receptor” model [2] comprises 22 places and 33 transitions, and reproduces a bifurcation process into different cell fates, one of which has been declared as bad (apoptosis). In these two cases, the minimally doomed configurations identify configurations in which a decisive event has just taken place, committing the system to the attractor marked as bad. The “Budding yeast cell cycle” model [19] comprises 18 places and 32 transitions, and represents the oscillation of gene activity during the cell cycle. In this model, the cycle can exit and eventually reach a marking corresponding to all genes being inactive, which is our bad marking. In this later case, the minimally doomed configurations identify precisely when the system exits its oscillatory behavior.

It appears that the computation time for identifying minimally doomed configurations seems mostly affected by the size of Π_1 for the verification of the doom property of a configuration by ASP solving, implementing the conditions of Theorem 1. In each case, the number of minimally doomed configurations is a fraction of the size of the finite complete prefix Π_0 . Future work may explore compact representations of the set of minimally doomed configurations, as they typically share a large amount of events, and may ease biological interpretations.

4 Protectedness

4.1 Cliff-Edges and Ridges.

From the minimal doomed configurations, we derive the critical ‘points’ at which a run becomes doomed:

Definition 11 *An event set $\gamma \subseteq E$ is called a cliff-edge iff there exists a minimally doomed configuration*

$C \in \check{\mathcal{D}}$ such that $\gamma = \mathbf{crest}(C)$. The set of cliff-edges is denoted Γ . The folding $\chi \triangleq \pi(\gamma) \subseteq T$ of a cliff-edge γ is called a ridge.

To complete the map of the evolutionary landscape for \mathcal{N} , it is important to find, in a bounded prefix of the unfolding, all ridges that determine the viability of a trajectory. Notice that the completeness of prefix Π_0 only guarantees that all reachable *markings* of \mathcal{N} are represented by at least one configuration of Π_0 ; this does not extend to a guarantee that all concurrent steps that lead into a doomed marking can be found in Π_0 as well. Fortunately, one has:

Lemma 4 *For every ridge χ of \mathcal{N} there is a witness in Π_0 , i.e. there exists a minimally doomed configuration C in Π_1 such that $\pi(\mathbf{crest}(C)) = \chi$.*

Proof: Fix χ , and let C_χ be any configuration such that $\pi(\mathbf{crest}(C_\chi)) = \chi$; set $M^C \triangleq \mathbf{Mark}(C_\chi)$, and let M_χ^C the unique reachable marking such that $M_\chi^C \xrightarrow{\chi} M^C$. Then any such M_χ^C is represented by some C^χ in Π_0 . By construction, there exists a cliff-edge γ such that $C^\chi \xrightarrow{\gamma}$ and $\pi(\gamma) = \chi$. Then $C \triangleq C^\chi \cup \gamma$ is a minimally doomed configuration that lies within Π_1 . \square

4.2 Measuring the Distance from Doom

With the above, we have the tools to draw a map of the ‘landscape’ in which the system evolves, with doomed zones and cliff-edges highlighted. What we wish to add now is to assist *navigation* in this landscape: we intend to give a meaningful measure of how well, or badly, a current system state is protected against falling from a cliff-edge. We chose to measure this distance not in terms of the *length* of paths, or similar notions, but rather in terms of the *choices* that are made by the system in following a particular path.

Consider a configuration C and the nonsequential process that it represents. Some of the events in C can be seen as representing a *decision*, in the sense that their occurrence took place in conflict with some event that was enabled by some prefix of C . The number of such events gives a measure of the information contained in C , in terms of the decisions necessary to obtain C :

Definition 12 *Let $C \in \mathcal{C}^\mathcal{A}$, and define*

$$\mathbf{dech}(C) \triangleq |\{e \in C : \exists e' \in E : e \#_\sigma^C e'\}|,$$

where $\#_\sigma$ is the strict C -conflict relation defined, for all $e \in C$, by

$$e \#_\sigma^C e' \iff e \#_\delta e' \wedge \langle e' \rangle \subseteq C.$$

$\mathbf{dech}(C)$ is called the decisional height of C .

In Figure 2, the configuration $C_1 = \{\xi_1, \alpha_1, \gamma_1\}$ satisfies $\mathbf{dech}(C_1) = 2$, whereas for $C_0 = \{\beta_1\}$, one has $\mathbf{dech}(C_0) = 1$.

Note that $\#_\sigma^C$ is more restrictive than direct conflict $\#_\delta$; it is also more restrictive than the *immediate conflict* in the literature (e.g. [1]). It is closely dependent on the configuration C under study, and describes precisely those events *against* which the process had to decide in performing C .

Now, for any free marking M (or, equivalently, any free configuration C such that $\mathbf{Mark}(C) = M$), we wish to measure the threat represented by doomed markings reachable from M : how far away from doom is the system when it is in M ? Using the decisional height introduced above, we can define a height difference in terms of the conflicts that lead from one marking to another:

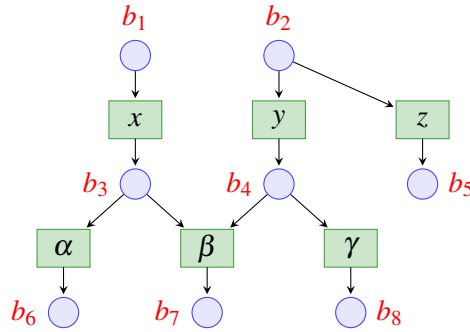


Figure 4: Illustration of direct conflict.

Definition 13 For $C \in \mathcal{C}^{\dagger}$, let

$$\check{\mathcal{D}}_C \triangleq \begin{cases} \{C' \in \check{\mathcal{D}} : C \subseteq C'\} & : C \in \mathcal{F} \\ \{C\} & : C \in \mathcal{B} \end{cases} \quad (2)$$

The protectedness of C is then

$$\mathbf{prot}(C) \triangleq \min_{C' \in \check{\mathcal{D}}_C} \{\mathbf{dech}(C' \otimes C)\} \quad (3)$$

In Figure 3, with the definitions introduced there, $\mathbf{prot}(C) = \mathbf{prot}(C') = 0$. Setting $C_1 \triangleq \{x\}$, $C_2 \triangleq \{x, y\}$, $C_3 \triangleq \{x, z\}$, $C_4 \triangleq C_2 \cup C_3$, $C_5 \triangleq C_4 \cup \{\beta\}$, and $C_6 \triangleq C_4 \cup \{\gamma\}$, one further has

$$\begin{aligned} \mathbf{prot}(C_1) = \mathbf{prot}(C_2) = \mathbf{prot}(C_3) &= 2 \\ \mathbf{prot}(C_4) = \mathbf{prot}(C_5) &= 1. \end{aligned}$$

Returning to Figure 4, suppose that $C' = \{x, y, \beta\}$ is the only minimally doomed configuration. Then for $C = \{x, z, \alpha\}$ as above, we have $\mathbf{prot}(C) = 1$, because the only direct conflict here is the one between z and y .

Note that the definition of protectedness is parametrized by the choice of conflict relation in computing $\mathbf{dech}(\bullet)$. Using direct conflict instead of strict conflict would increase $\mathbf{dech}(\bullet)$ and lead to an overevaluation of protectedness.

To see the point, consider the occurrence net in Figure 4. Let $C_\alpha = \{x, z, \alpha\}$, $C_\beta = \{x, y, \beta\}$ and $C_\gamma = \{x, y, \alpha, \gamma\}$. We have $\mathbf{dech}(C_\alpha) = 1$, $\mathbf{dech}(C_\beta) = 3$ and $\mathbf{dech}(C_\gamma) = 2$. Were $\#_\sigma$ replaced by $\#_\delta$ in the computation of $\mathbf{dech}(\bullet)$, these values would not change *except* for C_α where it would change to 2. As a result, if $C \in \check{\mathcal{D}}$, the protectedness of the empty configuration would be evaluated as 2, whereas by our definition $\mathbf{prot}(\emptyset) = 1$. Indeed, \emptyset is *just one wrong decision away from doom*, and this is what protectness is meant to express.

4.3 Computing Protectedness is Feasible

Computation of $\mathbf{prot}(\bullet)$ does not require any larger data structure than those already required for computing $\check{\mathcal{D}}$ according to Lemma 2:

Lemma 5 *There is a complete prefix scheme producing a complete prefix Π_0 whose size is bounded by the number of reachable markings, and such that for every finite configuration C , $\mathbf{prot}(C)$ can be computed on $\Pi_0(\text{Mark}(C))$.*

Proof: If $\check{\mathcal{J}} \cap \mathcal{C}(\Pi_0) = \emptyset$, then all extensions of C are free, and we are done. Otherwise, the crucial step is to find an adequate *total* order \prec on finite configurations, that ensures that Π_0 contains at least one minimally doomed configuration that minimizes $\mathbf{dech}(\bullet)$ over all minimally doomed configurations in $\mathcal{U}(\text{Mark}(C))$. The following order \prec is obtained by modifying the total order \prec_F introduced in [7], Def. 6.2.: For $C_1, C_2 \in \mathcal{C}^f$, write $C_1 \prec C_2$ iff either

- $\mathbf{dech}(C_1) < \mathbf{dech}(C_2)$, or
- $\mathbf{dech}(C_1) = \mathbf{dech}(C_2)$ and $C_1 \ll C_2$, or
- $\mathbf{dech}(C_1) = \mathbf{dech}(C_2)$ and $C_1 \equiv C_2$, and $FC(C_1) \ll FC(C_2)$,

where $\ll (\equiv)$ denote lexicographic ordering (lexicographic equivalence) wrt some total ordering of the transition set T , and FC denotes Cartier-Foata normal form. The proof of Theorem 6.4. of [7] extends immediately, proving that \prec is an adequate total order; therefore, Lemma 5.3. of [7] applies, hence any complete prefix Π_0^\prec obtained via the scheme using \prec is bounded in size by the reachability graph. Now, let \mathcal{C}^* be the set of configurations from $\check{\mathcal{J}}(\text{Mark}(C))$ that minimize $\mathbf{dech}(\bullet)$; by construction of \prec , one has $\mathcal{C}^* \cap \mathcal{C}(\Pi_0^\prec) \neq \emptyset$. \square

5 Discussion

The results presented here give a toolkit for the analysis of tipping situations in a safe Petri net, i.e. when and how a basin boundary is crossed; an algorithmic method for finding minimally doomed configuration has been developed, implemented and tested.

Moreover, we have introduced a measure of *protectedness* that indicates the number of *decisions* that separate a free state from doom. It uses an intrinsic notion of decisional height that allows to warn about impending dangerous scenarios; at the same time, this height is also 'natural' for unfoldings, in the sense that it induces an adequate linear order that allows to compute complete prefixes of bounded size.

On a more general level, the results here are part of a broader effort to provide a discrete, Petri-net based framework for dynamical systems analysis in the life sciences. The applications that we target lie in systems biology and ecology.

Future work will investigate possibilities for Doom Avoidance Control, i.e. devising strategies that allow to steer away from doom; we expect to complement the existing approaches via structural methods of e.g. Antsaklis et al [13], and also the unfolding construction of Giua and Xie [10]. A crucial question is the knowledge that any control player can be assumed to have, as a basis for choosing control actions. We believe the protectedness measure is a valid candidate for coding this information, so that a controller may take action when the system is too close to doom (wrt some thresholds to be calibrated) but there still remain decisions that can be taken to avoid it. Evaluating this option, along with other approaches, must, however, be left to future work.

References

- [1] Samy Abbes & Albert Benveniste (2006): *Probabilistic models for true-concurrency: branching cells and distributed probabilities for event structures*. *Information and Computation* 204(2), pp. 231–274, doi:10.1016/j.ic.2005.10.001.

- [2] Laurence Calzone, Laurent Tournier, Simon Fourquet, Denis Thieffry, Boris Zhivotovsky, Emmanuel Barillot & Andrei Zinovyev (2010): *Mathematical Modelling of Cell-Fate Decision in Response to Death Receptor Engagement*. *PLoS Computational Biology* 6(3), p. e1000702, doi:10.1371/journal.pcbi.1000702.
- [3] Thomas Chatain, Stefan Haar, Loïc Jezequel, Loïc Paulevé & Stefan Schwoon (2014): *Characterization of Reachable Attractors Using Petri Net Unfoldings*. In Pedro Mendes, editor: *Proceedings of the 12th Conference on Computational Methods in System Biology (CMSB'14)*, *Lecture Notes in Bioinformatics* 8859, Springer-Verlag, Manchester, UK, pp. 129–142, doi:10.1007/978-3-319-12982-2_10.
- [4] Thomas Chatain, Stefan Haar, Juraj Kolcák, Loïc Paulevé & Aalok Thakkar (2020): *Concurrency in Boolean networks*. *Nat. Comput.* 19(1), pp. 91–109, doi:10.1007/s11047-019-09748-4.
- [5] David P. A. Cohen, Loredana Martignetti, Sylvie Robine, Emmanuel Barillot, Andrei Zinovyev & Laurence Calzone (2015): *Mathematical Modelling of Molecular Pathways Enabling Tumour Cell Invasion and Migration*. *PLoS Comput Biol* 11(11), p. e1004571, doi:10.1371/journal.pcbi.1004571.
- [6] J. Esparza & K. Heljanko (2008): *Unfoldings – A Partial-Order Approach to Model Checking*. Springer. ISBN: 978-3-540-77426-6.
- [7] J. Esparza, S. Römer & W. Vogler (2002): *An Improvement of McMillan's Unfolding Algorithm*. *FMSD* 20, pp. 285–310, doi:10.1023/A:1014746130920.
- [8] Louis Fippo Fitime, Olivier Roux, Carito Guziolowski & Loïc Paulevé (2017): *Identification of bifurcation transitions in biological regulatory networks using Answer-Set Programming*. *Algorithms for Molecular Biology* 12(1), p. 19, doi:10.1186/s13015-017-0110-3.
- [9] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2014): *Clingo = ASP + Control: Preliminary Report*. *CoRR* abs/1405.3694, doi:10.48550/arXiv.1405.3694.
- [10] A. Giua & X. Xie (2005): *Control of safe ordinary Petri nets using unfolding*. *Discrete Event Dynamic Systems* 15(4), pp. 349–373, doi:10.1007/s10626-005-4057-z.
- [11] Stefan Haar, Loïc Paulevé & Stefan Schwoon (2020): *Drawing the Line: Basin Boundaries in Safe Petri Nets*. In Alessandro Abate, Tatjana Petrov & Verena Wolf, editors: *Proc.18th Conf. on Computational Methods in System Biology (CMSB'20)*, *Lecture Notes in Bioinformatics* 12314, Springer, pp. 321–336, doi:10.1007/978-3-030-60327-4_17.
- [12] Stefan Haar, César Rodríguez & Stefan Schwoon (2013): *Reveal Your Faults: It's Only Fair!* In Marta Pietkiewicz-Koutny & Mihai Teodor Lazarescu, editors: *Proc. 13th Int. Conf. on Application of Concurrency to System Design (ACSD'13)*, IEEE Computer Society Press, Barcelona, Spain, pp. 120–129, doi:10.1109/ACSD.2013.15.
- [13] Marian V. Iordache & Panos J. Antsaklis (2006): *Supervisory Control of Concurrent Systems: A Petri Net Structural Approach*. Birkhäuser, Boston, Basel, Berlin.
- [14] H. Klarner, H. Siebert, S. Nee & F. Heintz (2018): *Basins of Attraction, Commitment Sets and Phenotypes of Boolean Networks*. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, doi:10.1109/TCBB.2018.2879097.
- [15] K. L. McMillan (1992): *Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits*. In: *CAV*, pp. 164–177, doi:10.1007/3-540-56496-9_14.
- [16] Nuno D. Mendes, Rui Henriques, Elisabeth Remy, Jorge Carneiro, Pedro T. Monteiro & Claudine Chauviya (2018): *Estimating Attractor Reachability in Asynchronous Logical Models*. *Frontiers in Physiology* 9, doi:10.3389/fphys.2018.01161.
- [17] T. Murata (1989): *Petri nets: Properties, analysis and applications*. *Proc. of the IEEE* 77(4), pp. 541–580, doi:10.1109/5.24143.
- [18] M. Nielsen, G. D. Plotkin & G. Winskel (1979): *Petri Nets, Event Structures and Domains*. In: *SCC*, pp. 266–284, doi:10.1016/0304-3975(81)90112-2.

- [19] David A. Orlando, Charles Y. Lin, Allister Bernard, Jean Y. Wang, Joshua E. S. Socolar, Edwin S. Iversen, Alexander J. Hartemink & Steven B. Haase (2008): *Global control of cell-cycle transcription by coupled CDK and network oscillators*. *Nature* 453(7197), pp. 944–947, doi:10.1038/nature06955.
- [20] Ertugrul M. Ozbudak, Mukund Thattai, Han N. Lim, Boris I. Shraiman & Alexander van Oudenaarden (2004): *Multistability in the lactose utilization network of Escherichia coli*. *Nature* 427(6976), pp. 737–740, doi:10.1038/nature02298.
- [21] Loïc Paulevé: *Implementation of the search for minimal doomed configurations*. Available at <https://gitub.u-bordeaux.fr/lpauleve/doomed-configurations>.
- [22] Loïc Paulevé (2017): *Pint: a static analyzer for transient dynamics of qualitative networks with IPython interface*. In: *CMSB 2017 - 15th conference on Computational Methods for Systems Biology, Lecture Notes in Computer Science* 10545, Springer International Publishing, pp. 309–316, doi:10.1007/978-3-319-67471-1_20.
- [23] Alexander N. Pisarchik & Ulrike Feudel (2014): *Control of multistability*. *Physics Reports* 540(4), pp. 167–218, doi:10.1016/j.physrep.2014.02.007.
- [24] Erik Plahte, Thomas Mestl & Stig W. Omholt (1995): *Feedback Loops, Stability and Multistationarity in Dynamical Systems*. *J. Biol. Syst.* 03(02), pp. 409–413, doi:10.1142/s0218339095000381.
- [25] Franck Pommereau, Colin Thomas & Cédric Gaucherel (2022): *Petri Nets Semantics of Reaction Rules (RR), a Language for Ecosystems Modelling*. In: *Proc. 43rd Int. Conf. on Application and Theory of Petri Nets and Concurrency*, Bergen, Norway, pp. 1–20, doi:10.1007/978-3-031-06653-5_10.
- [26] Adrien Richard (2019): *Positive and negative cycles in Boolean networks*. *Journal of Theoretical Biology* 463, pp. 67–76, doi:10.1016/j.jtbi.2018.11.028.
- [27] S. Schwoon (2014): *The MOLE Tool*. URL: <http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/>.
- [28] Denis Thieffry & René Thomas (1995): *Dynamical behaviour of biological regulatory networks—II. Immunity control in bacteriophage lambda*. *Bulletin of Mathematical Biology* 57, pp. 277–297, doi:10.1007/BF02460619.
- [29] R. Thomas (1980): *On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations*. *Springer Series in Synergies* 9, pp. 180–193, doi:10.1007/978-3-642-81703-8_24.
- [30] René Thomas & Richard d’Ari (1990): *Biological Feedback*. CRC Press, Boca Raton, Florida, USA.

Comparing Channel Restrictions of Communicating State Machines, High-level Message Sequence Charts, and Multiparty Session Types

Felix Stutz Damien Zufferey

MPI-SWS, Kaiserslautern, Germany

{fstutz,zufferey}@mpi-sws.org

Communicating state machines provide a formal foundation for distributed computation. Unfortunately, they are Turing-complete and, thus, challenging to analyse. In this paper, we classify restrictions on channels which have been proposed to work around the undecidability of verification questions. We compare half-duplex communication, existential B -boundedness, and k -synchronisability. These restrictions do not prevent the communication channels from growing arbitrarily large but still restrict the power of the model. Each restriction gives rise to a set of languages so, for every pair of restrictions, we check whether one subsumes the other or if they are incomparable. We investigate their relationship in two different contexts: first, the one of communicating state machines, and, second, the one of communication protocol specifications using high-level message sequence charts. Surprisingly, these two contexts yield different conclusions. In addition, we integrate multiparty session types, another approach to specify communication protocols, into our classification. We show that multiparty session type languages are half-duplex, existentially 1-bounded, and 1-synchronisable. To show this result, we provide the first formal embedding of multiparty session types into high-level message sequence charts.

Acknowledgements and Funding. The authors would like to thank Emanuele D’Osualdo, Georg Zetsche and the anonymous reviewers for their feedback and suggestions. This research was funded in part by the Deutsche Forschungsgemeinschaft project 389792660-TRR 248.

Extended Version: <http://arxiv.org/abs/2208.05559>

1 Introduction

Communicating state machines (CSMs) are one of the foundational models of message-passing concurrency. Unfortunately, the combination of multiple processes and unbounded FIFO channels yields a Turing-complete model of computation even when the processes are finite-state [14]. The communication channels can be used as memory and, therefore, most verification questions for CSMs are not algorithmically solvable. To regain decidability, one needs to exploit properties of specific systems. For instance, if all the runs of some communicating state machine use finite memory, it is possible to verify this system. This restriction, known as universal boundedness [24], admits only systems with finitely many reachable states.

In this paper, we compare three channel restrictions which allow infinite state systems while making interesting verification questions decidable. We compare half-duplex communication [17], existential B -boundedness [24], and k -synchronisability [13, 28].

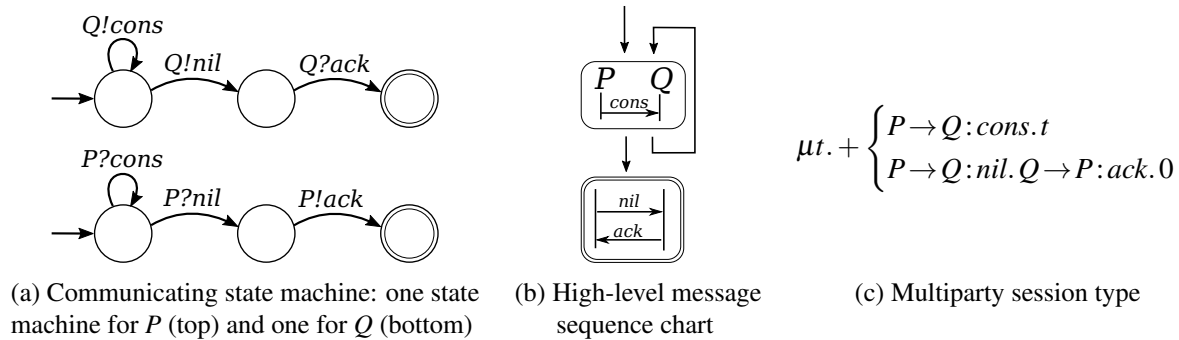


Figure 1: Sending a list expressed in different formalisms. The left part is an implementation of the protocol specified in the middle and right parts.

We explain all three restrictions with the CSM in Fig. 1a. There, a process P sends a list, element by element, to a process Q . After receiving the list's end, Q sends an acknowledgement back to P .

Half-duplex communication requires that, at all times, at least one of both channels between two processes is empty. While P sends the list, the channel can grow arbitrarily large. However, Q always receives all the messages until nil before replying. When Q replies, the channel from P to Q is empty. Hence, the CSM is half-duplex.

Existential B -boundedness means that, for every execution, we can reorder the sends and receptions such that the channels carry at most B messages. This CSM is existentially 1-bounded. Each reception is possible directly after the send.

k -synchronisability requires that every execution can be reordered and split into phases where up to k messages are first sent and then received. This CSM is 1-synchronisable because every message can be received directly after it was sent.

The original definitions of channel restrictions are phrased in terms of executions of a CSM. We present a characterisation for each restriction which only considers the generated language. This also allows us to reason about languages specified or generated in different ways. We consider languages given by protocol specifications and implementations. For implementations, we consider *CSM-definable languages*, i.e., languages which can be generated by a CSM.

Interestingly, for CSMs, these channel restrictions have not yet been compared thoroughly. In this paper, we close this gap and provide a classification of channel restrictions for CSM-definable languages. For instance, this answers a question for the FIFO point-to-point setting which has been posed for the mailbox setting by Bouajjani et al. [13] as we prove that existential B -boundedness and k -synchronisability are incomparable for CSM-definable languages. Overall, we give examples for every possible intersection and, thus, prove that none of the restrictions subsumes another one in this context. Our results for CSM-definable languages are summarised in Fig. 2a. In fact, we disprove one of the three known results from the literature [35, Thm. 7.1] which has been cited recently as part of a summary [12, Prop. 41]. This indicates that, despite their simplicity, these definitions hide some subtleties. Our classification provides a careful treatment — giving minimal examples for the sake of understandability.

Such a classification is interesting as focusing on languages or systems adhering to one of the channel restrictions can be key for solving verification problems algorithmically. For instance, control-state reachability and model checking LCPDL (propositional dynamic logic with loop and converse) formulas are decidable for k -synchronisable systems [28, 12]. Later, we highlight the impact of channel restrictions on verification questions and whether one can check if a system adheres to a restriction.

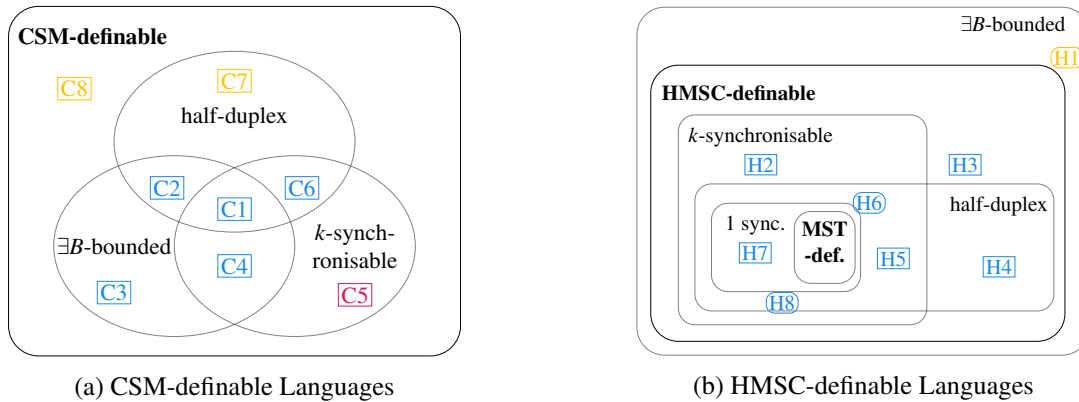


Figure 2: Comparing half-duplex, existential B -bounded, and k -synchronisable systems. The **results** are known results, **results** are new, and the **result** disproves an existing result. Hypotheses with **rounded** corners indicate inclusions while **pointed** corners indicate incomparability results.

Protocol Specifications. Instead of considering arbitrary CSMs, it is possible to start with a global description written in a dedicated protocol specification formalism such as High-level Message Sequence Charts (HMSCs) [6, 26], Multiparty Session Types (MSTs) [32, 33], or Choreography Automata (CA) [7]. A protocol is a global specification of all the processes' actions together while an implementation only gives the local actions of each process. Fig. 1 shows, along the CSM, two protocol specifications. The key difference between a protocol specification and an implementation is that the protocol specification explicitly connects a send event to the corresponding receive event. In the HMSC (Fig. 1b), the arrows connect sends to receptions. The MST¹ (Fig. 1c) specifies communication by *sender* \rightarrow *receiver*:*message*. The CSM (Fig. 1a) does not specify this connection upfront and it may not exist. This makes CSMs strictly more general than protocols. For instance, an incorrect implementation of the protocol could have P terminate before receiving the acknowledgement.

The CSM, HMSC, and MST all have the same language. Thus, our observations on channel restrictions also hold for the HMSC and the MST. We also say that the CSM *implements* the protocol specified by the HMSC (or the MST) as they accept the same language and the CSM is deadlock free. In general, there are several approaches to obtain a CSM which implements a protocol (if one exists). For instance, a protocol specification can be projected on to each process. In this paper, we do not consider this problem. A protocol specification gives rise to a language, i.e., the protocol. We only need the protocol as our definitions for channel restrictions apply to languages, e.g., *HMSC-definable languages*.

For protocols, the classification of channel restrictions was less studied than for CSMs. Fig. 2b summarises our results. It was only known that each HMSC-definable language is existentially B -bounded for some B [24]. Surprisingly, the classification changes in the context of protocols. For restrictions which differ ($\overline{H2}$ to $\overline{H5}$, and $\overline{H7}$), we give distinguishing examples. When one restriction subsumes another one ($\overline{H1}$, $\overline{H6}$, and $\overline{H8}$), we prove it. For instance, $\overline{H6}$ proves that 1-synchronisability entails half-duplex communication while $\overline{H5}$ is an example which is half-duplex, existentially B -bounded, k -synchronisable but not 1-synchronisable.

Embedding MSTs into HMSCs. In addition to our results about CSM- and HMSC-definable languages, we provide the first formal embedding from MSTs into HMSCs. The contribution is two-fold. First, we situate MSTs in the picture of common channel restrictions and prove that languages specified by

¹ We actually present a global type in an MST framework here but only use the term after its formal introduction in Section 5

multiparty session types are half-duplex, existentially 1-bounded, and 1-synchronisable. This sheds a new light on why MSTs are effectively analysable. Second, we did recently show that using insights from the domain of HMSCs in the domain of MSTs is a promising research direction as we made the effective MST verification techniques applicable to patterns from distributed computing [38]. Hence, our formal embedding can act as a crucial building block for further advances which are facilitated by insights from both domains.

Contributions. In this paper, we make three main contributions. (1) We provide an exhaustive classification of channel restrictions for CSM-definable languages. In this process, we disprove a recent result from the literature. (2) We provide an exhaustive classification of channel restrictions for HMSC- and MST-definable languages. (3) We give the first formal embedding of MSTs into HMSCs.

Outline. After providing some preliminary definitions in Section 2, we define the channel restrictions formally in Section 3 and summarise their impact on the decidability of verification questions. Subsequently, we establish our results on HMSCs (Section 4), MSTs (Section 5), and CSMs (Section 6). We discuss related work in Section 7.

2 Preliminaries

Finite and Infinite Words. For an alphabet Σ , the set of finite words over Σ is denoted by Σ^* , the set of infinite words by Σ^ω , while we write $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ for their union. For two strings $u \in \Sigma^*$ and $v \in \Sigma^\infty$, u is said to be a *prefix* of v , denoted by $u \leq v$, if there is some $w \in \Sigma^\infty$ such that $u \cdot w = v$. For two alphabets Σ and Δ with $\Delta \subseteq \Sigma$, the *projection* of $w \in \Sigma^\infty$ on to Δ , denoted by $w \downarrow_\Delta$, is the word which is obtained by omitting every letter in w that does not belong to Δ .

Message Alphabet. \mathcal{P} is a finite set of processes, ranged over by P, Q, R, \dots , and \mathcal{V} a finite set of messages. For a process P , we define the alphabet $\Sigma_P = \{P \triangleright Q!m, P \triangleleft Q?m \mid Q \in \mathcal{P}, m \in \mathcal{V}\}$ of events. The event $P \triangleright Q!m$ denotes process P sending a message m to Q , and $P \triangleleft Q?m$ denotes process P receiving a message m from Q . Note that the process performing the action is always the first one, e.g., the receiver P in $P \triangleleft Q?m$. The alphabet $\Sigma = \bigcup_{P \in \mathcal{P}} \Sigma_P$ denotes all send and receive events while $\Sigma_{sync} = \{P \rightarrow Q : m \mid P, Q \in \mathcal{P} \text{ and } m \in \mathcal{V}\}$ is the set where sending and receiving a message is specified at the same time. We fix \mathcal{P} , \mathcal{V} , Σ , and Σ_{sync} in the rest of the paper. We write $w \downarrow_{P \triangleright Q! _}$ to select all send events in w where P sends a message to Q and $\mathcal{V}(w)$ to project the send and receive events to their message values.

Distributed Executions. We use these specialised alphabets to model specifications in which multiple distributed processes communicate by exchanging messages. Furthermore, these executions cannot be any word but need to comply with conditions that correspond to the asynchronous communication over reliable FIFO channels. We call such words channel-compliant.

Definition 1 ([38]) A protocol is a set of complete channel-compliant words where:

1. **Channel-compliant:** A word $w \in \Sigma^\infty$ is channel-compliant if messages are received after they are sent and, between two processes, the reception order is the same as the send order. Formally, for each prefix w' of w , we require $\mathcal{V}(w' \downarrow_{Q \triangleleft P? _})$ to be a prefix of $\mathcal{V}(w' \downarrow_{P \triangleright Q! _})$, for every $P, Q \in \mathcal{P}$.
2. **Complete:** A channel-compliant word $w \in \Sigma^\infty$ is complete if it is infinite or the send and receive events match: if $w \in \Sigma^*$, then $\mathcal{V}(w \downarrow_{P \triangleright Q! _}) = \mathcal{V}(w \downarrow_{Q \triangleleft P? _})$ for every $P, Q \in \mathcal{P}$.

To pinpoint the corresponding send and receive events, we define a notion of *matching*.

Definition 2 (Matching Sends and Receptions) In a word $w = e_1 \dots \in \Sigma^\infty$, a send event $e_i = P \triangleright Q!m$ is matched by a receive event $e_j = Q \triangleleft P?m$, denoted by $e_i \vdash e_j$, if $i < j$ and $\mathcal{V}((e_1 \dots e_i) \downarrow_{P \triangleright Q!}) = \mathcal{V}((e_1 \dots e_j) \downarrow_{Q \triangleleft P?})$. A send event e_i is unmatched if there is no such receive event e_j .

If a sequence of events is channel-compliant, it is trivial that for each channel between two processes, either all send events are matched or there is an index from which all send events are unmatched.

In this paper, we consider protocols that can be specified with high-level messages sequence charts. We define *prefix message sequence charts* to allow unmatched send events, inspired by the work of Genest et al. [24, Def. 3.1]. The definition of a (prefix) MSC can look intimidating. In Fig. 3, we show pictorially what each component corresponds to.

Definition 3 ((Prefix) Message Sequence Charts) A prefix message sequence chart is a 5-tuple $M = (N, p, f, l, (\leq_P)_{P \in \mathcal{P}})$ where

- N is a set of send (S) and receive (R) event nodes ($N = S \uplus R$),
- $p: N \rightarrow \mathcal{P}$ maps each event node to the process acting on it,
- $f: S \rightarrow R$ is an injective partial function linking corresponding send and receive event nodes,
- $l: N \rightarrow \Sigma$ labels every event node with an event, and
- $(\leq_P)_{P \in \mathcal{P}}$ is a family of total orders for the event nodes of each process: $\leq_P \subseteq p^{-1}(P) \times p^{-1}(P)$.

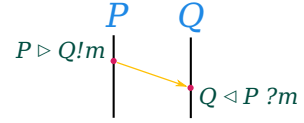


Figure 3: Highlighting the elements of a (prefix) MSC: $(N, p, f, l, (\leq_P)_{P \in \mathcal{P}})$

A prefix MSC M induces a partial order \leq_M on N that is defined co-inductively²:

$$\frac{e \leq_P e'}{e \leq_M e'} \text{ PROC} \quad \frac{s \in S}{s \leq_M f(s)} \text{ SND-RCV} \quad \frac{}{e \leq_M e} \text{ REFL} \quad \frac{e \leq_M e' \quad e' \leq_M e''}{e \leq_M e''} \text{ TRANS}$$

The labelling function l respects the function f between S and R : for every pair of event nodes $e, e' \in N$ with $f(e) = e'$, we have $l(e) = p(e) \triangleright p(e)!m$ and $l(e') = p(e') \triangleleft p(e)?m$ for some $m \in \mathcal{V}$ and for every e where $f(e)$ is undefined, we have $l(e) = p(e) \triangleright P!m$ for some $P \neq p(e)$ according to its destination.

We say that M is *degenerate* if there is some P and Q such that there are $e_1, e_2 \in p^{-1}(P)$ with $e_1 \neq e_2$, $l(e_1) = l(e_2)$, $e_1 \leq_P e_2$ and $f(e_2) \leq_Q f(e_1)$. We say that M *respects FIFO order* if M is not degenerate and for every pair of processes P, Q , and for every two event nodes $e_1 \leq_M e_2$ with $l(e_i) = P \triangleright Q!_-$ for $i \in \{1, 2\}$, it holds that $f(e_2)$ is undefined if $f(e_1)$ is undefined as well as that it holds that $\mathcal{V}(w_P) = \mathcal{V}(f(w_P))$ where w_P is the (unique) linearisation of $p^{-1}(P)$.

In this paper, we do only consider prefix message sequence charts that respect FIFO order.

If f is total, we omit the term *prefix* and call M a *message sequence chart (MSC)*. If N is finite for an MSC M , we call M a *basic MSC (BMSC)*. We denote the set of BMSCs by \mathcal{M} . When M is clear from context, we simply write \leq instead of \leq_M . For a prefix MSC M , the language $\mathcal{L}(M)$ contains a sequence $l(w)$ for each linearisation w of N compatible with \leq_M . When unambiguous, we may refer to event nodes or sequences thereof by their (event) labels or omit the label function l .

A prefix MSC, in contrast to an MSC, allows send event nodes for any channel to be unmatched from some point on. The concatenation $M_1 \cdot M_2$, or simply $M_1 M_2$, of an MSC M_1 and a prefix MSC M_2 is

²Note that we cannot use the standard reflexive and transitive closure since we consider infinite sequences of events. Co-induction lifts the reflexive, transitive closure of the union of the send-receive relation and all process orders, i.e., $(\{(s, f(s)) \mid s \in S\} \cup \bigcup_{P \in \mathcal{P}} \leq_P)^*$, to infinite sets of event nodes.

defined as expected (see the technical report [45] for the formal definition). The concatenation requires that, for any individual process, all event nodes in M_1 happen before the event nodes in M_2 . However, the induced partial order on N may permit linearisations in which an event node from M_2 of one process occurs before an event node from M_1 of another process.

For every channel-compliant word w , one can construct a unique prefix MSC M such that w is a linearisation of M .

Lemma 1 (msc(-) ([24], Section 3.1)) *Let $w \in \Sigma^\infty$ be a channel-compliant word. Then, there is unique prefix MSC, denoted by $\text{msc}(w)$, such that w is a linearisation of $\text{msc}(w)$. In case the above conditions are not satisfied, $\text{msc}(w)$ is undefined.*

All sequences of events we consider in this work are channel-compliant. For sequences from MSCs (considered in Section 4), this trivially holds, while for sequences from execution prefixes of CSMs (considered in Section 6), we prove this in the technical report [45].

3 Channel Restrictions

In this section, we present different channel restrictions and their implications on decidability of interesting verification questions. Their application is discussed subsequently: for HMSCs in Section 4.1, for MSTs in Section 5.3, and for CSMs in Section 6.1.

3.1 Definitions

3.1.1 Half-duplex Communication

Cécé and Finkel [17, Def. 8] introduced the restriction of half-duplex communication which intuitively requires that, for any two processes P and Q , the channel from P to Q is empty before Q sends a message to P . We define the restriction of half-duplex on sequences of events and show that it is equivalent to the original definition in the technical report [45].

Definition 4 (Half-duplex) *A sequence of events w is called half-duplex if for every prefix w' of w and pair of processes P and Q , one of the following holds: $\mathcal{V}(w' \downarrow_{P \rightarrow Q!}) = \mathcal{V}(w' \downarrow_{Q \rightarrow P?})$ or $\mathcal{V}(w' \downarrow_{Q \rightarrow P!}) = \mathcal{V}(w' \downarrow_{P \rightarrow Q?})$. A language $L \subseteq \Sigma^\infty$ is half-duplex if every word $w \in L$ is.*

3.1.2 Existential B-boundedness

While the previous property restricts the channel for at least one direction to be empty, one can also bound the size of channels and consider linearisations that are possible adhering to such bounds. On the one hand, one can consider a universal bound that applies for every linearisation. However, this yields finite-state systems [24] and disallows very simple protocols, e.g., the example in Fig. 1. On the other hand, one can consider an existential bound on the channels which solely asks that there is one linearisation of the distributed execution for which the channels are bounded. This allows infinite-state systems and admits the earlier example.

Definition 5 (B-bounded [24]) *Let $B \in \mathbb{N}$ be a natural number. A word w is B-bounded if for every prefix w' of w and pair of processes P and Q , it holds that $|w' \downarrow_{P \rightarrow Q!}| - |w' \downarrow_{Q \rightarrow P?}| \leq B$.*

Definition 6 (Existentially B-bounded [24]) *Let $B \in \mathbb{N}$. A prefix MSC M is existentially B-bounded if there is a B-bounded linearisation for M . A sequence of events w is existentially B-bounded if $\text{msc}(w)$ is defined and existentially B-bounded. A language L is existentially B-bounded if every word $w \in L$ is. We may use not existentially bounded as abbreviation for not existentially B-bounded for any B .*

3.1.3 k -synchronisability

The restriction of k -synchronisability was introduced for mailbox communication [13] and later refined and adapted to the point-to-point setting [28]. We define k -synchronisability following definitions by Giusto et al. [28, Defs. 6 and 7]. The definition of k -synchronisability builds upon the notion when a prefix MSC is k -synchronous. Its first condition requires that there is some linearisation of the prefix MSC while its second condition requires causal delivery to hold. In contrast to the mailbox setting, the first condition always entails the second condition for the point-to-point setting.

Point-to-point Communication implies Causal Delivery. We first adapt the definition of causal delivery [28, Def. 4] for point-to-point FIFO channels [28, Section 6]. Unfortunately, this discussion leaves room for interpreting what causal delivery exactly is for point-to-point systems. Based on the description that a process P can receive messages from two distinct processes Q and R in any order, regardless of the dependency between the corresponding send events, we decided to literally adapt the definition of causal delivery as follows.

Definition 7 (Causal delivery) *Let $M = (N, p, f, l, (\leq_P)_{P \in \mathcal{P}})$ be an MSC. We say that M satisfies causal delivery if there is a linearisation $w = e_1 \dots$ of N such that for any two events $e_i \leq_M e_j$ with $e_i = P \triangleright Q!_-$ and $e_j = P \triangleright Q!_-,$ either e_j is unmatched in w or there are $e_{i'} \leq_M e_{j'}$ such that $e_i \vdash e_{i'}$ and $e_j \vdash e_{j'}$ in $w.$*

We show that $\text{msc}(w)$ for every w (if defined) satisfies causal delivery (as proven in the technical report [45]).

Lemma 2 *Let $w \in \Sigma^\infty$ such that $\text{msc}(w)$ is defined. Then, $\text{msc}(w)$ satisfies causal delivery.*

In combination with the fact that, given a linearisation w of a prefix MSC M , $\text{msc}(w)$ is isomorphic to M , this yields that causal delivery is satisfied if there is a linearisation.

Corollary 1 *Every prefix MSC with a linearisation satisfies causal delivery.*

With this, we can simplify the definition by omitting the second condition without changing its meaning. In addition, we extend it to apply for MSCs with infinite sets of event nodes.

Definition 8 (k -synchronous and k -synchronisable) *Let $k \in \mathbb{N}$ be a positive natural number. We say that a prefix MSC $M = (N, p, f, l, (\leq_P)_{P \in \mathcal{P}})$ is k -synchronous if*

1. *there is a linearisation of event nodes w compliant with \leq_M which can be split into a sequence of k -exchanges (also called message exchange if k not given or clear from context), i.e., $w = w_1 \dots$ such that for all i , it holds that $l(w_i) \in S^{\leq k} \cdot R^{\leq k}$; and*
2. *for all e, e' in w such that $e \vdash e'$, there is some i with e, e' in w_i .³*

A linearisation w is k -synchronisable⁴ if $\text{msc}(w)$ is k -synchronous. A language L is k -synchronisable if every word $w \in L$ is. We may use not synchronisable as abbreviation for not k -synchronisable for any k .

³This is equivalent to the following: for all e and $f(e)$ in w , there is some i with $e, f(e)$ in w_i .

⁴One could distinguish between universal and existential k -synchronisability, i.e., to distinguish the existence of a k -synchronisable linearisation rather than all linearisations being k -synchronisable. However, the universal version does not make much sense in practice. Thus, we omit the term existential.

3.2 Algorithmic Verification and Channel Restrictions

It is important to note that we use the term *restriction* as a property of a system which occurs naturally and not something that is imposed on its semantics. However, both have a tight connection: a system *naturally* satisfies a restriction if *imposing* the restriction does not change its possible behaviours. If this is the case, one can exploit this for algorithmic verification and only check behaviours that satisfy the restriction without harming correctness.

For each channel restriction, we recall known results about checking membership and which verification problems become decidable.

Half-duplex Communication. For CSMs with two processes, membership is decidable [17, Thm. 31]. The set of reachable configurations is computable in polynomial time which renders many verification questions like the *unspecified reception problem* decidable (see [17, Thm. 16] for a detailed list of verification problems) while model checking PLTL or CTL is still undecidable. Half-duplex CSMs with more than two processes are Turing-powerful [17, Thm. 38] so verification becomes undecidable and checking membership is of little interest.

Existential B -boundedness. For CSMs, membership is undecidable, unless CSMs are known to be deadlock free and B is given [24, Fig. 3]. For protocols, we will see that they are always existentially B -bounded for some B and thus a correct implementation of a protocol also is. It is quite straightforward that control-state reachability is decidable but not typically studied for these systems [12]. Intuitively, it can be solved by exhaustively enumerating the reachability graph of the CSM while pruning configurations exceeding the bound B . For HMSCs, model checking is undecidable for LTL [6, Thm. 3] and decidable for MSO [37, Thm. 1].

k -synchronisability. For CSMs, membership for a given k is decidable in EXPTIME [12, Rem. 30], originally shown decidable by Di Giusto et al. [28], while it is undecidable if k is not given [12, Thm. 22]. For HMSCs, both questions are decidable in polynomial time, while we show that MSTs are always 1-synchronisable. Model checking for k -synchronisable systems is decidable and in EXPTIME when formulas are represented in LCPDL. This follows from combining that such systems have bounded (special) tree-width [12, Prop. 28] and results by Bollig and Finkel [11]. Control-state reachability was shown to be decidable for k -synchronisable systems [28, Thm. 6].

4 High-level Message Sequence Charts

Message sequence charts have been used as compact representation for executions of CSMs. The (prefix) message sequence charts obtained from different executions of CSMs can be analysed to determine which channel restriction is satisfied [24, 28]. In addition, we do also use message sequence charts as building blocks for high-level message sequence charts [39, 48] which specify protocols.

We define these following the presentation by Alur et al. [4, 5]. A BMSC corresponds to “straight line code” in which each process follows a single sequence of event nodes. A *high-level message sequence chart* (HMSC) adds a regular control structure (branching and loops).

Definition 9 (High-Level Message Sequence Charts) A high-level message sequence chart (HMSC) is a structure (V, E, v^I, V^T, μ) where V is a finite set of vertices, $E \subseteq V \times V$ is a set of directed edges, $v^I \in V$ is an initial vertex, $V^T \subseteq V$ is a set of terminal vertices, and $\mu : V \rightarrow \mathcal{M}$ is a function mapping every vertex to a BMSC.

To obtain the language of an HMSC, we start with initial paths through the HMSC. As usual, we are interested only in maximal paths, i.e., either infinite or ending in a terminal vertex. We can expand

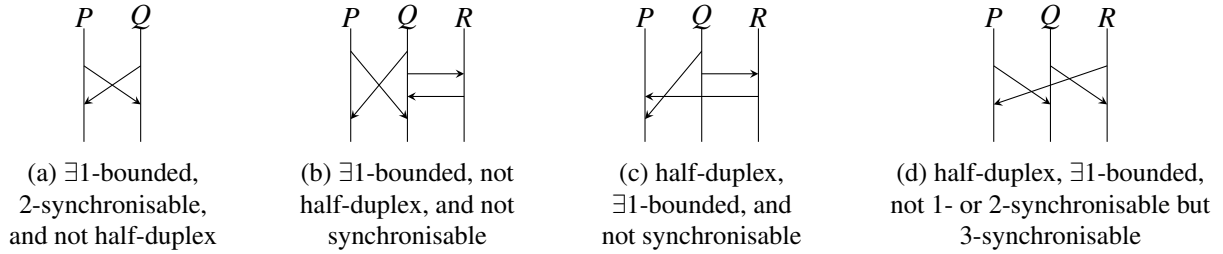


Figure 4: BMSCs which satisfy different channels restrictions

each such path into a sequence of BMSCs, concatenate this sequence of BMSCs, and take the language of the resulting MSC. The language of the HMSC is the union of the languages of the MSCs generated by all its initial paths – see the technical report [45] for the formal definition. For simplicity, we assume every vertex in an HMSC is reachable from the initial vertex and every initial non-maximal path can be completed to a maximal one.

Example 1 Figure 1b shows an HMSC composed of two BMSCs. MSCs of the HMSC are obtained by following the control structure and concatenating the corresponding BMSCs. The language contains all linearisations of these MSCs.

4.1 Channel Restrictions of HMSCs

We say that an HMSC is half-duplex, existentially B -bounded or k -synchronisable respectively if its language is. It is straightforward that checking an HMSC for k -synchronisability amounts to checking its BMSCs.

Proposition 1 *An HMSC H is k -synchronisable iff all BMSCs of H are k -synchronisable.*

For the presentation of our results, we follow the numbering laid out in Fig. 2b. Note that any BMSC can always be turned into a HMSC with a single initial and terminal vertex. Therefore, it is trivial that all BMSC examples also apply to HMSCs.

Lemma 3 ([24], Prop. 3.1) $\boxed{\text{H1}}$: *Any HMSC H is existentially B -bounded for some B .*

We prove this result in a slightly different way in the technical report [45]. Basically, one computes the bound for the BMSC of every vertex in H and takes the maximum. This works since every MSC of H is a concatenation of individual BMSCs which can be scheduled in a way that the channels are empty after each BMSC.

Example 2 $\boxed{\text{H2}}$: $\exists B$ -bounded, k -synchronisable, and not half-duplex. Consider the BMSC in Fig. 4a. It is existentially 1-bounded as there is one message per channel, 2-synchronisable since the message exchange can be split into one phase of two sends and two subsequent receives and not half-duplex because both messages can traverse their channel at the same time.

Example 3 $\boxed{\text{H3}}$: $\exists B$ -bounded, not half-duplex, and not synchronisable. It is obvious that the BMSC M in Fig. 4b is not half-duplex. We show that M is not k -synchronisable for any k . Let us denote the event nodes for each process P with p_1, \dots as ordered by the total process order. It is straightforward that one of p_1 and q_1 has to be part of the first k -exchange. However, since the respective corresponding reception

happens after the other's event node, both have to be a part of the first k -exchange. Since these receive event nodes (transitively) depend on all other event nodes, all event nodes have to be part of a single k -exchange for M . However, R first has to receive from Q in order to send back to it and therefore, there is no single k -exchange for M and M is not k -synchronisable for any k .

Example 4 [H4]: *half-duplex, $\exists B$ -bounded, and not synchronisable.* Let us consider the BMSC in Fig. 4c. It is straightforward that it is half-duplex and existentially 1-bounded. However, it is not k -synchronisable for any k . In particular, the first and last event node (of any total order induced by the BMSC) must belong to the same message exchange but two more linearly dependent message exchanges need to happen in between.

Example 5 [H5]: *half-duplex, $\exists B$ -bounded, k -synchronisable but not 1-synchronisable.* Consider the BMSC in Fig. 4d. It is easy to see that it is not 1- or 2-synchronisable but 3-synchronisable, half-duplex and existentially 1-bounded. Note that it is straightforward to amend the example such that it is still half-duplex but the parameters B and k need to be increased.

Lemma 4 [H6]: Every 1-synchronisable HMSC is half-duplex.

Intuitively, in any BMSC of an HMSC, every send event node has a corresponding receive event node. Therefore, a message that has been sent needs to have been received directly afterwards and the per-process order is total so any process has to receive a message before it sends a message back. The full proof can be found in the technical report [45].

5 Multiparty Session Types

In this section, we recall global types from Multiparty Session Types (MSTs) as a way to specify protocols. We present an embedding for MSTs into HMSCs, prove it correct, and use it to show that MSTs are half-duplex, existentially 1-bounded, and 1-synchronisable.

5.1 Specifying Protocols with Global Types

We now define global types in the framework of MSTs as a syntax for protocol specifications. The syntax of global types is defined following classical MST frameworks [44, Def. 3.2]. The calculus focuses on the core message-passing primitives of asynchronous MSTs and does not incorporate features like subsessions or delegation. However, it does incorporate a recent generalisation that allow a sender to send to different receivers upon branching [38].

Definition 10 (Syntax of Global Types [38]) Global types for MSTs are defined by the grammar:

$$G ::= 0 \mid \sum_{i \in I} P \rightarrow Q_i : m_i . G_i \mid \mu t . G \mid t$$

An expression $P \rightarrow Q : m$ stands for a send and receive event: $P \triangleright Q ! m$ and $Q \triangleleft P ? m$. Since global types always specify send and the receive events together, they specify complete channel-compliant sequences of events. For a *choice*, the sender process decides which branch to take and each branch of a choice needs to be uniquely distinguishable ($\forall i, j \in I. i \neq j \Rightarrow Q_i \neq Q_j \vee m_i \neq m_j$). If there is a single alternative (and no actual choice), we omit writing the sum operator. Loops are encoded by the least fixed point operator and recursion must be guarded, i.e., there is at least one message between μt and t . We assume, without loss of generality, that all occurrences of recursion variables t are bound and every variable t

is distinct. Recursion only happens at the tail (and there is no additional parameter) and, therefore, the language of a global type can be defined with an automaton – as expected by following the structure of a global type, splitting the message exchanges into send and receive events while not only accounting for finite but also infinite executions. We give one language as example and refer to the technical report [45] for a formal definition.

Example 6 *The type language for the global type in Fig. 1c, $\mu t. (P \rightarrow Q : \text{cons}. t + P \rightarrow Q : \text{nil}. Q \rightarrow P : \text{ack}. 0)$, is the union of a set of finite executions and infinite executions:*

$$(P \triangleright Q ! \text{cons}. Q \triangleleft P ? \text{cons})^* . P \triangleright Q ! \text{nil}. Q \triangleleft P ? \text{nil}. Q \triangleright P ! \text{ack}. P \triangleleft Q ? \text{ack} \quad \text{and} \quad (P \triangleright Q ! \text{cons}. Q \triangleleft P ? \text{cons})^\omega$$

Remark 1 *For readers familiar with MSTs, it may be strange that we do not define local types. In fact, one correctness criterion for local types requires that their composition generates the same language as the original global type. All channel restrictions are defined using languages, so it suffices to consider global types for our purposes.*

Example 7 *Consider the global type: $P \rightarrow Q : m_1. R \rightarrow S : m_2$. The type language for this type contains only the word $P \triangleright Q ! m_1. Q \triangleleft P ? m_1. R \triangleright S ! m_2. S \triangleleft R ? m_2$. On the other hand, if we want to describe the same protocol with a HMSC, it always allows any permutation of the events where $P \triangleright Q ! m_1$ occurs before $Q \triangleleft P ? m_1$ and $R \triangleright S ! m_2$ before $S \triangleleft R ? m_2$.*

Intuitively, some events in a distributed setting shall not be ordered since they are independent, e.g., happen on different processes as in the previous example. To this end, we recall an indistinguishability relation \sim that captures the reordering allowed by CSMs with FIFO channels (which will be defined in Definition 12). In MSTs, similar reordering rules are applied (e.g., [33, Def. 3.2 and 5.3]).

Definition 11 (Indistinguishability relation \sim [38]) *Let $\sim_i \subseteq \Sigma^* \times \Sigma^*$, for $i \geq 0$, be a family of indistinguishability relations. For all $w \in \Sigma^*$, we have $w \sim_0 w$. For $i = 1$, we define:*

- (1) *If $P \neq R$, then $w.P \triangleright Q ! m. R \triangleright S ! m'. u \sim_1 w.R \triangleright S ! m'. P \triangleright Q ! m. u$.*
- (2) *If $Q \neq S$, then $w.Q \triangleleft P ? m. S \triangleleft R ? m'. u \sim_1 w.S \triangleleft R ? m'. Q \triangleleft P ? m. u$.*
- (3) *If $P \neq S \wedge (P \neq R \vee Q \neq S)$, then $w.P \triangleright Q ! m. S \triangleleft R ? m'. u \sim_1 w.S \triangleleft R ? m'. P \triangleright Q ! m. u$.*
- (4) *If $|w \downarrow_{P \triangleright Q !} | > |w \downarrow_{Q \triangleleft P ?} |$, then $w.P \triangleright Q ! m. Q \triangleleft P ? m'. u \sim_1 w.Q \triangleleft P ? m'. P \triangleright Q ! m. u$.*

Let w, w', w'' be sequences of events such that $w \sim_1 w'$ and $w' \sim_i w''$ for some i . Then, $w \sim_{i+1} w''$. We define $w \sim u$ if there is n such that $w \sim_n u$.

This formalises how messages can be swapped for finite executions of protocols. The infinite case requires special technical treatment for which we refer to the work by Majumdar et al. [38].

The relation is lifted to languages as expected. For a language L , we have:

$$\mathcal{L}^\sim(L) = \left\{ w' \mid \bigvee \begin{array}{l} w' \in \Sigma^* \wedge \exists w \in \Sigma^*. w \in L \text{ and } w' \sim w \\ w' \in \Sigma^\omega \wedge \exists w \in \Sigma^\omega. w \in L \text{ and } w' \preceq^\omega w \end{array} \right\}.$$

The indistinguishability relation \sim does not change the order of send and receive events of a single process. The relation \sim captures all reorderings which naturally appear when global types from MSTs are implemented with CSMs. For a global type G , its semantics is given by its *execution language* $\mathcal{L}^\sim(\mathcal{L}(G))$. Furthermore, the indistinguishability relation captures exactly the events that are independent in any HMSC. Phrased differently, HMSC include these reorderings by design.

Lemma 5 *Let H be any HMSC. Then, $\mathcal{L}(H) = \mathcal{L}^\sim(\mathcal{L}(H))$.*

We prove this in the technical report [45]. A similar result for CSMs has been proven [38, Lemma 21].

Theorem 1 *For channel-compliant words, the indistinguishability relation \sim preserves satisfaction of half-duplex communication, existential B-boundedness, and k-synchronisability.*

The proof can be found in the technical report [45].

5.2 Encoding Global Types from MSTs into HMSCs

Global types from MSTs can be turned into HMSCs while preserving the protocol they specify. In this step, we account for the orders that can and cannot be enforced in an asynchronous point-to-point setting with the indistinguishability relation \sim . The main difference between the automata-based semantics of global types from MSTs and the semantics of HMSCs is that an automaton carries the events on the edges and an HMSC carries events as labels of the event nodes in the BMSCs associated with the vertices.

In the translation, we use the following notation. M_\emptyset is the empty BMSC ($N = \emptyset$) and $M(P \rightarrow Q : m)$ is the BMSC with two event nodes: e_1, e_2 such that $f(e_1) = e_2$, $l(e_1) = P \triangleright Q ! m$, and $l(e_2) = Q \triangleleft P ? m$.

From a global type G , we construct an HMSC $H(G) = (V, E, v^J, V^T, \mu, \lambda)$ with

$$\begin{aligned} V &= \{G' \mid G' \text{ is a subterm of } G\} \cup \{(\sum_{i \in I} P \rightarrow Q_i : m_i \cdot G_i, j) \mid \sum_{i \in I} P \rightarrow Q_i : m_i \cdot G_i \text{ occurs in } G \wedge j \in I\} \\ E &= \{(\mu t.G', G') \mid \mu t.G' \text{ occurs in } G\} \cup \{(t, \mu t.G') \mid t, \mu t.G' \text{ occurs in } G\} \\ &\quad \cup \{(\sum_{i \in I} P \rightarrow Q_i : m_i \cdot G_i, (\sum_{i \in I} P \rightarrow Q_i : m_i \cdot G_i, j)) \mid (\sum_{i \in I} P \rightarrow Q_i : m_i \cdot G_i, j) \in V\} \\ &\quad \cup \{((\sum_{i \in I} P \rightarrow Q_i : m_i \cdot G_i, j), G_j) \mid (\sum_{i \in I} P \rightarrow Q_i : m_i \cdot G_i, j) \in V\} \\ v^J &= G \quad V^T = \{0\} \quad \mu(v) = \begin{cases} M(P \rightarrow Q_i : m_j) & \text{if } v = (\sum_{i \in I} P \rightarrow Q_i : m_i \cdot G_i, j) \\ M_\emptyset & \text{otherwise} \end{cases} \end{aligned}$$

This translation does not yield the HMSC with the least number of vertices since vertices with a single successor could be merged to form larger BMSCs. Here, every BMSC contains at most one message exchange. We obtain the following correctness statement for the embedding:

Theorem 2 *For any global type G , it holds that $\mathcal{L}(G) \subseteq \mathcal{L}(H(G))$ and $\mathcal{C}^\sim(\mathcal{L}(G)) = \mathcal{C}^\sim(\mathcal{L}(H(G)))$.*

We provide the technical developments to show Theorem 2 in the technical report [45].

Remark 2 *The first part of Theorem 2 uses \subseteq instead of $=$ as HMSCs do not order indistinguishable events and we consider the type language of G . Example 7 shows that using the execution language rather than the type language in the second part is inevitable for equality and does not weaken the claim.*

5.3 Channel Restrictions of Global Types

Example 8 [H7]: *half-duplex, $\exists 1$ -bounded, 1-synchronisable but not in MSTs.*

Consider the HMSC in Fig. 5. It is straightforward that it is half-duplex, existentially 1-bounded, and 1-synchronisable. Both P and Q send the same message to R independently in each branch. Intuitively, R chooses which branch to take by the order it decides to receive both messages. Subsequently, it notifies Q about this choice. Such a communication pattern cannot be expressed in the MST framework. If one tried to model it with R actually choosing the branch, l and r would always occur before the receptions so the languages are different.

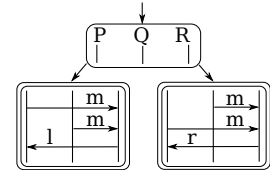


Figure 5: half-duplex, $\exists 1$ -bounded, and 1-synchronisable but not expressible in MSTs

We show that protocols specified as global types satisfy all discussed channel restrictions (with the minimal reasonable parameter).

Theorem 3 [H8]: *The execution language $\mathcal{C}^\sim(\mathcal{L}(G))$ is half-duplex, existentially 1-bounded, and 1-synchronisable for any global type G .*

The proof uses the embedding to obtain an HMSC built of BMSCs with at most one message exchange and exploits previously shown properties about HMSCs. Details can be found in the technical report [45].

Remark 3 (Choreography automata are half-duplex, $\exists 1$ -bounded, and 1-synchronisable)

In this section, we looked at MSTs which are rooted in process algebra. With choreography automata [7], a similar concept has been studied from automata theory perspective. Basically, a protocol specification is an automaton whose transitions are labelled by $P \rightarrow Q : m$. In contrast to global types from MSTs, they do not impose constraints on choice, i.e., there does not need to be a unique process chooses which branch to take next and do not employ an indistinguishability relation but require to explicitly spell out all possible reorderings. This feature can lead to complications w.r.t. implementing such protocols but does not change the satisfaction of channel restrictions. In fact, protocols specified by choreography automata are also half-duplex, existentially 1-bounded, and 1-synchronisable.

6 Communicating State Machines

In this section, we first present communicating state machines (CSMs) as formal model for distributed processes which communicate messages asynchronously via reliable point-to-point FIFO channels. If a CSM implements a protocol specification, both languages are the same – modulo \sim which does not alter satisfaction of channel restrictions. This entails that CSMs implementing protocol specifications satisfy the same channel restrictions as presented in previous sections. Here, we investigate the channel restrictions of general CSMs which might not implement a protocol specified as global type or HMSC.

Definition 12 (Communicating state machines) A state machine $A = (Q, \Delta, \delta, q_0, F)$ is a 5-tuple where Q is a finite set of states, Δ is an alphabet, $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is a transition relation, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of final states. We write $q \xrightarrow{a} q'$ for $(q, a, q') \in \delta$. A run of A is a sequence $\rho = q_0 \xrightarrow{w_0} q_1 \xrightarrow{w_1} \dots$, with $q_i \in Q$ and $w_i \in \Delta \cup \{\varepsilon\}$ for $i \geq 0$, such that q_0 is the initial state, and for each $i \geq 0$, it holds that $(q_i, w_i, q_{i+1}) \in \delta$. The trace of the run is the finite or infinite word $w_0 w_1 \dots \in \Sigma^\infty$. The path of the run is the finite or infinite sequence $q_0 q_1 \dots \in Q^\infty$. A run is called maximal if it is infinite or ends at a final state. Accordingly, the corresponding trace and path are called maximal. The language $\mathcal{L}(A)$ of A is the set of its maximal traces.

We call $\mathcal{A} = \{A_P\}_{P \in \mathcal{P}}$ a communicating state machine (CSM) over \mathcal{P} and \mathcal{V} if A_P is a finite state machine with alphabet Σ_P for every $P \in \mathcal{P}$. The state machine for P is denoted by $(Q_P, \Sigma_P, \delta_P, q_{0,P}, F_P)$. Intuitively, a CSM allows a set of state machines, one for each process in \mathcal{P} , to communicate by sending and receiving messages. For this, each pair of processes $P, Q \in \mathcal{P}$, $P \neq Q$, is connected by two directed message channels. A transition $q_P \xrightarrow{P \triangleright Q!m} q'_P$ in the state machine of P denotes that P sends message m to Q if P is in the state q and changes its local state to q' . The channel $\langle P, Q \rangle$ is appended by message m . For receptions, a transition $q_Q \xrightarrow{Q \triangleleft P?m} q'_Q$ in the state machine of Q corresponds to Q retrieving the message m from the head of the channel when its local state is \hat{q} which is updated to \hat{q}' . The run of a CSM always starts with empty channels and each finite state machine is its respective initial state. The formalisation of this intuition is standard and can be found in the technical report [45].

As for HMSCs, the language of a CSM is closed under \sim .

Lemma 6 ([38], Lemma 21) Let \mathcal{A} be a CSM. Then $\mathcal{L}(\mathcal{A}) = \mathcal{C}^\sim(\mathcal{L}(\mathcal{A}))$.

Implementing Protocol Specifications. Given a protocol specification, one can try to generate an implementation which admits the same language. This problem is known as implementability or realisability in the HMSC setting [27, 5, 36]. In the MST setting [32], this is done in two steps. First, the global type is projected on to local types. Second, a type system ensures that the implementations follow the local types, i.e., a refinement check. However, one can design CSMs from scratch that yield systems which cannot be captured by protocol specifications like HMSCs or global types from MSTs.

6.1 Channel Restrictions of CSMs

We say that an CSM is half-duplex, existentially B -bounded, or k -synchronisable respectively if its language is. Again, we follow the outline presented in Fig. 2a.

Example 9 $\boxed{\text{C1}}$: *half-duplex, $\exists B$ -bounded, and k -synchronisable. The CSM in Fig. 1a is $\exists 1$ -bounded, 1-synchronisable, and half-duplex.*

Any BMSC can easily be implemented with an CSM by simple letting each process follow its linear trajectory of eventnodes. We call this projection. Therefore, we can use three of the BMSCs presented in Fig. 4 to show the hypotheses for CSMs:

Example 10 For $\boxed{\text{C2}}$, the projection of Fig. 4c (used to show $\boxed{\text{H4}}$) is half-duplex, $\exists B$ -bounded, and not synchronisable. For $\boxed{\text{C3}}$, the projection of Fig. 4b (used to show $\boxed{\text{H3}}$) is $\exists B$ -bounded, not half-duplex, and not synchronisable. For $\boxed{\text{C4}}$, the projection of Fig. 4a (used to show $\boxed{\text{H2}}$) is $\exists B$ -bounded, k -synchronisable, and not half-duplex.

Example 11 $\boxed{\text{C5}}$: *k -synchronisable, not half-duplex and not \exists -bounded;*

$\boxed{\text{C6}}$: *k -synchronisable, half-duplex and not \exists -bounded.*

We consider two CSMs constructed from the state machines in Fig. 6. For $\boxed{\text{C5}}$, we consider the CSM consisting of both state machines. It is 1-synchronisable but not existentially bounded and not half-duplex. It is 1-synchronisable because every linearisation can be split into single send events that constitute 1-exchanges. It is neither existentially B -bounded for any B nor half-duplex since none of the messages will be received so both channels can grow arbitrarily. For $\boxed{\text{C6}}$, it can easily be turned into a half-duplex CSM by removing one of the send events. Then, the CSM is 1-synchronisable and half-duplex but not existentially bounded.

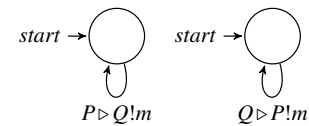


Figure 6: CSM with FSMs for P (left) and for Q (right)

This example disproves a result from the literature [35, Thm. 7.1], which states that every k -synchronisable system is existentially B -bounded for some B and has been cited recently as part of a summary [12, Prop. 41]. In the proof, it is neglected that unreceived messages remain in the channels after a message exchange. Our example satisfies their assumption that CSMs do not have states with mixed choice, i.e., each state either is final, has send options to choose from, or receive options to choose from. We do not impose any assumptions on mixed choice in this work. Still, all the presented examples do not have states with mixed choice so the presented relationships also hold for this subset of CSMs.

Corollary 2 *Existential B -boundedness and k -synchronisability for CSMs are incomparable.*

The previous result follows immediately from the CSMs constructed in Example 11. Our result considers the point-to-point FIFO setting. For the mailbox setting, the analogous question is an open problem [13].

Turing-powerful Encodings. On the one hand, it is well-known that CSMs are Turing-complete [14] and Cécé and Finkel [17, Thm. 36] showed that half-duplex communication does not impair expressiveness of CSMs with more than two processes. On the other hand, each of existential B -boundedness and k -synchronisability render some verification questions decidable. Therefore, the encodings of Turing-completeness [14, 17] are examples for CSMs which are not existentially B -bounded for any B nor k -synchronisable for any k and either half-duplex ($\boxed{\text{C7}}$) or not half-duplex ($\boxed{\text{C8}}$).

7 Related Work

We now cover related work which is not already cited in the earlier sections.

The origins of MSTs date back to 1993 when Honda et al. [31] proposed a binary version for typing communication in the domain of process algebra. In 2008, Honda et al. [32] generalised the idea to multiparty systems. While the connection of MSTs and CSMs has been studied soon after MSTs had been proposed [16, 19], we provide, to the best of our knowledge, the first formal connection of MSTs to HMSCs, even though HMSC-like visualisations have been used in the community of session types, e.g. [15, Fig. 1], [33, Figs. 1 and 2]. For binary session types, it is known how to compute the bound B of universally B -bounded types [20, 22]. Lange et al. [35] proposed k -multiparty consistency (k -MC) for CSMs as extension of multiparty consistency for MSTs. We did not consider k -MC in this work for two reasons. First, they assume an existential bound (of k) on channels. Second, as an extension of multiparty consistency, k -MC focuses on implementability rather than channel restrictions.

HMSCs and variants thereof have been extensively studied [26, 25, 23, 43]. The connection to CSMs has been investigated in particular for different forms of implementability [27, 5, 36], also called realisability [5], which is undecidable in general [26, 5], and implied scenarios [41, 40] which arise when implementing HMSCs with CSMs. Several restrictions to check implementability adopted a limited form of choice [8, 29, 41, 40, 18] which is similar to the one in global types from MSTs. For more details, we refer to work by Majumdar et al. [38].

While we consider finite state machines as model for processes, research has also been conducted on communicating systems where processes are given more computational power, e.g., pushdown automata [30, 47, 3]. However, as noted before, our setting is already Turing-powerful. In Section 3.2, we surveyed how channel restrictions can yield decidability. Incomplete approaches consider subclasses which enable the effective computation of symbolic representations (of channel contents) for reachable states [9, 34]. Other approaches change the semantics of channels, e.g., by making them lossy [2, 1, 34], input-bounded [10], or by restricting the communication topology [42, 46].

While we build on the most recent definitions of synchronisability [28], we refer to the work by Finkel and Lozes [21] and Bouajjani et al. [13] for earlier work on synchronisability. Bollig et. al [12] studied the connection of different notions of synchronisability for MSCs and MSO logic which yields interesting decidability results. We refer to their work for more details but briefly point to the slightly different use of terminology: k -synchronisability is called weak (k -)synchronisability by Bollig where the omission of k indicates a system is synchronisable for some k ; while strong (k -)synchronisability does solely apply to the mailbox setting.

8 Conclusion

We presented a comprehensive comparison of half-duplex, existential B -bounded, and k -synchronisable communication. We showed that the three restrictions are different for CSMs. For HMSCs, the half-duplex restriction and k -synchronisability are different and included in existential B -boundedness. Furthermore, all 1-synchronisable HMSC-definable languages are half-duplex. This subclass contains global types from MSTs which are also existentially 1-bounded. We established the first formal embedding of global types from MSTs into HMSCs which can be used to combine insights from both domains for further advances on implementing protocol specifications.

References

- [1] Parosh Aziz Abdulla, C. Aiswarya & Mohamed Faouzi Atig (2016): *Data Communicating Processes with Unreliable Channels*. In Martin Grohe, Eric Koskinen & Natarajan Shankar, editors: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, ACM, pp. 166–175, doi:10.1145/2933575.2934535.
- [2] Parosh Aziz Abdulla, Ahmed Bouajjani & Bengt Jonsson (1998): *On-the-Fly Analysis of Systems with Unbounded, Lossy FIFO Channels*. In Alan J. Hu & Moshe Y. Vardi, editors: *Computer Aided Verification, 10th International Conference, CAV'98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings, Lecture Notes in Computer Science 1427*, Springer, pp. 305–318, doi:10.1007/BFb0028754.
- [3] C. Aiswarya, Paul Gastin & K. Narayan Kumar (2014): *Verifying Communicating Multi-pushdown Systems via Split-Width*. In Franck Cassez & Jean-François Raskin, editors: *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings, Lecture Notes in Computer Science 8837*, Springer, pp. 1–17, doi:10.1007/978-3-319-11936-6_1.
- [4] Rajeev Alur, Kousha Etessami & Mihalis Yannakakis (2003): *Inference of Message Sequence Charts*. *IEEE Trans. Software Eng.* 29(7), pp. 623–633, doi:10.1109/TSE.2003.1214326.
- [5] Rajeev Alur, Kousha Etessami & Mihalis Yannakakis (2005): *Realizability and verification of MSC graphs*. *Theor. Comput. Sci.* 331(1), pp. 97–114, doi:10.1016/j.tcs.2004.09.034.
- [6] Rajeev Alur & Mihalis Yannakakis (1999): *Model Checking of Message Sequence Charts*. In Jos C. M. Baeten & Sjouke Mauw, editors: *CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings, Lecture Notes in Computer Science 1664*, Springer, pp. 114–129, doi:10.1007/3-540-48320-9_10.
- [7] Franco Barbanera, Ivan Lanese & Emilio Tuosto (2020): *Choreography Automata*. In Simon Bliudze & Laura Bocchi, editors: *Coordination Models and Languages - 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings, Lecture Notes in Computer Science 12134*, Springer, pp. 86–106, doi:10.1007/978-3-030-50029-0_6.
- [8] Hanène Ben-Abdallah & Stefan Leue (1997): *Syntactic Detection of Process Divergence and Non-local Choice in Message Sequence Charts*. In Ed Brinksma, editor: *Tools and Algorithms for Construction and Analysis of Systems, Third International Workshop, TACAS '97, Enschede, The Netherlands, April 2-4, 1997, Proceedings, Lecture Notes in Computer Science 1217*, Springer, pp. 259–274, doi:10.1007/BFb0035393.
- [9] Bernard Boigelot, Patrice Godefroid, Bernard Willems & Pierre Wolper (1997): *The Power of QDDs (Extended Abstract)*. In Pascal Van Hentenryck, editor: *Static Analysis, 4th International Symposium, SAS '97, Paris, France, September 8-10, 1997, Proceedings, Lecture Notes in Computer Science 1302*, Springer, pp. 172–186, doi:10.1007/BFb0032741.
- [10] Benedikt Bollig, Alain Finkel & Amrita Suresh (2020): *Bounded Reachability Problems Are Decidable in FIFO Machines*. In Igor Konnov & Laura Kovács, editors: *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference), LIPIcs 171, Schloss Dagstuhl - Leibniz-Zentrum für Informatik*, pp. 49:1–49:17, doi:10.4230/LIPIcs.CONCUR.2020.49.
- [11] Benedikt Bollig & Paul Gastin (2019): *Non-Sequential Theory of Distributed Systems*. *CoRR abs/1904.06942*, doi:10.48550/arXiv.1904.06942. arXiv:1904.06942.
- [12] Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Étienne Lozes & Amrita Suresh (2021): *A Unifying Framework for Deciding Synchronizability*. In Serge Haddad & Daniele Varacca, editors: *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference, LIPIcs 203, Schloss Dagstuhl - Leibniz-Zentrum für Informatik*, pp. 14:1–14:18, doi:10.4230/LIPIcs.CONCUR.2021.14.

- [13] Ahmed Bouajjani, Constantin Enea, Kailiang Ji & Shaz Qadeer (2018): *On the Completeness of Verifying Message Passing Programs Under Bounded Asynchrony*. In Hana Chockler & Georg Weissenbacher, editors: *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II, Lecture Notes in Computer Science 10982*, Springer, pp. 372–391, doi:10.1007/978-3-319-96142-2_23.
- [14] Daniel Brand & Pitro Zafiropulo (1983): *On Communicating Finite-State Machines*. *J. ACM* 30(2), pp. 323–342, doi:10.1145/322374.322380.
- [15] Marco Carbone, Kohei Honda, N. Yoshida, R. Milner, G. Brown & Steve Ross-Talbot (2005): *A Theoretical Basis of Communication-Centred Concurrent Programming*.
- [16] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini & Luca Padovani (2012): *On Global Types and Multi-Party Session*. *Log. Methods Comput. Sci.* 8(1), doi:10.2168/LMCS-8(1:24)2012.
- [17] Gérard Cécé & Alain Finkel (2005): *Verification of programs with half-duplex communication*. *Inf. Comput.* 202(2), pp. 166–190, doi:10.1016/j.ic.2005.05.006.
- [18] Haitao Dan, Robert M. Hierons & Steve Counsell (2010): *Non-local Choice and Implied Scenarios*. In José Luiz Fiadeiro, Stefania Gnesi & Andrea Maggiolo-Schettini, editors: *8th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2010, Pisa, Italy, 13-18 September 2010*, IEEE Computer Society, pp. 53–62, doi:10.1109/SEFM.2010.14.
- [19] Pierre-Malo Deniérou & Nobuko Yoshida (2012): *Multiparty Session Types Meet Communicating Automata*. In Helmut Seidl, editor: *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings, Lecture Notes in Computer Science 7211*, Springer, pp. 194–213, doi:10.1007/978-3-642-28869-2_10.
- [20] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen C. Hunt, James R. Larus & Steven Levi (2006): *Language support for fast and reliable message-based communication in singularity OS*. In Yolande Berbers & Willy Zwaenepoel, editors: *Proceedings of the 2006 EuroSys Conference, Leuven, Belgium, April 18-21, 2006*, ACM, pp. 177–190, doi:10.1145/1217935.1217953.
- [21] Alain Finkel & Étienne Lozes (2017): *Synchronizability of Communicating Finite State Machines is not Decidable*. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn & Anca Muscholl, editors: *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland, LIPIcs 80, Schloss Dagstuhl - Leibniz-Zentrum für Informatik*, pp. 122:1–122:14, doi:10.4230/LIPIcs.ICALP.2017.122.
- [22] Simon J. Gay & Vasco Thudichum Vasconcelos (2010): *Linear type theory for asynchronous session types*. *J. Funct. Program.* 20(1), pp. 19–50, doi:10.1017/S0956796809990268.
- [23] Thomas Gazagnaire, Blaise Genest, Loïc Hélouët, P. S. Thiagarajan & Shaofa Yang (2007): *Causal Message Sequence Charts*. In Luís Caires & Vasco Thudichum Vasconcelos, editors: *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings, Lecture Notes in Computer Science 4703*, Springer, pp. 166–180, doi:10.1007/978-3-540-74407-8_12.
- [24] Blaise Genest, Dietrich Kuske & Anca Muscholl (2007): *On Communicating Automata with Bounded Channels*. *Fundam. Inform.* 80(1-3), pp. 147–167. Available at <http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-09>.
- [25] Blaise Genest & Anca Muscholl (2005): *Message Sequence Charts: A Survey*. In: *Fifth International Conference on Application of Concurrency to System Design (ACSD 2005), 6-9 June 2005, St. Malo, France*, IEEE Computer Society, pp. 2–4, doi:10.1109/ACSD.2005.25.
- [26] Blaise Genest, Anca Muscholl & Doron A. Peled (2003): *Message Sequence Charts*. In Jörg Desel, Wolfgang Reisig & Grzegorz Rozenberg, editors: *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been*

- commissioned], *Lecture Notes in Computer Science* 3098, Springer, pp. 537–558, doi:10.1007/978-3-540-27755-2_15.
- [27] Blaise Genest, Anca Muscholl, Helmut Seidl & Marc Zeitoun (2006): *Infinite-state high-level MSCs: Model-checking and realizability*. *J. Comput. Syst. Sci.* 72(4), pp. 617–647, doi:10.1016/j.jcss.2005.09.007.
- [28] Cinzia Di Giusto, Laetitia Laversa & Étienne Lozes (2020): *On the k-synchronizability of Systems*. In Jean Goubault-Larrecq & Barbara König, editors: *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Lecture Notes in Computer Science* 12077, Springer, pp. 157–176, doi:10.1007/978-3-030-45231-5_9.
- [29] Loïc Hélouët & Claude Jard (2000): *Conditions for synthesis of communicating automata from HMSCs*. In: *In 5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*.
- [30] Alexander Heußner, Jérôme Leroux, Anca Muscholl & Grégoire Sutre (2012): *Reachability Analysis of Communicating Pushdown Systems*. *Log. Methods Comput. Sci.* 8(3), doi:10.2168/LMCS-8(3:23)2012.
- [31] Kohei Honda (1993): *Types for Dyadic Interaction*. In Eike Best, editor: *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings, Lecture Notes in Computer Science* 715, Springer, pp. 509–523, doi:10.1007/3-540-57208-2_35.
- [32] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In George C. Necula & Philip Wadler, editors: *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008, ACM*, pp. 273–284, doi:10.1145/1328438.1328472.
- [33] Kohei Honda, Nobuko Yoshida & Marco Carbone (2016): *Multiparty Asynchronous Session Types*. *J. ACM* 63(1), pp. 9:1–9:67, doi:10.1145/2827695.
- [34] Chris Köcher (2021): *Reachability Problems on Reliable and Lossy Queue Automata*. *Theory Comput. Syst.* 65(8), pp. 1211–1242, doi:10.1007/s00224-021-10031-2.
- [35] Julien Lange & Nobuko Yoshida (2019): *Verifying Asynchronous Interactions via Communicating Session Automata*. In Isil Dillig & Serdar Tasiran, editors: *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I, Lecture Notes in Computer Science* 11561, Springer, pp. 97–117, doi:10.1007/978-3-030-25540-4_6.
- [36] Markus Lohrey (2003): *Realizability of high-level message sequence charts: closing the gaps*. *Theor. Comput. Sci.* 309(1-3), pp. 529–554, doi:10.1016/j.tcs.2003.08.002.
- [37] P. Madhusudan (2001): *Reasoning about Sequential and Branching Behaviours of Message Sequence Graphs*. In Fernando Orejas, Paul G. Spirakis & Jan van Leeuwen, editors: *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings, Lecture Notes in Computer Science* 2076, Springer, pp. 809–820, doi:10.1007/3-540-48224-5_66.
- [38] Rupak Majumdar, Madhavan Mukund, Felix Stutz & Damien Zufferey (2021): *Generalising Projection in Asynchronous Multiparty Session Types*. In Serge Haddad & Daniele Varacca, editors: *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference, LIPIcs* 203, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 35:1–35:24, doi:10.4230/LIPIcs.CONCUR.2021.35.
- [39] Sjouke Mauw & Michel A. Reniers (1997): *High-level message sequence charts*. In Ana R. Cavalli & Amardeo Sarma, editors: *SDL '97 Time for Testing, SDL, MSC and Trends - 8th International SDL Forum, Evry, France, 23-29 September 1997, Proceedings*, Elsevier, pp. 291–306.
- [40] Arjan J. Mooij, Nicolae Goga & Judi Romijn (2005): *Non-local Choice and Beyond: Intricacies of MSC Choice Nodes*. In Maura Cerioli, editor: *Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings, Lecture Notes in Computer Science* 3442, Springer, pp. 273–288, doi:10.1007/978-3-540-31984-9_21.
- [41] Henry Muccini (2003): *Detecting Implied Scenarios Analyzing Non-local Branching Choices*. In Mauro Pezzè, editor: *Fundamental Approaches to Software Engineering, 6th International Conference, FASE 2003,*

- Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings, Lecture Notes in Computer Science 2621, Springer, pp. 372–386, doi:10.1007/3-540-36578-8_26.
- [42] Wuxu Peng & S. Purushothaman (1992): *Analysis of a Class of Communicating Finite State Machines*. *Acta Informatica* 29(6/7), pp. 499–522, doi:10.1007/BF01185558.
- [43] Abhik Roychoudhury, Ankit Goel & Bikram Sengupta (2012): *Symbolic Message Sequence Charts*. *ACM Trans. Softw. Eng. Methodol.* 21(2), pp. 12:1–12:44, doi:10.1145/2089116.2089122.
- [44] Alceste Scalas & Nobuko Yoshida (2019): *Less is more: multiparty session types revisited*. *Proc. ACM Program. Lang.* 3(POPL), pp. 30:1–30:29, doi:10.1145/3290343.
- [45] Felix Stutz & Damien Zufferey (2022): *Comparing Channel Restrictions of Communicating State Machines, High-level Message Sequence Charts, and Multiparty Session Types*. CoRR abs/2208.05559, doi:10.48550/arXiv.2208.05559. arXiv:2208.05559.
- [46] Salvatore La Torre, P. Madhusudan & Gennaro Parlato (2008): *Context-Bounded Analysis of Concurrent Queue Systems*. In C. R. Ramakrishnan & Jakob Rehof, editors: *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings, Lecture Notes in Computer Science 4963*, Springer, pp. 299–314, doi:10.1007/978-3-540-78800-3_21.
- [47] Tayssir Touili & Mohamed Faouzi Atig (2010): *Verifying parallel programs with dynamic communication structures*. *Theor. Comput. Sci.* 411(38-39), pp. 3460–3468, doi:10.1016/j.tcs.2010.05.028.
- [48] International Telecommunication Union (1996): *Z.120: Message Sequence Chart*. Technical Report, International Telecommunication Union. Available at <https://www.itu.int/rec/T-REC-Z.120>.

Characterizing the Decidability of Finite State Automata Team Games with Communication

Michael Coulombe

Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA
mcoulomb@mit.edu

Jayson Lynch

Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada
jayson.lynch@uwaterloo.ca

In this paper we define a new model of limited communication for multiplayer team games of imperfect information. We prove that the Team DFA Game and Team Formula Game, which have bounded state, remain undecidable when players have a rate of communication which is less than the rate at which they make moves in the game. We also show that meeting this communication threshold causes these games to be decidable.

1 Introduction

Deciding optimal play in multiplayer games of incomplete information is known to be an undecidable problem [11, 10]. This includes games where the state space is bounded, a surprising result first shown of a collection of abstract computation games [6] that has been extended to generalized versions of real games, like Team Fortress 2 and Super Smash Bros [5]. However, past work has relied on the complete inability of teammates to communicate during the game, which is often not a realistic assumption. In this paper we study deterministic models of communication between players in two of these computation games, the Team DFA Game and the Team Formula Game, and show a sharp change in computational complexity based on whether players are able to eventually communicate all of their moves or only able to communicate a constant fraction.

One motivation for this model is a better understanding of real world games. Many team games played in-person naturally permit free form communication between teammates to coordinate their actions, and online multiplayer video games often provide communication channels such as voice-chat, text, and emotes to simulate this in-person environment. These include many FPS games such as Team Fortress and Left4Dead, MOBAs such as DOTA2 and League of Legends, and RTS games such as Starcraft and Age of Empires. Some of these examples have drawn research interest in AI/ML [14, 3] as well as computational complexity [13, 5]. The real-time nature of these games ensures that communication channels are bounded; however, modeling free form communication, as well as efficiently implementing meaningful player choices in a real-time setting, makes it difficult to analyze these games with these communication features enabled.

Outside of the team setting, communication is a central aspect of many other games. For example, in the cooperative card game Hanabi players are unable to see their own hands of cards, but this information is visible to everyone else. In addition, players are not allowed to communicate except through actions in the game, one of which allows players to reveal partial information about what is in another players hand. A perfect information version of Hanabi was shown to be NP-complete [1]. The Crew: Quest for

Planet Nine is a cooperative trick taking card game which uses limited communication between players as a core mechanic. The complexity of this game was also studied in the perfect information setting [12]. Under the limited information setting, containment in NP for both of these games is not obvious, and we see a need for models of player communication in games. Other examples of cooperative boardgames with limited communication channels between players include *Mysterium*, *The Mind*, and *Magic Maze*.

Other games may limit communication simply with time pressure in the game. Examples of fully cooperative games with imperfect information that use time pressure to limit coordination and communication include *Space Alert*, *5-Minute Dungeon*, *Keep Talking and Nobody Explodes*, and *Spaceteam*.

Multi-agent, imperfect information games are also a topic of interest in Reinforcement Learning. In [4] algorithms are developed to address team extensive form games of imperfect information where communication is allowed at certain points during gameplay, with *Bridge* and collusion among some players between hands in *poker* being the motivating examples. Other work considers *Sequential Social Dilemmas*, a type of iterated economic game where in any given instant a player is incentivized towards non-cooperative behavior, but cooperative strategies can obtain higher payoff over the game as a whole. Learning algorithms for these models both with and without explicit bounded communication channels were studied in [8]. Purely cooperative settings have also been of interest [7].

One major achievement was human level performance on a limited version of *DOTA2*, a MOBA-style video game [3]. These are real-time, team games with partially observable state. Although both text and voice chat are typically allowed in professional play, the AI system did not utilize these explicit communication mechanisms. The board game *Diplomacy*, while not explicitly a team game, features coordination and temporary alliances between players as a core game dynamic. This game has also seen interest as a new challenge in the AI community, but focusing on *No-Press Diplomacy* which does not allow explicit communication between players [9, 2].

These examples of both AI and computational complexity research which considers games with cooperation and communication motivate, but frequently ignore the important role of communication in these games, motivates this paper.

Paper Organization. In Section 1.2 we formally define our model of communication for the Team DFA Game. In Section 2 we prove undecidability for a few simple communication patterns to help build intuition for the techniques used in the next section. In Section 3 we prove our main undecidability result for Team DFA Games with Communication. In Section 4 we prove the game becomes decidable when either both players can communicate all information about their moves, or one player receives no information but can communicate all of their moves to their teammate. In Section 5 we show that analogous algorithms and undecidability results hold for Team Formula Games.

1.1 Team DFA Game

The Team DFA Game is a bounded state, two-vs-one multiplayer game, defined by [5] as a simplification of the undecidable Team Computation Game [6]. It involves a team of two players, the \exists players, and an adversary, the \forall player, who take turns sending a bit to a deterministic finite automaton (DFA). Each team's goal is to put the DFA into any of a set of winning final states for their team. The \forall player knows the moves of the \exists players and thus the state of the DFA; however the \exists players do not know each other's moves, which they must make after learning only one of \forall 's moves each turn.

Definition 1. The Team DFA Game (TDG) is a two-versus-one team game. An instance of the game is a DFA $D = (\{0, 1\}, Q, q_0, \delta, F = F_{\exists} \cup F_{\forall})$. The existential team $\{\exists_0, \exists_1\}$ competes against the universal team $\{\forall\}$. The game starts with D in state q_0 and each round proceeds as follows:

1. If D 's state $q \in F_{\exists}$ then team existential wins. If $q \in F_{\forall}$ then team universal wins.
2. \forall learns the state q of D then inputs two bits b_0, b_1 into D .
3. \exists_0 learns b_0 then inputs one bit m_0 into D . \forall learns m_0 .
4. \exists_1 learns b_1 then inputs one bit m_1 into D . \forall learns m_1 .

The problem we consider in this paper is: given an instance of the game, does a particular team have a *forced win*? More formally, does there exist a strategy function s_i for each player i on the team, specifying on each turn which move to make based on any information they have learned so far, that when followed will guarantee that this team will win? In this paper, we define the complexity of a game as the complexity of whether a specified team has a forced win in the game, such as in the following:

Theorem 1 (previous work [5]). *The Team DFA Game is undecidable.*

A number of variations of this game, all undecidable in the general case, exist. These include Team Computation Game where players give inputs to a Turing machine on a bounded tape [11], Team Constraint Logic Game where players make moves in a partially observable constraint logic graph [6]; and Team Formula Game where players flip the values of Boolean variables trying to satisfy different formulas [10, 6].

1.2 Communication Model

We model communication in the Team DFA Game with a policy that specifies the bandwidth of a dynamic information channel, as one might have due to natural factors (e.g. playing a real-time game with voice chat) or intentional game design (e.g. Hanabi's card-revealing moves) allowing a predictable but bounded amount of player-to-player communication between moves. Specifically, a policy P is a DFA over a unary alphabet with functions P_{MID}, P_{END} over its states. In a round of the game in policy state p , $P_{MID}(p)$ is the number of bits which are exchanged simultaneously between \exists_0 and \exists_1 after (b_0, b_1) are revealed but before (m_0, m_1) must be determined, and similarly $P_{END}(p)$ is the number of bits to exchange after (m_0, m_1) are submitted but before the next round starts.

Definition 2. The Team DFA Game with Communication (TDGC) is a game of the existential team $\{\exists_0, \exists_1\}$ versus the universal team $\{\forall\}$, extending the Team DFA Game. An instance of the game is a pair of a game DFA $D = (\{0, 1\}, Q, q_0, \delta, F_{\exists} \cup F_{\forall})$ and a policy P , which consists of a policy DFA $(\{1\}, \Pi, p_0, \pi, \theta)$ and functions $P_{MID}, P_{END} : \Pi \rightarrow \mathbb{N} \times \mathbb{N}$. The game starts with D in state q_0 and the policy DFA in state p_0 , and each round proceeds with added communication steps as illustrated in Figure 1.

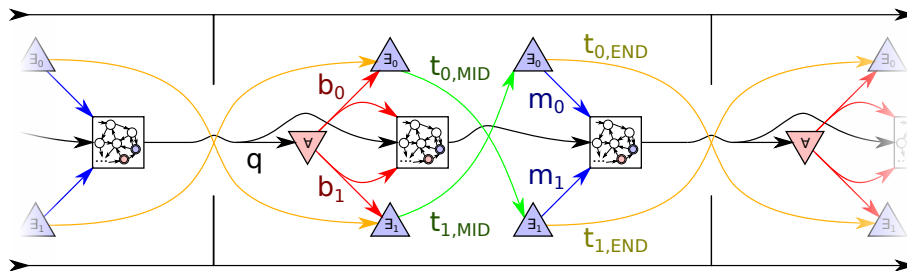


Figure 1: Information flow graph of one round of the Team DFA Game with Communication, including from the previous round and into the next round. New to this game are the mid-round transmissions, $t_{0,MID}$ and $t_{1,MID}$, and the end-of-round transmissions, $t_{0,END}$ and $t_{1,END}$, which have sizes determined by P_{MID} and P_{END} applied to the policy state.

There are two beneficial aspects of studying policies as DFAs on unary alphabets: bounding the state space allows for the policy to be computable by the mechanics of a bounded-state game (such as the DFA in the Team DFA Game), and giving every state exactly one next state (for the next round of the game) means the bandwidth every round will be known in advance when building our constructions, rather than being dependent on player actions. As a result of this choice, it is important to note that the shape of its state transition graph will always have the form: from the start state, there is an initial chain of unique states (possibly of length zero) that leads to a cycle of periodically-repeating states. Also shown in Figure 2.

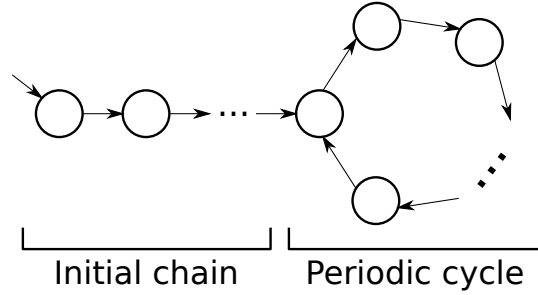


Figure 2: General form of a policy DFA: an initial chain followed by a cycle.

2 Undecidability of Simple Communication Games

In this section, we will explore some basic classes of policies that preserve the undecidability of the Team DFA Game with Communication. Our proof technique is to reduce from the zero-communication Team DFA Game, where we compensate for the message passing by “clogging the channel” with the forced transmission of garbage bits that do not facilitate information sharing. Section 3 builds upon these examples to obtain more general results, however proving the special cases in this section allows us to introduce ideas needed in the full proof and discuss some of the techniques more concretely.

For each class of policies \mathcal{P} below, we will show that given any policy $P \in \mathcal{P}$ and DFA D for playing TDG, we can produce a DFA D' for playing TDGC under P such that the \exists team has a forced win on D with no channel iff they have a forced win on D' given a channel following policy P . As TDG is undecidable, so will be TDGC under any policy $P \in \mathcal{P}$. For simplicity, we consider policies with DFA C_r , a length- r cycle of states $\Pi = \{0, 1, \dots, r-1\}$ with no initial chain, for arbitrary values of r .

Theorem 2. *TDGC is undecidable with a 1 bit mid-round exchange every $r \geq 2$ rounds: policies P where $P_{\text{MID}}(p) = (1, 1)$ if $p \equiv 0 \pmod r$, $P_{\text{MID}}(p) = (0, 0)$ otherwise, and $P_{\text{END}}(p) = (0, 0)$.*

Proof. We construct a DFA D' by first augmenting the state q of D with the state p of C_r . When $p \not\equiv 0 \pmod r$, D' simply simulates D for one round. However, when $p \equiv 0 \pmod r$, D' instead takes the inputs (b_0, b_1, m_0, m_1) and tests $b_0 = m_1 \wedge b_1 = m_0$. If the test fails, then D' enters a final state for \forall .

How D' clogs the channel is diagrammed in Figure 3. By tracking the round number, it knows exactly when \exists_0 and \exists_1 will exchange bits, and in that round D' expects \exists_0 to guess b_1 , a bit that \exists_0 does not learn by the game procedure, and vice-versa. \exists_0 and \exists_1 are forced to spend their single bit of communication on exchanging b_0 and b_1 to their teammate, in order to guarantee survival against any \forall strategy for choosing b_0 and b_1 .

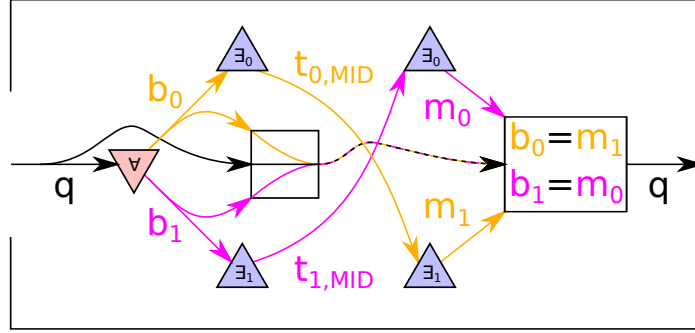


Figure 3: Mid-round 1-bit channel clogging technique. Values with the same color must be equal, namely $t_i = b_i = m_{1-i}$, or else the DFA permanently enters F_V .

Since \exists_0 and \exists_1 do not learn anything from each other or alter the simulated D 's state in the rounds with communication, they have a winning strategy on D' playing TDGC under P if and only if they have a strategy for the non-exchanging rounds (which happen infinitely-often since $r \geq 2$) that would give a winning strategy on D playing TDG. \square

Theorem 3. *TDGC is undecidable with n rounds of 1-bit mid-round exchanges across $r > n$ rounds: policies P where $P_{\text{MID}}(p) \in \{(0,0), (1,1)\}$ with pre-image size $|P_{\text{MID}}^{-1}((1,1))| = n$ and $P_{\text{END}}(p) = (0,0)$.*

Proof. We generalize Theorem 2 by constructing a DFA D' which clogs the channel on any round $p \pmod r$ in which $P_{\text{MID}}(p) = (1,1)$ and simulates D in the other $r - n > 0$ out of r rounds. By the same argument, this prevents communication between \exists_0 and \exists_1 while playing TDGC beyond the corresponding play of TDG taking place during non-exchanging rounds, and thus preserves forced win-ability. \square

3 Undecidability of General Communication Games

This section proves our main result: that a broad class of policies with sufficiently low communication rate remain undecidable for the Team DFA game. We now define this more general notion.

Definition 3. A policy is (r, x_0, x_1, N) -**rate-limited** if, after a fixed number of rounds N , the rate of transmission from player \exists_i to \exists_{1-i} is x_i during every period of r rounds. Specifically, it must satisfy

$$x_i = \sum_{k=k_0}^{k_0+r-1} P_{\text{MID}}(p_k)[i] + P_{\text{END}}(p_k)[i] \text{ for } k_0 = N + \ell r, \text{ where } \ell \in \mathbb{N} \text{ and } p_k \text{ is the policy state on round } k.$$

This now allows us to state the main theorem.

Theorem 4. *TDGC is undecidable under all (r, x_0, x_1, N) -rate-limited policies where $x_0, x_1 < r$.*

3.1 Properties of Rate-Limited Policies

Before proceeding to the proof of Theorem 4, we will establish useful lemmas about rate-limited policies.

First, we have the following two simple observations:

Lemma 1. *Any policy implemented as a unary-alphabet DFA with $n > 1$ states is (r, x_0, x_1, N) -rate-limited for some $1 < r \leq n$ and some initial segment of length $N \leq n$.*

Lemma 2. *Any (r, x_0, x_1, N) -rate-limited policy is also $(2r, 2x_0, 2x_1, N)$ -rate-limited if $r > 1$.*

Next, we will need the following property bounding the partial sums of certain repeated finite sequences for analyzing the transmission rates in each part of a round across a period.

Definition 4. Let a be any sequence of $2n$ natural numbers $(a_0, a_1, \dots, a_{2n-1})$ with sum at most $n - 1$, and let i be an index into a . ROTATE-BOUNDED(a, i) is the predicate that holds when the infinite sequence $b_j^{(i)} = a_{(i+j \bmod 2n)}$ with partial sums $B_j^{(i)} = \sum_{k=0}^{j-1} b_k^{(i)}$ satisfies $\forall j > 0. B_j^{(i)} < \frac{j}{2}$.

Lemma 3. For any such a , ROTATE-BOUNDED(a, i) holds for some even index i .

Proof. Let $C_j^{(i)} = B_j^{(i)} - \frac{j}{2}$. Let us find an even index i such that $C_j^{(i)} < 0$ for all $j > 0$, and consider the largest length $j^* < 2n$ which maximizes $C_{j^*}^{(0)}$. If $C_{j^*}^{(0)} < 0$ then $i = 0$ satisfies the claim, so suppose $C_{j^*}^{(0)} \geq 0$.

Because the sum of a is at most $n - 1$, notice that for any j , $j + 2n$ in the next period satisfies $C_{j+2n}^{(0)} = B_{j+2n}^{(0)} - \frac{j+2n}{2} < (B_j^{(0)} + n) - \left(\frac{j}{2} + n\right) = C_j^{(0)}$. Since j^* is the largest maximizer in the first period, for all $j > j^*$:

$$0 > C_j^{(0)} - C_{j^*}^{(0)} = \left(B_j^{(0)} - \frac{j}{2}\right) - \left(B_{j^*}^{(0)} - \frac{j^*}{2}\right) = B_{j-j^*}^{(j^*)} - \frac{j-j^*}{2} = C_{j-j^*}^{(j^*)}$$

and therefore $\forall j > 0. C_j^{(j^*)} < 0$, thus $i = j^*$ is an index for which a is rotate bounded. If j^* is even we are done. If j^* is odd, then we know that $j^* < 2n - 2$ because we supposed that $C_{j^*}^{(0)} \geq 0$ whereas $C_{2n-1}^{(0)} < 0$:

$$C_{2n-1}^{(0)} = B_{2n-1}^{(0)} - \frac{2n-1}{2} \leq (n-1 - a_{2n-1}) - \left(n - \frac{1}{2}\right) = -a_{2n-1} - \frac{1}{2} < 0$$

Thus, consider the even $j^* + 1$ and let $j' \in [j^* + 1, 2n)$ be the maximum length such that $C_{j'}^{(0)} = C_{j^*+1}^{(0)}$. $0 > C_{j^*+1}^{(0)} - C_{j^*}^{(0)} = C_1^{(j^*)} = b_0^{(j^*)} - \frac{1}{2}$ therefore $b_0^{(j^*)} = 0$ and $C_{j'}^{(0)} = C_{j^*+1}^{(0)} = C_{j^*}^{(0)} - \frac{1}{2}$. Since $B_{j^*}^{(0)}$ is an integer and j^* is odd, j' must also be even, and by similar arguments $\forall j > j'. C_j^{(0)} > C_{j'}^{(0)}$, thus $\forall j > 0. C_j^{(j')}$ is an integer and $C_j^{(j')} < 0$, so $i = j'$ satisfies the claim. \square

Corollary 1. For any such a , ROTATE-BOUNDED(a, i) holds for some odd index i .

Proof. Consider $a' = (a_1, a_2, \dots, a_{2n-1}, a_0)$. By Lemma 3, there is an even index i' which satisfies ROTATE-BOUNDED(a', i'). Thus $i \equiv i' + 1$ is an odd index such that ROTATE-BOUNDED(a, i). \square

3.2 Construction Outline

First, we introduce our reduction from the Team DFA Game to the Team DFA Game with Communication. Given an (r, x_0, x_1, N) -rate-limited policy P and an underlying DFA D , we create a DFA D' for playing the TDGC under P which simulates playing the TDG on D while completely clogging the communication between the \exists team to nullify any advantage such communication could bring. This lets us conclude that a winning strategy for TDGC on D' exists exactly when a winning strategy exists for TDG on D .

The reduction applies when each $x_i < r$, meaning the communication rate defined by P is eventually below an average of one bit per round. We also assume $r > 1$ and each $x_i > 0$: if there is no communication at all then TDGC is identical to TDG, and if communication only occurs in one direction then the

aspects of the construction that deal with the silent direction may be omitted. Lastly, by Lemma 2, we take period length r to be even without loss of generality.

The code for D' is fully shown in Algorithm 1. The behavior of D' is designed around what we call the *honest* strategy for the \exists team. We will show that it is the only strategy that guarantees the \exists team will pass validation checks by D' , but also that it requires using all available transmission bits, resulting in no information transfer between players for their additional benefit in the simulated TDG.

Along with the current state of D in the TDG, D' maintains two queues X_0, X_1 of clogging bits that have been given to each \exists player by the adversary \forall in specific rounds. These bits are expected to be submitted by the opposite \exists player to D' for validation in later rounds in order to avoid losing the game, so the players are forced to use transmission bandwidth to exchange this information. The honest \exists_i player sends these bits directly and as soon as possible to \exists_{1-i} , who maintains a “knowledge” queue K_{1-i} of all bits sent from \exists_i but not yet validated by D' . We note that $X_i \setminus K_{1-i}$ is thus the set of yet-to-be-transmitted private bits known only to \exists_i .

3.3 Build-up Phase

D' begins the build-up phase after N rounds, once P has started to repeat its policy states. This phase lasts for exactly r^2 rounds, or r periods of P 's cycle. D' starts with empty X_0 and X_1 , and every round D' simply enqueues b_0 and b_1 into the appropriate queues.

During these rounds, \forall can send one bit per round to \exists_i , who can transmit those bits to \exists_{1-i} , for each $i \in \{0, 1\}$. Because the rate of transmission can vary above or below one bit per round, there is some maximum amount $x'_i \leq x_i$ out of r bits that can reach \exists_{1-i} in the first r rounds. Each subsequent r rounds, x_i out of the r new bits can be sent (by the rate-limitedness of P), thus after r^2 rounds at most $x'_i + (r-1)x_i \leq rx_i < r(r-1)$ bits in X_i can be sent to \exists_{1-i} and thus at least r are not known. By this argument, at the end of the build-up phase we can say that an honest player's knowledge queue has size $|K_i| = x'_{1-i} + (r-1)x_{1-i} \in [r-1, r(r-1)]$, since we assume $x_{1-i} \geq 1$.

3.4 Clogging Phase

In the clogging phase, D' simulates playing TDG on D while clogging the transmissions between \exists players at a steady rate to keep $|X_i|$ and $|K_i|$ constant on period boundaries. In the last round of every period of r rounds, D' alternates between (1) having $\forall, \exists_0, \exists_1$ play one round of TDG, and (2) forcing \forall to tell the \exists team if they have won in the TDG yet and therefore if D' is going to start the next phase: the tear-down phase.

In the first $r-1$ rounds of each period in this phase, D' clogs the transmissions between \exists players by requiring that bits given to \exists_i by \forall (placed into queue X_i) are sent to \exists_{1-i} . This is done by dequeuing the oldest bit b from X_i and checking for \exists_{1-i} to submit $m_{1-i} = b$, otherwise they will lose the game. Specifically, to preserve the size of X_i and keep up with the rates at which the \exists players can transmit information to each other, D' will do $\text{ENQ}(X_i, b_i)$ then validate $\text{DEQ}(X_i) = m_{1-i}$ for the first x_i rounds of each period.

Across the whole period, K_i will gain x_{1-i} new bits transmitted from \exists_{1-i} (by the rate-limitedness of P). New available bits always exist because the number of private bits available to be sent is $|X_{1-i} \setminus K_i| \geq r > x_{1-i}$ at the start of the period. Additionally, across the first x_{1-i} rounds of the period, K_i will lose x_{1-i} bits submitted by \exists_i to D' , which are always known because $|K_i| \geq (r-1)x_{1-i} \geq x_{1-i}$ at the start. Overall, this means $|K_i|$ is preserved on period boundaries and honest players will always be able to submit the correct bit and pass the validation.

Labeling the first clogging period with index 0, at the end of every odd-indexed period, D' will simulate TDG by forwarding the inputs of all players directly to D . However, at the end of even-indexed periods, D' will ignore \exists_0, \exists_1 and expect both \forall bits to state whether or not the \exists team has won in TDG, specifically requiring that $b_0 = b_1 = [q \in F_{\exists}]$. If this validation fails, then D' will halt with a \exists team victory, so the \forall player must give the correct information to both \exists players to avoid losing, which it is always able to do.

Assuming validation never fails, which is achieved by the honest strategy, the clogging phase continues until the simulated Team DFA Game ends. If the \exists players lose in the simulation, they lose immediately, otherwise after the even-indexed period when the \exists players learn they have won, D' moves onto the tear-down phase to perform the final validation checks.

3.5 Tear-down Phase

The tear-down phase starts at the beginning of a period, so by the previous arguments for queue size preservation, it starts with $|X_i| = r^2$ and $|K_{1-i}| = x'_i + (r-1)x_i$. In order to ensure the \exists team's transmissions have been completely clogged all the way until the simulated victory, D' must validate that the remaining bits in K_i have actually been sent by this point.

This phase is split into two parts, with a boosting sub-phase to adjust the size of X_i and K_{1-i} for the following draining sub-phase that empties them. Once each queue has been drained and all validation checks have been passed, then D' will halt with an \exists team victory. We will need the following fact:

Lemma 4. *There exists a k_{end} such that, in every round up to the k_{end}^{th} round of a period, the cumulative number of bits \exists_0 will transmit to \exists_1 before \exists_1 submits a bit to D' in the k_{end}^{th} round is always upper-bounded by the cumulative number of bits \exists_1 will submit to D' in that time (from round N onwards).*

Proof. Say the period begins in round $m \geq N$, and recall that we can assume the period length r is even. Consider the sequence a_k of the number of bits transmitted from \exists_0 to \exists_1 in the k half-rounds starting in round m , so $a_k = P_{MID}(p_{m+k/2})[0]$ when k is even and $a_k = P_{END}(p_{m+(k-1)/2})[0]$ when k is odd. Since policy states repeat, $\forall k \geq 0. a_{k+r} = a_k$, and $a_0 + \dots + a_{r-1} = x_0 < r$, so we can apply Corollary 1 to the reversed sequence (a_{r-1}, \dots, a_0) to get an odd index i such that $\forall j > 0. B_j^{(i)} < \frac{j}{2}$.

Since $B_j^{(i)}$ is the cumulative number of bits transmitted from \exists_0 to \exists_1 across the j half-rounds ending when \exists_1 submits a bit to D' in round $m + \frac{r-1-i}{2}$, and $\lceil \frac{j+1}{2} \rceil \geq \frac{j}{2}$ is the cumulative number of bits \exists_1 submits to D' across the same set of j half-rounds, then round offset $k_{end} = \frac{r-i-1}{2}$ satisfies Lemma 4. \square

Draining Sub-Phase Given k_{end} from Lemma 4 (by symmetry, the lemma applies in both directions), let $t_{end} \leq k_{end}$ be the total number of bits transmitted from the beginning of a period until the bit submission in the k_{end}^{th} round. If a period starts with $|X_i| \leq k_{end}$ and $|X_i \setminus K_{1-i}| = t_{end}$, then we can have D' validate bits in the $|X_i|$ rounds before the k_{end}^{th} round and reach $|X_i| = |K_{1-i}| = 0$ where each of the t_{end} transmitted bits are clogging bits from $X_i \setminus K_{1-i}$ with no room for extra communication from \exists_i to \exists_{1-i} .

In order to ensure some period starts with $|X_i| \leq k_{end}$ and $|X_i \setminus K_{1-i}| = t_{end}$ we use some n_i periods beforehand to drain each queue appropriately. Since in each period there are x_i transmission bits (fixed) and up to r validated bits (based on D'), it suffices to have $|X_i| \leq n_i r + k_{end}$ and $|X_i \setminus K_{1-i}| = n_i x_i + t_{end}$.

Boosting Sub-Phase The tear-down phase must start with $|X_i \setminus K_{1-i}| = r^2 - (x'_i + (r-1)x_i) \geq r-1$, but this may not be $n_i x_i + t_{end}$ for any n_i , so before $n_i + 1$ draining periods, we will have additional periods

to increase the number of private bits by $\delta_i = (n_i x_i + t_{end}) - (r^2 - (x'_i + (r-1)x_i))$. So for $\delta_i \geq 0$, we can choose any sufficiently-large n_i .

After one period where \forall gives c_i new clogging bits to \exists_i and D' validates v_{1-i} bits from \exists_{1-i} , we would have $\Delta|X_i| = c_i - v_{1-i}$ and $\Delta|K_{1-i}| = x_i - v_{1-i}$ (given that \exists_i initially has $|X_i \setminus K_{1-i}| \geq x_i$ private bits to transmit to \exists_{1-i}), thus $\Delta|X_i \setminus K_{1-i}| = c_i - x_i$. Therefore, if we set $c_i = x_i + 1 \leq r$ and $v_{1-i} = x_i$, then we get $\Delta|X_i| = +1$, $\Delta|K_{1-i}| = 0$, and $\Delta|X_i \setminus K_{1-i}| = +1$. If $\delta_i < \delta_{1-i}$, then to delay we also need ‘‘filler’’ rounds with no change to the sizes of any queues, which can be achieved by setting $c_i = v_{1-i} = x_i$.

To ensure δ_i is positive and $|X_i| \leq n_i r + k_{end}$ at the *end* of this sub-phase, we need to choose an n_i that satisfies the following constraints at the *start* of the tear-down phase:

$$\begin{aligned} 0 &\leq \delta_i = (n_i x_i + t_{end}) - |X_i \setminus K_{1-i}| \\ n_i &\geq (|X_i \setminus K_{1-i}| - t_{end}) / x_i \end{aligned}$$

and

$$\begin{aligned} n_i r + k_{end} &\geq r^2 + \delta_i \\ n_i r + k_{end} &\geq |X_i| + (n_i x_i + t_{end}) - |X_i \setminus K_{1-i}| = |K_{1-i}| + (n_i x_i + t_{end}) \\ n_i &\geq (|K_{1-i}| + t_{end} - k_{end}) / (r - x_i) \end{aligned}$$

We pick n_i to be the smallest natural number satisfying both lower bounds:

$$\begin{aligned} n_i &= \left\lceil \max \left\{ \frac{|X_i \setminus K_{1-i}| - t_{end}}{x_i}, \frac{|K_{1-i}| + t_{end} - k_{end}}{r - x_i} \right\} \right\rceil \\ &= \left\lceil \max \left\{ \frac{r^2 - (x'_i + (r-1)x_i) - t_{end}}{x_i}, \frac{(x'_i + (r-1)x_i) + t_{end} - k_{end}}{r - x_i} \right\} \right\rceil \end{aligned}$$

Since $0 \leq x'_i \leq x_i < r$ and $0 \leq t_{end} \leq k_{end} < r$, we can upper bound $n_i = O(r^2)$.

Putting it all together At the beginning of the tear-down period, D' will run a set of δ_i periods where \forall produces $x_i + 1$ new bits and D' validates x_i bits, followed by $\max\{\delta_0, \delta_1\} - \delta_i$ periods of x_i new and validated bits. After $\delta_{\max} = \max\{\delta_0, \delta_1\}$ rounds, we will have $|X_i| = r^2 + \delta_i$ and $|X_i \setminus K_{1-i}| = n_i x_i + t_{end}$, preserving $|K_{1-i}| = x'_i + (r-1)x_i$. D' will then run n_i periods plus k_{end} rounds ignoring \forall and validating the remainder of X_i (starting $|X_i|$ rounds before the end).

3.6 Proof of Undecidability

Theorem 4. *TDGC is undecidable under all (r, x_0, x_1, N) -rate-limited policies where $x_0, x_1 < r$.*

Proof. We reduce from the Team DFA Game. For any (r, x_0, x_1, N) -rate-limited policy P where $x_0, x_1 < r$, given an input DFA D for playing the TDG, we construct the DFA D' described in Algorithm 1 for playing the TDGC under policy P . Since determining whether or not the \exists team has a forced win in the TDG is undecidable, this reduction will show that the same question of the TDGC under policy P is undecidable as well.

Given the analysis of D' from the previous sections, we first note that D' is indeed a finite automaton: the waiting counter takes on N values; each queue X_i has maximum size $r^2 + \delta_i$ bits, where $n_i = O(r^2)$ so $\delta_i = O(r^3)$; the state q of D has $|Q|$ possible values; and the various other counters require $O(\log r)$ bits each. From beginning to end, the maximum memory requirement is $O(\max\{\log N, r^2 + \log |Q|, r^3\})$ bits, summarizing Table 3.6.

State Category	Space Needed (bits)
HALT(<i>winner</i>)	$\Theta(1)$
WAITING(<i>w</i>)	$\Theta(\log N)$
BUILD-UP(X_0, X_1)	$\Theta(r^2)$
CLOG($X_0, X_1, q, p, k, c_{01}, c_{10}$)	$\Theta(r^2 + \log Q)$
BOOST($X_0, X_1, d_{01}, d_{10}, k, c_{01}, c_{10}$)	$\Theta(r^2 + \delta_{\max})$
DRAIN(X_0, X_1, c_{01}, c_{10})	$\Theta(r^2 + \delta_{\max})$

Table 1: Memory Requirements of D' over the course of the TDGC.

If there is a winning strategy S for the \exists team on D in the TDG, then the corresponding honest strategy described above that plays the simulated TDG using S will be a winning strategy for the \exists team on D' in the TDGC under policy P .

If there is a winning strategy for the \exists team on D' in the TDGC under policy P , then consider any winning execution γ . Since winning requires termination, let C be the number of periods in the clogging phase.

If γ reaches HALT(\exists) in the clogging phase because \forall did not correctly tell the \exists team whether or not $q \in F_{\exists}$, then \forall did not play optimally. Since \forall has perfect information and is allowed to give either 0 or 1 by the game rules, there is an alternate execution γ' where \forall gives the correct answer instead and the game continues, so no \exists team strategy can force a win in this way.

The only other way for the \exists team to win is for γ to reach HALT(\exists) at the end of the draining phase, which means they must pass all of the validation checks by D' .

Phase	ENQ(X_i) Count	DEQ(X_i) Count	Information $\exists_i \rightarrow \exists_{1-i}$
Build-up	r^2	0	$x'_i + (r-1) \times x_i$
Clogging	$C \times x_i$	$C \times x_i$	$C \times x_i$
Boosting	$\delta_{\max} \times x_i + \delta_i$	$\delta_{\max} \times x_i$	$\delta_{\max} \times x_i$
Draining	0	$r^2 + \delta_i$	$n_i \times x_i + t_{end_i}$

Table 2: Accounting of ENQ(X_i), DEQ(X_i), and Information Transfer between players in each phase

Table 2 details the value of three quantities in each phase of the game: the number of bits enqueued into X_i , the number of bits dequeued from X_i , and the amount of meaningful bits of information that can be transmitted from \exists_i to \exists_{1-i} . By the definition of δ_i and some algebra, it can be seen that each column has the same sum; let I be this total quantity of bits.

Because D' validates the value of each dequeued bit, in order for \exists_i to guarantee they pass all validation checks, they must send I bits of information to \exists_{1-i} . However, because I is the maximum amount of information \exists_i can send to \exists_{1-i} , no further information can be sent, which means that in every round in which D' simulates the TDG on D' , \exists_i has the same amount of information about the state q of D as it would when actually playing TDG on D . Therefore, if the \exists team has a winning strategy for playing TDGC on D' under policy P , within it is a winning sub-strategy for them to play the TDG on D . \square

Algorithm 1 Pseudocode for the D' internal update function per round

```

1:  $q' \leftarrow \text{WAITING}(1)$   $\triangleright$  Initial state
2: function DFA-ROUND-UPDATE( $q', b_0, b_1, m_0, m_1$ )
3:   switch  $q'$ 
4:     case HALT(winner)  $\triangleright$  Game is over, with  $q' \in F'_{\text{winner}}$ 
5:       return HALT(winner)
6:     case WAITING( $w$ )  $\triangleright$  Waiting Phase, delaying until policy starts repeating
7:       if  $w < N$  then return WAITING( $w + 1$ )
8:       return BUILD-UP( $\square, \square$ )
9:     case BUILD-UP( $X_0, X_1$ )  $\triangleright$  Build-up Phase, filling up  $X_i$  queues
10:      ENQ( $X_0, b_0$ )
11:      ENQ( $X_1, b_1$ )
12:      if LENGTH( $X_0$ )  $< r^2$  then return BUILD-UP( $X_0, X_1$ )
13:      return CLOG( $X_0, X_1, q_0, \text{EVEN}, r, x_0, x_1$ )
14:     case CLOG( $X_0, X_1, q, p, k, c_{01}, c_{10}$ ) given  $k > 1$   $\triangleright$  Clogging Phase, boosting
15:       for all  $i \in \{0, 1\}$ 
16:         if  $c_{i,1-i} > 0$  then
17:           ENQ( $X_i, b_i$ )
18:           if DEQ( $X_i$ )  $\neq m_{1-i}$  then return HALT( $\forall$ )
19:            $c_{i,1-i} \leftarrow c_{i,1-i} - 1$ 
20:         return CLOG( $X_0, X_1, q, p, k - 1, c_{01}, c_{10}$ )
21:     case CLOG( $X_0, X_1, q, \text{ODD}, 1, 0, 0$ )  $\triangleright$  Clogging Phase, simulating  $D$ 
22:        $q \leftarrow \delta(\delta(\delta(\delta(q, b_0), b_1), m_0), m_1)$ 
23:       return CLOG( $X_0, X_1, q, \text{EVEN}, r, x_0, x_1$ )
24:     case CLOG( $X_0, X_1, q, \text{EVEN}, 1, 0, 0$ )  $\triangleright$  Clogging Phase, testing for  $\exists$  win
25:       if  $\neg(b_0 = b_1 = [q \in F_{\exists}])$  then return HALT( $\exists$ )
26:       if  $q \in F_{\exists}$  then return BOOST( $X_0, X_1, \delta_0, \delta_1, r, x_0, x_1$ )
27:       if  $q \in F_{\forall}$  then return HALT( $\forall$ )
28:       return CLOG( $X_0, X_1, q, \text{ODD}, r, x_0, x_1$ )
29:     case BOOST( $X_0, X_1, d_{01}, d_{10}, k, c_{01}, c_{10}$ ) given  $k > 1$   $\triangleright$  Boost Phase, clogging
30:       for all  $i \in \{0, 1\}$ 
31:         if  $c_{i,1-i} > 0$  then
32:           ENQ( $X_i, b_i$ )
33:           if DEQ( $X_i$ )  $\neq m_{1-i}$  then return HALT( $\forall$ )  $\triangleright$  Return from caller
34:            $c_{i,1-i} \leftarrow c_{i,1-i} - 1$ 
35:         return BOOST( $X_0, X_1, d_{01}, d_{10}, k - 1, c_{01}, c_{10}$ )
36:     case BOOST( $X_0, X_1, d_{01}, d_{10}, 1, 0, 0$ )  $\triangleright$  Boost Phase, new boost bits
37:       for all  $i \in \{0, 1\}$ 
38:         if  $d_{i,1-i} > 0$  then
39:           ENQ( $X_i, b_i$ )
40:            $d_{i,1-i} \leftarrow d_{i,1-i} - 1$ 
41:       if  $d_{01} + d_{10} > 0$  then return BOOST( $X_0, X_1, d_{01}, d_{10}, r, x_0, x_1$ )
42:       return DRAIN( $X_0, X_1, r \times n_0 + k_{\text{end}_0}, r \times n_1 + k_{\text{end}_1}$ )
43:     case DRAIN( $X_0, X_1, c_{01}, c_{10}$ ) given  $c_{01} + c_{10} > 0$   $\triangleright$  Drain Phase, emptying queues
44:       for all  $i \in \{0, 1\}$ 
45:         if  $c_{i,1-i} > 0$  then
46:           if  $|X_i| = c_{i,1-i} \wedge \text{DEQ}(X_i) \neq m_{1-i}$  then return HALT( $\forall$ )
47:            $c_{i,1-i} \leftarrow c_{i,1-i} - 1$ 
48:         return DRAIN( $X_0, X_1, c_{01}, c_{10}$ )
49:     case DRAIN( $\square, \square, 0, 0$ )  $\triangleright$  Drain Phase, finished!
50:     return HALT( $\exists$ )

```

4 Decidability

We show that our general construction from the previous section is tight with respect to the transmission rate between \exists players.

For our precise bounds, we assume the straightforward encoding of the input DFA D with n states as a table for δ containing $2n$ states, a state q_0 , and the states in F_\exists and F_\forall , thus the input size is $\Theta(|Q|)$.

First, we demonstrate $(r, r, r, 0)$ -rate-limited policies under which the Team DFA Game with Communication is not only decidable but in PSPACE. Later we will show more restrictive communication patterns are in EXPSPACE. Recall $(r, r, r, 0)$ -rate-limited policies are the case where both players are allowed to exchange r bits over the course of a period of length r .

Theorem 5. *TDGC is decidable in PSPACE with a 1-bit, mid-round exchange in both directions every round: policies P with $P_{\text{MID}}(p) = (1, 1)$ and $P_{\text{END}}(p) = (0, 0)$ for all $p \in \Pi$.*

Proof. Under such a policy, TDGC becomes a perfect information game. In each round of the game, the optimal play for \exists_i is to send b_i to \exists_{1-i} immediately after receiving it, meaning \exists_{1-i} will know both b_0 and b_1 before it chooses m_{1-i} . Since the \exists team knows the initial state q_0 of D , we can consider strategy functions $s : (q, b_0, b_1) \mapsto (m_0, m_1)$, which both players can use to decide their own next move and know what move their teammate will perform as well, letting them use δ to learn the state q of D in the next round and beyond.

Note that it suffices for the \exists team to have a memoryless strategy because the policy P is constant per round, DFA transitions do not depend on the history of the game, and the adversarial \forall player's choices are not bound by the history either. It also suffices to have a deterministic strategy: if there exists a non-deterministic winning strategy s' , then we can fix $s(q, b_0, b_1)$ to be some (m_0, m_1) with $\Pr[s'(q, b_0, b_1) = (m_0, m_1)] > 0$ because all game executions in which the \exists team plays with deterministic strategy s are possible executions when playing with strategy s' , thus must also be winning.

We show that deciding whether or not the \exists team has a forced win in TDGC under policy P is in PSPACE by giving a brute-force search algorithm. For every strategy s among the $4^{4|Q|}$ possible strategy functions, we construct a game graph G_s where each state $q \in Q \setminus F_\exists$ is a vertex and for all $b_0, b_1 \in \{0, 1\}$, q has an edge to $q' = \delta(\delta(\delta(\delta(q, b_0), b_1), m_0), m_1)$ where $(m_0, m_1) = s(q, b_0, b_1)$ as long as $q' \notin F_\exists$. This means s is a winning strategy if and only if all $q \in F_\forall$ and all cycles are not reachable from q_0 in G_s , since otherwise the traversal corresponds to a losing execution or the start of a potentially non-terminating execution of the game that the \forall player can force to occur. We can thus perform an exhaustive depth-first search from q_0 for a counterexample (of length at most $|Q|$) to decide whether or not s is a winning strategy. Since we only need $\Theta(|Q|)$ space to store the current s , G_s , and depth-first search stack, this algorithm runs in PSPACE. \square

Since it is sufficient to send only one bit of useful information mid-round, we can extend Theorem 5 to higher transmission rates.

Corollary 2. *TDGC is decidable in PSPACE with at least a 1-bit, mid-round exchange in both directions every round: policies P with $P_{\text{MID}}(p)[i] \geq 1$ for all $p \in \Pi$ and each $i \in \{0, 1\}$.*

Next, we consider the decidability of TDGC under $(r, r, 0, 0)$ -rate-limited policies, which is tight given the undecidability of $(r, r - 1, 0, 0)$ -rate-limited policies. This shows that only one member of the team needs to have perfect information.

Theorem 6. *TDGC is decidable in EXPSPACE with a 1-bit, mid-round exchange every round from \exists_0 to \exists_1 , but none from \exists_1 to \exists_0 : policies P with $P_{\text{MID}}(p) = (1, 0)$ and $P_{\text{END}}(p) = (0, 0)$ for all $p \in \Pi$.*

Proof. As described in the proof of Theorem 5, \exists_0 can and should send b_0 to \exists_1 each round to give \exists_1 perfect information, but \exists_0 itself can learn nothing about b_1 . Using the terminology from [11], this asymmetry makes TDGC under P a hierarchical team game. To decide the existence of a winning strategy, we adapt ideas from the proof of Theorem 4 in the same paper that shows $\text{DTIME}\left(2^{2^{2^{cS(n)}}}\right) \supseteq \text{MPA}_2\text{-SPACE}(S(n))$, the languages decided by hierarchical 2-vs-1 private alternation Turing machines in $S(n)$ space.

Consider the set of all possible mid-round configurations (q, b_0, b_1) of the game, which are fully known to \forall and \exists_1 . Define C be the set of possible configurations (b_0, u) of \exists_0 's mid-round knowledge: the known b_0 and the set $u \in \mathcal{P}(Q \times \{b_0\} \times \{0, 1\})$ of possible mid-round configurations given the history of the game thus far. Since two game states with the same $c \in C$ are strategically equivalent from the perspective of \exists_0 (and thus \exists_1 too), a winning strategy only needs to account for the $|C| = 2^{2|Q|+1}$ knowledge configurations in its decision-making.

Given this, we can do a brute-force search as in Theorem 5 over the space of deterministic \exists team strategies $s : c \in C \mapsto (m_0, m_1)$ of size $4^{|C|}$. For each s , we construct the game graph G_s , where $c \in C$ has an outgoing edge representing the outcome of each b_0, b_1 choice of \forall after the \exists players use s to make their moves and \exists_0 updates their knowledge, and then search for counter-example game executions with length up to $|C|$ to decide whether s is a winning strategy. Therefore, TDGC under P is decidable in $\Theta(|C|)$ space, which is exponential in $|Q|$. \square

As before, Theorem 5 extends to higher transmission rates from \exists_0 to \exists_1 (or vice versa), as long as the receiver stays silent.

Corollary 3. *TDGC is decidable in PSPACE with at least a 1-bit, mid-round exchange in one direction every round, but none in the other direction: policies P with $P_{\text{MID}}(p)[i] \geq 1$ and $P_{\text{MID}}(p)[1-i] = P_{\text{END}}(p)[1-i] = 0$ for all $p \in \Pi$ and some $i \in \{0, 1\}$.*

5 Team Formula Games with Communication

Formula games model many types of games. The Team Formula Game was defined and proven undecidable in [6]. We define a communication version of this game and prove results analogous to the ones for TDGC.

Definition 5. A *Team Formula Game* (TFG) instance consists of sets of Boolean variables X, X', Y_1, Y_2 and their initial values; variables $h_0, h_1 \in X$; and Boolean formulas $F(X, X', Y_0, Y_1)$, $F'(X, X')$, and $G(X)$ such that F implies $\neg F'$. The TFG problem asks whether $\{W_0, W_1\}$, team White, has a forced win against $\{B\}$, team Black, in the game that repeats the following steps in order ad infinitum:

1. B sets X to any values. If F and G are true, then Black wins. If F is false, White wins.
2. B sets X' to any values. If F' is false, then White wins.
3. W_0 sets Y_1 to any values.
4. W_1 sets Y_2 to any values.

where B has perfect information but W_i can only see the values of Y_i and h_i .

Definition 6. *Team Formula Game with Communication* (TFGC) is TFG along with a policy P which specifies a number of bits to be transmitted between W_0 and W_1 mid-round (before each step 3) and at the end of the round (after each step 4)

Theorem 7. *TFGC is undecidable under all (r, x_0, x_1, N) -rate-limited policies where $x_0, x_1 < r$.*

Proof. For any such policy P , we reduce from the Team DFA Game with Communication under the same policy P , adapting the reduction done in Theorem 8 of [6] from the Team Computation Game to the Team Formula Game. In the reduction, the White team plays as the \exists team and B plays as \forall while also facilitating the simulation of TDGC in TFGC.

Given a DFA D to play TDGC under P , we first augment D so it will be suitable for the simulation. To each state, we add a 3-value counter to eliminate any four-edge cycles in the transition graph ($t \xrightarrow{\delta} (t+1) \xrightarrow{\delta} (t+2) \xrightarrow{\delta} t \xrightarrow{\delta} (t+1) \neq t$). Also, we add four new states in a path $q_0 \xrightarrow{\delta} q_0^{(1)} \xrightarrow{\delta} q_0^{(2)} \xrightarrow{\delta} q_0^{(3)} \xrightarrow{\delta} q_0^{(4)}$ from a new initial state q_0 to the original initial state $q_0^{(4)}$ in order to delay the first meaningful state transitions until the start of the second round, which is when the first set of player inputs are available.

In the instance of TFGC, we will have (1) variables $h_i = b_i \in X$ and $b'_i \in X'$, representing the \forall player's chosen bits in the current and previous round; (2) $Y_i = \{m_i\}$, containing the \exists_i player's message bit each round; (3) sets of $\Theta(\log |Q|)$ variables $\langle q' \rangle \subset X'$ and $\langle q \rangle \subset X$ that encode the previous state q' and current state q ; (4) and two parity bits $p \in X$ and $p' \in X'$ which B will be required to flip each round. We also choose the initial value of q' to be q_0 so that in step 1 of the first round B will be forced to set q to $q_0^{(4)}$; other initial values are arbitrary.

In step 1, formula F holds if B sets X so $q = \delta(\delta(\delta(\delta(q', b'_0), b'_1), m_0), m_1), q \notin F_{\exists})$, and $p' \neq p$. Formula G will be true if the current state $q \in F_{\forall}$. Thus, when F and G are both true, then in the TDGC the state transition function was correctly implemented and led to a final state where \forall has won, and thus Black wins the TFGC. On the other hand, if F is false, then either B violated the simulation or the TDGC led to a final state where \exists team has won, and thus White wins the TFGC.

In step 2, formula F' will be true if B sets X' such that $q' = q$ and $p' = p$, updating the previous state for the next round to the new state. If F' is false, then B violated the simulation, and thus White wins the TFGC. Additionally, the parity bit checks guarantee that F implies $\neg F'$.

Since this is a faithful simulation where each round of TFGC corresponds exactly to one round of TDGC, and by Theorem 4 it is undecidable whether or not there exists a winning strategy for the \exists team playing TDGC under P , it is also undecidable whether or not there exists a winning strategy for White playing TFGC under the same policy P . \square

The strategy for proving decidability results of Team DFA Game with Communication also be used to give the following tight decidability results on the Team Formula Game with Communication.

Theorem 8. *TFGC is decidable in 2-EXPSpace with a 1-bit, mid-round exchange in both directions every round: policies P with $P_{\text{MID}}(p) = (1, 1)$ and $P_{\text{END}}(p) = (0, 0)$ for all $p \in \Pi$.*

Theorem 9. *TFGC is decidable in 3-EXPSpace with a 1-bit, mid-round exchange every round from W_0 to W_1 , but none from W_1 to W_0 : policies P with $P_{\text{MID}}(p) = (1, 0)$ and $P_{\text{END}}(p) = (0, 0)$ for all $p \in \Pi$.*

6 Open Problems

One exciting question is whether we can prove computational complexity results about real games with communication. It seems plausible that TDGC may be sufficient for applications to games with highly structured communication. We present a number of questions that we think may help strengthen results to allow their application to more real world scenarios or questions we find particularly interesting for their own sake.

One of the main technical questions left open by this work is the complexity for rate-limited policies with $x_0 \geq r$ and $r > x_1 > 0$. We conjecture this case is decidable but our current arguments rely on both players either having full information or no information.

Looking further, there are many interesting variations and extensions of this model to study. Our arguments rely heavily on communication policies having some bounded period which is useful both for algorithms to bound the uncertainty in the game and for undecidability to allow for constructions that simulate a step in a zero information game after a bounded number of rounds. What happens if our policy is described by something more general than a DFA, such as a sequence recognizable by a pushdown automaton?

Similarly, some of our arguments rely on the fact that the game is played on something with bounded state, such as a DFA or Boolean Formula. What happens with team games on more general systems, such as a pushdown automaton or a bounded space Turing Machine?

Many realistic scenarios have noisy communication channels. How does the computational complexity change under different models of noise? We conjecture that there will again be a cutoff based on whether the information capacity of the channel is sufficiently high. However, it is also possible that the small probability of error will compound over these games of unbounded length resulting in different behavior. It would also be interesting to understand what happens when other sources of inherent randomness are introduced to these games.

It is also often the case that one's ability to communicate depends on the state of the environment and potentially the actions of the people involved. Thus having communication policies that depend on player actions or the game state would be another interesting generalization.

We also only consider two players on the Existential Team. We believe that when more players are added, undecidability will emerge if at least two players have imperfect information. However, this should be verified and the details around more complex communication patterns may lead to richer behavior.

Finally, there is an issue when trying to apply these results to real games or real world problems. Our characterization in some sense relies on communication being high or low compared to critical or meaningful choices in the games. Many natural scenarios have a much larger action space than communication rate, however many of those choices may be essentially equivalent or strategically inadvisable. Undecidability proofs such as those for Team Fortress 2 [5] have very inefficient reductions and require significant numbers of in-game actions to simulate one move in the DFA game. This makes a direct application of our results difficult.

Acknowledgements

We would like to thank Erik Demaine and other participants in the class 6.892 Algorithmic Lower Bounds: Fun with Hardness Proofs (Spring 2019) for useful discussion and the suggestion of potential applications. Thanks to Sophie Monahan for editing assistance.

References

- [1] Jean-François Baffier, Man-Kwun Chiu, Yago Diez, Matias Korman, Valia Mitsou, André van Renssen, Marcel Roeloffzen & Yushi Uno (2017): *Hanabi is NP-hard, even for cheaters who look at their cards*. 675, pp. 43–55, doi:10.1016/j.tcs.2017.02.024.
- [2] Anton Bakhtin, David J. Wu, Adam Lerer & Noam Brown (2021): *No-Press Diplomacy from Scratch*. *Advances in Neural Information Processing Systems 34: Annual Conference*

- on *Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 18063–18074. Available at <https://proceedings.neurips.cc/paper/2021/hash/95f2b84de5660ddf45c8a34933a2e66f-Abstract.html>.
- [3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębniak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse et al. (2019): *Dota 2 with large scale deep reinforcement learning*. arXiv:1912.06680.
- [4] Andrea Celli, Marco Ciccone, Raffaele Bongo & Nicola Gatti (2019): *Coordination in adversarial sequential team games via multi-agent deep reinforcement learning*. arXiv:1912.07712.
- [5] Michael J. Coulombe & Jayson Lynch (2018): *Cooperating in Video Games? Impossible! Undecidability of Team Multiplayer Games*. *9th International Conference on Fun with Algorithms (FUN 2018)* 100, pp. 14:1–14:16, doi:10.4230/LIPIcs.FUN.2018.14.
- [6] Erik D. Demaine & Robert A. Hearn (2008): *Constraint logic: A uniform framework for modeling computation as games*. In: *2008 23rd Annual IEEE Conference on Computational Complexity*, IEEE, College Park, MD, USA, pp. 149–162, doi:10.1109/CCC.2008.35.
- [7] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas & Shimon Whiteson (2016): *Learning to Communicate with Deep Multi-Agent Reinforcement Learning*. *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2137–2145. Available at <https://proceedings.neurips.cc/paper/2016/hash/c7635bfd99248a2cdef8249ef7bfbef4-Abstract.html>.
- [8] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, Dj Strouse, Joel Z. Leibo & Nando De Freitas (2019): *Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning*. *Proceedings of the 36th International Conference on Machine Learning 97*, pp. 3040–3049. Available at <https://proceedings.mlr.press/v97/jaques19a.html>.
- [9] Philip Paquette, Yuchen Lu, Steven Bocco, Max O. Smith, Satya Ortiz-Gagne, Jonathan K. Kummerfeld, Joelle Pineau, Satinder Singh & Aaron C. Courville (2019): *No-Press Diplomacy: Modeling Multi-Agent Gameplay*. *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 4476–4487. Available at <https://proceedings.neurips.cc/paper/2019/hash/84b20b1f5a0d103f5710bb67a043cd78-Abstract.html>.
- [10] Gary Peterson, John Reif & Salman Azhar (2001): *Lower bounds for multiplayer noncooperative games of incomplete information*. *Computers & Mathematics with Applications* 41(7-8), pp. 957–992, doi:10.1016/S0898-1221(00)00333-3.
- [11] Gary L. Peterson & John H. Reif (1979): *Multiple-person alternation*. In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, IEEE, San Juan, Puerto Rico, pp. 348–363, doi:10.1109/SFCS.1979.25.
- [12] Frederick Reiber (2021): *The Crew: The Quest for Planet Nine is NP-Complete*. *CoRR*. arXiv:2110.11758.
- [13] Giovanni Viglietta (2014): *Gaming is a hard job, but someone has to do it! Theory of Computing Systems* 54(4), pp. 595–621, doi:10.1007/s00224-013-9497-5.
- [14] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis & David Silver (2019): *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>.