EPTCS 119

Proceedings of the

Fourth International Symposium on Games, Automata, Logics and Formal Verification

Borca di Cadore, Dolomites, Italy, 29-31th August 2013

Edited by: Gabriele Puppis and Tiziano Villa

Published: 16th July 2013
DOI: 10.4204/EPTCS.119
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Preface

This volume contains the proceedings of the *Fourth International Symposium on Games, Automata, Logic and Formal Verification* (GandALF 2013). The symposium took place in Borca di Cadore, Italy, from 29th to 31st of August 2013.

The GandALF symposium was established by a number of Italian computer scientists interested in mathematical logic, automata theory, game theory, and their applications to the specification, design, and verification of complex systems. It aims to provide a forum where people from different areas, and possibly with different backgrounds, can fruitfully interact. Even though the idea of the symposium emerged within the Italian research community, the event has a truly international nature, as witnessed by the composition of the conference committees and by the country distribution of the submitted papers.

In response to the Call for Papers, the program committee received 34 submissions and selected 17 of them to be included in the conference program. Each paper was revised by at least three referees and the selection was based on originality, quality, and relevance to the topics of the symposium. The scientific program consisted of papers on a wide variety of topics, including algorithmic and behavioral game theory, game semantics, formal languages and automata theory, modal and temporal logics, software verification, hybrid systems.

This fourth edition of GandALF has also hosted three invited talks:

- *Games with delay for automaton synthesis*, by Christof Löding (Lehrstuhl Informatik 7, RWTH Aachen University, Germany)
- *New trends in program synthesis*, by Thomas A. Henzinger (IST, Austria)
- *Temporal logic satisfiability for the design of complex systems*, by Alessandro Cimatti and Stefano Tonetta (Center for Information Technology, Fondazione Bruno Kessler, Trento, Italy)

We wish to express our thanks to the authors who submitted extended abstracts for consideration. We would like to thank also the steering committee for giving us the opportunity and the honor to supervise GandALF 2013, as well as the program committee members and the additional reviewers for their excellent work, fruitful discussions and active participation during the evaluation process.

We would like to thank the people, institutions, and companies for contributing to the success of this edition of GandALF. In particular, we gratefully acknowledge the financial support from private and public sponsors, including: Dipartimento d'Informatica - Università di Verona, Dipartimento d'Informatica - Università di Salerno, Comune di Borca di Cadore. We also thank the EasyChair organization for supporting all the tasks related to the selection of contributions, EPTCS and arXiv for hosting the proceedings.

Finally, we would like to extend special thanks to the organizing chair, Pietro Sala, for his care and tireless efforts in making the local arrangements and organizing an attractive social program. Without his dedicated help and diligent work the conference would not have been such a success.

August 2013,
Gabriele Puppis and Tiziano Villa

Program Committee

Luca Aceto, University of Reykjavik, Iceland
Rajeev Alur, University of Pennsylvania, United States
Arnaud Carayol, IGM, Marne-la-Vallee, France
Anuj Dawar, University of Cambridge, United Kingdom
Stephane Demri, CNRS/New York University, United States
Volker Diekert, University of Stuttgart, Germany
Javier Esparza, University of Munich, Germany
Valentin Goranko, Technical University of Denmark, Denmark
Bakhadyr Khoussainov, University of Auckland, New Zealand
Naoki Kobayashi, University of Tokyo, Japan
Stephan Kreutzer, University of Berlin, Germany
Marta Kwiatkowska, University of Oxford, United Kingdom
Martin Lange, University of Kassel, Germany
Angelo Montanari, University of Udine, Italy
Mimmo Parente, University of Salerno, Italy
Adriano Peron, University of Naples, Italy
Gabriele Puppis, LaBRI, Bordeaux, France (*co-chair*)
Alexander Rabinovich, University of Tel Aviv, Israel
Ramaswamy Ramanujam, Institute of Mathematical Sciences, Chennai, India
Jean Francois Raskin, University of Bruxelles, Belgium
Davide Sangiorgi, University of Bologna, Italy
Olivier Serre, LIAFA, Paris, France
Sharon Shoham, Academic College of Tel Aviv Yaffo, Israel
Szymon Torunczyk, University of Warsaw, Poland
Tiziano Villa, University of Verona, Italy (*co-chair*)
Zhilin Wu, State Key Laboratory of Computer Science, China
Hsu-Chun Yen, National Taiwan University, Taiwan

Organizing Chair

Pietro Sala, University of Verona, Italy

Steering Committee

Mikolaj Bojanczyk, University of Warsaw, Poland
Javier Esparza, University of Munich, Germany
Andrea Maggiolo-Schettini, University of Pisa, Italy
Angelo Montanari, University of Udine, Italy
Margherita Napoli, University of Salerno, Italy
Mimmo Parente, University of Salerno, Italy
Wolfgang Thomas, RWTH Aachen University, Germany
Wieslaw Zielonka, University of Paris 7, France

Additional Reviewers

Games with delay for automaton synthesis

Christof Löding
Lehrstuhl Informatik 7
RWTH Aachen University
Germany
loeding@cs.rwth-aachen.de

Abstract

The framework of infinite two-player games is a powerful and flexible tool to verify and synthesize systems from given specifications. The origin of this work is the problem of automatic circuit synthesis from specifications, as posed in [3]. A circuit can be viewed as a device that transforms input sequences of bit vectors into output sequences of bit vectors. If the circuit acts as a kind of control device, then these sequences are assumed to be infinite because the computation should never halt.

The task in synthesis is to construct such a circuit based on a formal specification describing the desired input/output behaviour. This problem setting can be viewed as a game of infinite duration between two players: The first player provides the bit vectors for the input, and the second player produces the output bit vectors. The winning condition of the game is given by the specification. The goal is to find a strategy for the second player, such that all pairs of input/output sequences that can be produced according to the strategy, satisfy the specification. Such a strategy can be seen as a realisation of the specification.

This approach using games as a model for the synthesis problem has been taken in [1], where it is shown that the synthesis problem can be solved by an algorithm for specifications that are written in monadic second-order logic. Furthermore, for a given specification, one can construct a strategy represented by a finite transducer that reads the input sequence and synchronously produces an output sequence such that the resulting pair of input/output sequence satisfies the specification.

An interesting variation of the problem arises when the constructed strategy can use a lookahead: it does not need to produce an output in each step. In the corresponding game this means that the second player, who is in charge of the output, can delay some of his moves. An early decidability result in such a setting has been obtained in [6], where the strategy is allowed to skip a bounded number of moves in order to obtain a bounded look-ahead.

The aim of this presentation is to survey some recent results that have been obtained for games with delay, as for example, games with arbitrary (not necessarily bounded) delay [5], delay games with deterministic pushdown specifications [4], and delay games over finite words for the synthesis of sequential transducers [2].

References

[1] J. Richard Büchi & Lawrence H. Landweber (1969): *Solving sequential conditions by finite-state strategies*. *Transactions of the American Mathematical Society* 138, pp. 295–311, doi:10.2307/1994916.

[2] Arnaud Carayol & Christof Löding: *Uniformization in Automata Theory*. To appear in the Proceedings of the 14th Congress of Logic, Methodology and Philosophy of Science Nancy, July 19-26, 2011.

[3] Alonzo Church (1962): *Logic, Arithmetic and Automata*. In: *Proceedings of the International Congress of Mathematicians*, pp. 23–35.

[4] Wladimir Fridman, Christof Löding & Martin Zimmermann (2011): *Degrees of Lookahead in Context-free Infinite Games*. In Marc Bezem, editor: *Computer Science Logic (CSL'11) - 25th International Workshop/20th Annual Conference of the EACSL*, *Leibniz International Proceedings in Informatics (LIPIcs)* 12, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 264–276, doi:10.4230/LIPIcs.CSL.2011.264.

[5] Michael Holtmann, Łukasz Kaiser & Wolfgang Thomas (2010): *Degrees of Lookahead in Regular Infinite Games*. In: *Foundations of Software Science and Computational Structures*, *Lecture Notes in Computer Science* 6014, Springer, pp. 252–266, doi:10.1007/978-3-642-12032-9_18.

[6] Frederick A. Hosch & Lawrence H. Landweber (1972): *Finite Delay Solutions for Sequential Conditions*. In: *ICALP*, pp. 45–60.

New Trends in Program Synthesis

Thomas A. Hemzinger
IST Austria
tah@ist.ac.at

The synthesis of reactive programs from omega-regular specifications is based on finding winning strategies in graph games. In recent years, program synthesis has seen a revival due to several variations and extensions of this basic theme. We survey three such new trends. First, partial program synthesis shifts the emphasis from the problem of synthesizing whole programs from specifications to the problem of completing a partial program so that it satisfies a desired property. Second, quantitative program synthesis aims to find a solution that optimizes a given criterion, such as performance, robustness, or resource consumption. Third, concurrent program synthesis may require the computation of equilibria in graph games with multiple players that represent independent concurrent processes. Recent progress in these directions promises to replace some particularly intricate aspects of programming, such as the placement of synchronization or security primitives, by automatic code generation.

This work has been supported in part by an ERC Advanced Grant (QUAREM - Quantitative Reactive Models) and by an FWF National Research Network (RISE - Rigorous Systems Engineering).

Temporal logic satisfiability for the design of complex Systems

Alessandro Cimatti and Stefano Tonetta
Center for Information Technology, Fondazione Bruno Kessler, Trento, Italy
{cimatti,tonettas}@fbk.eu

The development of computer-based dynamic systems is a very hard task. On the one hand, the required functionalities are very complex, and often include inherently contradicting aspects (e.g. moving trains in a railways station versus avoiding crashes). On the other hand, it is required to integrate the continuous dynamics of physical plants with the discrete dynamics in the control modules and procedures. In addition, such systems often carry out critical functions, which calls for rigorous means to support a development process. The use of a formal approach, coupled with suitable reasoning tools, has found its way in several practical domains, such as railways [5], industrial production [20], hardware design [2, 13, 11], and avionics [15].

Most formal approaches focus on a behavioral characterization of a system, possibly expressed as an automaton or network/hierarchy of automata. Model-based approaches build this behavioral model as a result of semantics-preserving transformation of some design language, and use the model to verify the system description. The verification uses some properties, in form of first-order or temporal formulas, which represent the requirements and are typically assumed to be correct.

More recently, the role of properties is being recognized as increasingly important. For example, in hardware design, specification languages for properties (e.g. PSL [10], SVA [19]) have been introduced to increase expressive power (augmenting for example Linear-time Temporal Logic (LTL) with regular expressions) and usability (using natural language expressions and maximizing the syntactic sugar). The quality of the assertions expressed with such languages has emerged as a problem leading to the development of specialized techniques for their validation [3, 6]. Interestingly, the same type of problem has been addressed in requirements engineering, across domains, for many years. According to studies sponsored by NASA in 90s, many software bugs in safety-critical embedded systems were due to flaws in requirements [14]. The role of formal methods in finding such errors is becoming more and more important (e.g., [7]). The role of properties is also fundamental in compositional reasoning [17], where a global verification problem is decomposed into a number of localized problems. Finally, contract based design [18] allows to decompose the properties of the architectural blocks according to the hierarchical system decomposition, before behavioral descriptions are available, and provides a strong support for property-based refinement and reuse of components [9].

In the talk, we explore the role of temporal logic satisfiability in the design of complex systems, focusing on a property-based design, where behaviors of systems are expressed as formulas in temporal logics. We first discuss the challenges resulting in practice from requirements analysis, compositional reasoning, and contract-based design, showing that satisfiability of temporal formulas is a crucial problem.

Then, we analyze the satisfiability problem for various logics of interest. We adopt a linear model of time, and take into account two kinds of traces: discrete traces and hybrid traces. Properties are therefore represented by sets of traces and temporal formulas are used to specify such sets. We analyze two classes of temporal logics of practical interest. The first class is interpreted over discrete traces, that are sequences of states (assignments to sets of variables). It includes the usual temporal operators of Linear Temporal Logic (LTL) [16], regular expression and suffix operators [10, 4]. In addition, it allows for first order atoms, composed of symbols to be interpreted according to a background theory, similarly to Satisfiability Modulo Theories [1]. We call this class RELTL(T), LTL with regular expressions Modulo Theory. This class is decidable for specific classes of theories and if the variable interpretation is local to each state [12].

The second class, referred to as HRELTL, for Hybrid RELTL [8], is interpreted over hybrid traces. Hybrid traces are useful to model the behaviors of systems featuring continuous transitions, with discrete, instantaneous transitions. Continuous variables are interpreted as functions of time, and the predicates are required to have a uniform interpretation over all interval. The satisfiability problem for HRELTL is undecidable. However, there exists a satisfiability-preserving reduction from HRELTL to RELTL(T) over discrete traces [8]. The main idea is to introduce a sufficient number of constraints on the temporal evolution of the evaluation of predicates to guarantee that the nature of the hybrid dynamics is retained also in the discrete case.

We conclude the talk with an overview of the practical effectiveness of the current methods, and the open challenges in the area.

References

[1] C.W. Barrett, R. Sebastiani, S.A. Seshia & C. Tinelli (2009): *Satisfiability Modulo Theories*. In: *Handbook of Satisfiability*, pp. 825–885, doi:10.3233/978-1-58603-929-5-825.

[2] M. Bernardo & A. Cimatti, editors (2006): *Formal Methods for Hardware Verification, 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006, Bertinoro, Italy, May 22-27, 2006, Advanced Lectures*. *Lecture Notes in Computer Science* 3965, Springer.

[3] R. Bloem, R. Cavada, I. Pill, M. Roveri & A. Tchaltsev (2007): *RAT: A Tool for the Formal Analysis of Requirements*. In: *CAV*, pp. 263–267, doi:10.1007/978-3-540-73368-3_30.

[4] D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman & M.Y. Vardi (2005): *Regular Vacuity*. In: *CHARME*, pp. 191–206, doi:10.1007/11560548_16.

[5] A. Cimatti, R. Corvino, A. Lazzaro, I. Narasamdya, T. Rizzo, M. Roveri, A. Sanseviero & A. Tchaltsev (2012): *Formal Verification and Validation of ERTMS Industrial Railway Train Spacing System*. In: *CAV*, pp. 378–393, doi:10.1007/978-3-642-31424-7_29.

[6] A. Cimatti, M. Roveri, V. Schuppan & S. Tonetta (2007): *Boolean Abstraction for Temporal Logic Satisfiability*. In: *CAV*, pp. 532–546, doi:10.1007/978-3-540-73368-3_53.

[7] A. Cimatti, M. Roveri, A. Susi & S. Tonetta (2012): *Validation of requirements for hybrid systems: A formal approach*. *ACM Trans. Softw. Eng. Methodol.* 21(4), p. 22, doi:10.1145/2377656.2377659.

[8] A. Cimatti, M. Roveri & S. Tonetta (2009): *Requirements Validation for Hybrid Systems*. In: *CAV*, pp. 188–203, doi:10.1007/978-3-642-02658-4_17.

[9] A. Cimatti & S. Tonetta (2012): *A Property-Based Proof System for Contract-Based Design*. In: *EUROMICRO-SEAA*, pp. 21–28, doi:10.1109/SEAA.2012.68.

[10] C. Eisner & D. Fisman (2006): *A Practical Introduction to PSL (Series on Integrated Circuits and Systems)*. Springer-Verlag New York, Inc., doi:10.1007/978-0-387-36123-9.

[11] A. Franzén, A. Cimatti, A. Nadel, R. Sebastiani & J. Shalev (2010): *Applying SMT in symbolic execution of microcode*. In: *FMCAD*, pp. 121–128. Available at http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5770940.

[12] S. Ghilardi, E. Nicolini, S. Ranise & D. Zucchelli (2007): *Combination Methods for Satisfiability and Model-Checking of Infinite-State Systems*. In: *CADE*, pp. 362–378, doi:10.1007/978-3-540-73595-3_25.

[13] W.A. Hunt, Jr., S. Swords, J. Davis & A. Slobodová (2010): *Use of Formal Verification at Centaur Technology*. In: *Design and Verification of Microprocessor Systems for High-Assurance Applications*, Springer, pp. 65–88, doi:10.1007/978-1-4419-1539-9_3.

[14] R.R. Lutz (1993): *Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems*. In: *RE*, pp. 126–133, doi:10.1109/ISRE.1993.324825.

[15] S.P. Miller, M.W. Whalen & D. D. Cofer (2010): *Software model checking takes off*. *Commun. ACM* 53(2), pp. 58–64, doi:10.1145/1646353.1646372.

[16] A. Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS*, pp. 46–57, doi:10.1109/SFCS.1977.32.

[17] W.P. de Roever, F.S. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel & J. Zwiers (2001): *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. *Cambridge Tracts in Theoretical Computer Science* 54, Cambridge University Press.

[18] A.L. Sangiovanni-Vincentelli, W. Damm & R. Passerone (2012): *Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems*. *Eur. J. Control* 18(3), pp. 217–238, doi:10.3166/ejc.18.217-238.

[19] S. Vijayaraghavan & M. Ramanathan (2005): *A Practical Guide for SystemVerilog Assertions*. Springer, doi:10.1007/b137011.

[20] M. Weißmann, S. Bedenk, C. Buckl & A. Knoll (2011): *Model Checking Industrial Robot Systems*. In: *SPIN*, pp. 161–176, doi:10.1007/978-3-642-22306-8_11.

Zielonka's Recursive Algorithm: dull, weak and solitaire games and tighter bounds

Maciej Gazda and Tim A.C. Willemse
Eindhoven University of Technology, The Netherlands

Dull, weak and nested solitaire games are important classes of parity games, capturing, among others, alternation-free μ-calculus and ECTL* model checking problems. These classes can be solved in polynomial time using dedicated algorithms. We investigate the complexity of Zielonka's *Recursive* algorithm for solving these special games, showing that the algorithm runs in $\mathcal{O}(d \cdot (n+m))$ on weak games, and, somewhat surprisingly, that it requires exponential time to solve dull games and (nested) solitaire games. For the latter classes, we provide a family of games $\mathcal{G}$, allowing us to establish a lower bound of $\Omega(2^{n/3})$. We show that an optimisation of Zielonka's algorithm permits solving games from all three classes in polynomial time. Moreover, we show that there is a family of (non-special) games $\mathcal{M}$ that permits us to establish a lower bound of $\Omega(2^{n/3})$, improving on the previous lower bound for the algorithm.

1 Introduction

Parity games [5, 15, 18] are infinite duration, two player games played on a finite directed graph. Each vertex in the graph is owned by one of the two players and vertices are assigned a priority. The game is played by moving a single token along the edges in the graph; the choice where to move next is decided by the player owning the vertex on which the token currently resides. A parity winning condition determines the winner of this infinite play; a vertex in the game is won by the player that can play such that, no matter how the opponent plays, every play from that vertex is won by her, and the winner of each vertex is uniquely determined [15]. From a practical point of view, parity games are interesting as they underpin verification, satisfiability and synthesis problems, see [4, 5, 1].

The simplicity of the gameplay is fiendishly deceptive. Despite continued effort, no polynomial algorithm for solving such games (*i.e.* computing the set of vertices won by each player) has been found. Solving a parity game is known to be in UP $\cap$ coUP [10], a class that neither precludes nor predicts the existence of a polynomial algorithm. In the past, non-trivial classes of parity games have been identified for which polynomial time solving algorithms exist. These classes include *weak* and *dull* games, which arise naturally from alternation-free modal μ-calculus model checking, see [3], and *nested solitaire* games which are obtained from *e.g.* the L_2 fragment of the modal μ-calculus, see [3, 6]. Weak and dull games can be solved in $\mathcal{O}(n+m)$, where n is the number of vertices and m is the number of edges, whereas (nested) solitaire games can be solved in $\mathcal{O}(\log(d) \cdot (n+m))$, where d is the number of different priorities in the game.

One of the most fundamental algorithms for solving parity games is Zielonka's *recursive algorithm* [18]. With a complexity of $\mathcal{O}(n^d)$, the algorithm is theoretically less attractive than *e.g.* Jurdziński's *small progress measures* algorithm [11], Schewe's *bigstep* algorithm [16] or the sub-exponential algorithm due to Jurdziński *et al.* [12]. However, as observed in [8], Zielonka's algorithm is particularly effective in practice, typically beating other algorithms. In view of this, one might therefore ask whether the algorithm is particularly apt at solving natural classes of games, taking advantage of the special structure of these games. We explore this question by investigating the complexity of solving weak, dull and nested solitaire classes using Zielonka's algorithm. Our findings are as follows:

- in Section 4.1, we prove that Zielonka's algorithm solves weak games in polynomial time.
- in Section 4.2, we demonstrate that, somewhat surprisingly, Zielonka's algorithm is exponential on dull games and solitaire games.

The exponential lower bounds we obtain utilise a family of dull, solitaire games $\mathscr{G}^k$ with $3k$ vertices on which the algorithm requires 2^k iterations, allowing us to establish a lower bound of $\Omega(2^{n/3})$. This lower bound improves on previously documented lower bounds for this algorithm (*e.g.*, in [7] a lower bound of $\Omega(1.6^{n/5})$ is established).

In addition to the above complexity results we investigate whether the most common improvement of the algorithm permits it to run in polynomial time for all three special classes of games. That is, we prove in Section 5 that integrating Zielonka's algorithm in a *strongly connected component* decomposition algorithm, as suggested in [11, 8], permits solving all three classes in polynomial time. We analyse the complexity of the resulting algorithm for these three classes, showing that the optimised algorithm runs in $\mathscr{O}(n \cdot (n+m))$ for weak, dull and (nested) solitaire games. Note that these worst-case complexities are slightly worse than those for the dedicated algorithms, but that the applicability of the algorithm remains universal; *e.g.*, it is capable of solving arbitrary nestings of dull and solitaire games, and it does not depend on dedicated algorithms for detecting whether the game is special.

The optimised algorithm still requires exponential time on non-special games. For instance, Friedmann's games are resilient to all known optimisations. Drawing inspiration from our family of games $\mathscr{G}^k$ and the games of [7], we define a new family of games $\mathscr{M}^k$ containing $3k$ vertices, that is also resilient to all known optimisations and requires 2^k iterations of the algorithm. This again allows us to establish a lower bound of $\Omega(2^{n/3})$, also improving on the lower bound established by Friedmann in [7]. We experimentally compare the running time of the optimised algorithm on our games to those of Friedmann.

Outline. Before we present our results, we briefly describe parity games in Section 2 and Zielonka's algorithm in Section 3. Our runtime analysis of Zielonka's original algorithm on special games is presented in Section 4. We prove that an optimisation of the algorithm runs in polynomial time on special games in Section 5, and we prove that, in general, the optimisation's complexity is $\Omega(2^{n/3})$ in Section 6. In Section 7, we wrap up with some conclusions.

2 Parity Games

A parity game is an infinite duration game, played by players *odd*, denoted by $\Box$ and *even*, denoted by $\Diamond$, on a directed, finite graph. The game is formally defined as follows.

Definition 1 A pseudo parity game is a tuple $(V, E, \mathscr{P}, (V_\Diamond, V_\Box))$, where

- V is a finite set of vertices, partitioned in a set $V_\Diamond$ of vertices owned by player $\Diamond$, and a set of vertices $V_\Box$ owned by player $\Box$,
- $E \subseteq V \times V$ is an edge relation,
- $\mathscr{P}: V \to \mathbb{N}$ is a priority function that assigns priorities to vertices, players.

We write $v \to w$ iff $(v, w) \in E$. A pseudo parity game is a *parity game* if the edge relation is total; *i.e.* for each $v \in V$ there is at least one $w \in V$ such that $(v, w) \in E$.

We depict (pseudo) parity games as graphs in which diamond-shaped vertices represent vertices owned by player $\Diamond$ and box-shaped vertices represent vertices owned by player $\Box$. Priorities, associated with vertices, are written inside vertices.

For a given (pseudo) parity game, we are often interested in the subgame that is obtained by restricting the game to a given set of vertices in some way. Formally, we define such subgames as follows.

Definition 2 Let $G = (V, E, \mathscr{P}, (V_\Diamond, V_\Box))$ be a (pseudo) parity game and let $A \subseteq V$ be an arbitrary non-empty set. The (pseudo) parity game $G \cap A$ is the tuple $(A, E \cap (A \times A), \mathscr{P}|_A, (V_\Diamond \cap A, V_\Box \cap A))$. The (pseudo) parity game $G \setminus A$ is defined as the game $G \cap (V \setminus A)$.

Throughout this section, assume that $G = (V, E, \mathscr{P}, (V_\Diamond, V_\Box))$ is an arbitrary pseudo parity game. Note that in general, whenever G is a *parity game* then it is not necessarily the case that the pseudo parity games $G \setminus A$ and $G \cap A$ are again parity games, as totality may not be preserved. However, in what follows, we only consider constructs in which these operations guarantee that totality *is* preserved.

The game G is said to be *strongly connected*, see [17], if for all pairs of vertices $v, w \in V$, we have $v \to^* w$ and $w \to^* v$, where $\to^*$ denotes the transitive closure of $\to$. A *strongly connected component* of G is a maximal set $C \subseteq V$ for which $G \cap C$ is strongly connected.

Lemma 1 Let $C \subseteq V$ be a strongly connected component. If G is a parity game, then so is $G \cap C$.

Henceforth, we assume that G is a parity game (*i.e.* its edge relation is total), and $\bigcirc$ denotes an arbitrary player. We write $\overline{\bigcirc}$ for $\bigcirc$'s opponent; *i.e.* $\overline{\Diamond} = \Box$ and $\overline{\Box} = \Diamond$. A sequence of vertices $v_1, \ldots, v_n$ is a *path* if $v_m \to v_{m+1}$ for all $1 \leq m < n$. Infinite paths are defined in a similar manner. We write p_n to denote the n^{th} vertex in a path p.

A game starts by placing a token on a vertex $v \in V$. Players move the token indefinitely according to a simple rule: if the token is on some vertex $v \in V_\bigcirc$, player $\bigcirc$ gets to move the token to an adjacent vertex. The choice where to move the token next is determined by a partial function $\sigma{:}V^+ \to V$, called a *strategy*. Formally, a strategy σ for player $\bigcirc$ is a function satisfying that whenever it is defined for a finite path $v_1, \ldots, v_n$, we have $\sigma(v_1, \ldots, v_n) \in \{w \in V \mid v \to w\}$ and $v_n \in V_\bigcirc$. We say that an infinite path $v_1, v_2, \ldots$ is *consistent* with a strategy σ for player $\bigcirc$ if for all finite prefixes $v_1, \ldots, v_n$ for which σ is defined, we have $\sigma(v_1, \ldots, v_n) = v_{n+1}$. An infinite path induced by strategies for both players is called a *play*.

The winner of a play is determined by the *parity* of the *highest* priority that occurs infinitely often on it: player $\Diamond$ wins if, and only if this priority is even. That is, we here consider *max* parity games. Note that, alternatively, one could demand that the *lowest* priority that occurs infinitely often along a play determines the winner; such games would be *min* parity games.

A strategy σ for player $\bigcirc$ is *winning* from a vertex v if and only if $\bigcirc$ is the winner of every play starting in v that is consistent with σ. A vertex is won by $\bigcirc$ if $\bigcirc$ has a winning strategy from that vertex. Note that parity games are *positionally determined*, meaning that a vertex is won by player $\bigcirc$ if $\bigcirc$ has a winning *positional strategy*: a strategy that determines where to move the token next based solely on the vertex on which the token currently resides. Such strategies can be represented by a function $\sigma{:}V_\bigcirc \to V$. A consequence of positional determinacy is that vertices are won by exactly one player [5]. *Solving* a parity game essentially is computing the partition $(W_\Diamond, W_\Box)$ of V of vertices won by player $\Diamond$ and player $\Box$, respectively. We say that a game G is a *paradise* for player $\bigcirc$ if all vertices in G are won by $\bigcirc$.

Special games. Parity games pop up in a variety of practical problems. These include model checking problems for fixed point logics [5], behavioural equivalence checking problems [4] and satisfiability and synthesis problems [1]. In many cases, the parity games underlying such problems are *special games*: parity games with a particular structure. We here consider three such special games: *weak*, *dull* and *nested solitaire* games; these classes have previously been studied in the literature, see *e.g.* [3] and the references therein. The definitions that we present here are taken from [3].

Weak games are game graphs in which the priorities along paths are monotonically descending (this is not to be confused with parity games with *weak parity* conditions). That is, for each pair of vertices v, w in the graph, if $v \to w$, then $\mathscr{P}(v) \geq \mathscr{P}(w)$. Such games correspond naturally to model checking problems for the alternation-free modal μ-calculus.

Definition 3 A parity game is *weak* if the priorities along all paths are descending.

Dedicated solvers for weak games can solve these in $\mathscr{O}(|V| + |E|)$. The algorithm that does so is rather straightforward. Since parity games are total, the set L of vertices with lowest priorities m are immediately won by player $\diamond$ iff m is even. Any vertex in the game that can be *forced* to L by the player winning L can then be removed from the game; technically, this is achieved by computing the *attractor set* (see the next section) into L. What remains is another weak parity game which can be solved following the same steps until no vertex is left.

Weak games are closely related to dull games: the latter are game graphs in which all *basic cycles* in the graph are disjoint. A basic cycle is a finite path $v_1, \ldots, v_n$ for which $v_n \to v_1$ and no vertex v_i occurs twice on the path. An *even* cycle is a cycle in which the dominating (*i.e.* highest) priority is even; the cycle is an *odd* cycle if the dominating priority occurring on the cycle is odd.

Definition 4 A parity game is *dull* if even cycles and odd cycles are disjoint.

Note that every weak game is dull; every dull game, on the other hand, can be converted in linear time to a weak game by changing priorities only. This is achieved by assigning a priority that has the same parity as the highest priority present in a strongly connected component to all vertices in that component. This is harmless as each strongly connected component is either entirely even dominated or entirely odd dominated: if not, even cycles and odd cycles would overlap, contradicting the fact that the game is dull. Working bottom-up, it is straightforward to ensure that the priorities along the paths in the game are descending. As a result, dull games can also be solved in $\mathscr{O}(|V| + |E|)$ using the same algorithm as that for solving weak games. Dull games, too, can be obtained from alternation-free μ-calculus model checking problems, and they correspond naturally to the alternation-free fragment of LFP, see [2].

Solitaire games are games in which only one of the two players gets to make non-trivial choices where to play the token next; nested solitaire games generalise solitaire games to games in which both players may make non-trivial moves, but the interactions between both players is still restricted. Such games arise from model checking problems for the fragment L_2 of the modal μ-calculus, see [6], and they correspond with the solitaire fragment of LFP [3].

Definition 5 A parity game is *solitaire* if all non-trivial moves are made by a single player. The game is *nested solitaire* if each strongly connected component induces a solitaire game.

Nested solitaire games can be solved in $\mathscr{O}(\log(d) \cdot (|V| + |E|))$, see [9], although most implementations use a somewhat less optimal implementation that runs in $\mathscr{O}(d \cdot (|V| + |E|))$, see [3]. The latter algorithm works by computing the strongly connected components of a graph and start searching for an even cycle if all non-trivial moves in the component are made by player $\diamond$ and an odd cycle otherwise.

Computing whether there is an even cycle (resp. odd cycle) can be done in $\mathscr{O}(\log(d)\cdot(|V|+|E|))$ using the techniques of [14] or, in $\mathscr{O}(d\cdot(|V|+|E|))$ by repeatedly conducting a depth-first search, starting at the lowest even priority in the component. Clearly, in a component where only player $\Diamond$ gets to make non-trivial moves, all vertices are won by player $\Diamond$ iff an even cycle is found. Iteratively solving the final strongly connected component and removing it together with the attractor for the winner of this component solves entire nested solitaire games.

3 Zielonka's Recursive Algorithm

Throughout this section, we assume G is a fixed parity game $(V,E,\mathscr{P},(V_\Diamond,V_\Box))$, and $\bigcirc$ is an arbitrary player.

Zielonka's algorithm for solving parity games, listed as Algorithm 1, is a divide and conquer algorithm. It constructs winning regions for both players out of the solution of subgames with fewer different priorities and fewer vertices. It removes the vertices with the highest priority from the game, together with all vertices *attracted* to this set of vertices. Attractor sets are formally defined as follows.

Definition 6 The $\bigcirc$-*attractor* into a set $U\subseteq V$, denoted $Attr_\bigcirc(U)$, is defined inductively as follows:

$$\begin{array}{lcl} Attr^0_\bigcirc(U) & = & U \\ Attr^{n+1}_\bigcirc(U) & = & Attr^n_\bigcirc(U) \\ & \cup & \{u\in V_\bigcirc \mid \exists v\in Attr^n_\bigcirc(U):\ u\to v\} \\ & \cup & \{u\in V_{\bar\bigcirc} \mid \forall v\in V: u\to v \implies v\in Attr^n_\bigcirc(U)\} \\ Attr_\bigcirc(U) & = & \bigcup\limits_{i\geq 0} Attr^i_\bigcirc(U) \end{array}$$

If needed for clarity, we write $Attr^G_\bigcirc(U)$ to indicate that the $\bigcirc$-attractor is computed in game graph G.

The lemma below states that whenever attractor sets are removed from a parity game, totality is preserved.

Lemma 2 Let $A=Attr_\bigcirc(U)\subseteq V$ be an arbitrary attractor set. If G is a parity game, then so is $G\setminus A$.

The correctness of Zielonka's algorithm hinges on the fact that higher priorities in the game dominate lower priorities, and that any forced revisit of these higher priorities is beneficial to the player aligning with the parity of the priority. For a detailed explanation of the algorithm and proof of its correctness, we refer to [18, 7].

4 Solving Special Games

Zielonka's algorithm is quite competitive on parity games that stem from practical verification problems [8, 13], often beating algorithms with better worst-case running time. While Zielonka's original algorithm is known to run in exponential time on games defined by Friedmann [7], its behaviour on special parity games has never before been studied. It might just be the case that this algorithm is particularly apt to solve such games. We partly confirm this hypothesis in Section 4.1 by proving that the algorithm indeed runs in polynomial time on weak games. Somewhat surprisingly, however, we also establish that Zielonka's algorithm performs poorly when solving dull and (nested) solitaire games, see Section 4.2.

Algorithm 1 Zielonka's Algorithm

```
1:  function ZIELONKA(G)
2:      if $V = \emptyset$ then
3:          $(W_\Diamond, W_\Box) \leftarrow (\emptyset, \emptyset)$
4:      else
5:          $m \leftarrow \max\{\mathscr{P}(v) \mid v \in V\}$
6:          if $m \bmod 2 = 0$ then $p \leftarrow \Diamond$ else $p \leftarrow \Box$ end if
7:          $U \leftarrow \{v \in V \mid \mathscr{P}(v) = m\}$
8:          $A \leftarrow Attr_p(U)$
9:          $(W'_\Diamond, W'_\Box) \leftarrow$ ZIELONKA$(G \setminus A)$
10:         if $W'_{\bar{p}} = \emptyset$ then
11:             $(W_p, W_{\bar{p}}) \leftarrow (A \cup W'_p, \emptyset)$
12:         else
13:             $B \leftarrow Attr_{\bar{p}}(W'_{\bar{p}})$
14:             $(W'_\Diamond, W'_\Box) \leftarrow$ ZIELONKA$(G \setminus B)$
15:             $(W_p, W_{\bar{p}}) \leftarrow (W'_p, W'_{\bar{p}} \cup B)$
16:         end if
17:     end if
18:     return $(W_\Diamond, W_\Box)$
19: end function
```

4.1 Weak Games

We start with a crucial observation —namely, that for weak games, ZIELONKA solves a paradise in polynomial time— which permits us to prove that solving weak games can be done in polynomial time using ZIELONKA. The proof of the latter, formalised as Proposition 1, depends on three observations, which we first prove in isolation in the following lemma.

Lemma 3 Let $G = (V, E, \mathscr{P}, (V_\Diamond, V_\Box))$ be a weak parity game. Suppose G is a paradise for player $\bigcirc$; *i.e.*, G is won entirely by $\bigcirc$. Then ZIELONKA, applied to G, has the following properties:

1. in the first recursive call in line 9, the argument $G \setminus A$ is also a paradise for player $\bigcirc$.
2. if the second recursive call (line 14) is reached, then its argument $(G \setminus B)$ is the empty set.
3. edges that are used in the computation of attractor sets (lines 8 and 13) are not considered in subroutines.

Proof: We prove all three statements below.

1. Observe that $A = Attr_p(U) = U$, since, in a weak game, no vertex with lower priority has an edge to a vertex in U. In particular, the subgame $G \setminus A$ is $\overline{\bigcirc}$-closed, and hence must be won entirely by $\bigcirc$, if G is a $\bigcirc$-paradise.
2. The second recursive call can be invoked only if $W'_{\bar{p}} \neq \emptyset$. From the above considerations we know that this implies $\bar{p} = \bigcirc$, and $W'_{\bar{p}} = G \setminus A$ is a paradise for $\bigcirc$. We also have $G = W'_{\bar{p}} \cup A$. Since every game staying in A would be losing for $\bigcirc$, it must be the case that $A \subseteq Attr_{\bar{p}}(W'_{\bar{p}})$. But then $B = Attr_{\bar{p}}(W'_{\bar{p}}) = G$, and hence $G \setminus B = \emptyset$.
3. Edges that are considered in the computation of both $Attr_p(U)$ (line 8) and $Attr_{\bar{p}}(W'_{\bar{p}})$ have sources only in U; since no vertices from U are included in the subgame considered in the first recursive

call, and the second call can only take the empty set as an argument. Therefore, these edges will not be considered in the subroutines.

□

Proposition 1 Let $G = (V, E, \mathscr{P}, (V_\Diamond, V_\Box))$ be a weak parity game. Suppose G is a paradise for player $\bigcirc$; *i.e.*, G is won entirely by $\bigcirc$. Then ZIELONKA runs in $\mathscr{O}(|V| + |E|)$.

Proof: We analyse the running time $T(k)$ of ZIELONKA when it is called on a subgame G_k of G with exactly k priorities. Let v_k denote the number of nodes with the highest priority in G_k, and with e_k the number of edges that are considered in the attractor computations (lines 8 and 13) on G_k.

If we assume that the representation of the game has some built-in functionality that allows us to inspect the nodes in the order of priority, then the time required to execute the specific lines of the procedure can be bounded as follows:

- line 7: $c \cdot v_k$ for some constant c
- lines 8 and 13 in total: $c \cdot e_k$ for some constant c
- the remaining lines: z for some constant $z \in \mathbb{N}$

We obtain:

$$\begin{array}{rcl} T(k) & \leq & c \cdot (v_i + e_i + z) + T(k-1) \\ T(k) & \leq & \sum_{i=1}^{k} c \cdot (v_i + e_i + z) \\ T(k) & \leq & c \cdot (\sum_{i=1}^{k} v_i + \sum_{i=1}^{k} e_i + \sum_{i=1}^{k} z) \end{array}$$

Let d denote the total number of priorities occurring in G. Observe that from Lemma 3, we have:

$$\sum_{i=1}^{d} v_i = |V| \text{ and } \sum_{i=1}^{d} e_i = |E|$$

The total execution time of ZIELONKA on G can be bounded by:

$$T(d, V, E) \leq c \cdot (|V| + |E| + \mathscr{O}(d))$$

Hence we obtain $T(d, V, E) = \mathscr{O}(|V| + |E|)$. □

The above proposition is used in our main theorem below to prove that solving weak games using ZIELONKA can be done in polynomial time: each second recursive call to ZIELONKA will effectively be issued on a paradise or an empty game. By proposition 1, we know that ZIELONKA will solve a paradise in linear time.

Theorem 1 ZIELONKA requires $\mathscr{O}(d \cdot (|V| + |E|))$ to solve weak games with d different priorities, $|V|$ vertices and $|E|$ edges.

Proof: The key observation is that ZIELONKA, upon entering the second recursive call in line 14 is invoked on a game that is a paradise. Consider the set of vertices $V \setminus B$ of the game G at that point. It contains the entire set W'_p, and possibly a subset of U. Now, if player $\bar{p}$ could force a play in a node $v \in W'_p$ to $W'_{\bar{p}}$, it could be done only via set U. But this would violate the weakness property. Player p has a winning strategy on $V \setminus B$, which combines the existing strategy on W'_p and if necessary any strategy on U (because whenever a play visits U infinitely often, it is won by p). Thus, the game $G \setminus B$ that is then considered is a p-paradise.

As a result, by Proposition 1, the game $G \setminus B$ is solved in $\mathcal{O}(|V|+|E|)$. Based on these observations, we obtain the following recurrence for ZIELONKA:

$$\begin{array}{ll} T(0,V,E) & \leq \mathcal{O}(1) \\ T(d+1,V,E) & \leq T(d,V,E)+\mathcal{O}(|V|)+\mathcal{O}(|E|) \end{array}$$

Thus, a non-trivial upper bound on the complexity is $\mathcal{O}(d\cdot(|V|+|E|))$. □

Next, we show this bound is tight. Consider the family of parity games $\mathscr{W}^n = (V^n, E^v, \mathscr{P}^n, (V^n_\Diamond, V^n_\Box))$, where priorities and edges are defined in Table 1 and V^n is defined as $V^n = \{v_1,\dots,v_{2n},u_0,u_1\}$.

Table 1: The family $\mathscr{W}$ of games; $1 \leq i \leq n$.

Vertex	Player	Priority	Successors
v_i	$\Diamond$	$i+2$	$\{v_{i-1}, v_{n+i}\} \cup \{u_0 \mid i=1\}$
v_{n+i}	$\Box$	$i+2$	$\{v_i, v_{n+i-1}\} \cup \{u_1 \mid i=1\}$
u_0	$\Diamond$	0	$\{u_0\}$
u_1	$\Box$	1	$\{u_1\}$

The game $\mathscr{W}^4$ is depicted in Figure 1. The family $\mathscr{W}$ has the following characteristics.

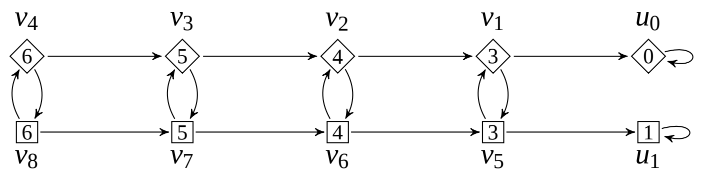


Figure 1: The game $\mathscr{W}^4$.

Proposition 2 The game $\mathscr{W}^n$ is of size linear in n; *i.e.*, $|\mathscr{W}^n| = \mathcal{O}(n)$, it contains $2n+2$ vertices, $4n+2$ edges and $n+2$ different priorities. Moreover, the game $\mathscr{W}^n$ is a weak game.

Lemma 4 In the game $\mathscr{W}^n$, vertices $\{u_0,v_1,\dots,v_n\}$ are won by player $\Diamond$, whereas vertices $\{u_1,v_{n+1},\dots,v_{2n}\}$ are won by player $\Box$.

Proof: Follows from the fact that, for $0 \leq j < n-1$, the strategy $v_{n-j} \to v_{n-j-1}$, $v_1 \to u_0$ and $u_0 \to u_0$ is winning for player $\Diamond$ for the set of vertices $\{u_0, v_1 1,\dots,v_n\}$ and the strategy $v_{2n-j} \to v_{2n-j-1}$, $v_{n+1} \to u_1$ and $u_1 \to u_1$ is winning for player $\Box$ from the set of vertices $\{u_1,v_{n+1},\dots,v_{2n}\}$. □

We next analyse the runtime of Zielonka's algorithm on the family $\mathscr{W}$. Let a_n be defined through the following recurrence relation:

$$\begin{array}{ll} a_0 & = 1 \\ a_{n+1} & = a_n+n+1 \end{array}$$

Observe that the function $\frac{1}{2}n^2$ approximates a_n from below. The proposition below states that solving the family $\mathscr{W}$ of weak parity games requires a quadratic number of recursions.

Proposition 3 Solving $\mathscr{W}^n$, for $n > 0$, requires at least a_n calls to ZIELONKA.

Proof: Follows from the observation that each game $\mathscr{W}^{n+1}$ involves:

1. a first recursive call to ZIELONKA for solving the game $\mathscr{W}^n$.

2. a second recursive call to ZIELONKA for solving either $\mathscr{W}^n \setminus \{v_{2n}, \dots, v_{n+1}, u_1\}$ or $\mathscr{W}^n \setminus \{v_n, \dots, v_1, u_0\}$; both require $n+1$ recursive calls to ZIELONKA.

□

Theorem 2 Solving weak games using ZIELONKA requires $\Theta(d \cdot (|V| + |E|))$.

Note that this complexity is a factor d worse than that of the dedicated algorithm. For practical problems such as when solving parity games that come from model checking problems d is relatively small; we expect that for such cases, the difference between the dedicated algorithm and Zielonka's algorithm to be small.

4.2 Dull and Nested Solitaire Games

We next prove that dull games and (nested) solitaire require exponential time to solve using ZIELONKA. Given that dull games can be converted to weak games in linear time, and that Zielonka solves weak games in polynomial time, this may be unexpected.

Our focus is on solitaire games first. We construct a family of parity games $\mathscr{G}^n = (V^n, E^n, \mathscr{P}^n, (V^n_\Diamond, V^n_\Box))$ with vertices $V^n = \{v_0, \dots, v_{2n-1}, u_1, \dots u_n\}$. All vertices belong to player $\Diamond$; that is, $V^n_\Diamond = V^n$ and $V^n_\Box = \emptyset$. The priorities and the edges are described by Table 2.

Table 2: The family $\mathscr{G}$ of games; $1 \leq i < 2n, 1 \leq j \leq n$.

Vertex	Priority	Successors
v_i	$i+2$	$\{v_{i-1}\}$
v_0	2	$\{v_0\}$
u_j	1	$\{u_j, v_{2j-1}\}$

Proposition 4 The game $\mathscr{G}^n$ is of size linear in n; *i.e.* $|\mathscr{G}^n| = \mathscr{O}(n)$, it has $3n$ vertices, $4n$ edges and $2n+1$ different priorities. Moreover, the game $\mathscr{G}^n$ is a (nested) solitaire game.

The game $\mathscr{G}^3$ is depicted in Figure 2. Observe that in this game, vertex v_5 has the maximal priority and that this priority is odd. This means that Zielonka's algorithm will compute the odd-attractor to v_5 in line 8 of the algorithm, *i.e.* $Attr_\Box(\{v_5\}) = \{v_5\}$. We can generalise this observation for arbitrary game $\mathscr{G}^n$: in such a game, $Attr_\Box(\{v_{2n-1}\}) = \{v_{2n-1}\}$. Henceforth, we denote the subgame $\mathscr{G}^n \setminus \{v_{2n-1}\}$ by $\mathscr{G}^{n,-}$.

Lemma 5 The game $\mathscr{G}^n$ is won by player $\Diamond$. In the game $\mathscr{G}^{n,-}$, all vertices except for vertex u_n, are won by player $\Diamond$.

Proof: The fact that $\mathscr{G}^n$ is won by player $\Diamond$ follows immediately from the strategy $\sigma: V^n \to V^n$, defined as $\sigma(v_i) = v_{i-1}$ for all $1 \leq i < 2n$, $\sigma(u_i) = v_{2i-1}$ for all $i \leq n$ and $\sigma(v_0) = v_0$, which is winning for player $\Diamond$. For the game $\mathscr{G}^{n,-}$, a strategy σ' can be used that is as strategy σ for all vertices $v \neq u_n$; for vertex u_n, we are forced to choose $\sigma'(u_n) = u_n$, since u_n is the sole successor of u_n in $\mathscr{G}^{n,-}$. Since the priority of u_n is odd, the vertex is won by player $\Box$. □

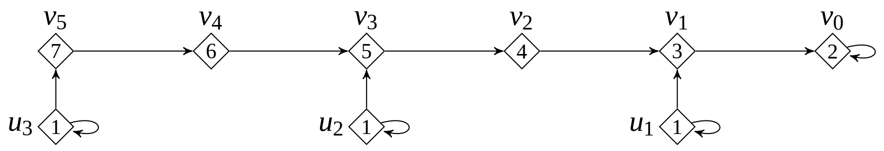


Figure 2: The game $\mathscr{G}^3$.

We now proceed to the runtime of Zielonka's algorithm on the family $\mathscr{G}$.

Proposition 5 Solving $\mathscr{G}^n$, for $n > 0$, requires at least 2^n calls to ZIELONKA.

Proof: Solving the game $\mathscr{G}^1$ requires at least one call to ZIELONKA.

Consider the game $\mathscr{G}^n$, for $n > 1$. Observe that ZIELONKA is invoked recursively on the game $\mathscr{G}^{n,-}$ in the first recursion on line 9. We focus on solving the latter game.

The vertex with the highest priority in $\mathscr{G}^{n,-}$ is $v_{2(n-1)}$. Observe that $A = Attr_{\Box}(\{v_{2(n-1)}\}) = \{v_{2(n-1)}\}$. Note that $\mathscr{G}^{n,-} \setminus A$ contains $\mathscr{G}^{n-1}$ as a separate subgame. The first recursive call in solving $\mathscr{G}^{n,-}$ will therefore also solve the subgame $\mathscr{G}^{n-1}$.

Next, observe that u_n (and u_n alone) is won by player $\Box$, see Lemma 5. We therefore need to compute $B = Attr_{\Diamond}(\{u_n\}) = \{u_n\}$. Now, note that $\mathscr{G}^{n,-} \setminus B$ subsumes the subgame $\mathscr{G}^{n-1}$, which is a separate game in $\mathscr{G}^{n,-} \setminus B$. Therefore, also the second recursive call to ZIELONKA involves solving the subgame $\mathscr{G}^{n-1}$. $\Box$

The lower bound on the number of iterations for ZIELONKA is thus exponential in the number of vertices.

Theorem 3 Solving (nested) solitaire games using ZIELONKA requires $\Omega(2^{|V|/3})$.

We note that this improves on the bounds of $\Omega(1.6^{|V|/5})$ established by Friedmann. Being structurally more complex, however, his games are robust to typical (currently known) improvements to Zielonka's algorithm such as the one presented in the next section (although this is not mentioned or proved in [7]). Still, we feel that the simplicity of our family $\mathscr{G}$ fosters a better understanding of the algorithm.

Observe that the family $\mathscr{G}$ is also a family of dull games. As a result, we immediately have the following theorem.

Theorem 4 Solving dull games using ZIELONKA requires $\Omega(2^{|V|/3})$.

5 Recursively Solving Special Games in Polynomial Time

The $\mathscr{G}$ family of games of the previous section are easily solved when preprocessing the games using priority propagation and self-loop elimination. However, it is straightforward to make the family robust to such heuristics by duplicating the vertices that have odd priority, effectively creating odd loops that are not detected by such preprocessing steps. In a similar vein, the commonly suggested optimisation to use a strongly connected component decomposition as a preprocessing step can be shown to be insufficient to solve (nested) solitaire games. The family $\mathscr{G}$ can easily be made robust to this preprocessing step: by adding edges from v_0 to all u_i, each game in $\mathscr{G}$ becomes a single SCC.

In this section, we investigate the complexity of a tight integration of a strongly connected component decomposition and Zielonka's algorithm, as suggested by *e.g.* [11, 8]. By decomposing the game each time Zielonka is invoked, large SCCs are broken down in smaller SCCs, potentially increasing the effectiveness of the optimisation. The resulting algorithm is listed as Algorithm 2.

We will need the following lemma:

Lemma 6 If algorithm 2 is invoked on a game that is either dull or (nested) solitaire, then in the entire recursion tree all second recursive calls (line 16) are trivial (with empty set as an argument).

Proof: In case of dull games, since game H is a connected component, each of its subgames is won by player corresponding to $m \bmod 2$, namely p. Hence after the line 11 is executed, we obtain $W'_p = H \setminus A$ and $W'_{\overline{p}} = \emptyset$. The second recursive call will therefore never be invoked.

Algorithm 2 Optimised Zielonka's Algorithm

```
function ZIELONKA_SCC(G)
    $(W^G_\Diamond, W^G_\Box) \leftarrow (\emptyset, \emptyset)$
    if $V \neq \emptyset$ then
        $\mathscr{S}$ := SCC_GRAPH_DECOMPOSITION($G$)
        for each final SCC $C \in \mathscr{S}$ do
            $H \leftarrow G \cap C$
            $m \leftarrow \max\{\mathscr{P}(v) \mid v \in C\}$
            if $m \bmod 2 = 0$ then $p \leftarrow \Diamond$ else $p \leftarrow \Box$ end if
            $U \leftarrow \{v \in C \mid \mathscr{P}(v) = m\}$
            $A \leftarrow Attr^H_p(U)$
            $(W'_\Diamond, W'_\Box) \leftarrow$ ZIELONKA_SCC($H \setminus A$)
            if $W'_{\bar{p}} = \emptyset$ then
                $(W_p, W_{\bar{p}}) \leftarrow (A \cup W'_p, \emptyset)$
            else
                $B \leftarrow Attr^H_{\bar{p}}(W'_{\bar{p}})$
                $(W'_\Diamond, W'_\Box) \leftarrow$ ZIELONKA_SCC($H \setminus B$)
                $(W_p, W_{\bar{p}}) \leftarrow (W'_p, W'_{\bar{p}} \cup B)$
            end if
            $(W^G_\Diamond, W^G_\Box) \leftarrow (W^G_\Diamond \cup Attr^G_\Diamond(W_\Diamond), W^G_\Box \cup Attr^G_\Box(W_\Box))$
            $\mathscr{S}$ := SCC_GRAPH_DECOMPOSITION($G \setminus (W^G_\Diamond \cup W^G_\Box)$)
        end for
    end if
    return $(W^G_\Diamond, W^G_\Box)$
end function
```

Now assume that the game is solitaire and owned by player q. If $p = q$, then $Attr^H_p(U) = H$ (the game is p-owned, and strongly connected), and the second call is not invoked at all. Otherwise, the second call is invoked only if $W'_{\bar{p}} \neq \emptyset$. But then $B = Attr^H_{\bar{p}}(W'_{\bar{p}}) = Attr^H_q(W'_{\bar{p}}) = H$ (the game is owned by $\bar{p}$, and strongly connected), and $H \setminus B = \emptyset$. $\Box$

We will now prove that the optimisation suffices to solve special parity games in polynomial time.

Theorem 5 Algorithm 2 solves dull and (nested) solitaire games in $\mathscr{O}(|V| \cdot (|V| + |E|))$ time.

Proof: Let $\#\mathsf{for}(V)$ denote the total number of iterations of the for loop in the entire recursion tree. Observe that the total execution time of ZIELONKA_SCC can be bounded from above as follows:

$$T(V,E) = \mathscr{O}(\#\mathsf{for}(V) \cdot (|V| + |E|))$$

Indeed, every iteration of the loop (not counting the iterations in subroutines) contributes a maximal factor of $\mathscr{O}(|V| + |E|)$ running time, which results from the attractor computation and SCC decomposition.

We will use subscripts for the values of the algorithm variables in iteration $i \in \{1, \dots, k\}$, e.g. the value of variable C in iteration i is C_i. Furthermore, by V_i we will denote the set of vertices in the subgame considered in the first recursive call, i.e. $V_i = C_i \setminus A_i$.

We will show that $\#\mathsf{for}(V) \leq |V|$. We have:

$$\begin{array}{lcll} \#\mathsf{for}(V) & \leq & 1 & \text{for } |V| \leq 1 \\ \#\mathsf{for}(V) & \leq & \#\mathsf{for}(V_1) + \cdots + \#\mathsf{for}(V_k) + k & \text{for } |V| > 1 \end{array} \quad (*)$$

In the second inequality, k is the total number of bottom SCCs considered in line 5. Each of these SCCs may give rise to a recursive call (at most one, see Lemma 6). This recursive call contributes in turn $\#\mathsf{for}(V_i)$ iterations.

We proceed to show $\#\mathsf{for}(V) \leq |V|$ by induction on $|V|$. The base holds immediately from the first inequality. Now assume that $\#\mathsf{for}(V) \leq |V|$ for $|V| < m$.

Obviously $|C_1| + \cdots + |C_k| \leq |V|$. Observe that in every iteration i the set A_i is nonempty, therefore $V_i < C_i$. Therefore $|V_1| + \cdots + |V_k| \leq |V| - k$, or equivalently $|V_1| + \cdots + |V_k| + k \leq |V|$.

Applying the induction hypothesis in the right-hand side of (*) yields $\#\mathsf{for}(V) \leq |V_1| + \cdots + |V_k| + k$, and due to the above observation we finally obtain $\#\mathsf{for}(V) \leq |V|$. □

The above upper bound is slower by a factor V compared to the dedicated algorithms for solving weak and dull games. For nested solitaire games, the optimised recursive algorithm has an above upper bound comparable to that of standard dedicated algorithms for nested solitaire games when the number of different priorities is of $\mathcal{O}(V)$, and it is a factor $V/\log(d)$ slower compared to the most efficient algorithm for solving nested solitaire games.

6 A Tighter Exponential Bound for Zielonka's Optimised Algorithm

In view of the findings of the previous section, it seems beneficial to always integrate Zielonka's recursive algorithm with SCC decomposition. Observe that the family of games we used to establish the lower bound of $\Omega(2^{|V|/3})$ in Section 4 does not permit us to prove the same lower bound for the optimised algorithm. As a result, the current best known lower bound for the algorithm is still $\Omega(1.6^{|V|/5})$. In this section, we show that the complexity of the optimised algorithm is actually also $\Omega(2^{|V|/3})$. The family of games we construct is, like Friedmann's family, resilient to all optimisations we are aware of.

Let $\mathscr{M}^n = (V^n, E^n, \mathscr{P}^n, (V^n_\diamond, V^n_\Box))$, for $n \geq 1$ be a family of parity games with set of vertices $V^n = \{v_i, u_i, w_i \mid 1 \leq i \leq n\}$. The sets $V^n_\diamond$ and $V^n_\Box$, the priority function $\mathscr{P}^n$ and the set of edges are described by Table 3. We depict the game $\mathscr{M}^4$ in Figure 3.

Table 3: The family $\mathscr{M}$ of games; $1 \leq i \leq n$.

Vertex	Player	Priority	Successors
v_i	$\Box$ iff $i \bmod 2 = 0$	$i+1$	$\{u_i\} \cup \{v_{i+1} \mid i < n\}$
u_i	$\Box$ iff $i \bmod 2 = 0$	$i \bmod 2$	$\{w_i\} \cup \{v_{i+1} \mid i < n\}$
w_i	$\diamond$ iff $i \bmod 2 = 0$	$i \bmod 2$	$\{u_i\} \cup \{w_{i-1} \mid 1 < i\}$

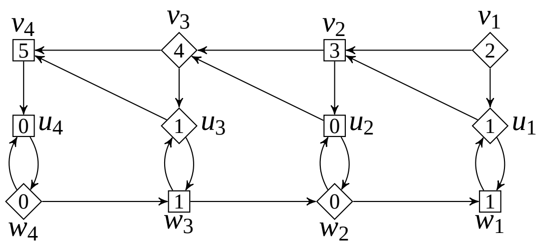


Figure 3: The game $\mathscr{M}^4$.

Proposition 6 The game $\mathscr{M}^n$ is won entirely by player $\diamond$ for even n and entirely by player $\Box$ for odd n.

Theorem 6 Solving $\mathscr{M}^n$ using either ZIELONKA or ZIELONKA_SCC requires $\Omega(2^{|V|/3})$ time.

Proof: The proof is similar to Prop. 5; we can show that the game $\mathscr{M}^n$ requires 2^n calls to either ZIELONKA or ZIELONKA_SCC. The only significant difference in case of ZIELONKA_SCC is that the game may be potentially simplified in line 4 of Alg.2. However, each game $\mathscr{M}^n$ constitutes a strongly connected subgame, and therefore will not be decomposed. $\square$

We compared the performance of the PGSolver tool, a publicly available tool that contains an implementation of the optimised recursive algorithm, on the family $\mathscr{M}$ to that of Friedmann's family of games (denoted with $\mathscr{F}$), see Figure 4. The figure plots the number of vertices (horizontal axis) and the time required to solve the games (vertical log scale axis), clearly illustrating that $\mathscr{M}$ games are harder.

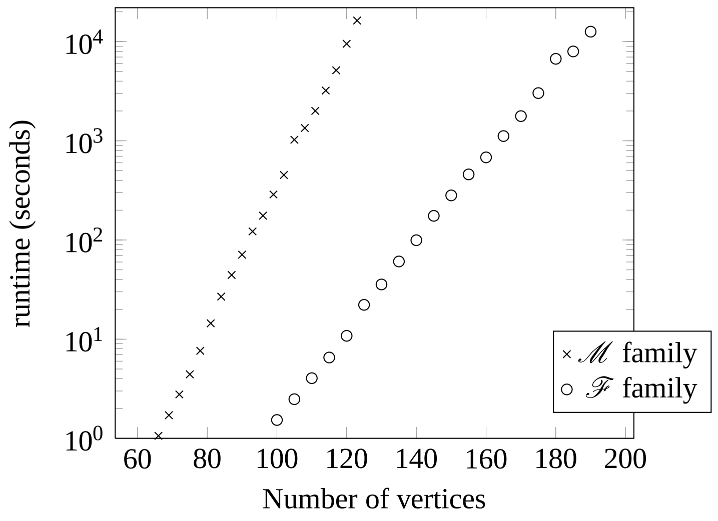


Figure 4: Runtime of the optimised recursive algorithm (vertical log scale axis) in seconds versus number of vertices of the games (horizontal axis).

7 Conclusions

We explored the complexity of solving special parity games using Zielonka's recursive algorithm, proving that weak games are solved in polynomial time and dull and nested solitaire games require exponential time. The family of games $\mathscr{G}$ we used to prove the exponential lower bounds in addition tighten the lower bound to $\Omega(2^{|V|/3})$ for the original algorithm by Zielonka.

We show that a standard optimisation of the algorithm permits solving all three classes of games in polynomial time. The technique used in the optimisation (a tight integration of a strongly connected component decomposition and Zielonka's algorithm) has been previously implemented in [8] and was observed to work well in practice. Our results provide theoretical explanations for these observations.

We furthermore studied the lower bounds of Zielonka's algorithm *with* optimisation. In the last section, we improve on Friedmann's lower bound and arrive at a lower bound of $\Omega(2^{|V|/3})$ for the optimised algorithm. For this, we used a family of games $\mathscr{M}$ for which we drew inspiration from the family $\mathscr{G}$ and the games defined in [7]. We believe that an additional advantage of the families of games $\mathscr{G}$ and $\mathscr{M}$ we defined in this paper over Friedmann's games lies in their (structural) simplicity.

Our complexity analysis for the special games offers additional insight into the complexity of Zielonka's algorithm and its optimisation and may inspire future optimisations of the algorithm. In a similar vein, the same type of analysis can be performed on other parity game solving algorithms from the literature, *e.g.* strategy improvement algorithms.

References

[1] A. Arnold, A. Vincent & I. Walukiewicz (2003): *Games for synthesis of controllers with partial observation*. *TCS* 303(1), pp. 7–34, doi:10.1016/S0304-3975(02)00442-5.

[2] D. Berwanger & E. Grädel (2001): *Games and Model Checking for Guarded Logics*. In: *LPAR*, *LNCS* 2250, Springer, pp. 70–84, doi:10.1007/3-540-45653-8_5.

[3] D. Berwanger & E. Grädel (2004): *Fixed-Point Logics and Solitaire Games*. *Theory Comput. Syst.* 37(6), pp. 675–694. Available at `http://dx.doi.org/10.1007/s00224-004-1147-5`.

[4] T. Chen, B. Ploeger, J. van de Pol & T.A.C. Willemse (2007): *Equivalence Checking for Infinite Systems Using Parameterized Boolean Equation Systems*. In: *CONCUR'07*, pp. 120–135, doi:10.1007/978-3-540-74407-8_9.

[5] E.A. Emerson & C.S. Jutla (1991): *Tree automata, Mu-Calculus and determinacy*. In: *FOCS'91*, IEEE Computer Society, Washington, DC, USA, pp. 368–377, doi:10.1109/SFCS.1991.185392.

[6] E.A. Emerson, C.S. Jutla & A.P. Sistla (2001): *On model checking for the μ-calculus and its fragments*. *Theor. Comput. Sci.* 258(1-2), pp. 491–522, doi:10.1016/S0304-3975(00)00034-7.

[7] O. Friedmann (2011): *Recursive algorithm for parity games requires exponential time*. *RAIRO – Theor. Inf. and Applic.* 45(4), pp. 449–457, doi:10.1051/ita/2011124.

[8] O. Friedmann & M. Lange (2009): *Solving Parity Games in Practice*. In: *ATVA'09*, *LNCS* 5799, Springer, pp. 182–196, doi:10.1007/978-3-642-04761-9_15.

[9] J.F. Groote & M. Keinänen (2005): *A Sub-quadratic Algorithm for Conjunctive and Disjunctive Boolean Equation Systems*. In: *ICTAC*, *LNCS* 3722, Springer, pp. 532–545, doi:10.1007/11560647_35.

[10] M. Jurdziński (1998): *Deciding the Winner in Parity Games is in UP $\cap$ co-UP*. *IPL* 68(3), pp. 119–124, doi:10.1016/S0020-0190(98)00150-1.

[11] M. Jurdziński (2000): *Small Progress Measures for Solving Parity Games*. In: *STACS'00*, *LNCS* 1770, Springer, pp. 290–301, doi:10.1007/3-540-46541-3_24.

[12] M. Jurdziński, M. Paterson & U. Zwick (2006): *A Deterministic Subexponential Algorithm for Solving Parity Games*. In: *SODA'06*, ACM/SIAM, pp. 117–123, doi:10.1145/1109557.1109571.

[13] J.J.A. Keiren (2009): *An experimental study of algorithms and optimisations for parity games, with an application to Boolean Equation Systems*. Master's thesis, Eindhoven University of Technology.

[14] V. King, O. Kupferman & M.Y. Vardi (2001): *On the Complexity of Parity Word Automata*. In: *FoSSaCS*, *LNCS* 2030, Springer, pp. 276–286, doi:10.1007/3-540-45315-6_18.

[15] R. McNaughton (1993): *Infinite games played on finite graphs*. *APAL* 65(2), pp. 149–184, doi:10.1016/0168-0072(93)90036-D.

[16] S. Schewe (2007): *Solving Parity Games in Big Steps*. In: *FSTTCS'07*, *LNCS* 4855, Springer, pp. 449–460, doi:10.1007/978-3-540-77050-3.

[17] R.E. Tarjan (1972): *Depth-First Search and Linear Graph Algorithms*. *SIAM J. Comput.* 1(2), pp. 146–160, doi:10.1137/0201010.

[18] W. Zielonka (1998): *Infinite games on finitely coloured graphs with applications to automata on infinite trees*. *TCS* 200(1-2), pp. 135 – 183, doi:10.1016/S0304-3975(98)00009-7.

Simple strategies for Banach-Mazur games and fairly correct systems [*]

Thomas Brihaye
UMONS
Mons, Belgium
University of Mons
Place du Parc 20, 7000 Mons, Belgium
thomas.brihaye@umons.ac.be

Quentin Menet [†]
UMONS
Mons, Belgium
University of Mons
Place du Parc 20, 7000 Mons, Belgium
quentin.menet@umons.ac.be

In 2006, Varacca and Völzer proved that on finite graphs, ω-regular large sets coincide with ω-regular sets of probability 1, by using the existence of positional strategies in the related Banach-Mazur games. Motivated by this result, we try to understand relations between sets of probability 1 and various notions of simple strategies (including those introduced in a recent paper of Grädel and Leßenich). Then, we introduce a generalisation of the classical Banach-Mazur game and in particular, a probabilistic version whose goal is to characterise sets of probability 1 (as classical Banach-Mazur games characterise large sets). We obtain a determinacy result for these games, when the winning set is a countable intersection of open sets.

1 Introduction

Systems (automatically) controlled by computer programs abound in our everyday life. Clearly enough, it is of a capital importance to know whether the programs governing these systems are *correct*. Over the last thirty years, formal methods for verifying computerised systems have been developed for validating the adequation of the systems against their requirements. Model checking is one such approach: it consists first in modelling the system under study (for instance by an automaton), and then in applying algorithms for comparing the behaviours of that model against a specification (modelled for instance by a logical formula). Model checking has now reached maturity, through the development of efficient symbolic techniques, state-of-the-art tool support, and numerous successful applications to various areas.

As argued in [9]: *'Sometimes, a model of a concurrent or reactive system does not satisfy a desired linear-time temporal specification but the runs violating the specification seem to be artificial and rare'*. As a naive example of this phenomenon, consider a coin flipped an infinite number of times. Classical verification will assure that the property stating *"one day, we will observe at least one head"* is false, since there exists a unique execution of the system violating the property. In some situations, for instance when modeling non-critical systems, one could prefer to know whether the system is *fairly correct*. Roughly speaking, a system is fairly correct against a property if the set of executions of the system violating the property is *"very small"*; or equivalently if the set of executions of the system satisfying the property is *"very big"*. A first natural notion of fairly correct system is related to probability: *almost-sure correctness*. A system is almost-surely correct against a property if the set of executions of the system satisfying the property has probability 1. Another interesting notion of fairly correct system is related to topology: *large correctness*. A system is largely correct against a property if the set of executions of the

[*]This work has been partly supported by a grant from the National Bank of Belgium, the ARC project (number AUWB-2010-10/15-UMONS-3), and the FRFC project (number 2.4545.11).
[†]The second author is supported by a grant of FRIA.

system satisfying the property is *large* (in the topological sense). There exists a lovely characterisation of *large sets* by means of the *Banach-Mazur games*. In [8], it has been shown that a set W is large if and only if a player has a winning strategy in the related Banach-Mazur game.

Although, the two notions of *fairly correct systems* do not coincide in general, in [9], the authors proved (amongst other results) the following result: when considering ω-regular properties on finite systems, the *almost-sure correctness* and the *large correctness* coincide, for bounded Borel measures. Motivated by this very nice result, we intend to extend it to a larger class of specifications. The key ingredient to prove the previously mentioned result of [9] is that when considering ω-regular properties, *positional* strategies are sufficient in order to win the related Banach-Mazur game [1]. For this reason, we investigate *simple strategies* in Banach-Mazur games, inspired by the recent work [4] where infinite graphs are studied.

Our contributions. In this paper, we first compare various notions of simple strategies on finite graphs (including *bounded* and *move-counting* strategies), and their relations with the sets of probability 1. Given a set W, the existence of a bounded (resp. move-counting) winning strategy in the related Banach-Mazur game implies that W is a set of probability 1. However there exist sets W of probability 1 for which there is no bounded and no move-counting winning strategy in the related Banach-Mazur game. Therefore, we introduce a generalisation of the classical Banach-Mazur game and in particular, a probabilistic version whose goal is to characterise sets of probability 1 (as classical Banach-Mazur games characterise large sets). We obtain the desired characterisation in the case of countable intersections of open sets. This is the main contribution of the paper. As a byproduct of the latter, we get a determinacy result for our probabilistic version of the Banach-Mazur game for countable intersections of open sets.

2 Banach-Mazur Games on finite graphs

Let $(X,\mathscr{T})$ be a topological space. A notion of topological "bigness" is given by large sets. A subset $W\subset X$ is said to be *nowhere dense* if the closure of W has empty interior. A subset $W\subset X$ is said to be *meagre* if it can be expressed as the union of countably many nowhere dense sets and a subset $W\subset X$ is said to be large if W^c is meagre. In particular, we remark that a countable intersection of large sets is still large and that if $W\subset X$ is large, then any set $Y\supset W$ is large.

If $G=(V,E)$ is a finite directed graph and $v_0\in V$, then the space of infinite paths in G from v_0, denoted $\mathrm{Paths}(G,v_0)$, can be endowed with the complete metric

$$d((\sigma_n)_{n\geq 0},(\rho_n)_{n\geq 0})=2^{-k}\quad\text{where}\quad k=\min\{n\geq 0:\sigma_n\neq\rho_n\}\tag{2.1}$$

with the conventions that $\min\emptyset=\infty$ and $2^{-\infty}=0$. In other words, the open sets in $\mathrm{Paths}(G,v_0)$ endowed with this metric are the countable unions of cylinders, where a cylinder is a set of the form $\{\rho\in\mathrm{Paths}(G,v_0)\mid\pi\text{ is a prefix of }\rho\}$ for some finite path π in G from v_0.

We can therefore study the large subsets of the metric space $(\mathrm{Paths}(G,v_0),d)$. Banach-Mazur games allow us to characterise large subsets of this metric space through the existence of winning strategies.

Definition 2.1. A *Banach-Mazur game* $\mathscr{G}$ on a finite graph is a triplet (G,v_0,W) where $G=(V,E)$ is a finite directed graph where every vertex has a successor, $v_0\in V$ is the initial state, W is a subset of the infinite paths in G starting in v_0.

A Banach-Mazur game $\mathscr{G}=(G,v_0,W)$ on a finite graph is a two-player game where Pl. 0 and Pl. 1 alternate in choosing a finite path as follows: Pl. 1 begins with choosing a finite[1] path π_1 starting in v_0;

[1]In this paper, we always assume that a finite path is non-empty.

Pl. 0 then prolongs π_1 by choosing another finite path π_2 and so on. A play of $\mathscr{G}$ is thus an infinite path in G and we say that Pl. 0 wins if this path belongs to W, while Pl. 1 wins if this path does not belong to W. The set W is called the winning condition. It is important to remark that, in general, in the literature, Pl. 0 moves first in Banach-Mazur games but in this paper, we always assume that Pl. 1 moves first in order to bring out the notion of large set (rather than meagre set). The main result about Banach-Mazur games can then be stated as follows:

Theorem 2.2 ([8]). *Let $\mathscr{G} = (G, v_0, W)$ be a Banach-Mazur game on a finite graph. Pl. 0 has a winning strategy for $\mathscr{G}$ if and only if W is large.*

3 Simple strategies in Banach-Mazur games

In a Banach-Mazur game (G, v_0, W) on a finite graph, a strategy for Pl. 0 is given by a function f defined on $\mathrm{FinPaths}(G, v_0)$, the set of finite paths of G starting from v_0, such that for any $\pi \in \mathrm{FinPaths}(G, v_0)$, we have $f(\pi) \in \mathrm{FinPaths}(G, \mathrm{last}(\pi))$. However, we can imagine some restrictions on the strategies of Pl. 0:

1. A strategy f is said to be *positional* if it only depends on the current vertex, i.e f is a function defined on V such that for any $v \in V$, $f(v) \in \mathrm{FinPaths}(G, v)$ and a play ρ is consistent with f if ρ is of the form $(\pi_i f(\mathrm{last}(\pi_i)))_{i \geq 1}$.

2. A strategy f is said to be *finite-memory* if it only depends on the current vertex and a finite memory (see [3] for the precise definition of a finite-memory strategy).

3. A strategy f is said to be *b-bounded* if for any $\pi \in \mathrm{FinPaths}(G, v_0)$, $f(\pi)$ has length less than b and a strategy is said to be *bounded* if there is $b \geq 1$ such that f is b-bounded.

4. A strategy f is said to be *move-counting* if it only depends on the current vertex and the number of moves already played, i.e. f is a function defined on $V \times \mathbb{N}$ such that for any $v \in V$, any $n \in \mathbb{N}$, $f(v,n) \in \mathrm{FinPaths}(G, v)$ and a play ρ is consistent with f if ρ is of the form $(\pi_i f(\mathrm{last}(\pi_i), i))_{i \geq 1}$.

5. A strategy f is said to be *length-counting* if it only depends on the current vertex and the length of the prefix already played, i.e. f is a function defined on $V \times \mathbb{N}$ such that for any $v \in V$, any $n \in \mathbb{N}$, $f(v,n) \in \mathrm{FinPaths}(G, v)$ and a play ρ is consistent with f if after a prefix π, the move of Pl. 0 is given by $f(\mathrm{last}(\pi), |\pi|)$.

The notions of positional and finite memory strategies are classical, bounded strategies are present in [9], move-counting and length-counting strategies have been introduced in [4]. We first remark that, by definition, the existence of a positional winning strategy implies the existence of finite-memory/move-counting/length-counting winning strategies. Moreover, since G is a finite graph, a positional strategy is always bounded. In [3], it is proved that the existence of a finite-memory winning strategy implies the existence of a positional winning strategy.

Proposition 3.1 ([3]). *Let $\mathscr{G} = (G, v_0, W)$ be a Banach-Mazur game. Pl. 0 has a finite-memory winning strategy if and only if Pl. 0 has a positional winning strategy.*

Using the ideas of the proof of the above proposition, we can also show that the existence of a winning strategy implies the existence of a length-counting winning strategy.

Proposition 3.2. *Let $\mathscr{G} = (G, v_0, W)$ be a Banach-Mazur game on a finite graph. Pl. 0 has a length-counting winning strategy if and only if Pl. 0 has a winning strategy.*

Proof. Let f be a winning strategy for Pl. 0. Since G is a finite graph, for any $n \geq 0$ and any $v \in V$, we can consider an enumeration $\pi_1, \dots, \pi_m$ of finite paths in $\mathrm{FinPaths}(G, v_0)$ of length n such that $\mathrm{last}(\pi_i) = v$. We then let

$$h(v,n) = f\big(\pi_1\big) f\big(\pi_2 f(\pi_1)\big) f\big(\pi_3 f(\pi_1) f(\pi_2 f(\pi_1))\big) \dots f\big(\pi_m f(\pi_1) f(\pi_2 f(\pi_1)) \cdots\big).$$

If ρ is a play consistent with h, then ρ is a play where the strategy f is applied infinitely often. Thus such a play ρ can be seen as a play $\sigma_1\tau_1\sigma_2\tau_2\cdots$ where the τ_i's (resp. the σ_i's) are the moves of Pl. 0 (resp. Pl. 1.) and where $f(\sigma_1\tau_1\cdots\sigma_i) = \tau_i$. Each play consistent with h can thus be seen as a play consistent with f, and we deduce that the strategy h is a length-counting winning strategy. □

On the other side, the notions of move-counting winning strategies and bounded winning strategies are incomparable.

Example 3.3 (Set with a move-counting winning strategy and without a bounded winning strategy). We consider the complete graph $G_{0,1}$ on $\{0,1\}$. Let W be the set of any sequences $(\sigma_n)_{n\geq 1}$ in $\{0,1\}^\omega$ with $\sigma_1 = 0$ such that $(\sigma_n)_{n\geq 1}$ contains a finite sequence of 1 strictly longer than the initial finite sequence of 0. In other words, $(\sigma_n)_{n\geq 1} \in W$ if $\sigma_1 = 0$ and if there exist $j \geq 1$ and $k \geq 1$ such that $\sigma_j = 1$ and $\sigma_{k+1} = \cdots = \sigma_{k+j} = 1$. Let $\mathscr{G} = (G_{0,1}, 0, W)$. The strategy $f(\cdot, n) = 1^n$ is a move-counting winning strategy for Pl. 0 for the game $\mathscr{G}$. On the other hand, there does not exist a bounded winning strategy for Pl. 0 for the game $\mathscr{G}$. Indeed, if f is a b-bounded strategy of Pl. 0, then Pl. 1 can start by playing 0^b and then, always play 0.

Example 3.4 (Set with a bounded winning strategy and without a move-counting winning strategy). We consider the complete graph $G_{0,1}$ on $\{0,1\}$. Let $(\pi_n)_{n\geq 0}$ be an enumeration of $\mathrm{FinPaths}(G)$ with $\pi_0 = 0$. We let W be the set of any sequences in $\{0,1\}^\omega$ starting by 0 except the sequence $\rho = \pi_0\pi_1\pi_2\dots$. Let $\mathscr{G} = (G_{0,1}, 0, W)$. It is obvious that Pl. 0 has a 1-bounded winning strategy for $\mathscr{G}$ but we can also prove that Pl. 0 has no move-counting winning strategy. Indeed, if h is a move-counting strategy of Pl. 0, then Pl. 1 can start by playing a prefix π of ρ so that $\pi h(\mathrm{last}(\pi), 1)$ is a prefix of ρ. Afterwards, Pl. 1 can play π' such that $\pi h(\mathrm{last}(\pi), 1)\pi' h(\mathrm{last}(\pi'), 2)$ is a prefix of ρ and so on.

We remark that the sets W considered in these examples are *open* sets, i.e. sets on a low level of the Borel hierarchy. Moreover, by Proposition 3.2, there also exist length-counting winning strategies for these two examples. The relations between the simple strategies are thus completely characterised and are summarised in Figure 1. This Figure also contains other simple strategies which will be discussed later.

4 Link with the sets of probability 1

Let $G = (V, E)$ be a finite directed graph. We can easily define a probability measure P, on the set of infinite paths in G, by giving a weight $w_e > 0$ at each edge $e \in E$ and by considering that for any $v, v' \in V$, $p_w(v, v') = 0$ if $(v, v') \notin E$ and $p_w(v, v') = \frac{w_{(v,v')}}{\sum_{e' \text{ enabled from v}} w_{e'}}$ else, where $p_w(v, v')$ denotes the probability of taking edge (v, v') from state v. Given $v_1 \cdots v_n \in \mathrm{FinPaths}(G, v_1)$, we recall that we denote by $\mathrm{Cyl}(v_1 \cdots v_n)$ the cylinder generated by $v_1 \cdots v_n$ and defined as $\mathrm{Cyl}(v_1 \cdots v_n) = \{\rho \in \mathrm{Paths}(G, v_1) \mid v_1 \cdots v_n \text{ is a prefix of } \rho\}$.

Definition 4.1. Let $G = (V, E)$ be a finite directed graph and $w = (w_e)_{e\in E}$ a family of positive weights. We define the probability measure P_w by the relation

$$P_w(\mathrm{Cyl}(v_1 \cdots v_n)) = p_w(v_1, v_2) \cdot \cdots \cdot p_w(v_{n-1}, v_n) \tag{4.1}$$

and we say that such a probability measure is *reasonable*.

We are interested in characterising the sets W of probability 1 and their links with the different notions of simple winning strategies. We remark that, in general, Banach-Mazur games do not characterise sets of probability 1. In other words, the notions of large sets and sets of probability 1 do not coincide in general on finite graphs. Indeed, there exist some large sets of probability 0. We present here an example of such sets:

Example 4.2 (Large set of probability 0**).** We consider the complete graph $G_{0,1,2}$ on $\{0,1,2\}$ and the set $W = \{(w_i w_i^R)_{i\geq 0} \in \mathrm{Paths}(G_{0,1,2},2) : w_i \in \{0,1,2\}^*\}$, where for any finite word $\sigma \in \{0,1,2\}^*$ given by $\sigma = \sigma(1)\cdots\sigma(n)$ with $\sigma(i) \in \{0,1,2\}$, we let $\sigma^R = \sigma(n)\cdots\sigma(1)$. In other words, W is the set of runs ρ starting from 2 that we can divide into a consecutive sequence of finite words and their reverse. It is obvious that Pl. 0 has a winning strategy for the Banach-Mazur game $(G_{0,1,2},2,W)$ and thus that W is large. On the other hand, if P is the reasonable probability measure given by the weights $w_e = 1$ for any $e \in E$, then we can verify that $P(W) = 0$. Indeed, we have

$$\begin{aligned}
P(W) &\leq \sum_{n=1}^{\infty} P(\{w_0 w_0^R (w_i w_i^R)_{i\geq 1} \in W : |w_0| = n\})\\
&= \sum_{n=1}^{\infty} P(\{w_0 w_0^R w \in \mathrm{Paths}(G_{0,1,2},2) : |w_0| = n\}) \cdot P(W)\\
&\leq \sum_{n=1}^{\infty} \frac{P(W)}{3^n} = \frac{1}{2}P(W).
\end{aligned}$$

For certain families of sets, we can however have an equivalence between the notion of large set and the notion of set of probability 1. It is the case for the family of sets W representing ω-regular properties on finite graphs (see [9]). In order to prove this equivalence for ω-regular sets, Varacca and Völzer have in fact used the fact that for these sets, the Banach-Mazur game is positionally determined ([1]) and that the existence of a positional winning strategy for Pl. 0 implies $P(W) = 1$. This latter assertion follows from the fact that every positional strategy is bounded and that, by the Borel-Cantelli lemma, the set of plays consistent with a bounded strategy is a set of probability 1. Nevertheless, if W does not represent an ω-regular properties, it is possible that W is a large set of probability 1 and that there is no positional winning strategy for Pl. 0 and even no bounded or move-counting winning strategy.

Example 4.3 (Large set of probability 1 **without a positional/ bounded/ move-counting winning strategy).** We consider the complete graph $G_{0,1}$ on $\{0,1\}$ and the reasonable probability measure P given by $w_e = 1$ for any $e \in E$. Let $a_n = \sum_{k=1}^{n} k$. We let $W = \{(\sigma_k)_{k\geq 1} \in \{0,1\}^\omega : \sigma_1 = 0 \text{ and } \sigma_{a_n} = 1 \text{ for some } n > 1\}$ and $\mathscr{G} = (G_{0,1},0,W)$. Since Pl. 0 has a winning strategy for $\mathscr{G}$, we deduce that W is a large set. We can also compute that $P(W) = 1$ because if we denote by A_n, $n > 1$, the set

$$A_n := \{(\sigma_k)_{k\geq 1} \in \{0,1\}^\omega : \sigma_{a_n} = 1 \text{ and } \sigma_{a_m} = 0 \text{ for any } m < n\},$$

we have:

$$W = \dot{\bigcup}_{n>1} A_n \quad \text{and} \quad P(A_n) = \frac{1}{2^{n-1}}.$$

On the other hand, there does not exist any positional (resp. bounded) winning strategy f for Pl. 0. Indeed, if f is a positional (resp. bounded) strategy for Pl. 0 such that $f(0)$ (resp. $f(\pi)$ for any π) has length less than n, then Pl. 1 has just to start by playing a_n zeros so that Pl. 1 does not reach the index a_{n+1} and afterwards to complete the sequence by a finite number of zeros to reach the next index a_k, and so on. Moreover, there does not exist any move-counting winning strategy h for Pl. 0 because Pl. 1 can start by playing a_n zeros so that $|h(0,1)| \leq n$ and because, at each step k, Pl. 1 can complete the sequence by a finite number of zeros to reach a new index a_n such that $|h(0,k)| \leq n$.

On the other hand, we can show that the existence of a move-counting winning strategy for Pl. 0 implies $P(W) = 1$. The key idea is to realise that given a move-counting winning strategy h, the strategy $h(\cdot, n)$ is positional.

Proposition 4.4. *Let $\mathcal{G} = (G, v_0, W)$ be a Banach-Mazur game on a finite graph and P a reasonable probability measure. If Pl. 0 has a move-counting winning strategy for $\mathcal{G}$, then $P(W) = 1$.*

Proof. Let h be a move-counting winning strategy of $Pl.$ 0. We denote by f_n the strategy $h(\cdot, n)$. Each set

$$M_n := \{\rho \in \text{Paths}(G, v_0) \ : \ \rho \text{ is a play consistent with } f_n\}$$

has probability 1 since f_n is a positional winning strategy for the Banach-Mazur game (G, v_0, M_n). Moreover, if ρ is a play consistent with f_n for each $n \geq 1$, then ρ is a play consistent with h. In other words, since h is a winning strategy, we get $\bigcap_n M_n \subset W$. Therefore, as $P(M_n) = 1$ for all n, we know that $P(\bigcap_n M_n) = 1$ and we conclude that $P(W) = 1$. □

Let us notice that the converse of Proposition 4.4 is false in general. Indeed, Example 4.3 exhibit a large set W of probability 1 such that Pl. 0 has no move-counting winning strategy. However, if W is a countable intersection of ω-regular sets, then the existence of a winning strategy for Pl. 0 implies the existence of a move-counting winning strategy for Pl. 0.

Proposition 4.5. *Let $\mathcal{G} = (G, v_0, W)$ be a Banach-Mazur game on a finite graph where W is a countable intersection of ω-regular sets W_n. Pl. 0 has a winning strategy if and only if Pl. 0 has a move-counting winning strategy.*

Proof. Let $W = \bigcap_{n \geq 1} W_n$ where W_n is an ω-regular set and f a winning strategy of Pl. 0 for $\mathcal{G}$. For any $n \geq 1$, the strategy f is a winning strategy for the Banach-Mazur game (G, v_0, W_n). Thanks to [1], we therefore know that for any $n \geq 1$, there exists a positional winning strategy $\tilde{f}_n$ of Pl. 0 for (G, v_0, W_n).

Let $\phi : \mathbb{N} \to \mathbb{N}$ such that for any $k \geq 1$, $\{n \in \mathbb{N} : \phi(n) = k\}$ is an infinite[2] set. We consider the move-counting strategy $h(v, n) = \tilde{f}_{\phi(n)}(v)$. This strategy is winning because each play ρ consistent with h is a play consistent with $\tilde{f}_n$ for any n and thus

$$\begin{aligned} &\{\rho \in \text{Paths}(G, v_0) \ : \ \rho \text{ is a play consistent with } h\} \\ &\quad \subseteq \bigcap_n \{\rho \in \text{Paths}(G, v_0) \ : \ \rho \text{ is a play consistent with } \tilde{f}_n\} \\ &\quad \subseteq \bigcap_n W_n = W. \end{aligned}$$

□

Remark 4.6. We cannot extend this result to countable unions of ω-regular sets because the set of countable unions of ω-regular sets contains the open sets and Example 3.4 exhibited a Banach-Mazur game where W is an open set and Pl. 0 has a winning strategy but no move-counting winning strategy.

[2]Such a map ϕ exists because one could build a surjection $\psi : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ and then let $\phi = \psi_1$ where $\psi(n) = (\psi_1(n), \psi_2(n))$.

Remark 4.7. We also notice that if W is a countable intersection of ω-regular sets, then W is large if and only if W is a set of probability 1. Indeed, the notions of large sets and sets of probability 1 are stable by countable intersection and we know that a ω-regular set is large if and only if it is of probability 1 [9].

As a consequence of Remark 4.7, we have that if W is a ωS-regular sets, as defined in [2], the set W is large if and only if W is a set of probability 1. Indeed, it is shown in [6, 7] that ωS-regular sets are countable intersection of ω-regular sets. Nevertheless, the following example shows that, unlike the case of ω-regular sets, positional strategies are not sufficient for ωS-regular sets.

Example 4.8 (**ωS-regular set with a move-counting winning strategy and without a positional/ bounded winning strategy**)**.** We consider the complete graph $G_{0,1}$ on $\{0,1\}$ and the set W corresponding to the ωS-regular expression $((0^*1)^*0^S1)^\omega$, which corresponds to the language of words where the number of consecutive 0 is unbounded. The move-counting strategy which consists in playing n consecutive 0's at the nth step is winning for Pl. 0. However, clearly enough Pl. 0 does not have a positional (nor bounded) winning strategy for W.

Example 4.2 shows that Remark 4.7 does not extend to ω-context-free sets. Another notion of simple strategies, natural inspired by Example 4.2, is the notion of last-move strategy. A strategy f for Pl. 0 is said to be *last-move* if it only depends on the last move of Pl. 1, i.e. for any $v \in V$, for any $\pi \in \text{FinPaths}(G,v)$, $f(\pi) \in \text{FinPaths}(G,\text{last}(\pi))$ and a play ρ is consistent with f if it is of the form $(\pi_i f(\pi_i))_{i\geq 1}$. It is obvious that there exists a last-move winning strategy for Pl. 0 in the game described in Example 4.2. In particular, we deduce that the existence of a last-move winning strategy for W does not imply that W has probability 1. Example 4.2 allows also us to see that the existence of a last-move winning strategy does not imply in general the existence of a move-counting winning strategy or a bounded winning strategy. Indeed, let W be the set $\{(w_i w_i^R)_i \in \text{Paths}(G_{0,1,2},2) : w_i \in \{0,1,2\}^*\}$. Since $P(W) = 0$ (and thus $P(W) \neq 1$), we know that Pl. 0 has no move-counting winning strategy by Proposition 4.4 and no bounded winning strategy.

The notion of last-move winning strategy is in fact incomparable with the notion of move-counting winning strategy and the notion of bounded winning strategy. Indeed, on the complete graph $G_{0,1}$ on $\{0,1\}$, if we denote by W the set of runs in $G_{0,1}$ such that for any $n \geq 1$, the word 1^n appears, then Pl. 0 has a move-counting winning strategy for the game $(G_{0,1},0,W)$ but no last-move winning strategy. In the same way, if we denote by W the set of aperiodic runs on $G_{0,1}$ then Pl. 0 has a 1-bounded winning strategy for the game $(G_{0,1},0,W)$ but no last-move winning strategy (it suffices for Pl. 1 to play at each time the same word).

5 Generalised Banach-Mazur games

Let $\mathscr{G} = (G,v_0,W)$ be a Banach-Mazur game on a finite graph. We know that the existence of a bounded winning strategy or a move-counting winning strategy of Pl. 0 for $\mathscr{G}$ implies that $P(W) = 1$ for every reasonable probability measure P. Nevertheless, it is possible that $P(W) = 1$ and Pl. 0 has no bounded winning strategy and no move-counting winning strategy (Example 4.3). We therefore search a new notion of strategy such that the existence of such a winning strategy implies $P(W) = 1$ and the existence of a bounded winning strategy or a move-counting winning strategy imply the existence of such a winning strategy. To this end, we introduce a new type of Banach-Mazur games:

Definition 5.1. A *generalised Banach-Mazur game* $\mathscr{G}$ on a finite graph is a tuple (G,v_0,ϕ_0,ϕ_1,W) where $G = (V,E)$ is a finite directed graph where every vertex has a successor, $v_0 \in V$ is the initial state, $W \subset \text{Paths}(G,v_0)$, and ϕ_i is a map on $\text{FinPaths}(G,v_0)$ such that for any $\pi \in \text{FinPaths}(G,v_0)$,

$$\phi_i(\pi) \subset \mathscr{P}\big(\text{FinPaths}(G,\text{last}(\pi))\big) \setminus \{\emptyset\} \text{ and } \phi_i(\pi) \neq \emptyset.$$

A generalised[3] Banach-Mazur game $\mathscr{G} = (G, v_0, \phi_0, \phi_1, W)$ on a finite graph is a two-player game where Pl. 0 and Pl. 1 alternate in choosing *sets of finite paths* as follows: Pl. 1 begins with choosing a set of finite paths $\Pi_1 \in \phi_1(v_0)$; Pl. 0 selects a finite path $\pi_1 \in \Pi_1$ and chooses a set of finite paths $\Pi_2 \in \phi_0(\pi_1)$; Pl 1. then selects $\pi_2 \in \Pi_2$ and proposes a set $\Pi_3 \in \phi_1(\pi_1\pi_2)$ and so on. A play of $\mathscr{G}$ is thus an infinite path $\pi_1\pi_2\pi_3\dots$ in G and we say that Pl. 0 wins if this path belongs to W, while Pl. 1 wins if this path does not belong to W.

We remark that if we let $\phi_{\text{ball}}(\pi) := \{\{\pi'\} : \pi' \in \text{FinPaths}(G, \text{last}(\pi))\}$ for any $\pi \in \text{FinPaths}(G, v_0)$, then the generalised Banach-Mazur game given by $(G, v_0, \phi_{\text{ball}}, \phi_{\text{ball}}, W)$ coincides with the classical Banach-Mazur game (G, v_0, W). We also obtain a game similar to the classical Banach-Mazur game if we consider the function $\phi(\pi) = \mathscr{P}(\text{FinPaths}(G, \text{last}(\pi)))$. On the other hand, if we consider $\phi(\pi) := \{\{\pi'\} : \pi' \in \text{FinPaths}(G, \text{last}(\pi)), |\pi'| = 1\}$, we obtain the classical games on graphs such as the ones studied in [5].

We are interested in defining a map ϕ_0 such that Pl. 0 has a winning strategy for $(G, v_0, \phi_0, \phi_{\text{ball}}, W)$ if and only if $P(W) = 1$. To this end, we notice that we can restrict actions of Pl. 0 by forcing each set in $\phi_0(\pi)$ to be "big" in some sense. The idea to characterise $P(W) = 1$ is therefore to force Pl. 0 to play with finite sets of finite paths of conditional probability bigger than α for some $\alpha > 0$.

Definition 5.2. Let $\mathscr{G} = (G, v_0, W)$ be a Banach-Mazur game on a finite graph, P a reasonable probability measure and $\alpha > 0$. An *α-strategy* of Pl. 0 for $\mathscr{G}$ is a strategy of Pl. 0 for the generalised Banach-Mazur game $\mathscr{G}_\alpha = (G, v_0, \phi_\alpha, \phi_{\text{ball}}, W)$ where

$$\phi_\alpha(\pi) = \Big\{\Pi \subset \text{FinPaths}(G, \text{last}(\pi)) : P\Big(\bigcup_{\pi' \in \Pi} \text{Cyl}(\pi\pi') \Big| \text{Cyl}(\pi)\Big) \geq \alpha \text{ and } \Pi \text{ is finite}\Big\}.$$

We recall that, given two events A, B with $P(B) > 0$, the conditional probability $P(A|B)$ is defined by $P(A|B) := P(A \cap B)/P(B)$.

We notice that every bounded strategy can be seen as an α-strategy for some $\alpha > 0$, since for any $N \geq 1$, there exists $\alpha > 0$ such that for any π of length less than N, we have $P(\{\pi\}) \geq \alpha$. We can also show that the existence of a move-counting winning strategy for Pl. 0 implies the existence of a winning α-strategy for Pl. 0 for every $0 < \alpha < 1$.

Proposition 5.3. *Let $\mathscr{G} = (G, v_0, W)$ be a Banach-Mazur game on a finite graph. If Pl. 0 has a move-counting winning strategy, then Pl. 0 has a winning α-strategy for every* $0 < \alpha < 1$.

Proof. Let P be a reasonable probability measure, h a move-counting winning strategy for Pl. 0 and $0 < \alpha < 1$. We denote by g_n the positional strategy defined by

$$g_n(v) = h(v,1)\ h\big(\text{last}(h(v,1)), 2\big)\ \cdots\ h\big(\text{last}(h(v,1)\ h(\text{last}(h(v,1)), 2) \cdots), n\big).$$

Let us notice that the definition of the g_n's implies that for any increasing sequence (n_k), a play of the form

$$\pi_1\ g_{n_1}(\text{last}(\pi_1))\ \pi_2\ g_{n_2}(\text{last}(\pi_2))\ \cdots\ \pi_k\ g_{n_k}(\text{last}(\pi_k))\ \cdots \tag{5.1}$$

is consistent with h. Since g_n is a positional strategy, we know that each set

$$M_n := \{\rho \in \text{Paths}(G, v_0)\ :\ \rho \text{ is a play consistent with } g_n\}$$

[3]We only present here a generalisation of Banach-Mazur games on finite graphs but this generalisation could be extended to Banach-Mazur games on topological spaces by asking that for any non-empty open set O, $\phi_i(O)$ is a collection of non-empty open subsets of O.

has probability 1. In particular, for any $\pi_0 \in \mathrm{FinPaths}(G, v_0)$, we deduce that $P(M_n|\mathrm{Cyl}(\pi_0)) = 1$. Since

$$M_n \cap \mathrm{Cyl}(\pi_0) \subseteq \bigcup_{\pi \in \mathrm{FinPaths}(G,\mathrm{last}(\pi_0))} \mathrm{Cyl}\big(\pi_0 \pi g_n(\mathrm{last}(\pi))\big),$$

we have

$$P\Big(\bigcup_{\pi \in \mathrm{FinPaths}(G,\mathrm{last}(\pi_0))} \mathrm{Cyl}\big(\pi_0 \pi g_n(\mathrm{last}(\pi))\big) \Big| \mathrm{Cyl}(\pi_0)\Big) = 1$$

and since $\mathrm{FinPaths}(G, \mathrm{last}(\pi_0))$ is countable, we deduce that for any $n \geq 1$, any $\pi_0 \in \mathrm{FinPaths}(G, v_0)$, there exists a finite subset $\Pi_n(\pi_0) \subset \mathrm{FinPaths}(G, \mathrm{last}(\pi_0))$ such that

$$P\Big(\bigcup_{\pi \in \Pi_n(\pi_0)} \mathrm{Cyl}\big(\pi_0 \pi g_n(\mathrm{last}(\pi))\big) \Big| \mathrm{Cyl}(\pi_0)\Big) \geq \alpha.$$

We denote by $\Pi'_n(\pi_0)$ the set $\{\pi g_n(\mathrm{last}(\pi)) : \pi \in \Pi_n(\pi_0)\}$ and we let

$$f(\pi_0) := \Pi'_{|\pi_0|}(\pi_0).$$

The above-defined strategy f is therefore a winning α-strategy for Pl. 0 since each play consistent with f is of the form (5.1) for some sequence (n_k) and thus consistent with h. $\square$

Moreover, the existence of a winning α-strategy for some $\alpha > 0$ still implies $P(W) = 1$.

Theorem 5.4. *Let $\mathscr{G} = (G, v_0, W)$ be a Banach-Mazur game on a finite graph and P a reasonable probability measure. If Pl. 0 has a winning α-strategy for some $\alpha > 0$, then $P(W) = 1$.*

Proof. Let f be a winning α-strategy. We consider an increasing sequence $(a_n)_{n\geq 1}$ such that for any $n \geq 1$, any π of length a_n, each $\pi' \in f(\pi)$ has length less than $a_{n+1} - a_n$; this is possible because for any π, $f(\pi)$ is a finite set by definition of α-strategy. Without loss of generality[4], we can even assume that for any $n \geq 1$, any π of length a_n, each $\pi' \in f(\pi)$ has exactly length $a_{n+1} - a_n$. We therefore let

$$A := \{(\sigma_k)_{k\geq 1} \in \mathrm{Paths}(G, v_0) : \#\{n : (\sigma_k)_{a_n+1 \leq k \leq a_{n+1}} \in f((\sigma_k)_{1\leq k \leq a_n})\} = \infty\}.$$

In other words, $(\sigma_k)_{k\geq 1} \in A$ if (σ_k) can be seen as a play where f has been played on an infinite number of indices a_n. Since f is a winning strategy, A is included in W and it thus suffices to prove that $P(A) = 1$.

We first notice that for any $m \geq 1$, any $n \geq m$, if we let

$$B_{m,n} = \{(\sigma_k)_{k\geq 1} \in \mathrm{Paths}(G, v_0) : (\sigma_k)_{a_j+1 \leq k \leq a_{j+1}} \notin f((\sigma_k)_{1\leq k \leq a_j}),\ \forall m \leq j \leq n\},$$

then $P(B_{m,n}) \leq (1-\alpha)^{n+1-m}$ as f is an α-strategy. We therefore deduce that for any $m \geq 1$,

$$P\Big(\bigcap_{n=m}^{\infty} B_{m,n}\Big) = 0$$

and since $A^c = \bigcup_{m\geq 1} \bigcap_{n=m}^{\infty} B_{m,n}$, we conclude that $P(A) = 1$. $\square$

[4]Let π be a finite path and $n_\pi \geq \max\{|\tau| \text{ such that } \tau \in f(\pi)\}$. One can define $\tilde{f}(\pi)$ as the set of finite paths σ of length n_π such that τ is a prefix of σ, for some $\tau \in f(\pi)$. Given a play ρ, one can show that ρ is consistent with f if and only if ρ is consistent with $\tilde{f}$.

If W is a countable intersection of open sets, we can prove the converse of Theorem 5.4 and so obtain a characterisation of sets of probability 1.

Theorem 5.5. *Let $\mathscr{G} = (G, v_0, W)$ be a Banach-Mazur game on a finite graph where W is a countable intersection of open sets and P a reasonable probability measure. Then the following assertions are equivalent:*

1. $P(W) = 1$,
2. *Pl. 0 has a winning α-strategy for some $\alpha > 0$,*
3. *Pl. 0 has a winning α-strategy for all $0 < \alpha < 1$.*

Proof. We have already proved 2. $\Rightarrow$ 1., and 3. $\Rightarrow$ 2. is obvious.
1. $\Rightarrow$ 3. Let $0 < \alpha < 1$. Let $W = \bigcap_{n=1}^{\infty} W_n$ where W_n's are open sets. Since $P(W) = 1$, we deduce that for any $n \geq 1$, $P(W_n) = 1$. We can therefore define a winning α-strategy f of Pl. 0 as follows: if $\mathrm{Cyl}(\pi) \subset \bigcap_{k=1}^{n-1} W_k$ and $\mathrm{Cyl}(\pi) \not\subset W_n$, we let $f(\pi)$ be a finite set $\Pi \subset \mathrm{FinPaths}(G, \mathrm{last}(\pi))$ such that $P\Big(\bigcup_{\pi' \in \Pi} \mathrm{Cyl}(\pi\pi') | \mathrm{Cyl}(\pi)\Big) \geq \alpha$ and for any $\pi' \in \Pi$, $\mathrm{Cyl}(\pi\pi') \subset W_n$. Such a finite set Π exists because W_n has probability 1 and W_n is an open set, i.e. a countable union of cylinders. This concludes the proof. □

Remark 5.6. We cannot hope to generalise the latter result to any set W. More precisely, there exist sets of probability 1 for which no winning α-strategy exists. Indeed, given a set W, on the one hand, the existence of a winning α-strategy for W implies the existence of a winning strategy for W, and thus in particular such a W is large. On the other hand, we know that there exists some meagre (in particular not large) set of probability 1 (see Example 4.2). However, one can ask whether the existence of a winning α-strategy is equivalent to the fact that W is a large set of probability 1.

When W is a countable intersection of open sets, we remark that the generalised Banach-Mazur game $\mathscr{G}_\alpha = (G, v_0, \phi_\alpha, \phi_{\mathrm{ball}}, W)$ is in fact determined.

Theorem 5.7. *Let $\mathscr{G}_\alpha$ be the generalised Banach-Mazur game given by $\mathscr{G}_\alpha = (G, v_0, \phi_\alpha, \phi_{ball}, W)$ where G is a finite graph, W is a countable intersection of open sets and P a reasonable probability measure. Then the following assertions are equivalent:*

1. $P(W) < 1$,
2. *Pl. 1 has a winning strategy for $\mathscr{G}_\alpha$ for some $\alpha > 0$,*
3. *Pl. 1 has a winning strategy for $\mathscr{G}_\alpha$ for all $0 < \alpha < 1$.*

Proof. We deduce from Theorem 5.5 that 2. $\Rightarrow$ 1. because $\mathscr{G}_\alpha$ is a zero-sum game, and 3. $\Rightarrow$ 2. is obvious.
1. $\Rightarrow$ 3. Let $W = \cap_{n=1}^{\infty} W_n$ with $P(W) < 1$ and W_n open. We know that there exists $n \geq 1$ such that $P(W_n) < 1$. It then suffices to prove that Pl. 1 has a winning strategy for the generalised Banach-Mazur game $(G, v_0, \phi_\alpha, \phi_{\mathrm{ball}}, W_n)$ for all $0 < \alpha < 1$. Without loss of generality, we can thus assume that W is an open set. We recall that W is open if and only if it is a countable union of cylinders. Since any strategy of Pl. 1 is winning if $W = \emptyset$, we also suppose that $W \neq \emptyset$.

Let $0 < \alpha < 1$. We first show that there exists a finite path $\pi_1 \in \mathrm{FinPaths}(G, v_0)$ such that any set $\Pi_2 \in \phi_\alpha(\pi_1)$ contains a finite path π_2 satisfying

$$P(W | \mathrm{Cyl}(\pi_1\pi_2)) \leq P(W) < 1. \tag{5.2}$$

Let

$$I_W := \inf\{P(W | \mathrm{Cyl}(\pi)) : \pi \in \mathrm{FinPaths}(G, v_0)\}. \tag{5.3}$$

Since W is a non-empty union of cylinders, there exists $\sigma \in \text{FinPaths}(G, v_0)$ such that $P(W|\text{Cyl}(\sigma)) = 1$. We remark that $P(W) = \sum_{\pi:|\pi|=|\sigma|} P(W|\text{Cyl}(\pi))P(\text{Cyl}(\pi))$ and $\sum_{\pi:|\pi|=|\sigma|} P(\text{Cyl}(\pi)) = 1$. Therefore, since $P(W|\text{Cyl}(\sigma)) > P(W)$, we deduce that there exists $\pi \in \text{FinPaths}(G, v_0)$ with $|\pi| = |\sigma|$ such that $P(W|\text{Cyl}(\pi)) < P(W)$. We conclude that $I_W < P(W)$ and thus, by definition of I_W, there exists $\pi_1 \in \text{FinPaths}(G, v_0)$ such that

$$I_W + \frac{1}{\alpha}(P(W|\text{Cyl}(\pi_1)) - I_W) < P(W). \tag{5.4}$$

Let $\Pi_2 \in \phi_\alpha(\pi_1)$. We consider $\tau_1, \dots, \tau_n \in \Pi_2$ and $\sigma_1, \dots, \sigma_m \in \text{FinPaths}(G, \text{last}(\pi_1))$ such that cylinders $\text{Cyl}(\tau_i)$, $\text{Cyl}(\sigma_j)$ are pairwise disjoint, $\bigcup_{\pi\in\Pi_2} \text{Cyl}(\pi) \subset \bigcup_{i=1}^n \text{Cyl}(\tau_i)$ and

$$\text{Paths}(G, \text{last}(\pi_1)) = \bigcup_{i=1}^{n} \text{Cyl}(\tau_i) \cup \bigcup_{j=1}^{m} \text{Cyl}(\sigma_j). \tag{5.5}$$

Assume that for all $1 \le i \le n$, we have

$$P(W|\text{Cyl}(\pi_1\tau_i)) > P(W). \tag{5.6}$$

Then, we get

$$\begin{aligned}
&P(W|\text{Cyl}(\pi_1))\\
&= \sum_{i=1}^{n} P(W \cap \text{Cyl}(\pi_1\tau_i)|\text{Cyl}(\pi_1)) + \sum_{j=1}^{m} P(W \cap \text{Cyl}(\pi_1\sigma_j)|\text{Cyl}(\pi_1)) \;\; \text{by disjointness and (5.5)}\\
&= \sum_{i=1}^{n} P(W|\text{Cyl}(\pi_1\tau_i))P(\text{Cyl}(\pi_1\tau_i)|\text{Cyl}(\pi_1)) + \sum_{j=1}^{m} P(W|\text{Cyl}(\pi_1\sigma_j))P(\text{Cyl}(\pi_1\sigma_j)|\text{Cyl}(\pi_1))\\
&\ge P(W)\sum_{i=1}^{n} P(\text{Cyl}(\pi_1\tau_i)|\text{Cyl}(\pi_1)) + I_W \sum_{j=1}^{m} P(\text{Cyl}(\pi_1\sigma_j)|\text{Cyl}(\pi_1)) \;\; \text{by (5.6) and (5.3)}\\
&\ge P(W)\sum_{i=1}^{n} P(\text{Cyl}(\pi_1\tau_i)|\text{Cyl}(\pi_1)) + I_W\big(1 - \sum_{i=1}^{n} P(\text{Cyl}(\pi_1\tau_i)|\text{Cyl}(\pi_1))\big) \;\; \text{by (5.5)}\\
&\ge P(W)P\big(\bigcup_{\pi\in\Pi_2} \text{Cyl}(\pi_1\pi)|\text{Cyl}(\pi_1)\big) + I_W\big(1 - P\big(\bigcup_{\pi\in\Pi_2} \text{Cyl}(\pi_1\pi)|\text{Cyl}(\pi_1)\big)\big) \;\; \text{by properties of } \tau_i\text{'s}\\
&\ge P(W)\alpha + I_W(1-\alpha) \quad (\text{because } \Pi_2 \in \phi_\alpha(\pi_1) \text{ and } P(W) > I_W)
\end{aligned}$$

and thus $P(W) \le I_W + \frac{1}{\alpha}(P(W|\text{Cyl}(\pi_1)) - I_W)$ which is a contradiction with (5.4). We conclude that if π_1 is given by (5.4), then any set $\Pi_2 \in \phi_\alpha(\pi_1)$ contains a finite path π_2 satisfying (5.2).

We can now exhibit a winning strategy for Pl. 1. We assume that Pl. 1 begins with playing a finite path π_1 satisfying (5.4). Let f be an α-strategy. We know that Pl. 1 can select a finite path $\pi_2 \in f(\pi_1)$ satisfying (5.2), i.e. $P(W|\text{Cyl}(\pi_1\pi_2)) \le P(W)$. By repeating the above method from $\pi_1\pi_2$, we also deduce the existence of a finite path π_3 such that any set $\Pi_4 \in \phi_\alpha(\pi_1\pi_2\pi_3)$ contains a finite path π_4 satisfying $P(W|\text{Cyl}(\pi_1\pi_2\pi_3\pi_4)) \le P(W)$. We can thus assume that Pl. 1 plays such a finite path π_3 and then selects $\pi_4 \in f(\pi_1\pi_2\pi_3)$ such that $P(W|\text{Cyl}(\pi_1\pi_2\pi_3\pi_4)) \le P(W)$. This strategy is a winning strategy for Pl. 1. Indeed, as W is an open set and thus a countable union of cylinders, if $P(W|\text{Cyl}(\pi_1 \cdots \pi_{2n})) \le P(W) < 1$ for any n, then $\pi_1\pi_2\pi_3 \cdots \notin W$. □

Corollary 5.8. *Let $0 < \alpha < 1$. The generalised Banach-Mazur game $\mathcal{G}_\alpha = (G, v_0, \phi_\alpha, \phi_{ball}, W)$ is determined when W is a countable intersection of open sets. More precisely, Pl. 0 has a winning strategy for $\mathcal{G}_\alpha$ if and only if $P(W) = 1$, and Pl. 1 has a winning strategy for $\mathcal{G}_\alpha$ if and only if $P(W) < 1$.*

Since the existence of a bounded winning strategy for Pl. 0 implies the existence of a winning α-strategy for Pl. 0 and the existence of a move-counting winning strategy for Pl. 0 implies the existence of a winning α-strategy for Pl. 0, we deduce from Example 3.3 and Example 3.4 that in general, the existence of a winning α-strategy for Pl. 0 does not imply the existence of a move-counting winning strategy Pl. 0 and the existence of a bounded winning strategy for Pl. 0. On the other hand, we know that there exists a Banach-Mazur game for which Pl. 0 has a bounded winning strategy and no last-move winning strategy. The existence of a winning α-strategy thus does not imply in general the existence of a last-move winning strategy. Conversely, if we consider the game $(G_{0,1}, 0, W)$ described in Example 4.2, Pl. 0 has a last-move winning strategy but no winning α-strategy (as $P(W) = 0$). The notion of α-strategy is thus incomparable with the notion of last-move strategy.

6 More on simple strategies

We finish this paper by considering the crossings between the different notions of simple strategies and the notion of bounded strategy i.e. the bounded length-counting strategies, the bounded move-counting strategies and the bounded last-move strategies. Obviously, the existence of a bounded length-counting winning strategy for Pl. 0 implies the existence of a length-counting winning strategy for Pl. 0, and we have this implication for each notion of bounded strategies and their no bounded counterpart. We start by noticing that the existence of a bounded move-counting winning strategy is equivalent to the existence of a positional winning strategy.

Proposition 6.1. *Let $\mathscr{G} = (G, v_0, W)$ be a Banach-Mazur game on a finite graph. Pl. 0 has a bounded move-counting winning strategy if and only if Pl. 0 has a positional winning strategy.*

Proof. Let h be a bounded move-counting winning strategy for Pl. 0. We denote by $C_1, \dots, C_N$ the bottom strongly connected components (BSCC) of G. Let $1 \le i \le N$. Since h is a bounded strategy and G is finite, there exist some finite paths $w_1^{(i)}, \dots, w_{k_i}^{(i)} \subset C_i$ such that for any $v \in C_i$, for any $n \ge 1$,

$$h(v,n) \in \{w_1^{(i)}, \dots, w_{k_i}^{(i)}\}.$$

Let $v \in V$. If $v \in C_i$, we let $f(v) = \sigma_0 w_1^{(i)} \sigma_1 w_2^{(i)} \sigma_2 \dots w_{k_i}^{(i)}$ where σ_l are finite paths in C_i such that $f(v)$ is a finite path in C_i starting from v. If $v \notin \bigcup_i C_i$, we let $f(v) = \sigma_v$ where σ_v starts from v and leads into a BSCC of G. The positional strategy f is therefore winning as each play ρ consistent with f can be seen as a play consistent with h. □

The other notions of bounded strategies are not equivalent to any other notion of simple strategy.

Example 6.2 (Set with a bounded length-counting winning strategy and without a positional winning strategy). Let $G_{0,1}$ be the complete graph on $\{0,1\}$, (ρ_n) an enumeration of finite words in $\{0,1\}$ and $\rho_{\text{target}} = 0\rho_1\rho_2\cdots$. We consider the set $W = \{\sigma \in \{0,1\}^\omega : \#\{i \ge 1 : \sigma(i) = \rho_{\text{target}}(i)\} = \infty\}$. It is evident that Pl. 0 has a bounded length-counting winning strategy for the game $(G_{0,1}, 0, W)$. However, Pl. 0 has no positional winning strategy. Indeed, if f is a positional strategy such that $f(0) = a(1)\cdots a(k)$, then Pl. 1 can play according to the strategy h defined by $h(\sigma(1)\cdots\sigma(n)) = \sigma(n+1)\cdots\sigma(N)\, 0$ such that for any $n+1 \le i \le N$, $\sigma(i) \ne \rho_{\text{target}}(i)$, $\rho_{\text{target}}(N+1) \ne 0$ and for any $1 \le i \le k$, $a(i) \ne \rho_{\text{target}}(N+i+1)$.

Example 6.3 (Set with a bounded last-move winning strategy and without a positional winning strategy). Let $G_{0,1,2}$ be the complete graph on $\{0,1,2\}$. For any $\phi : \{0,1,2\}^* \to \{0,1\}$, if we consider the set $W := \{(\pi_i\phi(\pi_i))_{i\ge 1} : \pi_i \in \{0,1,2\}^*\}$, then Pl. 0 has a 1-bounded last-move winning strategy given by ϕ for the game $(G_{0,1,2}, 2, W)$. On the other hand, we can choose ϕ such that Pl. 0 has no positional winning strategy. Indeed, it suffices to choose $\phi : \{0,1,2\}^* \to \{0,1\}$ such that for any $\pi \in \{0,1,2\}^*$, any $n \geq 1$, any $\sigma(1), \ldots, \sigma(n) \in \{0,1,2\}$, there exists $k \geq 1$ such that $\phi(\pi 2^k) \neq \sigma(1)$ and for any $1 \leq i \leq n-1$, $\phi(\pi 2^k \sigma(1) \cdots \sigma(i)) \neq \sigma(i+1)$. Such a function exists because the set $\{0,1,2\}^*$ is countable. Therefore, Pl. 0 has no positional winning strategy for the game $(G_{0,1,2}, 2, W)$ because, if f is a positional strategy and $f(2) = \sigma(1) \ldots \sigma(n)$, then Pl. 1 can play consistent with the strategy h defined by $h(\pi) = 2^k$ such that $\phi(\pi 2^k) \neq \sigma(1)$ and for any $1 \leq i \leq n-1$, $\phi(\pi 2^k \sigma(1) \cdots \sigma(i)) \neq \sigma(i+1)$. Pl. 0 has thus a 1-bounded last-move winning strategy and no positional winning strategy for the game $(G_{0,1,2}, 2, W)$.

Example 6.4 (Set with a bounded winning strategy and without a bounded length-counting winning strategy). Let $G_{0,1,2,3}$ be the complete graph on $\{0,1,2,3\}$. For any $\phi : \{0,1,2,3\}^* \to \{0,1\}$, if we denote by W the set of runs ρ such that $\#\{n \geq 1 : \phi(\rho(1) \ldots \rho(n)) = \rho(n+1)\} = \infty$, then Pl. 0 has a 1-bounded winning strategy given by ϕ for the game $(G_{0,1,2,3}, 2, W)$. We now show how we can define ϕ so that Pl. 0 has no bounded length-counting winning strategy. Let $n_k = \sum_{i=1}^{k} 3i$. We choose $\phi : \{0,1,2,3\}^* \to \{0,1\}$ such that for any $k \geq 1$, any $\pi \in \{0,1,2,3\}^*$ of length n_k and any $\sigma(1), \ldots, \sigma(k) \in \{0,1,2,3\}$, there exists $\tau \in \{2,3\}^*$ of length $2k$ such that $\phi(\pi\tau 2) \neq \sigma(1)$ and for any $1 \leq i \leq k-1$, $\phi(\pi\tau 2 \sigma(1) \cdots \sigma(i)) \neq \sigma(i+1)$. Such a function exists because the cardinality of $\{2,3\}^{2k}$ is equal to the cardinality of $\{0,1,2,3\}^k$ and the length of $\pi\tau 2 \sigma(1) \cdots \sigma(k) < n_{k+1}$. Therefore, Pl. 0 has no bounded length-counting winning strategy because if f is a k-bounded length-counting strategy (for some $k \in \mathbb{N}$) and $f(2, n_k + k + 1) = \sigma$, then Pl. 1 can start by playing $2^{n_k}\tau 2$, where $\tau \in \{2,3\}^*$ of length $2k$ such that $\phi(\pi\tau 2) \neq \sigma(1)$ and for any $1 \leq i \leq k-1$, $\phi(\pi\tau 2 \sigma(1) \cdots \sigma(i)) \neq \sigma(i+1)$, and if Pl. 1 keep playing with same philosophy, then Pl. 1 wins the play. Pl. 0 has thus a 1-bounded winning strategy and no bounded length-counting winning strategy for the game $(G_{0,1,2}, 2, W)$.

The relations between the different notions of simple strategies on a finite graph can be summarised as depicted in Figure 1. We draw attention to the fact that the situation is very different in the case of infinite graphs. For example, a positional strategy can be unbounded, the notion of length-counting winning strategy is not equivalent to the notion of winning strategy (except if the graph is finitely branching), and the notion of bounded move-counting winning strategy for Pl. 0 is not equivalent to the notion of positional winning strategy.

Example 6.5 (Set on an infinite graph with a bounded move-counting winning strategy and without a positional winning strategy). We consider the complete graph $G_{\mathbb{N}}$ on $\mathbb{N}$ and the game $\mathscr{G} = (G_{\mathbb{N}}, 0, W)$ where $W = \{(\sigma_k) \in \mathbb{N}^\omega : \forall\, n \geq 1,\ \exists\, k \geq 1,\ (\sigma_k, \sigma_{k+1}) = (n, n+1)\}$. Pl. 0 has a bounded move-counting winning strategy given by $h(v,n) = n\, n+1$ but no positional winning strategy.

References

[1] Dietmar Berwanger, Erich Grädel & Stephan Kreutzer (2003): *Once upon a Time in a West - Determinacy, Definability, and Complexity of Path Games*. In: *LPAR*, *Lecture Notes in Computer Science* 2850, Springer, pp. 229–243, doi:`10.1007/978-3-540-39813-4_16`.

[2] Mikolaj Bojanczyk & Thomas Colcombet (2006): *Bounds in w-Regularity*. In: *LICS*, IEEE Computer Society, pp. 285–296, doi:`10.1109/LICS.2006.17`.

[3] Erich Grädel (2008): *Banach-Mazur Games on Graphs*. In: *FSTTCS*, *LIPIcs* 2, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 364–382, doi:`10.4230/LIPIcs.FSTTCS.2008.1768`.

[4] Erich Grädel & Simon Leßenich (2012): *Banach-Mazur Games with Simple Winning Strategies*. In: *CSL*, *LIPIcs* 16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 305–319, doi:`10.4230/LIPIcs.CSL.2012.305`.

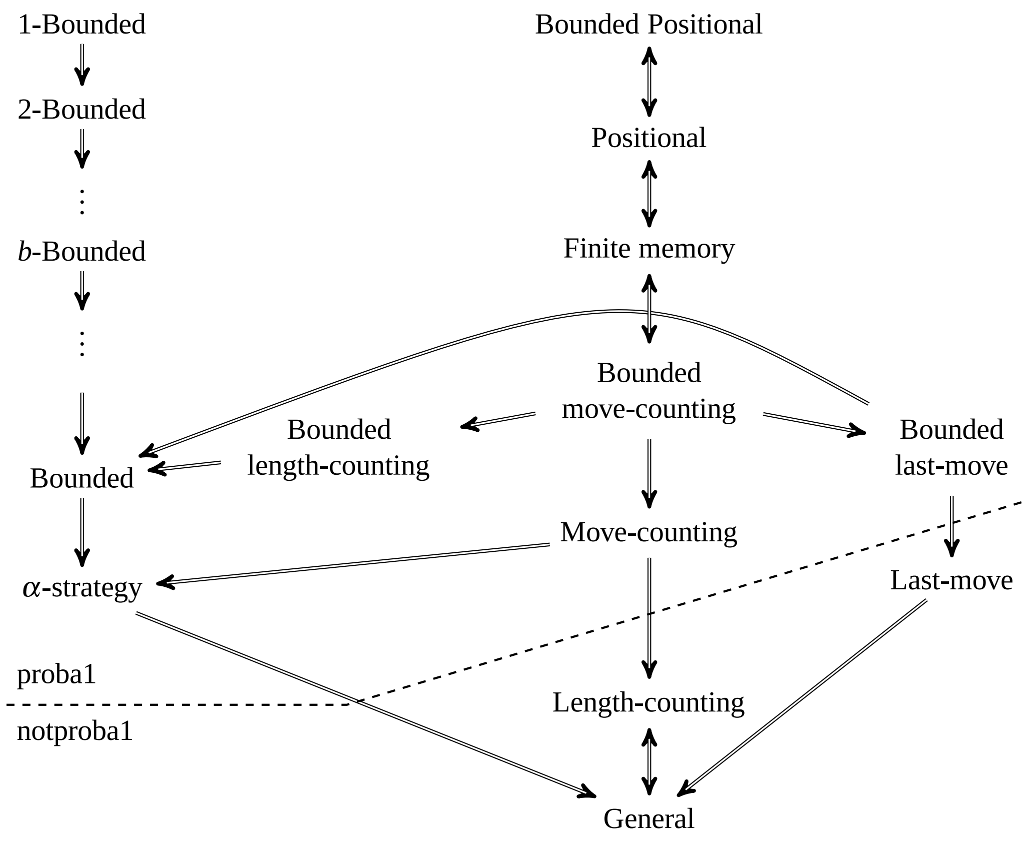


Figure 1: Winning strategies for Player 0 on finite graphs.

[5] Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors (2002): *Automata, Logics, and Infinite Games: A Guide to Current Research*. *LNCS* 2500, Springer.

[6] Szczepan Hummel & Michał Skrzypczak (2012): *The topological complexity of MSO+U and related automata models*. *Fund. Inform.* 119(1), pp. 87–111, doi:`10.3233/FI-2012-728`.

[7] Szczepan Hummel, Michał Skrzypczak & Szymon Torunczyk (2010): *On the Topological Complexity of MSO+U and Related Automata Models*. In: *MFCS*, *Lecture Notes in Computer Science* 6281, Springer, pp. 429–440, doi:`10.1007/978-3-642-15155-2_38`.

[8] John C. Oxtoby (1957): *The Banach-Mazur Game and Banach Category Theorem*. *Annals of Mathematical Studies* 39, pp. 159–163. Contributions to the Theory of Games, volume 3.

[9] Daniele Varacca & Hagen Völzer (2006): *Temporal Logics and Model Checking for Fairly Correct Systems.* In: *Proc. 21st Ann. Symp. Logic in Computer Science (LICS'06)*, IEEE Comp. Soc. Press, pp. 389–398, doi:`10.1109/LICS.2006.49`.

The Rabin index of parity games

Michael Huth and Jim Huan-Pu Kuo
Department of Computing, Imperial College London
London, SW7 2AZ, United Kingdom
{m.huth, jimhkuo}@imperial.ac.uk

Nir Piterman
Department of Computer Science, University of Leicester
Leicester, LE1 7RH, United Kingdom
nir.piterman@leicester.ac.uk

We study the descriptive complexity of parity games by taking into account the coloring of their game graphs whilst ignoring their ownership structure. Colored game graphs are identified if they determine the same winning regions and strategies, for *all* ownership structures of nodes. The Rabin index of a parity game is the minimum of the maximal color taken over all equivalent coloring functions. We show that deciding whether the Rabin index is at least k is in P for $k = 1$ but NP-hard for all *fixed* $k \geq 2$. We present an EXPTIME algorithm that computes the Rabin index by simplifying its input coloring function. When replacing simple cycle with cycle detection in that algorithm, its output over-approximates the Rabin index in polynomial time. Experimental results show that this approximation yields good values in practice.

1 Introduction

Parity games (see e.g. [11]) are infinite, 2-person, 0-sum, graph-based games that are hard to solve. Their nodes are colored with natural numbers, controlled by different players, and the winning condition of plays depends on the minimal color occurring in cycles. The condition for winning a node, therefore, is an alternation of existential and universal quantification. In practice, this means that the maximal color of its coloring function is the only exponential source for the worst-case complexity of most parity game solvers, e.g. for those in [11, 8, 9].

One approach taken in analyzing the complexity of parity games, and in so hopefully improving the complexity of their solution, is through the study of the descriptive complexity of their underlying game graph. This method therefore ignores the ownership structure on parity games.

An example of this approach is the notion of DAG-width in [1]. Every directed graph has a DAG-width, a natural number that specifies how well that graph can be decomposed into a directed acyclic graph (DAG). The decision problem for DAG-width, whether the DAG-width of a directed graph is at most k, is NP-complete in k [1]. But parity games whose DAG-width is below a given threshold have polynomial-time solutions [1]. The latter is a non-trivial result since DAG-width also ignores the colors of a parity game.

In this paper we want to develop a similar measure of the descriptive complexity of parity games, their *Rabin index*, a natural number that ignores the ownership of nodes, but does take into account the colors of a parity game. Intuitively, the Rabin index is the number of colors that are *required* to capture the complexity of the game structure. By measuring and reducing the number of colors we hope to improve the complexity of analyzing parity games. [1] The reductions we propose are related to priority compression and propagation in [6] but, in contrast, exploit the *cyclic* structure of game graphs.

[1] We note that if we also were to account for ownership, we could solve the parity game and assign color 0 to nodes won by player 0 and color 1 to nodes won by player 1. Thus, this would reduce the index of *all* games to at most 2. However, this would prevent a more fine-grained analysis of the structural complexity of the game and defeats the purpose of simplifying parity games *before* solving them.

Gabriele Puppis, Tiziano Villa (Eds.): Fourth International Symposium on Games, Automata, Logics and Formal Verification
EPTCS 119, 2013, pp. 35–49, doi:10.4204/EPTCS.119.6

The name for the measure developed here is inspired by related work on the Wagner hierarchy for automata on infinite words [10]: Carton and Maceiras use similar ideas to compute and minimize the Rabin index of deterministic parity automata on infinite words [2]. To the best of our knowledge, our work is the first to study this notion in the realm of infinite, 2-person games.

The idea behind our Rabin index is that one may change the coloring function of a parity game to another one if that change neither affects the winning regions nor the choices of winning strategies. This yields an equivalence relation between coloring functions. For the coloring function of a parity game, we then seek an equivalent coloring function with the smallest possible maximal color, and call that minimal maximum the Rabin index of the respective parity game.

The results we report here about this Rabin index are similar in spirit to those developed for DAG-width in [1] but there are important differences:

- We propose a measure of descriptive complexity that is closer to the structure of the parity game as it only forgets ownership of nodes and not their colors.
- We prove that for every *fixed* $k \geq 2$, deciding whether the Rabin index of a parity game is at least k is NP-hard.
- We can characterize the above equivalence relation in terms of the parities of minimal colors on *simple* cycles in the game graph.
- We use that characterization to design an algorithm that computes the Rabin index and a witnessing coloring function in exponential time.
- We show how the same algorithm efficiently computes sound approximations of the Rabin index when simple cycles are abstracted by cycles.
- We derive from that approximation an abstract Rabin index of parity games such that games with bounded abstract Rabin index are efficiently solvable.
- We conduct detailed experimental studies that corroborate the utility of that approximation, also as a preprocessor for solvers.

Outline of paper. Section 2 contains background for our technical develeopments. In Section 3, we define the equivalence between coloring functions, characterize it in terms of simple cycles, and use that characterization to define the Rabin index of parity games. In Section 4 we develop an algorithm that runs in exponential time and computes a coloring function which witnesses the Rabin index of the input coloring function. The complexity of the natural decision problems for the Rabin index is studied in Section 5. An abstract version of our algorithm is shown to soundly approximate that coloring function and Rabin index in Section 6. Section 7 contains our experimental results for this abstraction. And we conclude the paper in Section 9. An appendix contains selected proofs.

2 Background

We write $\mathbb{N}$ for the set $\{0,1,\dots\}$ of natural numbers. A parity game G is a tuple (V,V_0,V_1,E,c) where V is a non-empty set of nodes partitioned into possibly empty node sets V_0 and V_1, with an edge relation $E \subseteq V \times V$ (where for all v in V there is a w in V with (v,w) in E), and a coloring function $c\colon V \to \mathbb{N}$.

Throughout, we write s for one of 0 or 1. In a parity game, player s owns the nodes in V_s. A play from some node v_0 results in an infinite play $P = v_0 v_1 \dots$ in (V,E) where the player who owns v_i chooses the successor v_{i+1} such that (v_i, v_{i+1}) is in E. Let $\mathsf{Inf}(P)$ be the set of colors that occur in P infinitely often: $\mathsf{Inf}(P) = \{k \in \mathbb{N} \mid \forall j \in \mathbb{N}\colon \exists i \in \mathbb{N}\colon i > j \text{ and } k = c(v_i)\}$. Player 0 wins play P iff $\min \mathsf{Inf}(P)$ is even; otherwise player 1 wins play P.

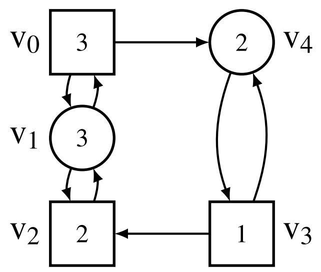


Figure 1: A parity game with winning regions $W_0 = \{v_1, v_2\}$ and $W_1 = \{v_0, v_3, v_4\}$; winning strategies for players 0 and 1 map v_1 to v_2, respectively v_0 and v_3 to v_4

A strategy for player s is a total function $\tau\colon V_s \to V$ such that $(v, \tau(v))$ is in E for all $v \in V_s$. A play P is consistent with τ if each node v_i in P owned by player s satisfies $v_{i+1} = \tau(v_i)$. It is well known that each parity game is determined: node set V is the disjoint union of two sets W_0 and W_1, the winning regions of players 0 and 1 (respectively), where one of W_0 and W_1 may be empty. Moreover, strategies $\sigma\colon V_0 \to V$ and $\pi\colon V_1 \to V$ can be computed such that

- all plays beginning in W_0 and consistent with σ are won by player 0; and
- all plays beginning in W_1 and consistent with π are won by player 1.

Solving a parity game means computing such data (W_0, W_1, σ, π). We show a parity game and one of its possible solutions in Figure 1.

3 Rabin Index

We now formalize the definition of equivalence for coloring functions, and then use that notion in order to formally define the Rabin index of a parity game.

We want to reduce the complexity of a coloring function c in a parity game (V, V_0, V_1, E, c) by transforming c to some coloring function c'. Since we do not want the transformation to be based on a solution of the game we design the transformation to ignore ownership of nodes. That is, it needs to be sound for *every possible* ownership structure $V = V_0 \cup V_1$. Therefore, for *all* such partitions $V = V_0 \cup V_1$, the two parity games (V, V_0, V_1, E, c) and (V, V_0, V_1, E, c') that differ only in colors need to be equivalent in that they have the same winning regions and the same sets of winning strategies. We formalize this notion.

Definition 1 *Let (V, E) be a directed graph and $c, c'\colon V \to \mathbb{N}$ two coloring functions. We say that c and c' are equivalent, written $c \equiv c'$, iff for all partitions $V_0 \cup V_1$ of V the resulting parity games (V, V_0, V_1, E, c) and (V, V_0, V_1, E, c') have the same winning regions and the same sets of winning strategies for both players.*

Intuitively, changing coloring function c to c' with $c \equiv c'$ is sound: regardless of what the actual partition of V is, we know that this change will neither affect the winning regions nor the choice of their supporting winning strategies. But the definition of $\equiv$ is not immediately amenable to algorithmic simplification of c to some c'. This definition quantifies over exponentially many partitions, and for each such partition it insists that certain sets of strategies be equal.

We need a more compact characterization of $\equiv$ as the basis for designing a static analysis. To that end, we require some concepts from graph theory first.

Definition 2 1. *A path P in a directed graph (V, E) is a sequence $v_0, v_1, \ldots, v_n$ of nodes in V such that (v_i, v_{i+1}) is in E for every i in $\{0, 1, \ldots, n-1\}$.*

2. *A cycle C in a directed graph (V, E) is a path $v_0, \ldots, v_n$ with (v_n, v_0) in E.*

3. *A simple cycle C in a directed graph (V,E) is a cycle $v_0, v_1, \dots, v_n$ such that for every $i \neq j$ in $\{0,1,\dots n\}$ we have $v_i \neq v_j$.*
4. *For (V,E,c), the c-color of a cycle $v_0, \dots, v_n$ in (V,E) is $\min_{0 \leq i \leq n} c(v_i)$.*

Simple cycles are paths that loop so that no node has more than one outgoing edge on that path. A cycle is defined similarly, except that it is allowed that v_i equals v_j for some $i \neq j$, so a node on that path may have more than one outgoing edge. The color of a cycle is the minimal color that occurs on it.

For example, for the parity game in Figure 1, a simple cycle is v_0, v_4, v_3, v_2, v_1 and its color is 1, a cycle that is not simple is v_0, v_1, v_2, v_1 and its color is 2.

We can now characterize $\equiv$ in terms of colors of simple cycles. Crucially, we make use of the fact that parity games have pure, positional strategies [3].

Proposition 1 *Let (V,E) be a directed graph and $c, c' \colon V \to \mathbb{N}$ two coloring functions. Then $c \equiv c'$ iff for all simple cycles C in (V,E), the c-color of C has the same parity as the c'-color of C.*

Proof Sketch: We write $c \sim c'$ iff for all simple cycles C in (V,E), the c-color of C has the same parity as the c'-color of C. We have to show $\sim$ equals $\equiv$.

To prove that $\sim$ is contained in $\equiv$, let $c \sim c'$ be given. For each subset V_0 of V we have parity games $G_c = (V, V_0, V \setminus V_0, c)$ and $G_{c'} = (V, V_0, V \setminus V_0, c')$. We write W_s (resp. W'_s) for the winning region of player s in G_c (resp. $G_{c'}$).

Now let σ be a strategy for player 0 that is winning on W_0 in G_c. We use that plays that begin in W_0 and are consistent with σ and any strategy π of player 1 are decided by their periodic suffix – which forms a simple cycle as both strategies are memoryless. As $c \sim c'$, that decision is the same in both parity games. So W_o is contained in W'_0 and σ is winning on W_0 in game $G_{c'}$ as well.

A symmetric argument for the winning region W_1 and a π for player 1 that is winning on W_1 in $G_{c'}$ then proves the claim by the determinacy of parity games.

To show that $\equiv$ is contained in $\sim$, let $c \equiv c'$ be given. We construct, for each simple cycle C, a 1-player parity game (so one of V_0 and V_1 is empty) which is controlled by the player that matches the parity of the c-color of C. From $c \sim c'$ is then follows that the c'-color of C also has that parity. (A full proof is contained in the appendix.) □

Next, we define the relevant measure of descriptive complexity, which will also serve as a measure of precision for the static analyses we will develop.

Definition 3 1. *For colored arena (V,E,c), its index $\mu(c)$ is $\max_{v \in V} c(v)$.*
2. *The Rabin index $\mathsf{RI}(c)$ of colored arena (V,E,c) is $\min\{\mu(c') \mid c \equiv c'\}$.*
3. *The Rabin index of parity game (V, V_0, V_1, E, c) is $\mathsf{RI}(c)$ for (V,E,c).*

The index $\mu(c)$ reflects the maximal color occurring in c. So for a coloring function $c \colon V \to \mathbb{N}$ on (V,E), its Rabin index is the minimal possible maximal color in a coloring function that is equivalent to c. This definition applies to colored arenas and parity games alike.

As an aside, is $\mu(c)$ a good measure, given that $\mu(c+n) = n + \mu(c)$ for $c+n$ with $(c+n)(v) = c(v) + n$ when n is even? And given that c may have large color gaps? Fortunately, this is not a concern for the Rabin index of c. This is so as for all $c' \equiv c$ with $\mu(c') = \mathsf{RI}(c)$ we know that the minimal color of c' is at most 1 and that c' has no color gaps – due to the minimality of the Rabin index.

Intuitively, in order to prove that $\mathsf{RI}(c) < k$ for some $k > 0$ one has to produce a coloring c' and show that all simple cycles in the graph have the same color under c and c'. As we will see below, deciding for a given colored arena (V,E,c) whether $\mathsf{RI}(c)$ is at least k is NP-hard for fixed $k \geq 2$.

Next, we present an algorithm that computes a coloring function which witnesses the Rabin index of a given c.

```
rabin($V$,$E$,$c$) {
  rank = $\sum_{v\in V}$ $c(v)$;
  do {
    cache = rank;
    cycle(); pop();
    rank = $\sum_{v\in V}$ $c(v)$;
  } while (cache != rank)
  return $c$;
}

cycle() {
  sort V in ascending c-color ordering $v_1,v_2,\ldots,v_n$;
  for ($i$=1..$n$) {
    $j$ = getAnchor($v_i$);
    if ($j$ == $-1$) { $c(v_i)$ = $c(v_i)\%2$; }
    else { c($v_i$) = $j+1$; }
  }
}

getAnchor($v_i$) {
  for ($\gamma$ = $c(v_i)-1$  down to $(c(v_i)-1)\%2$;  step size 2) {
     if ($\exists$ simple cycle $C$ with color $\gamma$ through  $v_i$) { return $\gamma$; }
  }
  return $-1$;
}

pop() {
  $m$ = max{ $c(v)$ |  $v\in V$};
  while (not $\exists$ simple cycle $C$ with color $m$) {
    for ($v$ in { $w\in V$ | $c(w) = m$}) { $c(v) = m - 1$; }
    $m = m - 1$;
  }
}
```

Figure 2: Algorithm `rabin` which relies on methods `cycle`, `getAnchor`, and `pop`.

4 Computing the Rabin Index

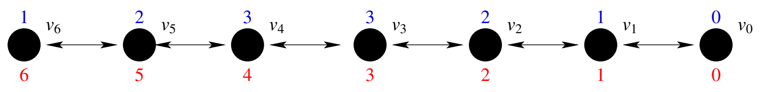
We now discuss our algorithm `rabin`, shown in Figure 2. It takes a coloring function as input and outputs an equivalent one whose index is the Rabin index of the input. Formally, `rabin` computes a coloring function c' with $c \equiv c'$ and where there is no $c \equiv c''$ with $\mu(c'') < \mu(c')$. Then, $\mathsf{RI}(c) = \mu(c')$ by definition.

Algorithm `rabin` uses a standard iteration pattern based on a rank function which sums up all colors of all nodes. In each iteration, two methods are called:

- `cycle` analyzes the cyclic structure of (V,E) and so reduces colors of nodes
- `pop` repeatedly lowers all occurrences of maximal colors by 1 until there is a simple cycle whose color is a maximal color.

These iterations proceed until neither `cycle` nor `pop` has an effect on the coloring function. Method `cycle` first sorts all nodes of (V,E,c) in ascending color values for c. It then processes each node v_i in that ascending order. For each node v_i it calls `getAnchor` to find (if possible) a maximal "anchor" for v_i.

If `getAnchor` returns -1, then v_i has no anchor as all simple cycles through v_i have color $c(v_i)$.



iteration	`cycle`	`pop`
1	*nil*	$c(v_6) = 5$
2	$c(v_6) = 1$	$c(v_5) = 4$
3	$c(v_5) = 2$	$c(v_4) = 3$

Figure 3: Colored arena (V,E,c) and table showing effects of iterations in `rabin`(V,E,c)

Therefore, it is sound to change $c(v_i)$ to its parity. Otherwise, `getAnchor` returns an index j to an "anchor" node that is maximal in that

- there is a simple cycle C through v_i whose color j is smaller and of different parity than that of v_i, and
- for all simple cycles C' through v_i, either they have a color that has the same parity as the color of v_i or they have a color that is less than or equal to j.

A node on this simple cycle C with color j is thus a maximal anchor for node v_i. Method `cycle` therefore resets $c(v_i)$ to $j+1$.

The idea behind `pop` is that one can safely lower maximal color m to $m-1$ if there is no simple cycle whose color is m. For then all occurrences of m are dominated by smaller colors on simple cycles.

We now prove the soundness of our algorithm `rabin`.

Lemma 1 *Let (V,E,c) be a given colored arena and let c' be the coloring function that is returned by the call* `rabin`(V,E,c)*. Then $c \equiv c'$ holds.*

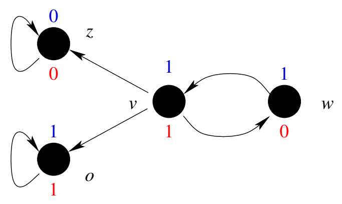
We show some example runs of `rabin`, starting with a detailed worked example, for the parity game in Figure 1. Let the initial sort of `cycle` be $v_3v_4v_2v_0v_1$. Then `cycle` changes no colors at v_3 (as the anchor of v_3 is -1), at v_4 (as the anchor of v_4 is 1 due to simple cycle v_4v_3), at v_2 (as the anchor of v_2 is 1 due to simple cycle $v_2v_1v_0v_4v_3$), but changes $c(v_0)$ to 1 (as the anchor of v_0 is -1). Also, $c(v_1)$ won't change (as the anchor of v_1 is 2 due to simple cycle v_1v_2).

Then `pop` changes $c(v_1)$ to 2 (as there is no simple cycle with color 3). Let the sort of the second call to `cycle` be $v_0v_3v_1v_2v_4$. Then the corresponding list of anchor values is $-1,-1,1,1,1$ and so `cycle` changes no colors. Therefore, the second call to `pop` changes no colors either. Thus the overall effect of `rabin` was to lower the index from 3 to 2 by lowering $c(v_1)$ to 2.

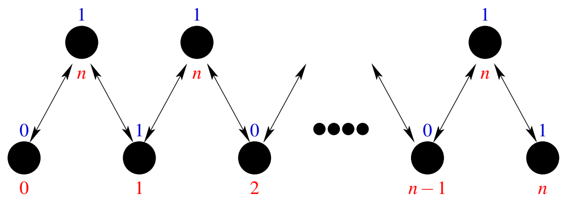
As a second example, in Figure 3, we see a colored arena with $c(v_i) = i$ (in red/bottom), the output `rabin`(V,E,c) (in blue/top), and a table showing how the coloring function changes through repeated calls to `cycle` and `pop`. Each iteration of `rabin` reduces the measure $\mu(c)$ by 1. This illustrates that the number of iterations of `rabin` is unbounded in general.

We note that $\equiv$ cannot be captured by just insisting that the winning regions of all abstracted parity games be the same. In Figure 4(a), we see a colored arena with two coloring functions c (in red/bottom) and c' (in blue/top). The player who owns node v will win all nodes as she chooses between z or o the node that has her parity. So c and c' are equivalent in that they always give rise to the same winning regions. But if v is owned by player 1, she has a winning strategy for c' (move from v to w) that is not winning for c.

In Figure 4(b), colored arena (V,E,c) has odd index n and Rabin index 2. Although there are cycles from all nodes with color n, e.g., to the node with color $n-1$, there are no *simple* such cycles. So all colors reduce to their parity.



(a) Coloring functions c and c' give rise to the same winning regions, but not the same winning strategies. Thus $c \not\equiv c'$



(b) Coloring function c has Rabin index 2, witnessed by c'

Figure 4: Two coloring functions c (in red/bottom) and c' (in blue/top) on the same game

Now we can prove that algorithm `rabin` is basically as precise as it could be. First, we state and prove an auxilliary lemma which provides sufficient conditions for a coloring function c to have its index $\mu(c)$ as its Rabin index $\mathsf{RI}(c)$. Then we show that the output of `rabin` meets these conditions.

Lemma 2 *Let (V,E,c) be a colored arena where*

1. there is a simple cycle in (V,E) whose color is the maximal one of c

2. for all v in V with $c(v) > 1$, node v is on a simple cycle C with color $c(v) - 1$.

Then there is no c' with $c \equiv c'$ and $\mu(c') < \mu(c)$. And so $\mu(c)$ equals $\mathsf{RI}(c)$.

Proof : Let k be the maximal color of c and consider an arbitrary c' with $c \equiv c'$.

Proof by contradiction: Let the maximal color k' of c' satisfy $k' < k$. By the first assumption, there is a simple cycle C_0 whose c-color is k. Since $k' < k$ and $c \equiv c'$, we know that the c'-color of C_0 can be at most $k-2$. Let v_0 be a node on C_0 such that $c'(v_0)$ is the c'-color of C_0. Then $c'(v_0) \leq k-2$. As all nodes on C_0 have c-color k, we have also $c(v_0) \geq k$. For $k < 2$, then $c'(v_0) \leq k-2$ gives us a contradiction $c'(v_0) < 0$. It thus remains to consider the case when $k \geq 2$.

By the second assumption, there is some simple cycle C_1 through v_0 such that the color of C_1 is $k-1$. In particular, there is some node v_0' in C_1 with color $k-1$. But $k-1$ cannot be the color of C_1 with respect to c' since v_0 is on C_1 and $c'(v_0) \leq k-2$. Since $c \equiv c'$, the c'-color of C_1 is therefore at most $k-3$. So there is some v_1 on C_1 such that $c'(v_1) \leq k-3 < k-1 \leq c(v_1)$.

If $c(v_1) > 1$, we repeat the above argument at node v_1 to construct a simple cycle C_2 through v_1 with color $c(v_1) - 1$. Again, there then have to be nodes v_1' and v_2 on C_2 such that the color $c'(v_1')$ is the c'-color of C_2, and such that $c'(v_2) \leq k-4 < k-2 \leq c(v_2)$ holds.

We can repeat the above argument to construct simple cycles $C_0, C_1, C_2, \dots$ and nodes $v_0, v_0', v_1, v_1', v_2, v_2', \dots$ such that $c'(v_j) \leq k-j-2 < k-j \leq c(v_j)$ until $k-j \leq c(v_j) \leq 1$. But then $c'(v_j) \leq k-j-2 \leq 1-2 = -1$, a contradiction. □

We now show that the output of `rabin` satisfies the assumptions of Lemma 2. Since `rabin` is sound for $\equiv$, we therefore infer that it computes a coloring function whose maximal color equals the Rabin index of its input coloring function.

Theorem 1 *Let (V,E,c) be a colored arena. And let c^* be the output of the call* `rabin`(V,E,c)*. Then $c \equiv c^*$ and $\mu(c^*)$ is the Rabin index of c.*

Proof : By Lemma 1, we have $c \equiv c^*$. Since $\equiv$ is clearly transitive, it suffices to show that there is no c' with $c^* \equiv c'$ and $\mu(c') < \mu(c^*)$. By Lemma 2, it therefore suffices to establish the two assumptions of that lemma for c^*. As c^* is returned by `rabin` neither `cycle` nor `pop` have an effect on it.

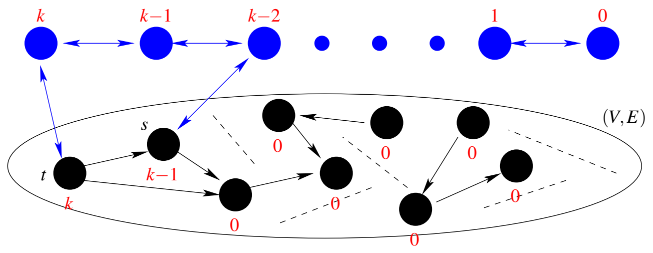


Figure 5: Construction for NP-hardness of deciding whether $\mathsf{RI}(c) \geq k$ for $k \geq 2$

The first assumption of Lemma 2 is therefore true since pop has no effect on c^* and so there must be a simple cycle in (V,E) whose color is the maximal one in c. This also applies to the case when c^* has only one color, as (V,E) has to contain cycles since it is finite and all nodes have outgoing edges.

As for the second assumption, let by way of contradiction there be some node v with $c^*(v) > 1$ and no simple cycle through v with color $c^*(v) - 1$. Then `cycle` would have an effect on $c^*(v)$ and would lower it, a contradiction. □

5 Complexity

We now discuss the complexity of algorithm `rabin` and of the decision problems associated with the Rabin index. We turn to the complexity of `rabin` first.

Let us assume that we have an oracle that checks for the existence of simple cycles. Then the computation of `rabin` is efficient modulo polynomially many calls (in the size of the game) to that oracle. Since deciding whether a simple cycle exists between two nodes in a directed graph is NP-complete (see e.g. [4, 5]), we infer that `rabin` can be implemented to run in exponential time.

Next, we study the complexity of deciding the value of the Rabin index. We can exploit the NP-hardness of simple cycle detection to show that the natural decision problem for the Rabin index, whether $\mathsf{RI}(c)$ is at least k, is NP-hard for fixed $k \geq 2$. In contrast, for $k = 1$, we show that this problem is in P.

Theorem 2 *Deciding whether the Rabin index of a colored arena (V,E,c) is at least k is NP-hard for every fixed $k \geq 2$, and is in P for $k = 1$.*

Proof : First consider the case when $k \geq 2$. We use the fact that deciding whether there is a simple cycle through nodes $s \neq t$ in a directed graph (V,E) is NP-complete (see e.g. [5]). Without loss of generality, for all v in V there is some w in V with (v,w) in E (we can add (v,v) to E otherwise). Our hardness reduction uses a colored arena (V',E',c), depicted in Figure 5, which we now describe:

We color s with $k-1$ and t with k, and color all remaining nodes of V with 0. Then we add $k+1$ many new nodes (shown in blue/top in the figure) to that graph that form a "spine" of descending colors from k down to 0, connected by simple cycles. Crucially, we also add a simple cycle between t and that new k node, and between s and the new $k-2$ node.

We claim that the Rabin index of (V',E',c) is at least k iff there is a simple cycle through s and t in the original directed graph (V,E).

1. Let there be a simple cycle through s and t in (V,E). Since there is a simple cycle between s and the new $k-2$ node, `cycle` does not change the color at s. As there is a simple cycle through s and t, method `cycle` also does not change the color at t. Clearly, no colors on the spine can be changed by `cycle`. Since there is a simple cycle between t and the new k node, method pop also does not change colors. But then the Rabin index of c is k and so at least k.

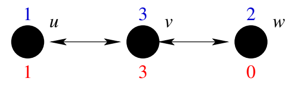


Figure 6: Coloring functions c (blue/top) and c' (red/bottom) with $c \equiv c'$ but $c \not\equiv^{\alpha} c'$

2. Conversely, assume that there is no simple cycle through s and t in the original graph (V,E). It follows that the anchor j of t has value 0 or, if k is even, has value -1. In this case, `cycle` changes the color at t to the parity of k. Then, `pop` reduces the color of the remaining node colored k to $k-1$. Thus, it cannot be the case that the Rabin index of c is at least k.

This therefore proves the claim. Second, consider the case when $k = 1$. Deciding whether $\mathsf{RI}(c)$ is at least 1 amounts to checking whether $c \equiv \vec{0}$ where $\vec{0}(v) = 0$ for all v in V. This is the case iff all simple cycles in (V,E,c) have even c-parity. But that is the case iff all cycles in (V,E,c) have even c-parity.

To see this, note that the "if" part is true as simple cycles are cycles. As for the "only if" part, this is true since if there were a cycle C with odd c-parity, then some node v on that cycle would have to have that minimal c-color, but v would then be on some simple cycle whose edges all belong to C.

Finally, checking whether all cycles in (V,E,c) have even c-parity is in P. □

The decision problem of whether $\mathsf{RI}(c) = 1$ cannot be in NP, unless NP equals coNP. Otherwise, the decision problem of whether $\mathsf{RI}(c) \leq 1$ would also be in NP, since we can decide in P whether $\mathsf{RI}(c) = 0$ and since NP is closed under unions. But then the complement decision problem of whether $\mathsf{RI}(c) \geq 2$ would be in coNP, and we have shown it to be NP-hard already. Therefore, all problems in NP would reduce to this problem and so be in coNP as well, a contradiction.

We now discuss an efficient version of `rabin` which replaces oracle calls for simple cycle detection with calls for over-approximating cycle detection.

6 Abstract Rabin index

We now discuss an efficient version of `rabin` which replaces oracle calls for simple cycle detection with over-approximating cycle detection. In fact, this static analysis computes an abstract Rabin index, whose definition is based on an abstract version of the equivalence relation $\equiv$. We define these notions formally.

Definition 4 *1. Let* `rabin`$^{\alpha}$ *be* `rabin` *where all existential quantifications over simple cycles are replaced with existential quantifications over cycles.*

2. Let (V,E) be a directed graph and $c, c' \colon V \to \mathbb{N}$ two coloring functions. Then:

(a) $c \equiv^{\alpha} c'$ iff for all cycles C, the parities of their c- and c'-colors are equal.

(b) The abstract Rabin index $\mathsf{RI}^{\alpha}(c)$ of (V,E,c) is $\min\{\mu(c') \mid c \equiv^{\alpha} c'\}$.

Thus `rabin`$^{\alpha}$ uses the set of cycles in (V,E) to overapproximate the set of simple cycles in (V,E). In particular, $c \equiv^{\alpha} c'$ implies $c \equiv c'$ but not the other way around, as can be seen in the example in Figure 6.

In that example, we have $c \equiv c'$ since all simple cycles have the same parity of color with respect to c and c'. But there is a cycle that reaches all three nodes and which has odd color for c and even color for c'. Thus, $c \not\equiv^{\alpha} c'$ follows.

We now show that the overapproximation `rabin`$^{\alpha}$ of `rabin` is sound in that its output coloring function is equivalent to its input coloring function. Below, in Theorem 3, we further show that this output yields an abstract Rabin index.

Lemma 3 *Let (V,E,c) be a colored arena and let* `rabin`$^{\alpha}(V,E,c)$ *return c'. Then $c \equiv^{\alpha} c'$ and $\mu(c') \geq \mathsf{RI}(c)$.*

To prove this lemma, it suffices to show $c \equiv^{\alpha} c'$, as $c \equiv c'$ follows from that, and then this in turn implies $\mu(c') \geq \mathsf{RI}(c)$ by the definition of the Rabin index.

Note that the definition of $\equiv^{\alpha}$ is like the characterization of $\equiv$ in Proposition 1, except that the universal quantification over simple cycles is being replaced by a universal quantification over cycles for $\equiv^{\alpha}$. In proving Lemma 3, we can thus reuse the proof for Lemma 1 where we replace $\equiv$ with $\equiv^{\alpha}$, `rabin` with $\texttt{rabin}^{\alpha}$, and "simple cycle" with "cycle" throughout in that proof.

We can now adapt the results for `rabin` to this abstract setting.

Lemma 4 *Let (V,E,c) be a colored arena where*

1. *there is a cycle in (V,E) whose color is the maximal one of c*
2. *for all v in V with $c(v) > 1$, node v is on a cycle C with color $c(v) - 1$.*

Then there is no c' with $c \equiv^{\alpha} c'$ and $\mu(c') < \mu(c)$, and so $\mu(c) = \mathsf{RI}^{\alpha}(c)$.

Similary to the case for algorithm `rabin`, we now show that the output of $\texttt{rabin}^{\alpha}$ satisfies the assumptions of Lemma 4. Since algorithm $\texttt{rabin}^{\alpha}$ is sound for $\equiv^{\alpha}$, we therefore infer that it computes coloring functions whose maximal color equals the abstract Rabin index of their input coloring function.

Theorem 3 *Let (V,E,c) be a colored arena. And let c^* be the output of the call $\texttt{rabin}^{\alpha}(V,E,c)$. Then $c \equiv^{\alpha} c^*$ and $\mu(c^*)$ is the abstract Rabin index $\mathsf{RI}^{\alpha}(c)$.*

We now study the sets of parity games whose abstract Rabin index is below a fixed bound. We define these sets formally.

Definition 5 *Let $\mathscr{P}_k^{\alpha}$ be the set of parity games (V,V_0,V_1,E,c) with $\mathsf{RI}^{\alpha}(c) < k$.*

We can now show that parity games in these sets are efficiently solvable, also in the sense that membership in such a set is efficiently decidable.

Theorem 4 *Let $k \geq 1$ be fixed. All parity games in $\mathscr{P}_k^{\alpha}$ can be solved in polynomial time. Moreover, membership in $\mathscr{P}_k^{\alpha}$ can be decided in polynomial time.*

Proof : For each parity game (V,V_0,V_1,E,c) in $\mathscr{P}_k^{\alpha}$, we first run $\texttt{rabin}^{\alpha}$ on it, which runs in polynomial time. By definition of $\mathscr{P}_k^{\alpha}$, the output coloring function c^* has index $< k$. Then we solve the parity game (V,V_0,V_1,E,c^*), which we can do in polynomial time as the index is bounded by k. But that solution is also one for (V,V_0,V_1,E,c) since $c \equiv^{\alpha} c^*$ by Lemma 3, and so $c \equiv c^*$ as well.

That the membership test is polynomial in the running time can be seen as follows: for coloring function c, compute $c' = \texttt{rabin}^{\alpha}(V,E,c)$ and return `true` if $\mu(c') < k$ and return `false` otherwise; this is correct by Theorem 3. □

We note that algorithm $\texttt{rabin}^{\alpha}$ is precise for colored arenas $A = (V,E,c)$ with Rabin index 0. These are colored arenas that have only simple cycles with even color. Since a colored arena has a cycle with odd color iff it has a simple cycle with odd color, $\texttt{rabin}^{\alpha}$ correctly reduces all colors to 0 for such arenas.

For Rabin index 1, the situation is more subtle. We cannot expect $\texttt{rabin}^{\alpha}$ to always be precise, as the decision problem for $\mathsf{RI}(c) \geq 2$ is NP-hard. Algorithm $\texttt{rabin}^{\alpha}$ will correctly compute Rabin index 1 for all those arenas that do not have a simple cycle with even color. But for c from Figure 6, e.g., algorithm $\texttt{rabin}^{\alpha}$ does not change c with index 3, although the Rabin index of c is 1.

Game Type	$\mu(c)$	$\mu(s(c))$	$\mathsf{RI}^\alpha(c)$	S	R	#I	Sol	Sol.S	Sol.R
`Clique[100]`	100	100	99	0.08	388.93	2	13.23	13.06	13.01
`Ladder[100]`	2	2	2	0.11	8.93	1	1.87	1.66	1.68
`Jurdziński[5 10]`	12	12	11	0.09	44.25	2	76.98	76.94	76.38
`Recursive Ladder[15]`	48	46	16	0.04	10.46	2	310.21	309.21	174.91
`Strategy Impr[8]`	237	181	9	0.10	54.01	2	194.96	45.46	8.99
`Model Checker Ladder[100]`	200	200	0	0.14	141.95	2	30.90	30.49	0.62
`Tower of Hanoi[5]`	2	2	1	0.46	261.10	2	29.43	29.61	45.41

Figure 7: Indices and average times (in *ms*) for 100 runs for game types named in first column. Next three columns: original, statically compressed, and `rabin`$^\alpha$-compressed index. Next three columns: times of static and `rabin`$^\alpha$-compression, and the number of iterations within `rabin`$^\alpha$. Last three columns: Times of solving the original, statically compressed, and `rabin`$^\alpha$-compressed games with Zielonka's solver

7 Experimental results

We now provide some experimental results. Our objective is to compare the effectiveness of color compression of `rabin`$^\alpha$ to a known color compression algorithm (called static compression), to observe the performance improvement in solving compressed games using Zielonka's parity game solver [11], and to get a feel for how much the abstract Rabin index reduces the index of random and non-random games.

Our implementation is written in Scala and realizes all game elements as objects to simplify implementation. Our main interest is in descriptive complexity measures and relative computation time.

We programed algorithm `rabin` with simple cycle detection reduced to incremental SAT solving. This did not scale to graphs with more than 40 nodes. But for those games for which we could compute the Rabin index, `rabin`$^\alpha(V,E,c)$ often computed the Rabin index $\mathsf{RI}(c)$ or did get very close to it.

Our implementation of `rabin`$^\alpha$ reduced cycle detection to the decomposition of the graph into strongly connected components, using Tarjan's algorithm (which is linear in the number of edges). The rank function is only needed for complexity and termination analysis, we replaced it with Booleans that flag whether `cycle` or `pop` had an effect.

The standard static compression algorithm simply removes gaps between colors, e.g. a set of colors $\{0,3,4,5,6,8\}$ is being compressed to $\{0,1,2,3,4\}$. Below, we write $s(c)$ for the statically compressed version of coloring function c.

The experiments are conducted on non-random and random games separately. Each run of the experiments generates a parity game $G = (V,V_0,V_1,E,c)$ of a selected configuration. Static compression and `rabin`$^\alpha$ are performed on these games. We report the time taken to execute static compression and `rabin`$^\alpha$, as well as the number of iterations that `rabin`$^\alpha$ runs until `cycle` and `pop` have no effect, i.e. the number of iterations needed for $\mu(c)$ to reach $\mathsf{RI}^\alpha(c)$. Finally, we record the wall-clock time required to solve original, statically compressed, and `rabin`$^\alpha$-compressed games, using Zielonka's solver [11].

We use PGSolver to generate non-random games, detailed descriptions on these games can be found in [7]. Each row in Figure 7 shows the average statistics from 100 runs of the experiments on corresponding non-random game. We see that `rabin`$^\alpha$ has significantly reduced the indices of `Recursive Ladder`, `Strategy Impr`, and `Model Checker Ladder`, where $\mathsf{RI}^\alpha(c)$ is 0% to 35% of the index $\mu(s(c))$ of the statically compressed coloring function.

Applying `rabin`$^\alpha$ improves performance of solvers. For all three game types, we observe 44% to 98% in solver time reduction between solving statically compressed and `rabin`$^\alpha$-compressed games.

The time required to perform static compression is low compared to the time needed for `rabin`$^\alpha$-compression, but `rabin`$^\alpha$-compression followed by solving the game is still faster than solving the original game for `Recursive Ladder`.

Game Configs	$\mu(c)$	$\mu(s(c))$	$\mathsf{RI}^{\alpha}(c)$	S	R	#I	Sol	Sol.S	Sol.R
100/1/20/100	99.16	45.34	35.97	0.48	57.04	2.05	6.71	5.21	4.84
200/1/40/200	198.97	91.91	80.29	0.12	441.29	2.03	12.40	11.49	11.43
400/1/80/400	399.28	184.34	172.30	0.24	4337.04	2.10	42.78	40.62	40.58
800/1/160/800	799.08	369.76	355.67	0.47	47241.70	2.05	181.73	173.59	173.83
1000/1/200/1000	999.14	462.48	447.37	0.59	106332.96	2.05	296.53	281.60	281.70

Figure 8: Indices and average times (in *ms*) for 100 runs of random games of various configurations listed in the first column. Next three columns: average original, statically compressed, and `rabin`$^{\alpha}$-compressed indices. The remaining columns are as in Figure 7

Games `Ladder` and `Tower of Hanoi` have very low indices and their colors cannot be compressed further. Method `cycle` has no effect on `Clique` games, but `pop` manages to reduce its index by 1.

We now discuss our experimental results on random games. The notation used to describe randomly generated parity games is $xx/yy/zz/cc$, where xx is the number of nodes (node ownership is determined by a fair coin flip for each node independently), with between yy to zz out-going edges for each node, and with colors at nodes chosen at random from $\{0,\dots,cc\}$. Also, the games used in the experiments have 1 as the minimum number of out-going edges. This means that the nodes have no dead-ends. We also disallow self-loops (no (v,v) in E).

Figure 8 shows the average statistics of 100 runs of experiments on five selected game configurations. (Our experiments on larger games are consistent with the data reported here, and so not reported here.) The results indicate that static compression is effective in reducing the colors for randomly generated games, it achieves around 54% index reduction for all game types. The `rabin`$^{\alpha}$-compression achieves further 3% to 21% reduction. Due to the relatively small index reduction by `rabin`$^{\alpha}$, we do not see much improvement in solving `rabin`$^{\alpha}$-compressed games over solving statically-compressed ones. In addition, `rabin`$^{\alpha}$ reduces $\mu(c)$ to $\mathsf{RI}^{\alpha}(c)$ in one iteration for all of the randomly generated games G.

The results in Figure 8 show that these games take an average of more than 2 `rabin`$^{\alpha}$ iterations. This indicates that certain game structure, such as the one found in the game in Figure 3, is present in our randomly generated games

The experimental results show that `rabin`$^{\alpha}$ is able to reduce the indices of parity games significantly and quickly, for certain structure such as `Recursive Ladder`. Hence it effectively improves the overall solver performance for those games.

However, algorithm `rabin`$^{\alpha}$ has a negative effect on the overall performance for other non-random games and experimented random games, when we consider `rabin`$^{\alpha}$-compression time plus solver time.

8 Related work

Carton and Maceiras develop an algorithm (denoted here `Rabin`$_a$) that computes and minimizes the Rabin index of deterministic parity word automata [2]. Deterministic parity word automata can be thought of as 1-player parity games, where the player chooses input letters. An infinite word can be compared to a strategy with memory for the player. The word is accepted if the strategy is winning, that is, if the minimal color to be visited infinitely often is even. Minimization of the Rabin index should preserve the language of the automaton or, put in our terms, every winning strategy should remain to be winning.

The pseudocode of `Rabin`$_a$ is shown in Figure 9. Algorithm `Rabin`$_a$ constructs the "coloring dependencies" of all states in an automaton arena by decomposing the automaton into maximal strongly connected components (*SCCs*). For each R being a maximal *SCC*, it removes the states with the maximal color (and pushes them onto a stack), then recursively *SCC* decomposes the remaining arena of R.

```
Rabin$_a$($V$,$E$,$c$) {
  define a new colouring function $c'$ for $(V,E,c)$;
  reduce($V$,$E$,$c$,$c'$);
  return $c'$;
}
reduce($V$,$E$,$c$,$c'$) {
  i = 0; decompose $(V,E)$ into maximal $SCCs$;
  for ($R$ $\in$ $SCCs$){
    if ($\pi(R)$ == 0) m = 0;
    else {
      $R'$ = $\{v \in R \mid c(v) \neq \pi(R)\}$; m = reduce($R'$,$E|_{R'}$,$c|_{R'}$,$c'|_{R'}$);
      if ($\pi(R)$ - m is odd) m = m + 1;
    }
    for ($v$ $\in$ $\{v \in R \mid c(v) = \pi(R)\}$)
      $c'(v)$ = m;
    i = max{i, m};
  }
  return i;
}
```

Figure 9: Algorithm to compute Rabin index [2] for a parity automaton $A = (V, E, c)$, where $R \subseteq V$, $\pi(R) = \max\{c(v) \mid v \in R\}$, $E|_R$ is E with restriction to nodes in R, and similarly for $c|_R$.

Eventually, the input arena is reduced to a set of states that exist in their own respective *SCCs* (hence do not exist in the same cycle as each other). These states are assigned the minimal colors m (which is 0 or 1 depending on their original parities). The algorithm then propagates the new colour m to the states in the "layer" above. Those states receive a new colour m or $m + 1$, depending on whether their original parities equal the parities of the states in the "layer" below. In essence, *SCC* decomposition is used to detect the cycle dependency of states and this techniques is also used in our implementation of $\texttt{rabin}^{\alpha}$.

Our notion of Rabin index is a natural generalization to 2-player games. We require that for every pair of strategies (σ, π), their outcome should not change. As mentioned, the weaker notion requiring to preserve winning strategies of each player separately is not interesting. Such a Rabin index associates rank 0 with the winning region of player 0 and 1 with the winning region of player 1. It can be computed by solving the game.

The transition from 1-player setting to 2-player setting requires a more elaborate algorithm for computing the Rabin index. Although presented differently, algorithm $\texttt{Rabin}_a$ has the same effect of `cycle` in $\texttt{rabin}^{\alpha}$, which approximates the Rabin index. In our context of 2-player games one has to replace SCC decomposition (or cycle detection) by simple-cycle detection. Furthermore, in order to compute the Rabin index of a 2-player game we have to add the procedure `pop`. These two additional components are crucial for the computation of the Rabin index of games (as shown in this paper).

The differences become crucially important in terms of the computational complexity and degree of possible color compression in the setting of parity games. Using the colored arena in Figure 3 as an example, $\texttt{Rabin}_a$ will make no change to the red coloring function, whereas $\texttt{rabin}^{\alpha}$ reduces its index to 5 (using `pop`), and `rabin` reduces it even to 3.

9 Conclusions

We have provided a descriptive measure of complexity for parity games that (essentially) measures the number of colors needed in a parity game if we forget the ownership structure of the game but if we do not compromise the winning regions or winning strategies by changing its colors.

We called this measure the Rabin index of a parity game. We then studied this concept in depth. By analyzing the structure of simple cycles in parity games, we arrived at an algorithm that computes this Rabin index in exponential time.

Then we studied the complexity of the decision problem of whether the Rabin index of a parity game is at least k for some fixed $k > 0$. For k equal to 1, we saw that this problem is in P, but we showed NP-hardness of this decision problem for all other values of k. These lower bounds therefore also apply to games that capture these decision problems in game-theoretic terms.

Next, we asked what happens if our algorithm `rabin` abstractly interprets all detection checks for simple cycles through detection checks for cycles. The resulting algorithm $\texttt{rabin}^{\alpha}$ was then shown to run in polynomial time, and to compute an abstract and sound approximation of the Rabin index.

Our experiments were performed on random and non-random games. We observed that $\texttt{rabin}^{\alpha}$-compression plus Zielonka's solver [11] in some cases speed up solving time. The combination achieved 29% and 85% time reduction for `Jurdziński` and `Recursive Ladder` games, respectively, over solving the original games. But for other game types and random games, no such reduction was observed. We also saw that for some structured game types, the abstract Rabin index is dramatically smaller than the index of the game.

In future work we mean to investigate properties of the measure $\mathsf{RI}^{\alpha}(c) - \mathsf{RI}(c)$. Intuitively, it measures the difference of the Rabin index based on the structure of cycles with that based on the structure of simple cycles. From Figure 4(b) we already know that this measure can be arbitrarily large.

It will also be of interest to study variants of $\mathsf{RI}(c)$ that are targeted for specific solvers. For example, the SPM solver in [8] favors fewer occurrences of odd colors but also favors lower index. This suggests a measure with a lexicographical order of the Rabin index followed by an occurrence count of odd colors.

References

[1] Dietmar Berwanger, Anuj Dawar, Paul Hunter & Stephan Kreutzer (2006): *DAG-Width and Parity Games*. In: *STACS 2006, Proceedings of the 23rd Symposium on Theoretical Aspects of Computer Science*, *LNCS* 3884, Springer-Verlag, pp. 524–436, doi:10.1007/11672142_43.

[2] Olivier Carton & Ramón Maceiras (1999): *Computing the Rabin Index of a Parity Automaton*. *ITA* 33(6), pp. 495–506.

[3] E.A. Emerson & C. Jutla (1991): *Tree Automata, μ-Calculus and Determinacy*. In: *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pp. 368–377, doi:10.1109/SFCS.1991.185392.

[4] Shimon Even, Alon Itai & Adi Shamir (1976): *On the Complexity of Timetable and Multicommodity Flow Problems*. *SIAM J. Comput.* 5(4), pp. 691–703, doi:10.1109/SFCS.1975.21.

[5] Steven Fortune, John Hofcroft & James Wyllie (1980): *The Directed Subgraph Homeomorphism Problem*. *Theoretical Computer Science* 10, pp. 111–121, doi:10.1016/0304-3975(80)90009-2.

[6] Oliver Friedmann & Martin Lange (2009): *Solving Parity Games in Practice*. In Zhiming Liu & Anders Ravn, editors: *Proc. of Automated Technology for Verification and Analysis*, *Lecture Notes in Computer Science* 5799, Springer, pp. 182–196, doi:10.1007/978-3-642-04761-9_15.

[7] Oliver Friedmann & Martin Lange (2010): *The* PGSolver *Collection of Parity Game Solvers*. Technical Report, Institut für Informatik, LMU Munich. Version 3.

[8] Marcin Jurdziński (2000): *Small Progress Measures for Solving Parity Games*. In: *STACS '00: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, Springer-Verlag, London, UK, pp. 290–301, doi:10.1007/3-540-46541-3_24.

[9] J. Vöge & M. Jurdziński (2000): *A Discrete Strategy Improvement Algorithm for Solving Parity Games*. In: *Proc 12th Int. Conf. on Computer Aided Verification*, *Lecture Notes in Computer Science* 1855, Springer, pp. 202–215, doi:10.1007/10722167_18.

[10] K. Wagner (1979): *On ω-Regular Sets*. *Information and Control* 43, pp. 123–177, doi:10.1016/S0019-9958(79)90653-3.

[11] Wieslaw Zielonka (1998): *Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees*. *Theoretical Computer Science* 200(12), pp. 135 – 183, doi:10.1016/S0304-3975(98)00009-7.

A Faster Tableau for CTL*

Mark Reynolds
School of Computer Science and Software Engineering, The University of Western Australia
`mark.reynolds@uwa.edu.au`

There have been several recent suggestions for tableau systems for deciding satisfiability in the practically important branching time temporal logic known as CTL*. In this paper we present a streamlined and more traditional tableau approach built upon the author's earlier theoretical work.

Soundness and completeness results are proved. A prototype implementation demonstrates the significantly improved performance of the new approach on a range of test formulas. We also see that it compares favourably to state of the art, game and automata based decision procedures.

1 Introduction

CTL* [5, 3] is an expressive branching-time temporal logic extending the standard linear PLTL [13]. The main uses of CTL* are for developing and checking the correctness of complex reactive systems [6] and as a basis for languages (like ATL*) for reasoning about multi-agent systems [8].

Validity of formulas of CTL* is known to be decidable with an automata-based decision procedure of deterministic double exponential time complexity [5, 4, 18]. There is also an axiomatization [14]. Long term interest in developing a tableau approach as well has been because they are often more suitable for automated reasoning, can quickly build models of satisfiable formulas and are more human-readable. Tableau-style elements have indeed appeared earlier in some model-checking tools for CTL* but tableau-based satisfiability decision procedures have only just started to be developed [17, 7].

Our CTL* tableau is of the tree, or top-down, form. To decide the validity of ϕ, we build a tree labelled with finite sets of sets of formulas using ideas called hues and colours originally from [14] and further developed in [16, 17]. The formulas in the labels come from a closure set containing only subformulas of the formula being decided, and their negations. Those earlier works proposed a tableau in the form of a roughly tree-shaped Hintikka-structure, that is, it utilised labels on nodes which were built from maximally consistent subsets of the closure set. Each formula or its negation had to be in each hue. In this paper we make the whole system much more efficient by showing how we only need to consider subformulas which are relevant to the decision.

In the older papers we identified two sorts of looping: good looping allowed up-links in our tableau tree while bad looping showed that a branch was just getting longer and longer in an indefinite way. In this paper we tackle only the good looping aspect and leave bad looping for a follow-on paper.

A publicly available prototype implementation of the approach here is available and comparisons with existing state of the art systems, and its Hintikka-style predecessor, show that we are achieving orders of magnitude speed-ups across a range of examples. As with any other pure tableau system, though, this one is better at deciding satisfiable formulas rather than unsatisfiable ones.

In section 2 we give a formal definition of CTL* before section 3 defines some basic building block concepts. Subsequent sections introduce the tableau shape, contain an example, look at a loop checking rule and show soundness. Section 7 presents the tableau construction rules and then we show completeness. Complexity, implementation and comparison issues are discussed briefly in section 10 before a conclusion. There is a longer version of this paper available as [15].

2 Syntax and Sematics

Fix a countable set $\mathscr{L}$ of atomic propositions. A (transition) structure is a triple $M = (S, R, g)$ where:

S	is the non-empty set of *states*
R	is a total binary relation $\subseteq S \times S$ i.e. for every $s \in S$, there is some $t \in S$ such that $(s,t) \in R$.
g	$: S \to \mathscr{P}(\mathscr{L})$ is a labelling of the states with sets of atoms.

Formulas are defined along ω-long sequences of states. A *fullpath* in (S, R) is an infinite sequence $\langle s_0, s_1, s_2, ...\rangle$ of states such that for each i, $(s_i, s_{i+1}) \in R$. For the fullpath $\sigma = \langle s_0, s_1, s_2, ...\rangle$, and any $i \geq 0$, we write σ_i for the state s_i and $\sigma_{\geq i}$ for the fullpath $\langle s_i, s_{i+1}, s_{i+2}, ...\rangle$.

The formulas of CTL* are built from the atomic propositions in $\mathscr{L}$ recursively using classical connectives $\neg$ and $\wedge$ as well as the temporal connectives X, U and A. We use the standard abbreviations, **true**, **false**, $\vee$, $\to$, $\leftrightarrow$, $F\alpha \equiv \mathbf{true}\, U\alpha$, $G\alpha \equiv \neg F \neg \alpha$, and $E\alpha \equiv \neg A \neg \alpha$.

Truth of formulas is evaluated at fullpaths in structures. We write $M, \sigma \models \alpha$ iff the formula α is true of the fullpath σ in the structure $M = (S, R, g)$. This is defined recursively by:

$M, \sigma \models p$	iff	$p \in g(\sigma_0)$, any $p \in \mathscr{L}$
$M, \sigma \models \neg\alpha$	iff	$M, \sigma \not\models \alpha$
$M, \sigma \models \alpha \wedge \beta$	iff	$M, \sigma \models \alpha$ and $M, \sigma \models \beta$
$M, \sigma \models X\alpha$	iff	$M, \sigma_{\geq 1} \models \alpha$
$M, \sigma \models \alpha U \beta$	iff	there is $i \geq 0$ such that $M, \sigma_{\geq i} \models \beta$ and for each j, if $0 \leq j < i$ then $M, \sigma_{\geq j} \models \alpha$
$M, \sigma \models A\alpha$	iff	for all fullpaths σ' such that $\sigma_0 = \sigma'_0$ we have $M, \sigma' \models \alpha$

We say that α is *valid* in CTL*, iff for all transition structures M, for all fullpaths σ in M, we have $M, \sigma \models \alpha$. Say α is *satisfiable* in CTL* iff for some transition structure M and for some fullpath σ in M, we have $M, \sigma \models \alpha$. Clearly α is satisfiable iff $\neg\alpha$ is not valid.

3 Hues, Colours and Hintikka Structures

Fix the formula ϕ whose satisfiability we are interested in. We write $\psi \leq \phi$ if ψ is a subformula of ϕ. The length of ϕ is $|\phi|$. The *closure set* for ϕ is $\mathbf{cl}\,\phi = \{\psi, \neg\psi \mid \psi \leq \phi\}$.

Definition. [MPC] Say that $a \subseteq \mathbf{cl}\,\phi$ is *maximally propositionally consistent (MPC)* for ϕ iff for all $\alpha, \beta \in \mathbf{cl}\,\phi$, M1) if $\beta = \neg\alpha$ then ($\beta \in a$ iff $\alpha \notin a$); and M2) if $\alpha \wedge \beta \in \mathbf{cl}\,\phi$ then ($\alpha \wedge \beta \in a$ iff both $\alpha \in a$ and $\beta \in a$).

The concepts of hues and colours were originally invented in [14] but we use particular formal definitions as presented in [16, 17, 15]. A hue is supposed to capture (approximately) a set of formulas which could all hold together of one fullpath. Definition. [hue] $a \subseteq \mathbf{cl}\,\phi$ is a *hue* for ϕ, or ϕ-hue, iff all these conditions hold:

H1) a is MPC;
H2) if $\alpha U \beta \in a$ and $\beta \notin a$ then $\alpha \in a$;
H3) if $\alpha U \beta \in (\mathbf{cl}\,\phi) \setminus a$ then $\beta \notin a$;
H4) if $A\alpha \in a$ then $\alpha \in a$.

Further, let H_ϕ be the set of hues of ϕ.

For example, if $\neg(AG(p \to EXp) \to (p \to EGp))$, the example known as $\neg\theta_{12}$ in [17], then here is a hue known as $h38$:

$$\begin{aligned}&\{\neg(AG(p \to EXp) \to (p \to EGp)), (AG(p \to EXp) \wedge \neg(p \to EGp)),\\ &AG(p \to EXp), G(p \to EXp), \mathbf{true}, \neg\neg(p \to EXp),\\ &(p \to EXp), p, \neg\neg EXp, EXp, \neg\neg Xp, Xp,\\ &\neg(p \to EGp), (p \wedge \neg EGp), \neg EGp, A\neg Gp, \neg Gp, F\neg p, \neg\neg p\}\end{aligned}$$

The usual temporal successor relation plays a role in determining allowed steps in the tableau. The relation r_X is put between hues a and b if a fullpath σ satisfying a could have a one-step suffix $\sigma_{\geq 1}$ satisfying b: Definition. [r_X] For hues a and b, put $a\ r_X\ b$ iff the following four conditions all hold:

R1) if $X\alpha \in a$ then $\alpha \in b$;
R2) if $\neg X\alpha \in a$ then $\neg\alpha \in b$;
R3) if $\alpha\, U\beta \in a$ and $\neg\beta \in a$ then $\alpha\, U\beta \in b$; and
R4) if $\neg(\alpha\, U\beta) \in a$ and $\alpha \in a$ then $\neg(\alpha\, U\beta) \in b$.

We also introduced an equivalence relation aiming to tell whether two hues could correspond to fullpaths starting at the same state. We just need the hues to agree on atoms and on universal path quantified formulas: Definition. [r_A] For hues a and b, put $a\ r_A\ b$ iff the following two conditions both hold: A1) for all $p \in \mathscr{L}$, $p \in a$ iff $p \in b$; and A2) $A\alpha \in a$ iff $A\alpha \in b$.

Now we move up from the level of hues to the level of colours. Could a set of hues be exactly the hues corresponding to all the fullpaths starting at a particular state? We would need each pair of hues to satisfy r_A but we would also need hues to be in the set to witness all the existential path quantifications:

Definition. [colour] Non-empty $c \subseteq H_\phi$ is a *colour* of ϕ, or ϕ-colour, iff the following two conditions hold. For all $a, b \in c$, C1) $a\ r_A\ b$; and C2) if $a \in c$ and $\neg A\alpha \in a$ then there is $b \in c$ such that $\neg\alpha \in b$. Let C_ϕ be the set of colours of ϕ.

The formulas $\neg Xp, EXp$ are both in $h37$, another hue from the example in [17], so $\{h37\}$ is not a colour. However, $Xp \in h38$ witnesses the existential path quantification so $\{h37, h38\}$ is a colour.

We define a successor relation R_X between colours. It is defined in terms of the successor relation r_X between the component hues and it will be used to define the successor relation between tableau nodes, themselves corresponding to states in transition structures, in terms of the colours which they exhibit. Note that colours, and tableau nodes, will, in general, have a non-singleton range of successors and this relation R_X just tells us whether one node can be one of the successors of another node.

Definition. [R_X] For all $c, d \in C_\phi$, put $c\ R_X\ d$ iff for all $b \in d$ there is $a \in c$ such that $a\ r_X\ b$.

It is worth noting that colours and hues are induced by actual transition structures. We will need these concepts in our completeness proof.

Definition. [actual ϕ-hue] Suppose (S, R, g) is a transition structure. If σ is a fullpath through (S, R) then we say that $h = \{\alpha \in \mathbf{cl}\,\phi \mid (S, R, g), \sigma \models \alpha\}$ is the *actual (ϕ-) hue* of σ in (S, R, g).

It is straightforward to see that this is a ϕ-hue. It is also easy to show that along any fullpath σ, the relation r_X holds between the actual hue of σ and the actual hue of its successor fullpath $\sigma_{\geq 1}$.

Definition. [actual ϕ-colour] If $s \in S$ then the set of all actual hues of all fullpaths through (S, R) starting at s is called the *actual (ϕ-) colour* of s in (S, R, g).

Again, it is straightforward to show that this is indeed a ϕ-colour and also that R_X holds between the actual colour of any state and the actual colour of any of its successors.

4 Tableau

The tableaux we construct will be roughly tree-shaped: the traditional upside down tree with a root at the top, predecessors and ancestors above, successors and descendants below. However, we will allow

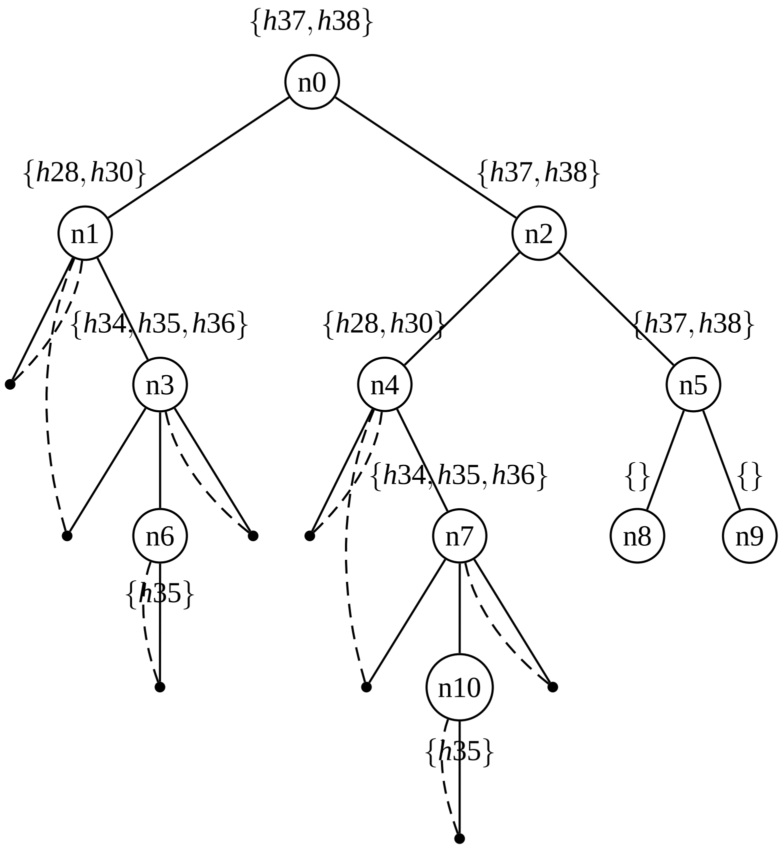


Figure 1: A Partial Tableau for $\neg\theta_{12}$

Definition. A *tableau* for $\phi \in L$ is a tuple (T, s, η, π) such that:

H1) T is a non-empty set of *nodes*; one distinguished element called the *root*;

H2) η is the phue label enumerator, so that for each $t \in T$, $\eta_t : \mathbb{N} \to 2^{\mathbf{cl}\,\phi}$ is a partial map,

H2.1) the domain of η_t is $\{0, 1, ..., n-1\}$ for some $n > 0$ denoted $|\eta_t|$;

H2.2) $\eta_t(i)$ is the *i*th label phue of t (if defined);

H3) s is the successor enumerator, so that for each $t \in T$, $s_t : \mathbb{N} \to T$ is a partial map,

H3.1) the domain of s_t is a subset of $\{0, 1, ..., |\eta_t| - 1\}$; $s_t(i)$ the *i*th successor of t;

H3.3) for each $t \in T$, there is a unique finite sequence $r_0, r_1, ..., r_k$ from T called the *ancestors* of t such that the r_i are all distinct, r_0 is the root, $r_k = t$ and for each j, r_{j+1} is a successor of r_j;

H4) $\phi \in \eta_{\mathbf{root}}(0)$;

H5) π is the predecessor map whereby if $t, u \in T$ then either π^t_u is undefined and we say that t is not a predecessor of u; or for all $j < |u|$, $\pi^t_u(j) = i < |t|$ and we say that the *i*th phue in t is a predecessor of the j th hue in u.

H6) if $s_t(i) = u$ then $\pi^t_u(0) = i$ (i.e. the *i*th phue in t is a predecessor of the 0th phue in $s_t(i)$);

Figure 2: Definition of Tableau

up-links from a node to one of its ancestors. Each node will be labelled with a finite sequence of sets of formulas from the closure set. We will call such a sequence of sets a *proto-colour* or *pcolour*. The sets, or *proto-hues (phues)*, in the pcolour are ordered and once completed the node will have one (ordered) successor for each phue.

The ordering of the successors will match the ordering of the hues (H3.1 and H6) so that we know there is a successor node containing a successor phue for each phue in the label. The respective orderings are otherwise arbitrary.

A *proto-hue (phue)* is just a subset of $\mathbf{cl}\,\phi$.

See Figure 2 for our definition of a tableau.

Definition. Say that the tableau (T, s, η, π) has *supported labelling* if each formula in each phue in each label is supported, as follows. Consider a formula $\alpha \in \eta_t(i)$. Determining whether α is support for not depends on the form of α:

- p is supported in $\eta_t(0)$. Otherwise, i.e. for $i > 0$, it is only supported if $p \in \eta_t(0)$.
- Same with $\neg p$.
- $\neg\neg\alpha$ supported iff $\alpha \in \eta_t(i)$.
- $\alpha \wedge \beta$ supported iff $\alpha \in \eta_t(i)$ and $\beta \in \eta_t(i)$.
- $\neg(\alpha \wedge \beta)$ supported iff either $\neg\alpha \in \eta_t(i)$ or $\neg\beta \in \eta_t(i)$.
- $X\alpha \in \eta_t(i)$ supported iff 1) there is $u \in T$ with $u = s_t(i)$ and 2) for all $u \in T$, for all j with $\pi^t_u(j) = i$, $\alpha \in \eta_u(j)$.
- $\neg X\alpha \in \eta_t(i)$ supported iff 1) there is $u \in T$ with $u = s_t(i)$ and 2) for all $u \in T$, for all j with $\pi^t_u(j) = i$, $\neg\alpha \in \eta_u(j)$.
- $\alpha U\beta \in \eta_t(i)$ supported iff 1) $\beta \in \eta_t(i)$; or 2) all 2.1) $\alpha \in \eta_t(i)$; 2.2) there is $u \in T$ with $u = s_t(i)$; and 2.3) for all $u \in T$, for all j with $\pi^t_u(j) = i$, $\alpha U\beta \in \eta_u(j)$.
- $\neg(\alpha U\beta) \in \eta_t(i)$ supported iff 1) $\neg\beta \in \eta_t(i)$; and 2) either 2.1) $\neg\alpha \in \eta_t(i)$; or 2.2) both 2.2.1) there is $u \in T$ with $u = s_t(i)$; and 2.2.2) for all $u \in T$, for all j with $\pi^t_u(j) = i$, $\neg(\alpha U\beta) \in \eta_u(j)$.
- $A\alpha \in \eta_t(i)$ supported iff for all $j < |\eta_t|$, $\alpha \in \eta_t(j)$.
- $\neg A\alpha \in \eta_t(i)$ supported iff there is some $j < |\eta_t|$, $\neg\alpha \in \eta_t(j)$.

A tableau is *successfully finished* iff it has no leaves, the predecessor relation is defined on all phues and the tableau does not fail any of the three checks that we introduce below: LG, NTP and the non-

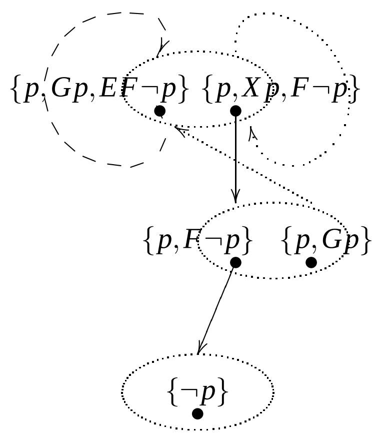


Figure 3: Example tableau.

existence of direct contradictions (or **false**) in phues.

It is common, in proving properties of tableau-theoretic approaches to reasoning, to refer to labelled structures as *Hintikka structures* if the labels are maximally complete (relative to a closure set). We say that one of our tableaux (T, s, η, π) is a Hintikka tableau iff the elements of each η_t are all hues (not just any phues). The older tableau approach in [17] was based on Hintikka tableaux.

5 Tableau Examples

Figure 1 is an example (unfinished) tableau illustrating general shape. There are 11 nodes, each with successors marked, and each labeled with a set of phues. Note that some of the successor relations involve up-links: $n1$ is a successor of $n3$. We just name the phues rather than listing their contents. There are more details about this example in [17] as, in fact, it is a Hintikka-tableau, which is a special type of the tableau we are introducing in this paper. We use Hintikka-tableaux later in the completeness proof here.

Figure 3 shows a smaller tableau in more detail. He we show the phues, which make up the pcolour labels of nodes and we show the predecessor or traceback map in some cases.

6 The LG test and Soundness

In this section we will briefly describe the LG rule which is a tableau construction rule that prevents bad up-links being added. LG is used to test and possibly fail a tableau. The test is designed to be used soon after any new up-link is added after being proposed by the LOOP rule. If the new tableau fails the LG test then “undo” the up-link and continue with alternative choices. We then show that if a tableau finishes, that is has no leaves, and passes the LG test then it guarantees satisfiability.

There was also a very similar LG test in the earlier work on the original slower tableau method [17]. In that paper, we show how to carry out the LG check on a tableau and we prove some results about its use. The check is very much like a model check on the tableau so far. We make sure that every phue in a label *matches*, or is a subset of an actual hue at that node in a transition structure defined using a

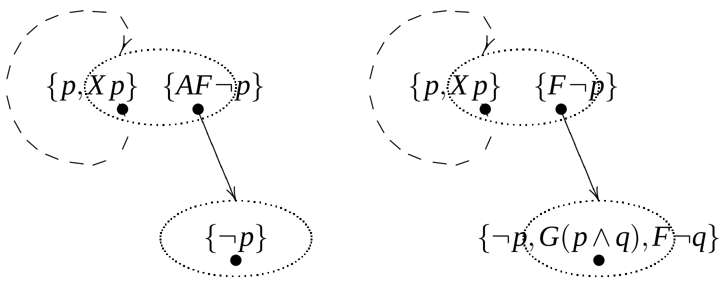


Figure 4: LG examples: left fails LG; right passes but eventually does not succeed

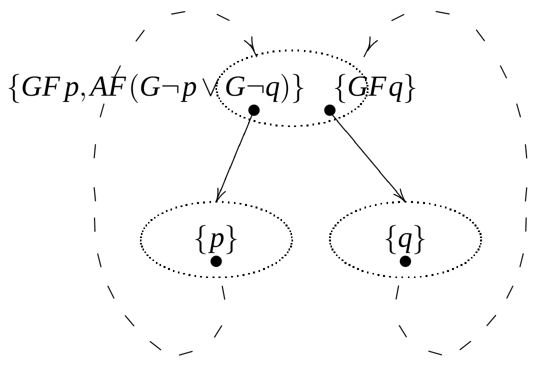


Figure 5: These two loops fail LG.

valuation of atoms based on the labels. It has polynomial running time in the size of the tableau so it is not a significant overhead on the overall tableau construction algorithm.

Due to space restrictions we do not go through the full details of the only very slightly different LG rule used for the faster tableaux here. Instead we give some brief motivation examples. The first example shows us that not all up-links are allowable: e.g., a node labelled with $p, AF\neg p$ which also has an immediate loop. See left hand example in Figure 4. The up-link would not be allowed by the LG rule.

The right hand example in Figure 4, with an allowable up-link and also separately an unsatisfiable leaf, is allowed by LG.

The example in Figure 5 has two loops, each one individually acceptable but not both. The LG rule fails the tableau when both up-links are added.

Now we show that if ϕ has a successfully finished tableau then ϕ is satisfiable. This is the soundness Lemma.

Lemma. If ϕ has a successfully finished tableau then ϕ is satisfiable.

Here we just outline the proof: details in [15]. Say that (T, s, η, π) is a successfully finished tableau for ϕ. Define a structure $M = (T, R, g)$ by interpreting the s relation as a transition relation g, and using η to define the valuation g on nodes.

By definition of matching, after a final check of LG there is some actual hue b of the root such that $\eta_{\mathbf{root}}(0) \subseteq b$. This means that ϕ holds along some fullpath in the final structure.

7 Building a tree

In this section we briefly describe how a tableau is built via some simple operations, or rules. We start with an initial tree of one root node labelled with just one phue containing only ϕ. The rules allow formulas to be added inside hues in labels, new hues to be added in labels and new nodes to be added as successors of existing nodes. The rules are generally non-deterministic allowing a finite range of options, or choices, at any application.

There are some properties to check such as LG, described above, and NTP described below. We also check that there are no hues containing both a formula and its negation, and we check that **false** is not contained in a phue. If these checks fail then the tableau has failed and we will need to backtrack to explore other possible options at choice points along the way.

The tableau succeeds if there are no leaves.

7.1 Basic Tableau Rules

Here are most of the basic rules, in an abbreviated notation:

2NEG: $\frac{\{\{\neg\neg\alpha\}\}}{\{\{\alpha\}\}}$ CONJ: $\frac{\{\{\alpha\wedge\beta\}\}}{\{\{\alpha,\beta\}\}}$ DIS: $\frac{\{\{\neg(\alpha\wedge\beta)\}\}}{\{\{\neg\alpha\}\} \mid \{\{\neg\beta\}\}}$ NEX: $\frac{\{\{X\alpha\}\}\to\{\{\}\}}{\{\{X\alpha\}\}\to\{\{\alpha\}\}}$ NNX: $\frac{\{\{\neg X\alpha\}\}\to\{\{\}\}}{\{\{\neg X\alpha\}\}\to\{\{\neg\alpha\}\}}$

UNT: $\frac{\{\{\alpha U\beta\}\}\to\{\{\}\}}{\{\{\alpha U\beta,\beta\}\}\to\{\} \mid \{\{\alpha U\beta,\alpha\}\}\to\{\{\alpha U\beta\}\}}$ NUN: $\frac{\{\{\neg(\alpha U\beta)\}\}\to\{\{\}\}}{\{\{\neg(\alpha U\beta),\neg\beta,\neg\alpha\}\}\to\{\} \mid \{\{\neg(\alpha U\beta),\neg\beta,\alpha\}\}\to\{\{\neg(\alpha U\beta)\}\}}$

ATM: $\frac{\{\{p\},\{\}\}}{\{\{p\},\{p\}\}}$ NAT: $\frac{\{\{\neg p\},\{\}\}}{\{\{\neg p\},\{\neg p\}\}}$ POS: $\frac{\{\{\neg A\alpha\}\}}{\{\{\neg A\alpha,\neg\alpha\}\} \mid \{\{\neg A\alpha\},\{\neg\alpha\}\}}$ NEC: $\frac{\{\{A\alpha\},\{\}\}}{\{\{A\alpha,\alpha\},\{\alpha\}\}}$

The rules are described in detail in [15] but the notation gives the main ideas. Here are details of a few of the rules above.

DIS: If $\neg(\alpha\wedge\beta)\in\eta_t(j)$ then can extend (T,s,η,π) to (T',s',η',π') via either: DIS1 or DIS2 as follows. DIS1 produces (T',s',η',π') such that $T'=T$, $s'=s$, and for all $t'\neq t$, $\eta_{t'}=\eta_t$ and for all $i'\neq i$, $\eta'_t(i')=\eta_t(i')$. However, $\eta'_t(i)=\eta_t(i)\cup\{\neg\alpha\}$. DIS2 is similar but use β instead of α.

NEX: If $X\alpha\in\eta_t(i)$ and there is $u\in T$ and j with $\pi^t_u(j)=i$ then can extend (T,s,η,π) to (T',s',η',π') such that $T'=T$, $s'=s$, and $\eta'_u(j)=\eta_u(j)\cup\{\alpha\}$. If $t\in T$ but there is no $s_t(j)\in T$ then extend (T,s,η,π) to (T',s',η',π') using new object t^+ such that $T'=T\cup\{t^+\}$, $s'_t(i)=t^+$, $\eta'_{t^+}(0)=\{\}$ and $\pi'^t_{t^+}(0)=i$. For all other arguments, s', η' and π' inherit values from s,η and π respectively.

ATM: If an atom $p\in\eta_t(j)$ and $k<|\eta_t|$ then can extend (T,s,η,π) to (T',s',η',π') such that $T'=T$, $s'=s$, and for all $t'\neq t$, $\eta_{t'}=\eta_t$ and for all $i'\neq k$, $\eta'_t(i')=\eta_t(i')$. However, $\eta'_t(k)=\eta_t(k)\cup\{p\}$.

POS: If $\neg A\alpha\in\eta_t(j)$ and $n=|\eta_t|$ then can extend (T,s,η,π) to (T',s',η',π') via one of POS_k for some $k=0,1,2,...,n$ as follows. For $k<n$, POS_k involves extending (T,s,η,π) to (T',s',η',π') where $T'=T$, $s'=s$, and for all $t'\neq t$, $\eta_{t'}=\eta_t$ and for all $i'\neq k$, $\eta'_t(i')=\eta_t(i')$. However, $\eta'_t(k)=\eta_t(k)\cup\{\neg\alpha\}$. However, POS_n involves extending (T,s,η,π) to (T',s',η',π') where $T'=T$, $s'=s$, and for all $t'\neq t$, $\eta_{t'}=\eta_t$ and for all $i'\neq k$, $\eta'_t(i')=\eta_t(i')$. However, $\eta'_t(k)=\eta_t(k)\cup\{\neg\alpha\}$.

There are also a couple of rules not sketched above.

PRED: If $t,u\in T$ and u is a successor of t but $\pi(t_u(j))$ is not defined then we can extend (T,s,η,π) to (T',s',η',π') via one of PRED_k for some $k=0,1,2,...,|\eta_t|-1$ as follows.

For $k<|\eta_t|$, PRED_k involves extending (T,s,η,π) to (T',s',η',π') where $T'=T$, $s'=s$, and $\eta'=\eta$. However, $\pi'^t_u(j)=k$.

For $k = |\eta_t|$, PRED$_k$ involves extending (T,s,η,π) to (T',s',η',π') where $T' = T$, but $\eta' = \eta$ but giving t an extra empty phue $\eta'_t(k) = \{\}$; and $s = s'$.

Later we need to add a kth successor for t and fill in formulas in $\eta'_t(k)$.

Note that t now potentially becomes unsupported, untraceable and unfinished, again.

LOOP: Suppose t is an ancestor of the parent u^- of u, then we can choose either to replace the u^- to u edge by an up-link from u^- to t, or to not do that replacement (and continue the branch normally).

(It is worth remembering which choice you make and not try that again if it did not work.)

Note that, as in normal successors, we will also put $s_{u^-}(i) = t$ and $\pi_t^{u^-}(0) = i$ where previously we had $s_{u^-}(i) = u$. All the other phues in η_t will also have to have predecessors chosen amongst the phues in η_{u^-}. We will use the PRED rule to do this for each one.

Note also that making such an up-link can possibly cause a subsequent consequential failure of the tableau. A contradiction could be introduced into the hues of t, the NTP could fail and/or the LG property could fail. It is possible to test for a few of these potential problems just before making use of this rule and act accordingly.

7.2 The NTP check: nominated thread property

The LG property check that every looping path is noticed by the labels in nodes. The converse requirement is taken care of by the much simpler NTP check.

We put a special significance on the initial hue in each colour label. This, along with the next condition, helps us ensure that each hue actually has a fullpath witnessing it. We are going to require the following property, NTP, of the tableaux which we construct.

First some auxiliary definitions: Definition. [hue thread] Suppose σ is a path through (T,s,η,π). A *hue thread* through σ is a sequence ξ of hues such that $|\xi| = |\sigma|$, for each $j < |\xi|$, $\xi_j \in \eta(\sigma_j)$ and for each $j < |\xi| - 1$, $\xi_j r_X \xi_{j+1}$.

Definition. [fulfilling hue thread] Suppose σ is a path through (T,s,η,π) and ξ is a hue thread through σ. We say that ξ is fulfilling iff either $|\sigma| < \omega$, or $|\sigma| = \omega$ and all the eventualities in each ξ_i are witnessed by some later ξ_j; i.e. if $\alpha U \beta \in \xi_i$ then there is $j \geq i$ such that $\beta \in \xi_j$.

Definition. [the nominated thread property] We say that the tableau (T,s,η,π) has the *nominated thread property* (NTP) iff the following holds. Suppose that for all $t \in T$ such that $0 < |s_t|$, $s_t(0)$ is an ancestor of t and that $t_0 = s_t(0), t_1, ..., t_k = t$ is a non-repeating sequence with each $t_{j+1} = s_{t_j}(0)$. Let σ be the fullpath $\langle t_0, t_1, ..., t_k, t_0, t_1, ..., t_k, t_0, t_1, ...\rangle$ and ξ be the sequence $\langle \eta_{t_0}(0), \eta_{t_1}(0), ..., \eta_{t_k}(0), \eta_{t_0}(0), ...\rangle$ of hues in σ. Then ξ is a fulfilling hue thread for σ.

It is straightforward to prove that this is equivalent to checking that each eventuality in $\eta_{t_0}(0)$ (or in all, or any, $\eta_{t_i}(0)$) is witnessed in at least one of the $\eta_{t_j}(0)$. So it is neither hard to implement nor computationally complex.

Using the rules described above, using any applicable one at any stage, allows construction of tableaux. We know that the LG rule ensures that any successful ones which we build thus will guarantee that ϕ is satisfiable. In the next section we consider whether we can build a successful tableau for any satisfiable formula in the way.

8 Completeness Using the Hintikka Tableau

In [17], the completeness result for the tableau in that paper, shows that for any satisfiable CTL* formula there is a finite model satisfying certain useful properties and from that we can find a successful tableau (as defined in that paper) for the formula. In fact the tableau constructed in that paper is just a special form of the tableaux that we are constructing in this paper: they are Hintikka structures.

Definition. A structure (T,s,η,π) is a *Standard Hintikka Tableau* for ϕ iff (T,s,η,π) is a finite finished successful tableau for ϕ and for each t, for each i, $\eta_t(i)$ is an MPC subset of $\mathbf{cl}\,(\phi)$.

Thus, in a Hintikka tableau, the labels tell us exactly which formulas hold there.

The completeness result in [17] shows the following, in terms of the concepts defined in this paper:

Lemma. If $\phi \in L$ is satisfiable then it has a Standard Hintikka Tableau.

The proof of this lemma is a straightforward translation of the definitions from [17] but we need to specify how to define our current predecessor relation π and we also need to check that the tableau is finished.

The predecessor relation π is not made explicit in the tableau structures of the earlier paper. Instead we require that the colour of a node t is related by a successor relation R_X between colours to the colour of any successor t'. This means that for any hue in the colour of t' there is a hue h in the colour of t such that h and h' are related by a successor relation between hues. We can use such a hue h as the predecessor of h' and so define π.

To show that the tableau (T,s,η,π) is finished, we just need to check all the rules of our tableau construction and make sure none require the tableau to be changed in any way. This needs to be done each rule at a time, and needs to be done carefully, although it is straightforward.

The proof in [17] uses a finite model theorem for CTL* to obtain a *branch boundedness* result on the Hintikka tableau. We can guarantee existence of a such a tableau with a certain function of the length of the formula bounding the length of each branch (before an up-link). The bound is triple exponential in the length of the formula, so rather large.

Thus we can conclude that each satisfiable formula has a tableau, but we can not yet claim that it is a tableau which can be constructed by our rules.

In the rest of this section we describe how we can show that if ϕ is satisfiable then there is a sequence of applications of our tableau rules that allow the construction of a successful tableau for ϕ. Suppose ϕ is satisfiable. From the lemma above we know that there is a successful, branch-bounded, supported tableau $T^{-\infty} = (T',s',\eta',\pi')$ for ϕ.

In [15], we show how to build a related, successful tableau for ϕ in a step by step manner only using the construction rules from section 7.1. Thus we make a sequence $T^0, T^1, \ldots$ of tableaux each one using a construction step to get to the next.

In order to use $T^{-\infty}$ to guide us, we also construct a sequence of maps $w_0, w_1, w_2, \ldots$, each w_i relating the phues of the labels of the nodes of T^i to the hues of the labels of the nodes of $T^{-\infty}$.

Thus each w_i maps ordered pairs which are nodes paired with indices to other such pairs. Suppose that $T^i = (T,s,\eta,\pi)$ and $T^{-\infty} = (T',s',\eta',\pi')$. Say $t \in T^i$ and $j < |\eta_t|$. Then $w_i(t,j)$ will be defined: say that $w_i(t,j) = (u,k)$ for $u \in T'$. Then $k < |\eta'_u|$. The idea in this example is that w_i is associating the jth phue of t with the kth phue of u.

All the while during the construction we ensure that w_i maps each node in T^i to a node in $T^{-\infty}$ which has a superset label.

We also show that the constructed tableau does not fail at any stage if one of the check rules such as LG, NTP or the existence of direct contradictions in phues. This follows from the fact that the phues in its labels are subsets of the hues in the labels of the Hintikka tableau.

If T is finished (leafless), supported and all predecessors exist then we are done. If T is not supported then choose any formula α in any phue in the label of any node that is not supported. Depending on the form of α we apply one of the tableau rules to add some successor, or some phue and/or some formula(s) in a phue that will ensure that α is then supported. See [15] for details.

There are only a finite number of formulas that can be added in hues in labels in a finite structure which is a subset of $T^{-\infty}$. This guarantees that the process will eventually terminate.

Thus every satisfiable formula has a successful tableau which can be found via our set of rules.

In fact, we can go further and get an even better completeness result. We can show that each formula ϕ only has a finite number of tableaux which respect the branch bounds and a simple bound on branching factor. Furthermore, if there is a successful tableau then there will be one obeying these bounds. There are at most $2^{|\phi|}$ hues and so each node in a Hintikka tableau has at most $2^{|\phi|}$ successors: by the form of completeness proof we can enforce the same bound on our more general tableaux. As we also have a finite bound on the length of branches there are clearly only finitely many tableaux for any particular ϕ.

Lemma. Given ϕ, there are only a finite number of tableaux which respect the branch length bound and the branching degree bounds.

In this definition of tableau we have guaranteed termination of any tableau construction algorithm by putting a simple but excessive bound on the length of branches. This allows us to conclude failure in a finite time and to also abbreviate the search for successful tableaux.

9 Stopping Repetition: coming up in follow-on paper

In this paper we have only briefly mentioned the limit on the length of branches as a way of guaranteeing that there are only finitely many tableau, and so that a search will terminate one way or another. The limit, based on a theoretical upper bound on the minimal CTL* model size, is very generous and hence this is an inefficient way of cutting short tableau searches. Being so generous slows down both negative and positive satisfiability reports.

In order to make some sort of working implementation to demonstrate the practicality of this tableau it is necessary to have a better way of preventing the construction of wastefully long branches. For want of better terminology we will call such a facility, a “repetition checker”.

The task of making a quick and more generally usable repetition checker will be left to be advanced and presented at a later date. In fact, eventually we hope to provide a useful set of criteria for earlier termination of construction of branches depending on the properties of the sequence of colours so far. A simple example of the sort of criterion is the repeated appearance of the same sequence of colours and hues along a non-branching path without being able to construct any up-links. Other more sophisticated ideas are easily suggested but we want to develop a more systematic set of tests before presenting this in future work.

In [17], we present some basic repetition checking tests for the Hintikka style tableau. These can be used in order to allow some faster automated tableau construction. The tests can be modified to work with our sparser labels, and we will present full details in a future paper. There are many opportunities for more thorough repetition checks as well.

10 Complexity, Implementation and Comparisons

Say that $|\phi| = l$. Thus ϕ has $\leq l$ subformulas and $\mathbf{cl}\,\phi$ contains at most $2l$ formulas. Since each hue contains, for each $\alpha \leq \phi$ at most one of α or $\neg\alpha$, there are at most $\leq 2^l$ hues. Thus there are less than 2^{2^l} colours. It is straightforward to see that there is a triple exponential upper bound if the tableau search algorithm uses the double exponential bound on branch length [17] to curtail searches down long branches.

A prototype implementation written by the author shows that for many interesting, albeit relatively small, formulas, the experimental performance of the system is relatively impressive. There are some preliminary results detailed in [15] which show a comparison of running times with the older Hintikka-style tableau technique of [17] and the state of the art game-based CTL* reasoner from [7]. In general the new reasoner is more than an order of magnitude quicker at deciding formulas from a range of basic and distinctive CTL* validities and their negations and a few other satisfiable formulas. The implementation is available as Java code for public download [15]. Online reasoner coming soon.

The implementation for the new technique that is used in these experiments, uses some basic repetition checking derived from the checks given earlier in the Hintikka-style system [17]. The new, slightly modified versions of these mechanisms are not described in the current paper. Instead they will be described in a future paper.

In [7], four series of formulas are suggested to examine asymptotic behaviour. Timing results for our system on these formulas are presented in Table 6. We compare the performance of our new tableau with the state of the art in game-based techniques for deciding CTL*. This is using published performance of the reasoner from [7] as reported in experiments in [11]. Consider the following series of formulas: $\alpha_1 = AFGq$, $\beta_1 = AFAGq$ and for each $i \geq 1$, $\alpha_{i+1} = AFG\alpha_i$ and $\beta_{i+1} = AFAG\beta_i$. In table 6, we compare the performance of the Hintikka-style tableau system from [17], the game-based reasoner from [7] and the new tableau system of this paper (using basic repetition checking) on the growing series built from these formulas. Although the running times, are on different computers, and so not directly comparable, we can see the difference in asymptotic performance. Running times greater than an hour or two are curtailed. From the results we see that we have achieved very noticeable and significant improvements in performance on the satisfiable examples.

Pure tableau-style reasoning on unsatisfiable formulas often involves exhaustive searches and the new technique is not immune to such problems. See the 400 series of examples in the asymptotic experiments. We will say more about these examples when proposing some new repetition mechanisms in the future.

There are some, more theoretical descriptions of other game-based and automata-based techniques for model-checking CTL* in older papers such as [10], [2] and [9]. However, these do not seem directly applicable to satisfiability decisions and/or there do not seem to be any easily publicly available implemented tools based on these approaches.

11 Conclusion

In this paper we have presented, albeit in a fairly high level sketch, a traditional tableau approach to reasoning with the important logic CTL*. Soundness and completeness results are proved and prototype implementation demonstrates the significantly improved performance of the new approach on a range of test formulas.

The next task in this direction is to build on the foundation here and present full details and proofs of the repetition checking mechanisms that can be used with the tableau construction. There are some basic repetition mechanisms available in the previous, Hintikka style tableau [17] but they need to be modified slightly. There are opportunities for additional techniques. It is also important to improve and document the rule-choice algorithms which have a bearing on running times.

In the future, it will be useful to develop reasoning tools which combine the latest in tableaux, automata and game-based approaches to CTL*. Having tools working in parallel should allow faster decisions. It will also be useful to extend the work to logics of multi-agent systems such as ATL* and strategy logic [12].

#	formula	length	sat?	MRH [17]	FLL [7]	NEW this paper
101	$\alpha_1 \rightarrow \beta_1$	20	Y	330	120	39
102	$\alpha_2 \rightarrow \beta_2$	35	Y	$> 10^5$	130	43
103	$\alpha_3 \rightarrow \beta_3$	50	Y	out of time	120	69
108	$\alpha_8 \rightarrow \beta_8$	125	Y	out of time	380	664
113	$\alpha_{13} \rightarrow \beta_{13}$	200	Y	out of time	$> 10^5$	2677
115	$\alpha_{15} \rightarrow \beta_{15}$	230	Y	out of time	$> 10^6$	4228
119	$\alpha_{19} \rightarrow \beta_{19}$	290	Y	out of time	out of time	9468
201	$\neg(\alpha_1 \rightarrow \beta_1)$	21	Y	350	120	172
202	$\neg(\alpha_2 \rightarrow \beta_2)$	36	Y	$> 10^5$	170	117
203	$\neg(\alpha_3 \rightarrow \beta_3)$	51	Y	out of time	2270	213
204	$\neg(\alpha_4 \rightarrow \beta_4)$	66	Y	out of time	$> 10^6$	377
205	$\neg(\alpha_5 \rightarrow \beta_5)$	81	Y	out of time	out of time	673
212	$\neg(\alpha_{12} \rightarrow \beta_{12})$	186	Y	out of time	out of time	7153
301	$\beta_1 \rightarrow \alpha_1$	20	Y	340	130	48
302	$\beta_2 \rightarrow \alpha_2$	35	Y	$> 10^5$	140	50
303	$\beta_3 \rightarrow \alpha_3$	50	Y	out of time	140	86
312	$\beta_{12} \rightarrow \alpha_{12}$	185	Y	out of time	30970	3333
314	$\beta_{14} \rightarrow \alpha_{14}$	215	Y	out of time	$> 10^6$	5512
316	$\beta_{16} \rightarrow \alpha_{16}$	245	Y	out of time	out of time	8627
319	$\beta_{19} \rightarrow \alpha_{19}$	290	Y	out of time	out of time	15615
401	$\neg(\beta_1 \rightarrow \alpha_1)$	21	N	400	760	1801
402	$\neg(\beta_2 \rightarrow \alpha_2)$	36	N	$> 10^5$	48670	$> 10^5$
403	$\neg(\beta_3 \rightarrow \alpha_3)$	51	N	out of time	$> 10^6$	out of time

Figure 6: Asymptotic Examples: Running Times (milliseconds)

References

[1] Sergei N. Artëmov & Anil Nerode, editors (2009): *Logical Foundations of Computer Science, International Symposium, LFCS 2009, Deerfield Beach, FL, USA, January 3-6, 2009. Proceedings*. *Lecture Notes in Computer Science* 5407, Springer. Available at `http://dx.doi.org/10.1007/978-3-540-92687-0`.

[2] Orna Bernholtz, Moshe Y. Vardi & Pierre Wolper (1994): *An Automata-Theoretic Approach to Branching-Time Model Checking (Extended Abstract)*. In David L. Dill, editor: *CAV*, *Lecture Notes in Computer Science* 818, Springer, pp. 142–155. Available at `http://dx.doi.org/10.1007/3-540-58179-0_50`.

[3] E. Emerson & J. Halpern (1986): *'Sometimes' and 'not never' revisited*. *J. ACM* 33. Available at `http://dx.doi.org/10.1145/4904.4999`.

[4] E. Emerson & C. Jutla (1988): *Complexity of Tree Automata and Modal Logics of Programs*. In: *29th IEEE Foundations of Computer Science, Proceedings*, IEEE.

[5] E. Emerson & A. Sistla (1984): *Deciding full branching time logic*. *Inf. and Control* 61, pp. 175 – 201. Available at `http://dx.doi.org/10.1016/S0019-9958(84)80047-9`.

[6] E.A. Emerson (1990): *Temporal and modal logic*. In J. van Leeuwen, ed.: *Hbk of Th. Comp. Sci.*, B, Elsevier.

[7] O. Friedmann, M. Latte & M. Lange (2010): *A Decision Procedure for CTL* Based on Tableaux and Automata*. In: *IJCAR'10*, pp. 331–345. Available at `http://dx.doi.org/10.1007/978-3-642-14203-1_28`.

[8] Valentin Goranko & Dmitry Shkatov (2009): *Tableau-Based Procedure for Deciding Satisfiability in the Full Coalitional Multiagent Epistemic Logic*. In Artëmov & Nerode [1], pp. 197–213. Available at `http://dx.doi.org/10.1007/978-3-540-92687-0_14`.

[9] Orna Kupferman & Moshe Y. Vardi (2005): *Safraless Decision Procedures*. In: *FOCS*, IEEE Computer Society, pp. 531–542. Available at `http://doi.ieeecomputersociety.org/10.1109/SFCS.2005.66`.

[10] M. Lange & C. Stirling (2000): *Model Checking Games for CTL**. In: *In ICTL'00*, pp. 115–125.

[11] J. McCabe-Dansted (2011): *A Rooted Tableau for BCTL*. *Electr. Notes Theor. Comput. Sci.* 278, pp. 145–158. Available at `http://dx.doi.org/10.1016/j.entcs.2011.10.012`.

[12] Fabio Mogavero, Aniello Murano, Giuseppe Perelli & Moshe Y. Vardi (2012): *What Makes ATL* Decidable? A Decidable Fragment of Strategy Logic*. In Maciej Koutny & Irek Ulidowski, editors: *CONCUR*, *Lecture Notes in Computer Science* 7454, Springer, pp. 193–208. Available at `http://dx.doi.org/10.1007/978-3-642-32940-1_15`.

[13] A. Pnueli (1977): *The temporal logic of programs*. In: *Proceedings of the Eighteenth Symposium on Foundations of Computer Science*, pp. 46–57. Providence, RI.

[14] M. Reynolds (2001): *An Axiomatization of Full Computation Tree Logic*. *J.S.L.* 66(3), pp. 1011–1057.

[15] M. Reynolds (May, 2013): *A Faster Tableau for CTL*, Long Version*. Technical Report, CSSE, UWA. Available at `http://www.csse.uwa.edu.au/~mark/research/Online/quicktab/`. Implemented reasoner also available here.

[16] Mark Reynolds (2009): *A Tableau for CTL**. In Ana Cavalcanti & Dennis Dams, editors: *FM 2009: Eindhoven, 2009. Proc.*, *Lecture Notes in Computer Science* 5850, Springer, pp. 403–418. Available at `http://dx.doi.org/10.1007/978-3-642-05089-3_26`.

[17] M. Reynolds (2011): *A tableau-based decision procedure for CTL**. *J. Formal Aspects of Comp.*, pp. 1–41. Available at `http://dx.doi.org/10.1007/s00165-011-0193-4`.

[18] M. Vardi & L. Stockmeyer (1985): *Improved Upper and Lower Bounds for Modal Logics of Programs*. In: *17th ACM Symp. on Theory of Computing, Proceedings*, ACM, pp. 240–251. Available at `http://dx.doi.org/10.1145/22145.22173`.

Deciding the Satisfiability of MITL Specifications[*]

Marcello M. Bersani[*] Matteo Rossi[*] Pierluigi San Pietro[*,+]
[*] Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy
[+]CNR IEIIT-MI, Milano, Italy
{marcellomaria.bersani,matteo.rossi,pierluigi.sanpietro}@polimi.it

In this paper we present a satisfiability-preserving reduction from MITL interpreted over finitely-variable continuous behaviors to Constraint LTL over clocks, a variant of CLTL that is decidable, and for which an SMT-based bounded satisfiability checker is available. The result is a new complete and effective decision procedure for MITL. Although decision procedures for MITL already exist, the automata-based techniques they employ appear to be very difficult to realize in practice, and, to the best of our knowledge, no implementation currently exists for them. A prototype tool for MITL based on the encoding presented here has, instead, been implemented and is publicly available.

1 Introduction

Computer systems are inherently discrete-time objects, but their application to control and monitoring of real-time systems often requires to deal with time-continuous external signals and variables, such as position, speed and acceleration or temperature and pressure. Hence, many continuous-time models have been developed for verification and validation of such systems, e.g., Timed Automata [3], or continuous-time temporal logics, such as MITL (Metric Interval Temporal Logic) [4].

In general, the role of temporal logics in verification and validation is two-fold. First, temporal logic allows abstract, concise and convenient expression of required properties of a system. Linear Temporal Logic (LTL) is often used with this goal in the verification of finite-state models, e.g., in model checking [5]. Second, temporal logic allows a descriptive approach to specification and modeling (see, e.g., [19, 14]). A descriptive model is based on axioms, written in some (temporal) logic, defining a system by means of its general properties, rather than by an operational model based on some kind of machine (e.g., a Timed Automaton) behaving in the desired way. In this case, verification typically consists of satisfiability checking of the conjunction of the model and of the (negation of) its desired properties. An example of the latter approach is Bounded Satisfiability Checking (BSC) [20], where Metric Temporal Logic (MTL) specifications on *discrete* time and properties are translated into Boolean logic, in an approach similar to Bounded Model Checking of LTL properties of finite-state machines.

In general, verification of continuous-time temporal logics is not as well sopported as for discrete-time models. Uppaal [6] is the de-facto standard tool for verification of Timed Automata. However, Uppaal does not support continuous-time temporal logics: not only satisfiability checking is not available in Uppaal, but even the formalization of system properties in temporal logic is not allowed, aside from rather simple invariants and reachability properties. Rather, non-trivial properties to be verified on an operational model must be expressed as other Timed Automata, i.e., at a lower level of abstraction. Indeed, there have been a few proposals for verifying continuous-time logics [17], but they do not appear to be actually implementable, and, to the best of our knowledge, in fact they have never been implemented.

This paper proposes a new technique, based on generalizing BSC to MITL, by reducing satisfiability of MITL to satisfiability of *Constraint LTL over clocks* (CLTL-oc), a new decidable variant of CLTL [12].

[*]This research was supported by the Programme IDEAS-ERC, Project 227977-SMScom.

$$\begin{aligned} M,t &\models p \Leftrightarrow p \in M(t) \qquad p \in AP \\ M,t &\models \neg\phi \Leftrightarrow M,t \not\models \phi \\ M,t &\models \phi \wedge \psi \Leftrightarrow M,t \models \phi \text{ and } M,t \models \psi \\ M,t &\models \phi \mathbf{U}_I \psi \Leftrightarrow \exists t' \in t+I : M,t' \models \psi \text{ and } M,t'' \models \phi \ \forall t'' \in (t,t') \end{aligned}$$

Table 1: Semantics of MITL.

In particular, a MITL formula may be encoded into an equisatisfiable CLTL-oc formula, which can then be solved through the same techniques of [7, 9, 8]. The latter approach generalizes BSC to CLTL, generating an encoding suitable for verification with standard Satisfiability Modulo Theories (SMT) solvers such as Z3 [18]. This new technique has been implemented in an open-source prototype tool [1].

Although MITL is known to be decidable over unrestricted behaviors [16], we focus on so-called finitely-variable models, i.e. such that in every bounded time interval there can only be a finite number of changes. This is a very common requirement for continuous-time models, which only rules out pathological behaviors (e.g., Zeno [14]) which do not have much practical interest. To define the encoding, we start by focusing on models in which intervals are closed on the left end and open on the right end. This restriction is later lifted to consider general, finitely-variable, signals.

The paper is organized as follows: Sect. 2 defines MITL and CLTL-oc, Sect. 3 defines a reduction from MITL to CLTL-oc, based on the restriction that intervals are closed to the left and open to the right; Sect. 4 generalizes the translation to intervals of any kind, also discussing the extension to include past operators. Sect. 5 concludes, discussing applications to other logics and presenting a prototype tool.

2 Languages

Let AP be a finite set of atomic propositions. The syntax of (well formed) formulae of MITL is defined as follows, with $p \in AP$ and I an interval of the form $\langle a,b\rangle$ or $\langle a,+\infty\rangle$, with $a,b \in \mathbb{N}$ constants, $a < b$:

$$\phi := p \mid \phi \wedge \phi \mid \neg\phi \mid \phi \mathbf{U}_I \phi$$

The semantics of MITL is defined in Table 1 with respect to *signals*. A signal is a function $M : \mathbb{R}_+ \to 2^{AP}$, with $\mathbb{R}_+$ the set of nonnegative reals. A MITL formula ϕ is *satisfiable* if there exists a signal M such that $M,0 \models \phi$ (in this case, M is called a *model* of ϕ). The *globally* $\mathbf{G}_I$ and *eventually* $\mathbf{F}_I$ operators can be defined by the usual abbreviations: $\mathbf{F}_I\phi = \top\mathbf{U}_I\phi$ and $\mathbf{G}_I\phi = \neg\mathbf{F}_I(\neg\phi)$.

Constraint LTL (CLTL [12, 9]) is used in Sect. 3 to solve the satisfiability problem of MITL. CLTL formulae are defined with respect to a finite set V of variables and a *constraint system* $\mathscr{D}$, which is a pair $(D,\mathscr{R})$ with D being a specific domain of interpretation for variables and constants and $\mathscr{R}$ being a family of relations on D, such that the set AP of atomic propositions coincides with set $\mathscr{R}_0$ of 0-ary relations. An *atomic constraint* is a term of the form $R(x_1,\dots,x_n)$, where R is an n-ary relation of $\mathscr{R}$ on domain D and $x_1,\dots,x_n$ are variables. A *valuation* is a mapping $v : V \to D$, i.e., an assignment of a value in D to each variable. A constraint is *satisfied* by v, written $v \models_{\mathscr{D}} R(x_1,\dots,x_n)$, if $(v(x_1),\dots,v(x_n)) \in R$. Given a variable $x \in V$ over domain D, *temporal terms* are defined by the syntax: $\alpha := c \mid x \mid \mathrm{X}\alpha$, where c is a constant in D and x denotes a variable over D. Operator X is very similar to $\mathbf{X}$, but it only applies to temporal terms, with the meaning that $\mathrm{X}\alpha$ is the *value* of temporal term α in the next time instant.

$$
\begin{aligned}
(\pi,\sigma),i \models p &\Leftrightarrow p \in \pi(i) \text{ for } p \in AP\\
(\pi,\sigma),i \models R(\alpha_1,\ldots,\alpha_n) &\Leftrightarrow (\sigma(i+|\alpha_1|,x_{\alpha_1}),\ldots,\sigma(i+|\alpha_n|,x_{\alpha_n})) \in R\\
(\pi,\sigma),i \models \neg\phi &\Leftrightarrow (\pi,\sigma),i \not\models \phi\\
(\pi,\sigma),i \models \phi \wedge \psi &\Leftrightarrow (\pi,\sigma),i \models \phi \text{ and } (\pi,\sigma),i \models \psi\\
(\pi,\sigma),i \models \mathbf{X}(\phi) &\Leftrightarrow (\pi,\sigma),i+1 \models \phi\\
(\pi,\sigma),i \models \mathbf{Y}(\phi) &\Leftrightarrow (\pi,\sigma),i-1 \models \phi \wedge i > 0\\
(\pi,\sigma),i \models \phi\mathbf{U}\psi &\Leftrightarrow \exists\, j \geqslant i : (\pi,\sigma),j \models \psi \ \wedge (\pi,\sigma),n \models \phi \ \forall\, i \leqslant n < j\\
(\pi,\sigma),i \models \phi\mathbf{S}\psi &\Leftrightarrow \exists\, 0 \leqslant j \leqslant i : (\pi,\sigma),j \models \psi \wedge (\pi,\sigma),n \models \phi \ \forall\, j < n \leqslant i
\end{aligned}
$$

Table 2: Semantics of CLTL.

Well-formed CLTL formulae are defined as follows:

$$\phi := R(\alpha_1,\ldots,\alpha_n) \mid \phi \wedge \phi \mid \neg\phi \mid \mathbf{X}(\phi) \mid \mathbf{Y}(\phi) \mid \phi\mathbf{U}\phi \mid \phi\mathbf{S}\phi$$

where α_i's are temporal terms, $R \in \mathscr{R}$, **X**, **Y**, **U** and **S** are the usual "next", "previous", "until" and "since" operators of LTL, with the same meaning. The dual operators "release" **R**, and "trigger" **T** may be defined as usual, i.e., $\phi\mathbf{R}\psi$ is $\neg(\neg\phi\mathbf{U}\neg\psi)$ and $\phi\mathbf{T}\psi$ is $\neg(\neg\phi\mathbf{S}\neg\psi)$.

The semantics of CLTL formulae is defined with respect to a strict linear order representing time $(\mathbb{N},<)$. Truth values of propositions in AP, and values of variables belonging to V are defined by a pair (π,σ) where $\sigma : \mathbb{N} \times V \to D$ is a function which defines the value of variables at each position in $\mathbb{N}$ and $\pi : \mathbb{N} \to \wp(AP)$ is a function associating a subset of the set of propositions with each element of $\mathbb{N}$. The value of terms is defined with respect to σ as follows:

$$\sigma(i,\alpha) = \sigma(i+|\alpha|,x_\alpha)$$

where x_α is the variable in V occurring in term α and $|\alpha|$ is the *depth* of a temporal term, namely the total amount of temporal shift needed in evaluating α: $|x| = 0$ when x is a variable, and $|\mathrm{X}\alpha| = |\alpha| + 1$. The semantics of a CLTL formula ϕ at instant $i \geqslant 0$ over a linear structure (π,σ) is recursively defined as in Table 2, where $R \in \mathscr{R}\backslash\mathscr{R}_0$. A formula $\phi \in$ CLTL is *satisfiable* if there exists a pair (π,σ) such that $(\pi,\sigma),0 \models \phi$.

In this paper, we consider a variant of CLTL, where arithmetic variables are evaluated as *clocks* and set $\mathscr{R}$ is $\{<,=\}$. A clock "measures" the time elapsed since the last time the clock was "reset" (i.e., the variable was equal to 0). By definition, in CLTL-oc each $i \in \mathbb{N}$ is associated with a "time delay" $\delta(i)$, where $\delta(i) > 0$ for all i, which corresponds to the "time elapsed" between i and the next state $i+1$. More precisely, for all clocks $x \in V$, $\sigma(i+1,x) = \sigma(i,x) + \delta(i)$, unless it is "reset" (i.e., $\sigma(i+1,x) = 0$).

3 Reduction of MITL to CLTL-over-clocks

This section devises a reduction from MITL to CLTL-oc. The inherent bounded variability of metric operators in MITL allows a translation of a MITL formula ϕ into a CLTL-oc formula with a bounded number of variables, depending on the subformulae of ϕ.

As in [17, 13], it is actually convenient to introduce the operators $\mathbf{U}_{(0,+\infty)}$ and $\mathbf{F}_I$ as primitive, and instead derive the metric until $\mathbf{U}_I$, as shown by the following result.

Lemma 1. *Let M be a signal. Then, for any $t \geqslant 0$,*

$$(1)\;\; M,t \models \phi\mathbf{U}_{[a,b\rangle}\psi \Leftrightarrow M,t \models \mathbf{G}_{[0,a)}(\phi\mathbf{U}_{(0,+\infty)}\psi) \wedge \mathbf{F}_{[a,b\rangle}\psi$$

$$(2)\;\; M,t \models \phi\mathbf{U}_{(a,b\rangle}\psi \Leftrightarrow M,t \models \mathbf{G}_{[0,a]}(\phi\mathbf{U}_{(0,+\infty)}\psi) \wedge \mathbf{F}_{(a,b\rangle}\psi$$

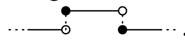
$$(3)\;\; M,t \models \phi\mathbf{U}_{\langle 0,b\rangle}\psi \Leftrightarrow M,t \models \phi\mathbf{U}_{\langle 0,+\infty)}\psi \wedge \mathbf{F}_{\langle 0,b\rangle}\psi$$

When b is $+\infty$, equivalences $(1),(2)$ can be simplified, respectively, in $\phi\mathbf{U}_{[a,+\infty)}\psi \equiv \mathbf{G}_{[0,a)}(\phi\mathbf{U}_{(0,+\infty)}\psi)$ and $\phi\mathbf{U}_{(a,+\infty)}\psi \equiv \mathbf{G}_{[0,a]}(\phi\mathbf{U}_{(0,+\infty)}\psi)$.

The above equivalences make it possible to base the CLTL-oc translation on the $\mathbf{U}_{(0,+\infty)}$ and $\mathbf{F}_I$ operators, instead of $\mathbf{U}_I$, therefore confining metric issues only to the translation of $\mathbf{F}_I$, which is much simpler than the translation of $\mathbf{U}_I$.

Reducing MITL to CLTL-oc requires a way to represent models of MITL formulae, i.e., continuous signals over a finite set of atomic propositions, by means of CLTL-oc models where time is discrete.

Discrete positions in CLTL-oc models represent, for each subformula θ of ϕ, the occurrence of an "event" at that point for the subformula. An "event" is a change of truth value ("become true" or "become false") of θ. Hence, the signal is "stable" (i.e., there is no change) in the interval between two events: a continuous-time signal is hence partitioned by the above events into intervals. Time progress between two discrete points is measured by CLTL variables behaving as clocks: for each subformula θ of ϕ, there are two clocks z^0_θ, z^1_θ measuring the time elapsed since the last "become true" and "become false" events, respectively (i.e., they are reset when the corresponding event occurs). In case of subformulae of the form $\theta = \mathbf{F}_{\langle a,b\rangle}\phi$, also a finite set of auxiliary clocks is introduced, whose cardinality depends on the values of a,b, namely $d = 2\left\lceil \frac{b}{b-a} \right\rceil$ auxiliary clocks x^j_θ $(0 \leqslant j \leqslant d-1)$. Therefore, a CLTL-oc model embeds, in every (discrete) position both the information defining the truth value of all the subformulae occurring in ϕ and also the time progress between two consecutive events. Then, every position in the CLTL-oc model captures the configuration of one of the intervals in which the MITL signals are partitioned by the events. Therefore, our reduction defines, by means of CLTL-oc formulae, the semantics of every subformula of ϕ.

We start by restricting the set of signals defining models of MITL formulae to signals where intervals are left-closed and right-open (*l.c.r.o.*), e.g.: . We will lift this restriction later in the paper. Hence, singularities (i.e., events being true in a single instant) cannot occur and may be ignored. However, the semantics given here does not exclude *a priori* Zeno behaviors [14]: it admits signals corresponding to an infinite sequence of events accumulating to the left of a time instant, i.e., where events do not advance beyond that instant. However, since these signals correspond to behaviors that are of little interest in practice, we restrict the set of models to non-Zeno signals, i.e., to models of CLTL-oc formulae where time diverges: $\sum_{i\in\mathbb{N}} \delta(i) = \infty$, by enforcing a suitable CLTL-oc constraint.

Let M be a signal, ϕ a MITL formula over AP and $sub(\phi)$ the set of all subformulae occurring in ϕ. We write $\uparrow_\theta$ for the occurrence of an event making $\theta \in sub(\phi)$ become true. With abuse of notation we extend $\models$ as follows:

$$M,t \models \uparrow_\theta \Leftrightarrow M,t \models \theta \text{ and } \begin{pmatrix} \exists\varepsilon > 0\ \forall t' \in (t, t+\varepsilon)\ M,t' \models \theta \text{ and} \\ t > 0 \Rightarrow \exists\varepsilon > 0\ \forall t' \in (t-\varepsilon, t)\ M,t' \models \neg\theta \end{pmatrix}$$

We define $\downarrow_\theta$ as an abbreviation for $\uparrow_{\neg\theta}$. These definitions impose that signals are defined over an infinite sequence of intervals of the form $[t_1, t_2)$ where $t_2 > t_1$.

Not all temporal operators preserve l.c.r.o. intervals. For example, let $\theta = \mathbf{F}_{\langle a,b)}\phi$ be a MITL formula and let ϕ hold on a l.c.r.o. signal; then, the corresponding signal for θ (i.e., the signal including also the values for $\uparrow_\theta$), is not l.c.r.o.. In fact, let $t > b$ be the first position such that $M,t \models \uparrow_\phi$. If the signal for θ were l.c.r.o., then it should be $M,t-b \models \uparrow_\theta$, which is impossible because $M,t-b \models \mathbf{F}_{\langle a,b\rangle}\phi \Leftrightarrow \exists t'' \in t-b+\langle a,b\rangle\ M,t'' \models \phi$ and $t'' < t$, but by hypothesis ϕ is false before t. Nevertheless, the next result shows that that Boolean connectives $\neg$, $\wedge$ and temporal operators $\mathbf{U}_{(0,+\infty)}$, $\mathbf{F}_{\langle a,b]}$, $\mathbf{F}_{\langle a,+\infty)}$ and $\mathbf{F}_{\langle 0,b]}$, do indeed preserve l.c.r.o. intervals.

We extend MITL models to any subformulae occurring in MITL formulae by defining a mapping $M_\theta : \mathbb{R}_+ \rightarrow \{\varnothing, \theta\}$ such that:

$$\theta \in M_\theta(t) \Leftrightarrow M,t \models \theta.$$

Lemma 2. *Let M be a l.c.r.o. signal, let ϕ,ψ be two formulae occurring in M and let θ be a formula $\neg\phi$, $\phi \wedge \psi$, $\mathbf{U}_{(0,+\infty)}(\phi,\psi), \mathbf{F}_{\langle a,b]}(\phi), \mathbf{F}_{\langle a,+\infty)}(\phi), \mathbf{F}_{\langle 0,b]}(\phi)$. Then, M_θ is a l.c.r.o. signal.*

In what follows, $\mathbf{F}_{\langle a,+\infty)}$ is defined as primitive, instead of applying the known equivalence $\mathbf{F}_{[a,+\infty)}\phi \equiv \top\mathbf{U}_{[a,+\infty)}\phi \equiv \mathbf{G}_{[0,a)}(\phi\mathbf{U}_{(0,+\infty)}\psi)$, as formula $\mathbf{G}_{[0,a)}\phi \equiv \neg\mathbf{F}_{[0,a)}\neg\phi$ violates the l.c.r.o. assumption.

We now show how to build a CLTL-oc model (π,σ) of ϕ from a signal M. For each subformula $\theta \in sub(\phi)$ we introduce two *clock variables* z^0_θ, z^1_θ and one atomic proposition $\overleftarrow{\theta}$. We will ensure that $\overleftarrow{\theta}$ is true at a position whenever θ is true in the interval corresponding to the position. To ease understanding, in the rest we use $\underleftarrow{\theta} = \neg\,\overleftarrow{\theta}$. We also introduce two abbreviations, $\lrcorner\!\ulcorner_\theta, \urcorner\!\llcorner_\theta$ that play the role of *event markers* (referred to as just "events" when the context is clear); more precisely, they denote, respectively, events $\uparrow_\theta$ and $\downarrow_\theta$, and are defined as follows:

$$\lrcorner\!\ulcorner_\xi = \neg\mathbf{Y}(\overleftarrow{\xi}) \wedge \overleftarrow{\xi} \qquad\qquad \urcorner\!\llcorner_\xi = \neg\mathbf{Y}(\underleftarrow{\xi}) \wedge \underleftarrow{\xi}$$

Note that, as $\neg\mathbf{Y}(\bullet)$ is true in the origin, no matter the argument, either $\lrcorner\!\ulcorner_\theta$ or $\urcorner\!\llcorner_\theta$ holds at 0.

For each $\theta = \mathbf{F}_{\langle a,b]}\psi \in sub(\phi)$ we introduce $d = 2\left\lceil \frac{b}{b-a} \right\rceil$ *auxiliary clocks* $x^0_\theta, \dots x^d_\theta$. The idea behind the above definitions is that at each occurrence of an event marker ($\lrcorner\!\ulcorner_\theta$ or $\urcorner\!\llcorner_\theta$), exactly one of the clocks z^0_θ, z^1_θ is equal to 0; the clock, then, measures the time elapsed from the last opposite event. Instead, the auxiliary clocks associated with formulae $\mathbf{F}_{\langle a,b]}\psi$ are used to store the time elapsed since the occurrence of events involving ψ between the current time instant t and $t+b$. In fact, [17] shows that formulae of the form $\mathbf{F}_{\langle a,b\rangle}\psi$ have inherent bounded variability (the result holds for signals with no l.c.r.o. restriction).

Lemma 3 ([17])**.** *Let $\theta = \mathbf{F}_{\langle a,b\rangle}\psi$, M be a signal and let $0 < t_1 < t_2$ be two instants such that $M,t_1 \models \uparrow_\theta$, $M,t_2 \models \downarrow_\theta$ and $\forall t \in (t',t'')\ M,t \models \theta$. Then, $t_2 - t_1 \geqslant b - a$.*

By Lemma 3, two consecutive events $\uparrow_\theta$ and $\downarrow_\theta$ for formulae $\theta = \mathbf{F}_{\langle a,b\rangle}\psi$ cannot occur at a distance less than $b-a$. However, this does not hold when $\uparrow_\theta$ occurs at $t = 0$ and ψ is true at 0, but it becomes false before b. For instance, let $M,a \models p$ and $M,a+\varepsilon \models \downarrow_p$, where $\varepsilon > 0$ is such that $a + \varepsilon < b$; assume for simplicity that p remains false, i.e., for all $t \in [a+\varepsilon,+\infty)$, $M,t \not\models \psi$. Then, we have that $M,0 \models \uparrow_\theta$ and $M,\varepsilon \models \downarrow_\theta$. This property will be exploited in Sect. 3.2 to define the translation of the $\mathbf{F}$ operator.

Corollary 1. *Let $\theta = \mathbf{F}_{\langle a,b]}\phi$ be a MITL formula, with $a > 0$, $b \neq \infty$, and let t be an instant of time. Then, in $[t,t+b]$ there are at most $d = 2\left\lceil \frac{b}{b-a} \right\rceil$ events $\uparrow_\theta, \downarrow_\theta$.*

The result of Corollary 1 can be significantly simplified for formulae of the form $\theta = \mathbf{F}_{\langle 0,b]}\phi$ or of the form $\theta = \mathbf{F}_{\langle a,+\infty)}\phi$. In fact, in the former case, let $t_2 > t_1 \geqslant 0$ be two time instants such that $M,t_1 \models \uparrow_\phi$, $M,t_2 \models \downarrow_\phi$ and $\forall t' \in [t_2, t_2+b]\ M,t' \not\models \phi$. Then, by definition, we have $M,t_1 - b \models \uparrow_\theta$, $M,t_2 \models \downarrow_\theta$ and $\forall t' \in [t_1 - b, t_2) \models \theta$. Therefore, no event for θ occurs over the interval $[t_1 - b, t_2)$. If $\theta = \mathbf{F}_{\langle a,+\infty)}\phi$, by definition, $M,t \models \theta \Leftrightarrow \exists t' \in \langle t+a,+\infty)\ M,t' \models \phi$; hence, $M,t \models \theta \Rightarrow M,0 \models \theta$, i.e., $M,0 \models \uparrow_\theta$. Event

$\uparrow_\theta$ occurs in 0 if, and only if: $\exists t \geqslant a\, M,t \models \uparrow_\phi$ or $\exists t > a\, M,t \models \downarrow_\phi$ or $\exists t < a\, M,t \models \uparrow_\phi \wedge \forall t' > t\, M,t' \models \phi$. Moreover, $M,t \not\models \theta \Rightarrow \forall t' \in \langle t+a, +\infty)\; M,t' \not\models \phi$, i.e., $M,t \models \downarrow_\theta \Leftrightarrow M,t+a \models \downarrow_\phi \wedge \mathbf{G}(\neg\phi)$. By the previous properties, the translation of formulae involving $\mathbf{F}_{\langle 0,b]}$ and $\mathbf{F}_{\langle a,+\infty)}$ is simpler than the case $a > 0$ and $b \neq \infty$, because auxiliary clocks are not needed to represent the formula. For this reason, we provide a direct translation for these subformulae.

Since signals are finitely variable, all the events in M can be enumerated as follows. A position $i \geqslant 0$ uniquely identifies a time instant along M. Let $T \subset \mathbb{R}_+$ be an infinite, but enumerable, set of time instants that includes 0 and every instant when at least one event occurs. Let $I : T \to \mathbb{N}$ be a one-to-one mapping, consistent with the ordering of time, i.e, $I(0) = 0$ and $I(t) < I(t') \Leftrightarrow t < t'$, and such that for all $t_1 < t_2 \in T$ $I(t_2) = I(t_1) + 1 \Leftrightarrow \neg\exists t\,(t_1 < t < t_2 \wedge t \in T)$. By definition, for each subformula θ an event (either $\lrcorner\!\ulcorner_\theta$ or $\urcorner\!\llcorner_\theta$) always occurs at $I(0) = 0$.

Now, given a MITL formula ϕ and a signal M such that $M,0 \models \phi$, we define how to build CLTL-oc interpretations from M. We will prove afterwards that this interpretation is a model for the CLTL-oc formula translating ϕ. We say that a clock v is *reset* at position i when $\sigma(i,v) = 0$.

Let (π,σ) be a CLTL-oc interpretation. If an event for $\theta \in sub(\phi)$ occurs at $t \geqslant 0$, the corresponding event marker ($\lrcorner\!\ulcorner_\theta$ or $\urcorner\!\llcorner_\theta$) labels $\pi(I(t))$ and a reset for one of z^0_θ, z^1_θ occurs at $I(t)$:

- $\bigvee_{i\in\{0,1\}} \sigma(I(t), z^i_\theta) = 0$ and $(\pi,\sigma), I(t) \models \lrcorner\!\ulcorner_\theta$ if $M,t \models \uparrow_\theta$
- $\bigvee_{i\in\{0,1\}} \sigma(I(t), z^i_\theta) = 0$ and $(\pi,\sigma), I(t) \models \urcorner\!\llcorner_\theta$ if $M,t \models \downarrow_\theta$.
- $\sigma(0, z^0_\theta) = 0$ for all θ.
- $\sigma(0, x^0_\theta) = 0$ for all θ of the form $\mathbf{F}_{\langle a,b\rangle}\psi$.

Note that, by definition, for all time instants $t \in T$ where no events for θ occur, neither $\lrcorner\!\ulcorner_\theta$ nor $\urcorner\!\llcorner_\theta$ hold in $\pi(I(t))$ (i.e., $(\pi,\sigma), I(t) \models \neg\lrcorner\!\ulcorner_\theta \wedge \neg\urcorner\!\llcorner_\theta$).

Now we define how CLTL-oc models represent time progress. Let $t, t' \in T$ be two time instants such that $I(t') = I(t) + 1$. For all clocks z^i_θ that are not reset in $I(t')$ we impose

$$\sigma(I(t'), z^i_\theta) = \sigma(I(t), z^i_\theta) + t' - t.$$

In addition, $\exists i \in \{0,1\}$ s.t. $\sigma(I(t), z^i_\theta) = 0$ if and only if $(\pi,\sigma), I(t) \models \lrcorner\!\ulcorner_\theta$ or $(\pi,\sigma), I(t) \models \urcorner\!\llcorner_\theta$. Clocks z^0_θ, z^1_θ cannot be reset at the same time, but alternate, and z^0_θ is reset in the origin. Clocks x^j_θ are dealt with analogously. As mentioned, there exist $d = 2\left\lceil \frac{b}{b-a} \right\rceil$ clocks x^j_θ for a formula $\mathbf{F}_{\langle a,b\rangle}\psi \in sub(\phi)$. First, for all positions $i \geqslant 0$, $\sigma(i, z^0_\theta) = 0$ or $\sigma(i, z^1_\theta) = 0$ if, and only if, $\bigvee_{j=0}^{d-1} \sigma(i, x^j_\theta) = 0$, i.e, whenever an event for θ occurs, (at least) one auxiliary clock is reset. To avoid simultaneous resets of different clocks, if x^j_θ is reset then no $x^{j'}_\theta$ is reset, for $j' \neq j$. Auxiliary clocks are circularly reset modulo d; i.e., if x^j_θ is reset at position i, then the next reset of x^j_θ, if it exists, occurs in a position $i' > i$ such that all other clocks $x^{j'}_\theta$ ($j' \neq j$) are reset, in order, exactly once in (i, i'). Note that, if a clock x^j_θ is reset at position $i = I(t)$, the next position $i' = I(t')$ when the clock is reset must be such that $t' > t + b$, i.e., given a formula $\theta = \mathbf{F}_{\langle a,b]}$, every clock x^j_θ is reset only once over intervals of length b. The sequence of resets starts with $x^0_\theta = 0$.

Finally, if ϕ is satisfiable and M is a signal such that $M,0 \models \phi$ i.e., $M,0 \models \uparrow_\phi$, then $(\pi,\sigma), 0 \models \lrcorner\!\ulcorner_\phi$.

Let $r_\phi(M)$ denote the (infinite) set of pairs (π,σ) obtained from M by means of the previous rules for a MITL formula ϕ. The inverse mapping r_ϕ^{-1} is also definable, but not all pairs (π,σ) represent legal signals. Hence, we restrict them to the set of CLTL-oc models that are images of a signal M under r_ϕ, i.e., (π,σ) is such that there exists a signal M such that $(\pi,\sigma) \in r_\phi(M)$. Sect. 3.1 provides a set of CLTL-oc formulae whose models are exactly the set of pairs (π,σ) such that $(\pi,\sigma) \in r_\phi(M)$. For these models the inverse map r^{-1} is well-defined.

3.1 Clocks and Events

The following formulae define how events $\lrcorner\!\ulcorner_\theta, \urcorner\!\llcorner_\theta$ occur, for $\theta \in sub(\phi)$, and when clocks z^0_θ, z^1_θ are reset. However, they do not capture the semantics of subformulae θ, which is the object of Sect. 3.2, but only the relations between events $\lrcorner\!\ulcorner_\theta$ and $\urcorner\!\llcorner_\theta$ and clock resets.

Formula (1) enforces that the occurrence of an event $\lrcorner\!\ulcorner_\theta, \urcorner\!\llcorner_\theta$ entails the reset of one of z^0_θ, z^1_θ. In addition, Formula $z^0_\theta = 0$ evaluated in the origin states that clock z^0_θ is reset in the origin.

$$\lrcorner\!\ulcorner_\theta \vee \urcorner\!\llcorner_\theta \Leftrightarrow z^0_\theta = 0 \vee z^1_\theta = 0 \tag{1}$$

Let $a \in \mathbb{N}$ and value $\overline{a}_k$ be $(a \bmod k)$. The clocks associated with a subformula θ are alternatively reset, as shown on an example in Figure 1. Hence, between any two resets of clock z^0_θ there must be a reset of clock z^1_θ, and vice-versa:

$$\Big(\bigwedge_{i\in\{0,1\}} (z^i_\theta = 0)\Big) \Rightarrow \mathbf{X}\left((z_\theta^{\overline{(i+1)}_2} = 0)\mathbf{R}(z^i_\theta \neq 0)\right). \tag{2}$$

For a position $i > 0$ it may happen that neither $\lrcorner\!\ulcorner_\theta$ nor $\urcorner\!\llcorner_\theta$ occur for any formula (i.e, no events occur). The assumption that intervals are l.c.r.o. entails that intervals have non-null durations, and events $\uparrow_\theta, \downarrow_\theta$ cannot occur at the same time. Define $\texttt{events}_\phi = \bigwedge_{\theta\in sub(\phi)}(z^0_\theta = 0) \wedge \mathbf{G}((1) \wedge (2))$.

Lemma 4. *Let θ be a symbol of a MITL formula. For any non-Zeno signal $M : \mathbb{R}_+ \to \{\emptyset, \theta\}$ for θ and for all $(\pi,\sigma) \in r_\theta(M)$, then $(\pi,\sigma), 0 \models \texttt{events}_\theta$. Conversely, given (π,σ) in which time is divergent and s.t. $(\pi,\sigma), 0 \models \texttt{events}_\theta$, there is exactly one non-Zeno signal M s.t. $M = r_\theta^{-1}((\pi,\sigma))$.*

Let θ be $\mathbf{F}_{\langle a,b]}\psi$. We introduce $d = 2\left\lceil \frac{b}{b-a} \right\rceil$ clocks x^j_θ, which behave in a similar way as z^0_θ, z^1_θ. Each x^j_θ is needed to store the time elapsed since the occurrence of the last event of θ ($\uparrow_\theta$ or $\downarrow_\theta$). When one of $\uparrow_\theta, \downarrow_\theta$ occurs, then a x^j_θ is reset, i.e., $x^j_\theta = 0$. Each reset event marked by $x^i_\theta = 0$ entails either $\lrcorner\!\ulcorner_\theta$ or $\urcorner\!\llcorner_\theta$ and all $\uparrow_\theta$, $\downarrow_\theta$ events are marked by a single reset $x^i_\theta = 0$ (Formula (3)).

$$\left(\lrcorner\!\ulcorner_\theta \vee \urcorner\!\llcorner_\theta \Leftrightarrow \bigvee_{j=0}^{d-1} x^j_\theta = 0\right) \wedge \left(\bigwedge_{i=0}^{d-1} \bigwedge_{j=0, i\neq j}^{d-1} \neg(x^i_\theta = 0 \wedge x^j_\theta = 0)\right) \tag{3}$$

The occurrence of resets for clocks x^i_θ is circularly ordered and the sequence of resets starts from the origin by x^0_θ (see an example in Figure 1). If $x^i_\theta = 0$, then, from the next position, all the other clocks are strictly greater than 0 until the next $x_\theta^{\overline{i+1}_d} = 0$ occurs.

$$\bigwedge_{i=0}^{d-1}\left(x^i_\theta = 0 \Rightarrow \mathbf{X}\left((x_\theta^{\overline{i+1}_d} = 0)\mathbf{R} \bigwedge_{j\in[0,d-1], j\neq i} (x_\theta^{\overline{j+1}_d} > 0)\right)\right) \tag{4}$$

Formula $x^0_\theta = 0$, evaluated at position 0, sets the first reset of the sequence, constrained by formulae (3)-(4). Moreover, we force all clock values to be strictly ordered in the origin by $x^0_\theta < x^{d-1}_\theta < \cdots < x^1_\theta$, guaranteeing that resets are correctly associated with events occurring after the origin.

The following lemma (whose proof is similar to the one for Lemma 4) shows that $\texttt{auxclocks}_\theta$, defined as $(x^0_\theta = 0) \wedge \mathbf{G}((3) \wedge (4))$ captures map r for $\mathbf{F}_{\langle a,b]}$ formulae. .

Lemma 5. *Let $\theta = \mathbf{F}_{\langle a,b]}\psi$. For any signal $M : \mathbb{R}_+ \to \{\emptyset, \theta\}$ for θ and for all $(\pi,\sigma) \in r_\theta(M)$, it is $(\pi,\sigma), 0 \models \texttt{auxclocks}_\theta$. Conversely, if $(\pi,\sigma), 0 \models \texttt{auxclocks}_\theta$, there exists one, and only one, signal M s.t. $M = r_\theta^{-1}((\pi,\sigma))$.*

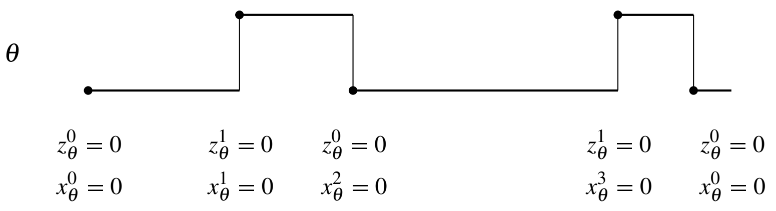


Figure 1: Sequence of circular resets for formula $\theta = \mathbf{F}_{\langle 2,1]}\psi$

3.2 Semantics of MITL Temporal Modalities

We now define a mapping m associating a MITL formula with an equisatisfiable CLTL-oc formula, thus capturing the semantics of MITL in CLTL-oc.

The cases for Boolean connectives and the non-metric $\mathbf{U}$ operator are straightforward. In the following we write O instead of $\neg\mathbf{Y}(\top)$ to represent the first position of CLTL-oc models.

- $\theta = p \in AP$: it follows from the definition of $\lrcorner\!\ulcorner_p$ and $\urcorner\!\llcorner_p$, representing events $\uparrow_p, \downarrow_p$ over discrete time.
- $\theta = \neg\psi$: in this case it is $m(\theta) = \overleftarrow{\theta} \Leftrightarrow \underleftarrow{\psi}$.
- $\theta = \gamma \wedge \psi$: we have: $m(\theta) = \overleftarrow{\theta} \Leftrightarrow \overleftarrow{\gamma} \wedge \overleftarrow{\psi}$.
- $\theta = \gamma\mathbf{U}_{(0,+\infty)}\psi$: similarly: $m(\theta) = \overleftarrow{\theta} \Leftrightarrow \overleftarrow{\gamma} \wedge \overleftarrow{\gamma}\,\mathbf{U}\,\overleftarrow{\psi}$.
- $\theta = \mathbf{F}_{\langle a,b]}\psi$: When an event $\uparrow_\theta$ occurs, a clock x_θ^j is reset, then event $\uparrow_\psi$ will eventually occur after b time units and it has to occur after $b-a$ instants from the last occurrence of $\downarrow_\psi$ (otherwise $\uparrow_\theta$ has already occurred in the past). The case for $t = 0$ is treated separately: $\uparrow_\theta$ occurs at 0 when there is an interval in which ψ holds that either starts in $[a,b]$ or it spans a. Clock x_θ^0 is used to measure the time elapsing from the origin. In fact, by Corollary 1, x_θ^0, which is reset at 0, can only be reset again after b.

$$
\lrcorner\!\ulcorner_\theta \Leftrightarrow
\begin{array}{c}
\neg O \wedge \bigvee_{j=0}^{d-1}(x_\theta^j = 0) \wedge \mathbf{X}\left(x_\theta^j > 0\,\mathbf{U}\left(\lrcorner\!\ulcorner_\psi \wedge x_\theta^j = b \wedge \bigvee_{i\in\{0,1\}} z_\psi^i > (b-a)\right)\right) \vee \\
O \wedge (O \vee x_\theta^0 > 0)\,\mathbf{U}\left(\overleftarrow{\psi} \wedge \left(a \leqslant x_\theta^0 \leqslant b \ \vee x_\theta^0 < a \wedge \mathbf{X}\left(x_\theta^0 > a\right)\right)\right)
\end{array}
\tag{5}
$$

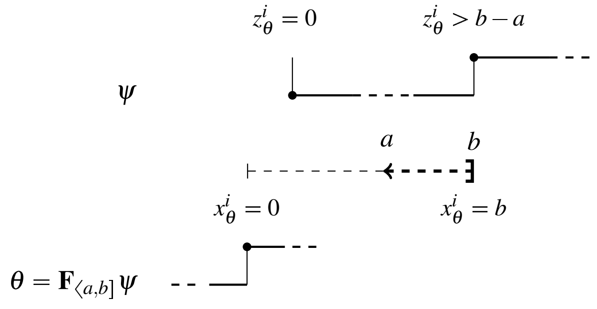


Figure 2: Rising edge

Formula (6) defines the condition to make $\lrcorner\!\ulcorner_\theta$ true exactly b instants before an event $\lrcorner\!\ulcorner_\psi$, provided that clock z_ψ^i is greater than $(b-a)$ when $\lrcorner\!\ulcorner_\psi$ occurs (i.e., the last time ψ became false was at least $b-a$

time units before). An illustration of Formulae (5) and (6) is in Figure 2.

$$\lrcorner\!\ulcorner_\psi \wedge \bigvee_{i\in\{0,1\}} z^i_\psi > (b-a) \Rightarrow \bigvee_{j=0}^{d-1} x^j_\theta = b \tag{6}$$

When an event $\downarrow_\theta$ occurs, a clock x^j_θ is reset, then the event $\downarrow_\psi$ will eventually occur after exactly a time units and the next $\uparrow_\psi$ cannot occur before another $b-a$ instants after that (otherwise $\downarrow_\theta$ cannot occur). In the origin, however, $\downarrow_\theta$ occurs also in the case that $\uparrow_\theta$ does not occur.

$$\urcorner\!\llcorner_\theta \Leftrightarrow \bigvee_{j=0}^{d-1}(x^j_\theta = 0) \wedge \mathbf{X}\left((x^j_\theta > 0)\mathbf{U}\left(\urcorner\!\llcorner_\psi \wedge x^j_\theta = a \;\wedge \lrcorner\!\ulcorner_\psi \mathbf{R}\neg\left(\lrcorner\!\ulcorner_\psi \wedge x^j_\theta \leqslant b\right)\right)\right) \;\vee\; (O \wedge \neg\lrcorner\!\ulcorner_\theta) \tag{7}$$

Formula (8) is the dual of (6) for a falling edge (Figure 3); it defines a sufficient condition forcing $\urcorner\!\llcorner_\theta$ when an event $\urcorner\!\llcorner_\psi$ occurs and $\lrcorner\!\ulcorner_\psi$ does not happen before $(b-a)$ time units have passed since $\urcorner\!\llcorner_\psi$.

$$\urcorner\!\llcorner_\psi \wedge \lrcorner\!\ulcorner_\psi \mathbf{R}\neg\left(\lrcorner\!\ulcorner_\psi \wedge \bigwedge_{i\in\{0,1\}} z^i_\psi \leqslant (b-a)\right) \Rightarrow \bigvee_{j=0}^{d-1} x^j_\theta = a \tag{8}$$

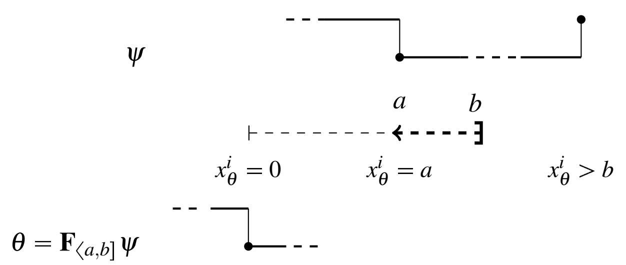


Figure 3: Falling edge

Formula $m(\theta)$ in this case is (5) $\wedge$ (6) $\wedge$ (7) $\wedge$ (8).

As already anticipated, we may study separately the case of formulae $\mathbf{F}_{\langle a,b]}\psi$ where $a = 0$ or $b = +\infty$. The translation in the two cases is simpler than the general one because auxiliary clocks are no longer required to measure the time elapsing between events involving signal for the formula.

- $\theta = \mathbf{F}_{\langle 0,b]}\psi$: the translation for event $\uparrow_\theta$ is analogous to the one of the general case where time elapsing is measured with respect to the clock z^j_θ that is reset when $\lrcorner\!\ulcorner_\theta$ holds (recall that, by Corollary 1, z^j_θ can be reset again only after the occurrence of $\lrcorner\!\ulcorner_\psi$). The semantics of $\downarrow_\theta$ in this case is simpler than for Formula (7) because events $\downarrow_\psi$ and $\downarrow_\theta$ always occur simultaneously, provided that the next $\uparrow_\psi$ does not occur within b time instants from $\downarrow_\psi$.

$$\lrcorner\!\ulcorner_\theta \Leftrightarrow \left(\begin{array}{c} \neg O \wedge \underset{\leftarrow}{\psi} \wedge \left(\bigvee_{j=0}^{1}(z^j_\theta = 0) \wedge \mathbf{X}\left(z^j_\theta > 0\,\mathbf{U}\left(\lrcorner\!\ulcorner_\psi \;\wedge z^j_\theta = b \wedge \bigvee_{i\in\{0,1\}} z^i_\psi > b\right)\right)\right) \;\vee \\ O \wedge (O \vee z^0_\theta > 0)\,\mathbf{U}(\overleftarrow{\psi} \wedge z^0_\theta \leqslant b) \end{array}\right) \tag{9}$$

$$\lrcorner\!\ulcorner_\psi \wedge \bigvee_{i\in\{0,1\}} z^i_\psi > b \Rightarrow \bigvee_{j\in\{0,1\}} z^j_\theta = b \tag{10}$$

$$\urcorner\!\!\llcorner_{\theta} \Leftrightarrow \urcorner\!\!\llcorner_{\psi} \wedge \lrcorner\!\!\ulcorner_{\psi} \mathbf{R} \neg \left(\lrcorner\!\!\ulcorner_{\psi} \wedge \bigwedge_{i \in \{0,1\}} z_{\psi}^{i} \leqslant b \right) \tag{11}$$

- $\theta = \mathbf{F}_{\langle a,+\infty)}\psi$: From the semantics of $\mathbf{F}_{a,+\infty}(\psi)$ it is easy to see that event $\uparrow_{\theta}$ may only occur at 0, if ψ eventually holds in the future after a instants from the origin. Similarly, event $\downarrow_{\theta}$ may only occur once, but not necessarily in the origin; more precisely, it holds at 0 if and only if $\uparrow_{\theta}$ does not hold at 0, while for every instant $t > 0$ it occurs when event $\downarrow_{\psi}$ occurs in $t + a$ and ψ is always false afterwards. As a consequence, z_{θ}^{1} is reset at most once, if $\urcorner\!\!\llcorner_{\theta}$ occurs in an instant other than the origin.

$$\lrcorner\!\!\ulcorner_{\theta} \Leftrightarrow O \wedge (O \vee z_{\theta}^{0} > 0)\,\mathbf{U}\left(\overleftarrow{\psi} \wedge \left(a \leqslant z_{\theta}^{0} \ \vee z_{\theta}^{0} < a \wedge \mathbf{X}\left(z_{\theta}^{0} > a\right)\right)\right) \tag{12}$$

$$\urcorner\!\!\llcorner_{\theta} \Leftrightarrow z_{\theta}^{1} = 0 \wedge \mathbf{X}\left(z_{\theta}^{1} > 0\,\mathbf{U}\left(\urcorner\!\!\llcorner_{\psi} \wedge z_{\theta}^{1} = a \ \wedge \mathbf{G}\left(\neg \lrcorner\!\!\ulcorner_{\psi}\right)\right)\right) \ \vee \ \left(O \wedge \neg \lrcorner\!\!\ulcorner_{\theta}\right) \tag{13}$$

$$\urcorner\!\!\llcorner_{\psi} \wedge \mathbf{G}\left(\neg \lrcorner\!\!\ulcorner_{\psi}\right) \Rightarrow z_{\theta}^{1} = a \tag{14}$$

3.3 Correctness

Let F be a set of formulae. We extend map r to $sub(\phi)$, written $r_{sub(\phi)}(M)$, to represent the set of CLTL-oc models where atomic propositions are symbols associated with each subformula in ϕ and variables includes all clocks $z_{\theta}^{0}, z_{\theta}^{1}$ and the auxiliary clocks for the case $\mathbf{F}_{\langle a,b]}$.

Lemma 6. *Let M be a signal, and ϕ a MITL formula. For any $(\pi,\sigma) \in r_{sub(\phi)}(M)$ it is:*

$$(\pi,\sigma),0 \models \bigwedge_{\theta \in sub(\phi)} \mathbf{G}(m(\theta)) \ \wedge \mathtt{events}_{\theta} \wedge \bigwedge_{\substack{\theta \in sub(\phi) \\ \theta = \mathbf{F}_{\langle a,b]}}} \mathtt{auxclocks}_{\theta}$$

and for all $k \in \mathbb{N}, \theta \in sub(\phi)$ it is $(\pi,\sigma),k \models m(\theta)$.

Lemma 7. *Let M be a signal and let ϕ be a MITL formula. If*

$$(\pi,\sigma),0 \models \bigwedge_{\theta \in sub(\phi)} \mathbf{G}(m(\theta)) \wedge \mathtt{events}_{\theta} \wedge \bigwedge_{\substack{\theta \in sub(\phi) \\ \theta = \mathbf{F}_{\langle a,b]}}} \mathtt{auxclocks}_{\theta}$$

and $M = r_{sub(\phi)}^{-1}((\pi,\sigma))$, then for all $t \in T$ it is $(\pi,\sigma),I(t) \models \lrcorner\!\!\ulcorner_{\phi}$ iff $M,t \models \uparrow_{\phi}$ (similarly for $\urcorner\!\!\llcorner_{\phi}$).

The main result, the equisatisfiability of MITL and of its CLTL-oc translation, follows.

Theorem 1. *A MITL formula ϕ is satisfiable if, and only if the following formula is satisfiable:*

$$\lrcorner\!\!\ulcorner_{\phi} \wedge \bigwedge_{\theta \in sub(\phi)} \mathbf{G}(m(\theta)) \wedge \mathtt{events}_{\phi} \wedge \bigwedge_{\substack{\theta \in sub(\phi) \\ \theta = \mathbf{F}_{\langle a,b]}}} \mathtt{auxclocks}_{\theta}. \tag{15}$$

3.4 Complexity

The reduction of MITL to CLTL-oc of Sect. 3.2 induces an EXPSPACE decision procedure for the satisfiability of MITL (the problem is actually EXPSPACE-complete). In fact, consider a MITL formula φ, and its CLTL-oc translation (15) obtained following the reduction of Sect. 3.2. In Formula (15) we introduce two clocks for each subformula of φ, unless the subformula is of the form $\mathbf{F}_{\langle a,b]}\psi$, in which case we introduce at most b clocks, since $a,b \in \mathbb{N}$. Then, the size of (15) is $O(|\varphi|K)$, where K is the maximum constant appearing in φ. It can be shown that satisfiability for a CLTL-oc formula ϕ_{CLTL} is PSPACE in the number of subformulae of ϕ_{CLTL} (which is $O(|\varphi|K)$ for Formula (15)) and in the size of the string encoding the maximum constant occurring in it (K for Formula (15)). Hence, the decision procedure induced by our encoding is in EXPSPACE when using a binary encoding of K. As remarked in [4], if the MITL formula φ does not contain subformulae of type $\mathbf{F}_{\langle a,b]}\psi$ (with $a > 0$ and $b \neq \infty$), the reduction of Sect. 3.2 only introduces one clock variable for each subformula. As a consequence, the size of Formula (15) is $O(|\varphi|)$ and the algorithm is in PSPACE.

4 Generalized translation

Our translation from MITL to CLTL-oc can be extended to represent general signals where no assumption is made on their shape, other than their finite variability, i.e., the l.c.r.o. assumption of Sect. 3 can be relaxed. In this more general case, the truth of a formula ϕ can change in a *singular* manner, that is, there can be instants where the value of ϕ is different than in a neighborhood thereof.

More precisely, we say that in a time instant t of a signal M formula ϕ has an "up-singularity" s^u_ϕ if it holds in t, but not before and after it; more precisely, we say that $M,t \models s^u_\phi$ if and only if $M,t \models \phi$ and $\exists \varepsilon > 0$ s.t. $\forall t' \neq t \in (t-\varepsilon, t+\varepsilon)$ it is $M,t' \not\models \phi$. We say that ϕ has a "down-singularity" s^d_ϕ when $\neg\phi$ has an up-singularity (i.e., ϕ does not hold in t, but it does before and after it). Note that, by their definition, singularities (either up or down), cannot occur in $t = 0$.

To represent general signals in CLTL-oc we "split" the representation of the value of subformulae θ in intervals $[t,t')$ in two parts: $\overset{\bullet}{|}_\theta$ captures the value of θ in t, whereas $\overleftarrow{\theta}$ corresponds to its value in (t,t'). With the new predicates, we can restrict represented signals to only include l.c.r.o. intervals by imposing the constraint $\overset{\bullet}{|}_\theta \Leftrightarrow \overleftarrow{\theta}$ for all θ. In addition, $\overset{\bullet\!-}{\theta}$ and $\underset{\bullet\!-}{\theta}$ become: $\overset{\bullet\!-}{\theta} = \overset{\bullet}{|}_\theta \wedge \overleftarrow{\theta}$ $\quad$ $\underset{\bullet\!-}{\theta} = \neg \overset{\bullet}{|}_\theta \wedge \neg \overleftarrow{\theta}$. Then, the encoding of Sect. 3 can be used also with the new atomic predicates, provided constraint $\overset{\bullet}{|}_\theta \Leftrightarrow \overleftarrow{\theta}$ is added for all subformulae. If, instead, general signals are to be allowed, the encoding must be extended to include also the cases in which the values of (sub)formulae change in singular manners.

To this end, we slightly modify the definition of $\lrcorner\!\ulcorner_\xi$ as $\neg\mathbf{Y}(\overleftarrow{\xi}) \wedge \overleftarrow{\xi}$ and $\urcorner\!\llcorner_\xi$ as $\neg\mathbf{Y}(\neg\overleftarrow{\xi}) \wedge \neg\overleftarrow{\xi}$ and we introduce the following abbreviations, which capture, respectively, up- and down-singularities (note that neither $\lrcorner\!\ulcorner\!\urcorner\!\llcorner_\xi$, nor $\urcorner\!\llcorner\!\lrcorner\!\ulcorner_\xi$ hold at 0, as $\mathbf{Y}(\bullet)$ is false there):

$$\lrcorner\!\ulcorner\!\urcorner\!\llcorner_\xi = \mathbf{Y}(\neg\overleftarrow{\xi}) \wedge \overset{\bullet}{|}_\xi \wedge \neg\overleftarrow{\xi} \qquad\qquad \urcorner\!\llcorner\!\lrcorner\!\ulcorner_\xi = \mathbf{Y}(\overleftarrow{\xi}) \wedge \neg \overset{\bullet}{|}_\xi \wedge \overleftarrow{\xi}$$

We also define the following: $\quad \overset{\xi}{\lrcorner\!\!\uparrow} = \lrcorner\!\ulcorner_\xi \vee \lrcorner\!\ulcorner\!\urcorner\!\llcorner_\xi \vee (O \wedge \overset{\bullet}{|}_\xi) \qquad \overset{\xi}{\hookrightarrow} = \urcorner\!\llcorner_\xi \vee \lrcorner\!\ulcorner\!\urcorner\!\llcorner_\xi$.

More precisely, $\overset{\xi}{\lrcorner\!\!\uparrow}$ corresponds to a situation where ξ does not hold the interval before the current one (if such interval exists), and it is true sometimes in the current one (either in its first instant, in which case ξ

can have a up-singularity, or in the rest of the interval). Dually, $\overset{\xi}{\hookrightarrow}$ holds if ξ is true in the first instant of the current interval, or in the interval before it, and from that moment on it is false.

When general signals are allowed, there is no need to restrict the temporal operators only to $\mathbf{F}_{\langle a,b]}(\psi)$. For simplicity, we focus on the encoding of case $\theta = \mathbf{F}_{(a,b)}(\psi)$, all other cases being similar.

- $\theta = \mathbf{F}_{(a,b)}\psi$: We have the following result.

Lemma 8. *If $\theta = \mathbf{F}_{(a,b)}\psi$ is a MITL formula and $M,t \models \theta$ then s $\exists\varepsilon \in \mathbb{R}_{>0}$ such that, for all $t' \in [t, t+\varepsilon]$ it is $M,t' \models \theta$ and, when $t > 0$, there is also $\varepsilon \in \mathbb{R}_{>0}$ such that $\varepsilon < t$ and for all $t' \in [t-\varepsilon, t]$ it is $M,t' \models \theta$.*

Because of Lemma 8, an up-singularity $\lrcorner\!\llcorner_\theta$ can never occur for $\theta = \mathbf{F}_{(a,b)}\psi$. In addition, if θ holds at the beginning of an interval (i.e., $\upharpoonleft_\theta$ holds), then it must hold also in the rest of the interval and, if $t > 0$, it must also hold in the interval before. Then, the following constraint holds in every instant:

$$\upharpoonleft_\theta \Rightarrow \overleftarrow{\theta} \wedge (\mathbf{Y}(\overleftarrow{\theta}) \vee O) \tag{16}$$

Formula (17) is similar to (5), but it specifies that, when θ becomes true outside of the origin, it must do so in a left-open manner (i.e., $\upharpoonleft_\theta$ does not hold with $\lrcorner\!\ulcorner_\theta$); also, there is one additional condition that makes θ become true in 0 when ψ becomes true exactly at b, in which case θ does not hold in 0.

$$\lrcorner\!\ulcorner_\theta \Leftrightarrow \begin{array}{c} \neg O \wedge \neg\upharpoonleft_\theta \wedge \bigvee_{j=0}^{d-1}(x_\theta^j = 0) \wedge \mathbf{X}\left(x_\theta^j > 0\,\mathbf{U}\left(\overset{\psi}{\lrcorner\!\uparrow} \wedge x_\theta^j = b \wedge \bigvee_{i=0}^{1} z_\psi^i > (b-a)\right)\right) \vee \\ O \wedge \neg\upharpoonleft_\theta \wedge \mathbf{X}\left(x_\theta^0 > 0\,\mathbf{U}\left(\overset{\psi}{\lrcorner\!\uparrow} \wedge x_\theta^0 = b \wedge \bigvee_{i=0}^{1} z_\psi^i \geqslant (b-a)\right)\right) \vee \\ O \wedge \upharpoonleft_\theta \wedge (O \vee x_\theta^0 > 0)\,\mathbf{U}\left((\upharpoonleft_\psi \vee \overleftarrow{\psi}) \wedge a < x_\theta^0 < b \quad \vee \quad \overleftarrow{\psi} \wedge x_\theta^0 < a \wedge \mathbf{X}\left(x_\theta^0 > a\right)\right) \end{array} \tag{17}$$

Formulae (18), (19) and (20) generalize, respectively, (6), (7) and (8) to include also the case in which ψ changes its value in a singular manner (i.e., with $\lrcorner\!\llcorner_\psi$ instead of $\lrcorner\!\ulcorner_\psi$ or $\urcorner\!\llcorner_\psi$).

$$\overset{\psi}{\lrcorner\!\uparrow} \wedge \bigvee_{i\in\{0,1\}} z_\psi^i \geqslant (b-a) \Rightarrow \bigvee_{j=0}^{d-1} x_\theta^j = b \tag{18}$$

$$\urcorner\!\llcorner_\theta \Leftrightarrow \bigvee_{j=0}^{d-1}(x_\theta^j = 0) \wedge \mathbf{X}\left((x_\theta^j > 0)\mathbf{U}\left(\overset{\psi}{\hookrightarrow} \wedge x_\theta^j = a \ \wedge \mathbf{X}\left(\overset{\psi}{\lrcorner\!\uparrow}\mathbf{R}\neg\left(\overset{\psi}{\lrcorner\!\uparrow} \wedge x_\theta^j \leqslant b\right)\right)\right)\right) \ \vee \ (O \wedge \neg\lrcorner\!\ulcorner_\theta) \tag{19}$$

$$\overset{\psi}{\hookrightarrow} \wedge \mathbf{X}\left(\overset{\psi}{\lrcorner\!\uparrow}\mathbf{R}\neg\left(\overset{\psi}{\lrcorner\!\uparrow} \wedge \bigwedge_{i=0}^{1} z_\psi^i \leqslant (b-a)\right)\right) \Rightarrow \bigvee_{j=0}^{d-1} x_\theta^j = a \tag{20}$$

Finally, we need to consider an additional shape in which θ can change value. More precisely, there is also the case in which θ becomes false with a down-singularity $\urcorner\!\ulcorner_\theta$. This occurs in an instant t (which must be > 0, as singularities cannot occur in the origin by definition) such that ψ becomes false at $t+a$, but it becomes true again at $t+b$ (and it stays false in interval $(t+a, t+b)$). This condition is captured by Formula (21), which is similar to Formula (19), except that it specifies that when ψ becomes true again, the clock x_θ^j that is reset when ϕ has the singularity has value b.

$$\urcorner\!\ulcorner_\theta \Leftrightarrow \neg O \wedge \bigvee_{j=0}^{d-1}(x_\theta^j = 0) \wedge \mathbf{X}\left((x_\theta^j > 0)\mathbf{U}\left(\overset{\psi}{\hookrightarrow} \wedge x_\theta^j = a \ \wedge \mathbf{X}\left(\overset{\psi}{\not\lrcorner\!\uparrow}\mathbf{U}\left(\overset{\psi}{\lrcorner\!\uparrow} \wedge x_\theta^j = b\right)\right)\right)\right) \tag{21}$$

Then, $m(\theta)$ is (16) $\wedge$ (17) $\wedge$ (18) $\wedge$ (19) $\wedge$ (20) $\wedge$ (21).

To allow for signals of general shape, the encoding for subformulae of the form $\gamma\mathbf{U}_{(0,+\infty)}\psi$ must also be revisited. As this is rather straightforward, we skip the details for reasons of brevity. Instead, we point out that it is possible to define a CLTL-oc encoding also for MITL *past* operators $\mathbf{S}$ and $\mathbf{P}_{\langle a,b\rangle}$. It is known that past operators increase the expressiveness of MITL [11], but do not impact on decidability. Hence, a decision procedure that also includes the possibility to handle past operators is more powerful than one dealing with the future-only fragment. To conclude this section, we show the encoding $m(\theta)$ for the $\mathbf{S}$ operator (whose semantics is symmetric to the one of $\mathbf{U}$ shown in Table 1). The case for operator $\mathbf{P}_{\langle a,b\rangle}$ is omitted for brevity.

- $\theta = \gamma\mathbf{S}_{(0,+\infty)}\psi$: In this case it can be shown that, if M is a finitely variable signal and θ holds in an instant t, then it must also hold in $(t-\varepsilon,t)$, for some $\varepsilon > 0$, and vice-versa. Then, in $t = 0$ θ is false, and there $\mathbf{S}$ formulae cannot have singularity points. In addition, when a $\mathbf{S}$ formula changes its value after the origin, it must do so in a left-open manner (i.e., the value at the changing point is the same as the one before the changing point). Then, we have

$$m(\theta) = (\uparrow_\theta \Leftrightarrow \mathbf{Y}(\overleftarrow{\theta})) \wedge (\overleftarrow{\theta} \Leftrightarrow \overset{\bullet\!-}{\gamma}\,\mathbf{S}((\uparrow_\psi \vee \overleftarrow{\psi}) \wedge \overleftarrow{\gamma})). \tag{22}$$

5 Conclusions

This paper investigates a bounded approach to satisfiability checking of the continuous-time temporal logic MITL. We showed an encoding of MITL into a decidable logic (CLTL-oc), which allows, both in principle and in practice, the use of SMT solvers to check satisfiability of MITL.

A decision procedure for CLTL-oc [10] is implemented in a plugin, called `ae`^{2}`zot`, of our Zot toolkit [2], whereas the reduction outlined in Sect. 3 and 4 is implemented in the `qtlsolver` tool, available from [1]. The tool translates MITL (or the expressively equivalent QTL logic [16]) into CLTL-oc, which can be checked for satisfiability by `ae`^{2}`zot`. The resulting toolkit has a 3-layered structure, where CLTL-oc is the intermediate layer between SMT-solvers and various temporal formalisms that can be reduced to CLTL-oc. This not only supports (bounded) satisfiability verification of different languages, but it also allows the expression of different degrees of abstraction. For instance, MITL abstracts away the notion of clocks, inherently encompassed within temporal modalities, which are instead explicit in CLTL-oc and actually available to a user, e.g., to express or verify properties where clocks are convenient. In fact, preliminary experimental results point out that the time required to solve CLTL-oc may be significantly smaller than the one needed for more abstract languages, such as MITL. This is caused by the "effort" required to capture the semantics of temporal modalities, which, on the other hand, allow for more concise and manageable high-level specifications. This layered structure also allows the resolution of a formula to be compliant with constraints imposed at lower layers, for instance by adding at the CLTL-oc layer some extra formula limiting the set of valid models (e.g., by discarding certain edges of some events or by adding particular timing requirements). Also the third layer (the SMT solver) may be used to add further constraints, e.g., to force the occurrence of a proposition or of a certain clock value at a specific discrete position of the finite model.

The current implementation of `qtlsolver` supports the MITL-to-CLTL-oc translation, both with or without the l.c.r.o. restriction. In fact, the following encodings are currently available:

MITL providing a direct definition of MITL operators, assuming l.c.r.o. intervals;

QTL providing the definition of generalized QTL operators (e.g., $\mathbf{F}_{(0,b)}$, $\mathbf{P}_{(0,b)}$) with unrestricted signals (other than they be finitely variable), and MITL operators through abbreviations.

We used the above two encodings to carry out some experiments (available from the `qtlsolver` website [1], or described in [10]). Let us illustrate one of them. MITL Formula (23) specifies that predicate p occurs in isolated points with a period of 100 (i.e., it occurs exactly at 0, 100, 200, etc.).

$$\mathbf{G}_{[0,\infty)}\big(\big(\mathbf{G}_{(0,100)}(\neg p) \Rightarrow \mathbf{G}_{(100,200)}(\neg p)\big) \wedge (p \Rightarrow \mathbf{F}_{(0,200)}(p))\big) \wedge p \wedge \mathbf{G}_{(0,100)}(\neg p) \tag{23}$$

`qtlsolver` was able to find a model for Formula (23) in around 10 seconds, using a bound of 10.[1] Note that, even if the constants appearing in Formula (23) are in the order of the hundreds, events in the corresponding models occur only sparsely, hence a bound of 10 is enough for `qtlsolver` to satisfy (23). If we add to the specification Formula (24), which states that q must hold within 1 time unit in the past or in the future of each p, the solver finds a model (again, with bound 10) in about 40 seconds.

$$\mathbf{G}_{(0,\infty)}\big(p \Rightarrow \mathbf{F}_{(0,1)}(q) \vee \mathbf{P}_{(0,1)}(q)\big) \tag{24}$$

Formula (24) does not impose that q be false in between occurrences of p. A more restricted behavior is obtained by adding also constraint (25), which imposes that q occurs only in isolated instants, and that there must be at least 100 time units between consecutive occurrences of q.

$$\mathbf{G}_{(0,\infty)}\big(q \Rightarrow \mathbf{G}_{(0,100)}(\neg q)\big) \tag{25}$$

`qtlsolver` was able to find a model (with bound 20, in this case) for formula (23) $\wedge$ (24) $\wedge$ (25) in around 10 minutes. As mentioned above, one can add constraints at different levels of abstraction. For example, we can add SMT constraints imposing that the *values* of the clocks (instead of the clock regions) associated with propositions p and q be periodic; this allows us to check that formula (23) $\wedge$ (24) $\wedge$ (25) admits periodic models (`qtlsolver` takes around 15 minutes to produce one with bound 20). Finally, if in Formula (25) we replace $\mathbf{G}_{(0,100)}$ with $\mathbf{G}_{(0,100]}$, the behavior becomes strictly aperiodic. In this case the solver takes around 80 minutes to find a model with bound 30, and in excess of 12 hours to show that, with that bound, no model exists in which p and q are periodic (i.e., that the specification, with the added constraint that the values of the clocks associated with p and q be periodic, is unsatisfiable).

While the results presented above are promising, further research will focus on optimizing the implementation of the solver and on extending the encoding to deal with richer constraints.

The techniques presented in this paper for MITL can be tailored also to other logics. We consider an example here. A syntactic fragment of MITL was proposed in [15], namely $\text{MTL}_{0,\infty}$, where temporal modalities are restricted only to intervals of the form $\langle 0,b\rangle$ or $\langle a,\infty)$ (e.g., the MITL formula $\mathbf{F}_{(2,3)}\phi$ is not acceptable). $\text{MTL}_{0,\infty}$ is complete in the sense that every MITL formula can be transformed into an equisatisfiable $\text{MTL}_{0,\infty}$ formula. However, the transformation may lead to an exponential blow-up, since satisfiability is EXPSPACE-complete for MITL and PSPACE-complete for $\text{MTL}_{0,\infty}$. In [15], $\text{MTL}_{0,\infty}$ was shown to be equivalent to a new temporal logic, called Event-Clock Logic (ECL), which is also in PSPACE. Although our work only concerns MITL (and actually $\text{MTL}_{0,\infty}$, which is considered by our translation provided that operator $\mathbf{F}_{\langle a,b]}$ is not primitive for the language), our results can directly be applied for solving the satisfiability of ($\text{MTL}_{0,\infty}$ and) ECL as well, by means of the above equivalence of the languages. However, an explicit encoding of ECL into CLTL-oc may be devised, since only a finite number of explicit clocks are enough to capture ECL semantics; this may allow solving satisfiability of both logics ($\text{MTL}_{0,\infty}$ and ECL) in PSPACE.

[1] All tests have been carried out on a desktop computer with a 2.8GHz AMD PhenomTMII processor and 8MB RAM; the solver was Microsoft Z3 3.2. The encoding used was the one for QTL, with unrestricted signals.

References

[1] *qtlsolver*. available from `qtlsolver.googlecode.com`.

[2] *Zot: a Bounded Satisfiability Checker*. available from `zot.googlecode.com`.

[3] Rajeev Alur & David L. Dill (1994): *A theory of timed automata*. Theor. Comp. Sci. 126(2), pp. 183–235. Available at `http://dx.doi.org/10.1016/0304-3975(94)90010-8`.

[4] Rajeev Alur, Tomás Feder & Thomas A. Henzinger (1996): *The Benefits of Relaxing Punctuality*. Journal of the ACM 43(1), pp. 116–146. Available at `http://doi.acm.org/10.1145/112600.112613`.

[5] Christel Baier & Joost-Pieter Katoen (2008): *Principles of Model Checking*. MIT Press.

[6] Johan Bengtsson & Wang Yi (2004): *Timed Automata: Semantics, Algorithms and Tools*. In: Lect. on Concurrency and Petri Nets, LNCS 3098, Springer, pp. 87–124. Available at `http://dx.doi.org/10.1007/978-3-540-27755-2_3`.

[7] Marcello M. Bersani, Achille Frigeri, Angelo Morzenti, Matteo Pradella, Matteo Rossi & Pierluigi San Pietro (2010): *Bounded Reachability for Temporal Logic over Constraint Systems*. In: TIME, IEEE Computer Society, pp. 43–50. Available at `http://dx.doi.org/10.1109/TIME.2010.21`.

[8] Marcello M. Bersani, Achille Frigeri, Angelo Morzenti, Matteo Pradella, Matteo Rossi & Pierluigi San Pietro (2012): *CLTL Satisfiability Checking without Automata*. arXiv:1205.0946v1.

[9] Marcello M. Bersani, Achille Frigeri, Matteo Rossi & Pierluigi San Pietro (2011): *Completeness of the Bounded Satisfiability Problem for Constraint LTL*. In: Reachability Problems, LNCS 6945, pp. 58–71. Available at `http://dx.doi.org/10.1007/978-3-642-24288-5_7`.

[10] Marcello M. Bersani, Matteo Rossi & Pierluigi San Pietro (2013): *A Tool for Deciding the Satisfiability of Continuous-time Metric Temporal Logic*. In: Proceedings of the International Symposium on Temporal Representation and Reasoning (TIME). To appear.

[11] Patricia Bouyer, Fabrice Chevalier & Nicolas Markey (2010): *On the expressiveness of TPTL and MTL*. Information and Computation 208(2), pp. 97 – 116. Available at `http://dx.doi.org/10.1016/j.ic.2009.10.004`.

[12] Stéphane Demri & Deepak D'Souza (2007): *An automata-theoretic approach to constraint LTL*. Information and Computation 205(3), pp. 380–415. Available at `http://dx.doi.org/10.1016/j.ic.2006.09.006`.

[13] Deepak D'Souza & Nicolas Tabareau (2004): *On Timed Automata with Input-Determined Guards*. In: Proc. of FORMATS/FTRTFT, LNCS 3253, Springer, pp. 68–83. Available at `http://dx.doi.org/10.1007/978-3-540-30206-3_7`.

[14] Carlo A. Furia, Dino Mandrioli, Angelo Morzenti & Matteo Rossi (2012): *Modeling Time in Computing*. EATCS Monographs in Theoretical Computer Science, Springer. Available at `http://dx.doi.org/10.1007/978-3-642-32332-4`.

[15] Thomas A. Henzinger, Jean F. Raskin & Pierre Y. Schobbens (1998): *The Regular Real-Time Languages*. In: Proc. of ICALP'98, LNCS 1343, pp. 580–591. Available at `http://dx.doi.org/10.1007/BFb0055086`.

[16] Yoram Hirshfeld & Alexander Moshe Rabinovich (2004): *Logics for Real Time: Decidability and Complexity*. Fundamenta Informaticae 62(1), pp. 1–28.

[17] Oded Maler, Dejan Nickovic & Amir Pnueli (2006): *From MITL to Timed Automata*. In: Proc. of FORMATS, LNCS 4202, pp. 274–289. Available at `http://dx.doi.org/10.1007/11867340_20`.

[18] Microsoft Research (2009): *Z3: An Efficient SMT Solver*. Available at: http://research.microsoft.com/en-us/um/redmond/projects/z3/.

[19] Angelo Morzenti & Pierluigi San Pietro (1994): *Object-Oriented Logical Specification of Time-Critical Systems*. ACM Transactions on Software Engineering and Methodology (TOSEM) 3(1), pp. 56–98. Available at `http://doi.acm.org/10.1145/174634.174636`.

[20] Matteo Pradella, Angelo Morzenti & Pierluigi San Pietro (2013): *Bounded Satisfiability Checking of Metric Temporal Logic Specifications*. ACM Trans. on Soft. Eng. and Meth. (TOSEM). To appear.

Improving HyLTL model checking of hybrid systems

Davide Bresolin
University of Verona (Italy)
davide.bresolin@univr.it

The problem of model-checking hybrid systems is a long-time challenge in the scientific community. Most of the existing approaches and tools are either limited on the properties that they can verify, or restricted to simplified classes of systems. To overcome those limitations, a temporal logic called HyLTL has been recently proposed. The model checking problem for this logic has been solved by translating the formula into an equivalent hybrid automaton, that can be analized using existing tools. The original construction employs a declarative procedure that generates exponentially many states upfront, and can be very inefficient when complex formulas are involved. In this paper we solve a technical issue in the construction that was not considered in previous works, and propose a new algorithm to translate HyLTL into hybrid automata, that exploits optimized techniques coming from the discrete LTL community to build smaller automata.

1 Introduction

Hybrid systems are heterogeneous systems characterized by a tight interaction between discrete and continuous components. Typical examples include discrete controllers that operate in a continuous environment, as in the case of manufacturing plants, robotic systems, and cyberphysical embedded systems. Because of their heterogeneous nature, hybrid systems cannot be faithfully modeled by discrete only nor by continuous only formalisms. In order to model and specify them in a formal way, the notion of *hybrid automata* has been introduced [1, 14]. Intuitively, a hybrid automaton is a "finite-state automaton" with continuous variables that evolve according to dynamics characterizing each discrete state (called a *location* or *mode*). Of particular importance in the analysis of hybrid automata is the *model checking problem*, that is, the problem of verifying whether a given hybrid automaton respects some property of interest. Unfortunately, the model checking problem is computationally very difficult. Indeed, even for simple properties and systems, this problem is not decidable [11].

For very simple classes of hybrid systems, like timed automata, the model checking problem can be solved exactly [2]. Tools like Kronos [20] and UPPAAL [13] can be used to verify properties of timed automata. For more complex classes of systems, the problem became undecidable, and many different approximation techniques may be used to obtain an answer, at least in some cases. Tools like PhaVer [8] and SpaceEx [9] can compute approximations of the reachable set of hybrid automata with linear dynamics, and thus can be used to verify safety properties. Other tools, like HSOLVER [17], and Ariadne [4], can manage systems with nonlinear dynamics, but are still limited to safety properties.

We are aware of only very few approaches that can specify and verify complex properties of hybrid systems in a systematic way. A first attempt was made in [12], where an extension of the Temporal Logic of Actions called TLA+ is used to specify and implement the well-known gas burner example. Later on, Signal Temporal Logic (STL), an extension of the well-known Metric Interval Logic to hybrid traces, has been introduced to monitor hybrid and continuous systems [15]. More recent approaches include the tool KeYmaera [16], that uses automated theorem proving techniques to verify nonlinear hybrid systems symbolically, and the logic HRELTL [6], that is supported by an extension of the discrete model checker NuSMV, but it is limited to systems with linear dynamics.

To overcome the limitations of the current technologies, an automata-theoretic approach for model checking hybrid systems has been recently proposed [5]. The work is based on an extension of the well-known temporal logic LTL to hybrid traces called HyLTL. The model checking problem for this logic has been solved by translating the formula into an equivalent hybrid automaton, reducing the model checking problem to a reachability problem that can be solved by existing tools. The original construction employs a declarative procedure that generates exponentially many states upfront, and can be very inefficient when complex formulas are involved.

In this paper we solve a technical issue in the construction that was not considered in previous works by identifying the precise fragment of HyLTL that can be translated into hybrid automata, and we propose a new algorithm to translate formulas into hybrid automata, that exploits optimized techniques coming from the discrete LTL community to be more efficient than the original declarative approach.

2 Preliminaries

Before formally defining hybrid automata and the syntax and semantics of HyLTL we need to introduce some basic terminology. Throughout the paper we fix the *time axis* to be the set of non-negative real numbers $\mathbb{R}^+$. An *interval* I is any convex subset of $\mathbb{R}^+$, usually denoted as $[t_1,t_2] = \{t \in \mathbb{R}^+ : t_1 \leq t \leq t_2\}$. We also fix a countable universal set $\mathcal{V}$ of *variables*, ranging over the reals. Given a finite set of variables $X \subseteq \mathcal{V}$, a *valuation* over X is a function $\mathbf{x} : X \mapsto \mathbb{R}^n$ that associates a value to every variable in X. The set $\mathrm{Val}(X)$ is the set of all valuations over X.

A notion that will play an important role in the paper is the one of *trajectory*. A trajectory over a set of variables X is a function $\tau : I \mapsto \mathrm{Val}(X)$, where I is a left-closed interval with left endpoint equal to 0. We assume trajectories to be differentiable almost everywhere on the domain, and we denote with $\dot{\tau}$ the corresponding (partial) function giving the value of the derivative of τ for every point in the interior of I where τ is differentiable (note that $\dot{\tau}$ might not be differentiable neither continuous). With $\mathrm{dom}(\tau)$ we denote the domain of τ, while with $\tau.\mathit{ltime}$ (the *limit time* of τ) we define the supremum of $\mathrm{dom}(\tau)$. The *first state* of a trajectory is $\tau.\mathit{fstate} = \tau(0)$, while, when $\mathrm{dom}(\tau)$ is right-closed, the *last state* of a trajectory is defined as $\tau.\mathit{lstate} = \tau(\tau.\mathit{ltime})$. We denote with $\mathit{Trajs}(X)$ the set of all trajectories over X. If $[t,t']$ is a subinterval of $\mathrm{dom}(\tau)$, we denote whith $\tau\downarrow_{[t,t']}$ the trajectory τ' such that $\mathrm{dom}(\tau') = [0,t'-t]$ and $\tau'(t'') = \tau(t''+t)$ for every $t'' \in \mathrm{dom}(\tau')$. Given two trajectories τ_1 and τ_2 such that $\tau_1.\mathit{ltime} < +\infty$, their concatenation $\tau_1 \cdot \tau_2$ is the trajectory with domain $[0, \tau_1.\mathit{ltime} + \tau_2.\mathit{ltime}]$ such that $\tau_1 \cdot \tau_2(t) = \tau_1(t)$ if $t \in \mathrm{dom}(\tau_1)$, $\tau_1 \cdot \tau_2(t) = \tau_2(t - \tau_1.\mathit{ltime})$ otherwise.

Variables will be used in the paper to build *constraints*: conditions on the value of variables and on their derivative that can define sets of valuations, sets of trajectories, and jump relations. Formally, given a set of variables X, and a set of mathematical operators OP (e.g. $+$, $-$, $\cdot$, exponentiation, sin, cos, ...), we define the corresponded set of *dotted variables* $\dot{X}$ as $\{\dot{x} | x \in X\}$ and the set of *tilde variables* $\tilde{X}$ as $\{\tilde{x} | x \in X\}$. We use OP, X, $\dot{X}$ and $\tilde{X}$ to define the following two classes of constraints.

- *Jump constraints*: expressions built up from variables in $X \cup \tilde{X}$, constants from $\mathbb{R}$, mathematical operators from OP and the usual equality and inequality relations ($\leq$, $=$, $>$, ...). Examples of jump constraints are $x = 4\tilde{y} + \tilde{z}$, $x^2 \leq \tilde{y}$, $\tilde{y} > \cos(y)$.
- *Flow constraints*: expressions built up from variables in $X \cup \dot{X}$, constants from $\mathbb{R}$, mathematical operators from OP and the usual equality and inequality relations ($\leq$, $=$, $>$, ...). Examples of flow constraints are $\dot{x} = 4y + z$, $\dot{x} + y \geq 0$, $\sin(x) > \cos(\dot{y})$.

We use jump constraints to give conditions on pairs of valuations $(\tilde{\mathbf{x}}, \mathbf{x})$. Given a jump constraint c, we say that $(\tilde{\mathbf{x}}, \mathbf{x})$ respects c, and we denote it with $(\tilde{\mathbf{x}}, \mathbf{x}) \vdash c$, when, by replacing every variable x with its value in $\mathbf{x}$ and every tilde variable $\tilde{x}$ with the value of the corresponding normal variable in $\tilde{\mathbf{x}}$ we obtain a solution for c. Flow constraints will be used to give conditions on trajectories. Given a flow constraint c, we say that a trajectory τ respects c, and we denote it with $\tau \vdash c$, if and only if for every time instant $t \in \mathrm{dom}(\tau)$, both the value of the trajectory $\tau(t)$ and the value of its derivative $\dot{\tau}(t)$ respect c (we assume that $\dot{\tau}(t)$ respects c when $\dot{\tau}$ is not defined on t).

3 HyLTL: syntax and semantics

The logic HyLTL is an extension of the well-known temporal logic LTL to hybrid systems. Given a *finite* set of actions A and a *finite* set of variables X, the language of HyLTL is defined from a set of *flow constraints FC* over X by the following grammar:

$$\varphi ::= f \in FC \mid a \in A \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \varphi\,\mathbf{R}\,\varphi \tag{1}$$

In HyLTL constraints from FC and actions from A take the role of propositional letters in standard temporal logics, $\neg$, $\wedge$ and $\vee$ are the usual boolean connectives, $\mathbf{X}$, $\mathbf{U}$ and $\mathbf{R}$ are hybrid counterpart of the standard *next*, *until* and *release* temporal operators.

The semantics of HyLTL is given in terms of *hybrid traces* mixing continuous trajectories with discrete events. Formally, given a set of actions A and a set of variables X, an *hybrid trace over A and X* is any infinite sequence $\alpha = \tau_1 a_1 \tau_2 a_2 \tau_3 a_3 \ldots$ such that τ_i is a trajectory over X and a_i is an action in A for every $i \geq 1$. For every $i > 0$, the truth value of a HyLTL formula φ over α at position i is given by the truth relation $\Vdash$, formally defined as follows:

- for every $f \in FC$, $\alpha, i \Vdash f$ if and only if $\tau_i \vdash f$;
- for every $a \in A$, $\alpha, i \Vdash a$ iff $i > 1$ and $a_{i-1} = a$;
- $\alpha, i \Vdash \neg\varphi$ if and only if $\alpha, i \nVdash \varphi$;
- $\alpha, i \Vdash \varphi \wedge \psi$ if and only if $\alpha, i \Vdash \varphi$ and $\alpha, i \Vdash \psi$;
- $\alpha, i \Vdash \varphi \vee \psi$ if and only if $\alpha, i \Vdash \varphi$ or $\alpha, i \Vdash \psi$;
- $\alpha, i \Vdash \mathbf{X}\varphi$ if and only if $\alpha, i+1 \Vdash \varphi$;
- $\alpha, i \Vdash \varphi\,\mathbf{U}\,\psi$ if and only if there exists $j \geq i$ such that $\alpha, j \Vdash \psi$, and for every $i \leq k < j$, $\alpha, k \Vdash \varphi$;
- $\alpha, i \Vdash \varphi\,\mathbf{R}\,\psi$ if and only if for all $j \geq i$, if for every $i \leq k < j$, $\alpha, k \nVdash \varphi$ then $\alpha, j \Vdash \psi$.

Other temporal operators, such as the "always" operator $\mathbf{G}$ and the "eventually" operator $\mathbf{F}$ can be defined as usual:

$$\mathbf{F}\varphi = \top\,\mathbf{U}\,\varphi \qquad\qquad \mathbf{G}\varphi = \neg\,\mathbf{F}\,\neg\varphi$$

3.1 HyLTL with positive constraints

In this paper we will pay a special attention on formulas of HyLTL where flow constraints from FC appears only in positive form, because it will turn out that they constitue the class of formulas that can be

when $a \in A$:	$\pi(a) = a$	$\pi(\neg a) = \neg a$
when $f \in FC$:	$\pi(f) = f \wedge \mathbf{X}((T \wedge f)\,\mathbf{U}\,\neg T)$	$\pi(\neg f) = \bar{f} \vee \mathbf{X}(T\,\mathbf{U}\,(T \wedge \bar{f}))$
	$\pi(\varphi \wedge \psi) = \pi(\varphi) \wedge \pi(\psi)$	$\pi(\varphi \vee \psi) = \pi(\varphi) \vee \pi(\psi)$
	$\pi(\varphi\,\mathbf{U}\,\psi) = (T \vee \pi(\varphi))\,\mathbf{U}\,(\neg T \wedge \pi(\psi))$	$\pi(\varphi\,\mathbf{R}\,\psi) = (\neg T \wedge \pi(\varphi))\,\mathbf{R}\,(T \vee \pi(\psi))$
	$\pi(\mathbf{X}\varphi) = \mathbf{X}(T\,\mathbf{U}\,(\neg T \wedge \pi(\varphi)))$	

Table 1: The translation function π from HyLTL to HyLTL$^+$

translated into hybrid automata. This particular fragment is called *HyLTL with positive flow constraints*, denoted by HyLTL$^+$, and formally defined by the following grammar:

$$\psi ::= f \in FC \mid a \in A \mid \neg a \in A \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\,\mathbf{U}\,\psi \mid \psi\,\mathbf{R}\,\psi \qquad (2)$$

Despite being a syntactical fragment, HyLTL$^+$ turns out to be equally expressive as the full language, at the price of adding an auxiliary action symbol. In the following, given a constraint c we denote with $\bar{c}$ the corresponding "dual" constraint obtained by replacing $<$ with $\geq$, $>$ with $\leq$, $=$ with $\neq$, and so on. Notice that a trajectory τ that satisfies the negation of a flow constraint $\neg c$ does not necessarily satisfy $\bar{c}$. Indeed, by the semantics of HyLTL we have that $\tau \vdash \neg c$ if *there exists* a time instant t such that $\tau(t) \not\vdash c$, while $\tau \vdash \bar{c}$ if *for all* time instants t we have that $\tau(t) \not\vdash c$.

Hence, given a trajectory τ with domain $\mathrm{dom}(\tau) = [0, t_{max}]$ such that $\tau \vdash \neg c$, it is possible to find a point $t \in [0, t_{max}]$ such that $\tau(t) \not\vdash c$ and we can split τ into three sub-trajectories τ_b, $\tau_{\bar{c}}$, τ_e such that $\tau_b = \tau{\downarrow_{[0,t]}}$, $\tau_{\bar{c}} = \tau{\downarrow_{[t,t]}}$ and $\tau_e = \tau{\downarrow_{[t,t_{max}]}}$: it is easy to see that $\tau_{\bar{c}} \vdash \bar{c}$. In the following, the auxiliary action symbol T will be used to represent the splitting points of trajectories when translating formulas with negated flow constraints to formulas with positive flow constraints only.

Given a formula of HyLTL in *in negated normal form* φ, consider the translation function π defined in Table 1. To compare hybrid traces satisfying the original formula φ with the ones satisfying $\pi(\varphi)$ we have to remove the occurrences of T from the latter. To this end, we define a suitable restriction operator over hybrid traces.

Definition 1. *Let A a set of action, and $B \subset A$. Given a hybrid trace $\alpha = \tau_1 a_2 \tau_2 a_2 \ldots$ over A we define its* restriction *to B as the hybrid trace $\alpha{\downarrow_B}$ obtained from α by first removing the actions not in B and then concatenating adjacent trajectories.*

The following lemma states that $\pi(\varphi)$ is a formula of HyLTL$^+$ equivalent to φ.

Lemma 1. *For every hybrid trace α over A and X and every HyLTL-formula φ we have that $\alpha, 1 \Vdash \varphi$ if and only if there exists a hybrid trace β over $A \cup \{T\}$ and X such that $\beta{\downarrow_A} = \alpha$ and $\beta, 1 \Vdash \pi(\varphi)$.*

Proof. Let $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ be an hybrid trace over A such that $\alpha, 1 \Vdash \varphi$, and let FC be the set of flow constraints that appears in φ. We will build a sequence of hybrid traces $\beta_0, \beta_1, \beta_2, \ldots$ over $A \cup \{T\}$ as follows.

1. β_0 is the empty sequence.
2. For every $i \geq 1$, consider the i-th trajectory τ_i in α, and let $C_i = \{f \in FC \mid \tau_i \not\vdash f\}$. Given an enumeration $f_1, \ldots, f_n$ of C_i, we have that it is possible to find a set of time instants $t_1, \ldots, t_n$ such that $\tau_i(t_j) \vdash \bar{f}$ for every $1 \leq j \leq n$. W.l.o.g., we can assume that $\tau_i.\mathit{ftime} = t_0 \leq t_1 \leq t_2 \leq \ldots \leq t_n \leq t_{n+1} = \tau_i.\mathit{ltime}$ and we can define the sequence of trajectories $\mu_1, \mu_2, \ldots, \mu_{2n+1}$ such that

$$\mu_1 = \tau_i{\downarrow_{[t_0,t_1]}}, \qquad \mu_{2j} = \tau_i{\downarrow_{[t_j,t_j]}}, \qquad \mu_{2j+1} = \tau_i{\downarrow_{[t_j,t_{j+1}]}} \qquad \text{for every } 1 \leq j \leq n \qquad (3)$$

We define $\beta_i = \beta_{i-1} \mu_1 T \mu_2 T \ldots T \mu_{2n+1} a_i$.

The hybrid trajectory we are looking for is the limit trajectory $\beta = \lim_{i\to\infty}\beta_i$.

Given an index i, we will denote by α^i and β^i the suffix of α and of β starting at position i. We show that β respects the following property: *"for every subformula ψ of φ and $i \geq 1$, $\alpha, i \Vdash \psi$ iff $\beta, j \Vdash \pi(\psi)$, where j is the unique index such that $\beta^j{\downarrow_A} = \alpha^i$"*. The proof is by induction on ψ.

- If $\psi = a$ or $\psi = \neg a$ for some $a \in A$ the property holds trivially.
- Suppose $\psi = f$ for some $f \in FC$. By the semantics, we have that $\tau_i \vdash f$. Consider now the sequence $\mu_1 T \mu_2 T \ldots T \mu_{2n+1} a_i$ built in the construction of β_i, and let j be the index of μ_1 in β. By (3) we have that $\mu_h \vdash f$ for every $1 \leq h \leq 2n+1$. This implies that $\beta, j \Vdash f \wedge \mathbf{X}((T \wedge f)\,\mathbf{U}\,\neg T)$.
- If $\psi = \neg f$ for some $f \in FC$ then we have that $\tau_i \not\vdash f$. Let $\mu_1 T \mu_2 T \ldots T \mu_{2n+1} a_i$ be the sequence built in the construction of β_i. Since $f \in C_i$, we have that there exists $t_0 \leq t_k \leq t_{n+1}$ such that $\tau_i(t_k) \vdash \bar{f}$. By (3), this implies that $\mu_k \vdash \bar{f}$. Let j be the index of μ_1 in β. Two case may arise: either $\mu_k = \mu_1$ and thus $\beta, j \Vdash \bar{f}$, or $\mu_k \neq \mu_1$ and then $\beta, j \Vdash \mathbf{X}(T\,\mathbf{U}\,(T \wedge \bar{f}))$. In both cases the property is satisfied.
- The cases of the boolean operators $\vee$ and $\wedge$ are trivial and can be skipped.
- Suppose $\psi = \psi_1\,\mathbf{U}\,\psi_2$, and let i be such that $\alpha, i \Vdash \psi_1\,\mathbf{U}\,\psi_2$. By the semantics, we have that there exists $k \geq i$ such that $\alpha, k \Vdash \psi_2$ and, for every $i \leq h < k$, $\alpha, h \Vdash \psi_1$. Now, let j and l be the two indexes such that $\beta^j{\downarrow_A} = \alpha^i$ and $\beta^l{\downarrow_A} = \alpha^k$. By inductive hypothesis we can assume that $\beta, l \Vdash \pi(\psi_2)$, while by the definition of the ${\downarrow_A}$ operator we have that $\beta, l \Vdash a_i \neq T$. Hence, $\beta, l \Vdash \neg T \wedge \pi(\psi_2)$. Consider now any index m such that $j \leq m < l$. Two cases may arise: either $\beta, m \Vdash T$, or not. In the latter case, we have that it is possible to find an index $i \leq h < k$ such that $\beta^m{\downarrow_A} = \alpha^h$. Since $\alpha, h \Vdash \psi_1$, by inductive hypothesis we have that $\beta, m \Vdash \pi(\psi_1)$. Hence, in both cases $\beta, m \Vdash T \vee \pi(\psi_1)$. This proves that $\beta, j \Vdash (T \vee \pi(\psi_1))\,\mathbf{U}\,(\neg T \wedge \pi(\psi_2)) = \pi(\psi)$.
  To prove the converse implication, suppose that $\beta, j \Vdash (T \vee \pi(\psi_1))\,\mathbf{U}\,(\neg T \wedge \pi(\psi_2))$. By the semantics, we have that there exists $l \geq j$ such that $\beta, l \Vdash \neg T \wedge \pi(\psi_2)$ and, for every $j \leq m < l$, $\beta, m \Vdash T \vee \pi(\psi_1)$. Since $\beta, l \Vdash \neg T$ it is possible to find an index k such that $\beta^l{\downarrow_A} = \alpha^k$. Hence, by inductive hypothesis we have that $\alpha, k \Vdash \psi_2$. Now, let h be such that $i \leq h < k$, and consider the index m such that $\beta^m{\downarrow_A} = \alpha^h$. By the semantics we have that $\beta, m \Vdash T \vee \pi(\psi_1)$. Since, by definition of the restriction operator, $\beta, m \nVdash T$, we have that $\beta, m \Vdash \pi(\psi_1)$ and thus, by inductive hypothesis, that $\alpha, h \Vdash \psi_1$. This proves that $\alpha, i \Vdash \psi_1\,\mathbf{U}\,\psi_2$.
- The cases of the temporal operators $\mathbf{X}$ and $\mathbf{R}$ can be proved by a similar argument.

By the property it is immediate to conclude that, since $\alpha, 1 \Vdash \varphi$ then $\beta, 1 \Vdash \pi(\varphi)$.

To conclude the proof, suppose that there exists a hybrid trace β such that $\beta, 1 \Vdash \pi(\varphi)$, and let $\alpha = \beta{\downarrow_A}$. By an induction on the structure of φ similar to the one above, we can prove that $\alpha, 1 \Vdash \varphi$. □

4 Hybrid Automata

An hybrid automaton is a finite state machine enriched with continuous dynamics labelling each discrete state (or *location*), that alternates continuous and discrete evolution. In continuous evolution, the discrete state does not change, while time passes and the evolution of the continuous state variables follows the dynamic law associated to the current location. A discrete evolution step consists of the activation of a *discrete transition* that can change both the current location and the value of the state variables, in accordance with the reset function associated to the transition.

In this section we recap the definition of Hybrid Automata introduced in [5] to solve the model checking problem for HyLTL.

Definition 2. *A* hybrid automaton *is a tuple* $\mathcal{H} = \langle \mathrm{Loc}, X, A, \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Res}, \mathrm{Init}\rangle$ *such that:*

1. Loc *is a finite set of* locations*;*
2. *X is a finite set of* variables*;*
3. *A is a finite set of* actions*;*
4. $\mathrm{Edg} \subseteq \mathrm{Loc} \times A \times \mathrm{Loc}$ *is a set of* discrete transitions*;*
5. Dyn *is a mapping that associates to every location* $\ell \in \mathrm{Loc}$ *a set of flow constraints* $\mathrm{Dyn}(\ell)$ *over* $X \cup \dot{X}$ *describing the* dynamics *of* ℓ*;*
6. Res *is a mapping that associates every discrete transition* $(\ell, e, \ell') \in \mathrm{Edg}$ *with a set of jump constraints* $\mathrm{Res}(\ell, e, \ell')$ *over* $\tilde{X} \cup X$ *describing the guard and reset function of the transition;*
7. $\mathrm{Init} \subseteq \mathrm{Loc}$ *is a set of* initial locations*.*

The *state* of a hybrid automaton $\mathcal{H}$ is a pair $(\ell, \mathbf{x})$, where $\ell \in \mathrm{Loc}$ is a location and $\mathbf{x} \in \mathrm{Val}(X)$ is a valuation for the continuous variables. A state $(\ell, \mathbf{x})$ is said to be *admissible* if $(\ell, \mathbf{x}) \vdash \mathrm{Dyn}(\ell)$. Transitions can be either *continuous*, capturing the continuous evolution of the state, or *discrete*, capturing instantaneous changes of the state.

Definition 3. *Let $\mathcal{H}$ be a hybrid automaton. The* continuous transition relation $\xrightarrow{\tau}$ *between admissible states, where τ is a bounded trajectory over X, is defined as follows:*

$$(\ell, \mathbf{x}) \xrightarrow{\tau} (\ell, \mathbf{x}') \iff \tau.\mathit{fstate} = \mathbf{x} \wedge \tau.\mathit{lstate} = \mathbf{x}' \wedge \tau \vdash \mathrm{Dyn}(\ell). \tag{4}$$

The discrete transition relation $\xrightarrow{a}$ *between admissible states, where $a \in A$, is defined as follows:*

$$(\ell, \mathbf{x}) \xrightarrow{a} (\ell', \mathbf{x}') \iff \mathbf{x} \vdash \mathrm{Dyn}(\ell) \wedge \mathbf{x}' \vdash \mathrm{Dyn}(\ell') \wedge (\mathbf{x}, \mathbf{x}') \vdash \mathrm{Res}(\ell, a, \ell'). \tag{5}$$

The above definitions allows an infinite sequence of discrete events to occur in a finite amount of time (Zeno behaviors). Such behaviors are physically meaningless, but very difficult to exclude completely from the semantics. In this paper we assume that all hybrid automata under consideration do not generate Zeno runs. This can be achieved, for instance, by adding an extra clock variable that guarantees that the delay between any two discrete actions is bounded from below by some constant. Moreover, we assume that all hybrid automata are *progressive*, that is, that all runs can be extended to an infinite one: it is not possible to stay forever in a location and never activate a new discrete action.

We can view progressive, non-Zeno hybrid automata as *generators* of hybrid traces, as formally expressed by the following definition.

Definition 4. *Let $\mathcal{H}$ be a progressive, non-Zeno hybrid automaton, and let $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ be a infinite hybrid trace over X and A. We say that α is* generated *by $\mathcal{H}$ if there exists a corresponding sequence of locations $\ell_1 \ell_2 \ldots$ such that $\ell_1 \in$* Init *and, for every $i \geq 1$: (i) $(\ell_i, \tau_i.\mathit{fstate}) \xrightarrow{\tau_i} (\ell_i, \tau_i.\mathit{lstate})$, and (ii) $(\ell_i, \tau_i.\mathit{lstate}) \xrightarrow{a_i} (\ell_{i+1}, \tau_{i+1}.\mathit{fstate})$.*

Our definition of hybrid automata admits composition, under the assumpion that all variables and actions are shared between the different automata. The formal definition of the parallel composition operator $||$ can be found in [5]. In this paper it is sufficient to recall that it respects the usual "compositionality property", that is, that the set of hybrid traces generated by a composition of hybrid automata corresponds to the intersection of the hybrid traces generated by the components (up to projection to the correct set of actions and variables).

5 Model checking HyLTL

In analogy with the classical automata-theoretic approach, in [5] the model checking problem for HyLTL has been solved by translating the HyLTL formula into an equivalent hybrid automaton, enriched with a suitable *Büchi acceptance condition* to identify the traces generated by the automaton that fulfills the semantics of HyLTL.

Definition 5. *A* Hybrid Automaton with Büchi condition (BHA) *is a tuple* $\mathcal{H} = \langle \text{Loc}, X, A, \text{Edg}, \text{Dyn}, \text{Res}, \text{Init}, \mathcal{F} \rangle$ *such that* $\langle \text{Loc}, X, A, \text{Edg}, \text{Dyn}, \text{Res}, \text{Init} \rangle$ *is a Hybrid Automaton, and* $\mathcal{F} \subseteq \text{Loc}$ *is a finite set final locations.*

We say that a hybrid trace $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ is *accepted* by a BHA $\mathcal{H}$ if there exists an *infinite* sequence of locations $\ell_1 \ell_2 \ldots$ such that:
(i) $\ell_1 \in \text{Init}$;
(ii) for every $i \geq 1$, $(\ell_i, \tau_i.\mathit{fstate}) \xrightarrow{\tau_i} (\ell_i, \tau_i.\mathit{lstate})$;
(iii) for every $i \geq 1$, $(\ell_i, \tau_i.\mathit{lstate}) \xrightarrow{a_i} (\ell_{i+1}, \tau_{i+1}.\mathit{fstate})$;
(iv) there exists $\ell_f \in \mathcal{F}$ that occurs infinitely often in the sequence.
By the above definition, not all sequences generated by the automaton are accepting: only those that respect the additional accepting condition are considered.

By the definition of the dynamics, hybrid automata can enforce only *positive constraints* on the continuous flow of the system. Hence, they can only recognize formulas of the positive flow fragment of HyLTL, as summarized by the following theorem.

Theorem 1 ([5]). *Given a* HyLTL$^+$ *formula* φ*, it is possible to build a BHA* $\mathcal{H}_\varphi$ *that accepts all and only those hybrid traces that satisfies* φ*.*

Theorem 1 and Lemma 1 can be exploited to solve the model checking problem for full HyLTL as follows. Let $\mathcal{H}_S$ be a hybrid automaton representing the system under verification, and let φ be the HyLTL formula representing a property that the system should respect. Consider the formula $\neg\varphi$ and its translation $\overline{\varphi} = \pi(\neg\varphi)$. By Lemma 1 we have that $\overline{\varphi}$ is a formula of HyLTL$^+$ that is equivalent to $\neg\varphi$, and thus we can build a BHA $\mathcal{H}_{\overline{\varphi}}$ that is equivalent to *the negation of the property*: it accepts all the hybrid traces that *violates* the property we want to verify. Now, if we compose the automaton for the system with the automaton for $\overline{\varphi}$ we obtain a BHA $\mathcal{H}_S \| \mathcal{H}_{\overline{\varphi}}$ that accepts only those hybrid traces that are generated by the system and violates the property. This means that $\mathcal{H}_S$ respects the property φ if and only $\mathcal{H}_S \| \mathcal{H}_{\overline{\varphi}}$ *does not accept any hybrid trace*.

It is worth pointing out that the reachability problem of hybrid automata is undecidable. This means that the model checking of HyLTL is an undecidable problem as well (reachability can be expressed by an eventuality formula). However, this does not mean that out logic is completely intractable. A number of different approximation techniques have been developed in the past years to obtain an answer to the reachability problem (at least in some cases), and they can be exploited to solve the model checking problem of HyLTL as well. Indeed, $\mathcal{H}_S \| \mathcal{H}_{\overline{\varphi}}$ accepts an hybrid trace if and only if there exists a loop that includes a final location and that is reachable from the initial states. As shown in [5], this property can be reduced to a reachability property that can be tested by existing tools for the analysis of hybrid automata. The only thing that one needs to do is to write a procedure implementing the construction of $\mathcal{H}_{\overline{\varphi}}$, and then send the results to the reachability analysis tool.

6 An improved construction algorithm

The algorithm presented in [5] to build a BHA equivalent to a HyLTL$^+$-formula φ is based on a declarative construction. While being simple to understand, it suffers of a major drawback from the efficiency point of view: it generates exponentially many locations upfront, even though many of them may be inconsistent, redundant or unreachable. This implies that the resulting BHA can be very big, even for very

$\gamma(\varphi) = \bigwedge_{i=0}^{n-1} \neg b_i \wedge \gamma_0(\varphi)$		
$\gamma_0(a) = \mathbf{b}(a)$	$\gamma_0(f) = f$	when $a \in A$ or $f \in FC$
$\gamma_0(\neg\varphi) = \neg\gamma_0(\varphi)$	$\gamma_0(\varphi \wedge \psi) = \gamma_0(\varphi) \wedge \gamma_0(\psi)$	$\gamma_0(\varphi \vee \psi) = \gamma_0(\varphi) \vee \gamma_0(\psi)$
$\gamma_0(\mathbf{X}\varphi) = \mathfrak{X}(\gamma_0(\varphi))$	$\gamma_0(\varphi\, \mathbf{U}\, \psi) = \gamma_0(\varphi)\, \mathfrak{U}\, \gamma_0(\psi)$	$\gamma_0(\varphi\, \mathbf{R}\, \psi) = \gamma_0(\varphi)\, \mathfrak{R}\, \gamma_0(\psi)$

Table 2: The translation function γ from HyLTL to LTL

simple formulas. In this section we describe an improved construction algorithm, based on the following steps:

A. the HyLTL$^+$-formula φ is first translated into a suitable formula of discrete LTL $\gamma(\varphi)$;

B. a discrete Büchi automaton $\mathcal{A}_{\gamma(\varphi)}$, equivalent to $\gamma(\varphi)$, is built using one of the many optimized tools available in the literature;

C. a BHA $\mathcal{H}_\varphi$, equivalent to φ, is built from $\mathcal{A}_{\gamma(\varphi)}$.

The new algorithm improves the original one by building a smaller BHA, thanks to the use of optimized tools for LTL in step B.

6.1 From HyLTL to discrete LTL

Let FC and A be respectively the set of all flow constraints and discrete actions appearing in φ. For the sake of simplicity, we will assume that $\|A \cup \{T\}\| = 2^n - 1$ for some $n \in \mathbb{N}$ (if this is not the case, we can always add some fresh action symbols to A that will not appear in the formula). Under this assumption we can represent action symbols from $A \cup \{T\}$ by means of a set of n propositional letters $B = \{b_0, \dots, b_{n-1}\}$, where every possible combination of the truth values, but the one where all letters are false, uniquely identify one action symbol. For every $a \in A \cup \{T\}$ let $\mathbf{b}(a)$ be the corresponding encoding. By definition, we put $\mathbf{b}(T) = \bigwedge_{i=0}^{n-1} b_i$.

If we consider $AP = FC \cup \{b_0, \dots, b_{n-1}\}$ as a set of propositional letters for discrete LTL, we have that we can transform any hybrid trace $\alpha = \tau_1 a_1 \tau_2 a_2 \dots$ into a discrete sequence $\Sigma(\alpha) = \sigma_1\sigma_2\sigma_3\dots$ where every element is a subset of AP defined as follows: $\sigma_1 = \{f \in FC \mid \tau_1 \vdash f\}$; for every $i > 1$, $\sigma_i = \{f \in FC \mid \tau_i \vdash f\} \cup \{b_j \in B \mid b_j \text{ holds true in } \mathbf{b}(a_{i-1})\}$.

Now, let $\gamma(\varphi)$ be the discrete LTL formula obtained from φ by means of the translation function γ defined in Table 2. It is easy to see that $\Sigma(\alpha)$ is a model for $\gamma(\varphi)$, as proved by the following lemma.

Lemma 2. *For every hybrid trace α, $\alpha, 1 \Vdash \varphi$ if and only if $\Sigma(\alpha), 1 \Vdash \gamma(\varphi)$.*

Proof. Let φ be a HyLTL formula, and α a hybrid trace. We prove the lemma by showing that the following stronger claim holds:

$$\text{for every } i \geq 1,\ \alpha, i \Vdash \varphi \text{ if and only if } \Sigma(\alpha), i \Vdash \gamma_0(\varphi).$$

We reason by induction on the structure of φ:

- if $\varphi = a$, with $a \in A$, then $\gamma_0(a) = \mathbf{b}(a)$ and the claim follows easily by the definition of $\Sigma(\alpha)$;
- if $\varphi = f$, with $f \in FC$, then $\gamma_0(f) = f$ and the claim follows easily by the definition of $\Sigma(\alpha)$;
- the boolean cases are trivial and thus skipped;
- when $\varphi = \mathbf{X}\psi$, we have that $\alpha, i \Vdash \mathbf{X}\psi$ iff $\alpha, i+1 \Vdash \psi$. By inductive hypothesis we have that $\Sigma(\alpha), i+1 \Vdash \gamma_0(\psi)$, from which we can conclude that $\Sigma(\alpha), i \Vdash \mathfrak{X}\gamma_0(\psi)$;

- suppose $\varphi = \psi_1 \mathbf{U} \psi_2$. By the semantic of HyLTL, we have that $\alpha, i \Vdash \psi_1 \mathbf{U} \psi_2$ iff there exists $j \geq i$ such that $\alpha, j \Vdash \psi_2$, and for every $i \leq k < j$, $\alpha, k \Vdash \psi_1$. By inductive hypothesis we have that $\Sigma(\alpha), j \Vdash \gamma_0(\psi_2)$ and that $\Sigma(\alpha), k \Vdash \gamma_0(\psi_1)$ for every $i \leq k < j$. Hence, $\Sigma(\alpha), i \Vdash \gamma_0(\psi_1)\, \mathfrak{U}\, \gamma_0(\psi_2)$ and the claim is proved.
- the case when $\varphi = \psi_1 \mathbf{R} \psi_2$ is analogous.

To conclude the proof it is sufficient to consider that, by definition, $\Sigma(\alpha), 1 \Vdash \bigwedge_{i=0}^{n-1} \neg b_i$. Hence, from the claim it is immediate to conclude that $\Sigma(\alpha), 1 \Vdash \bigwedge_{i=0}^{n-1} \neg b_i \wedge \gamma_0(\varphi)$ if and only if $\alpha, 1 \Vdash \varphi$. □

When φ is a formula of HyLTL$^+$ we have that also $\gamma(\varphi)$ is a formula where flow constraints appear only in positive form. Hence, $\gamma(\varphi)$ cannot force the negation of a flow constraint to hold in any of the elements σ_i of a discrete sequence, as formally stated by the following lemma.

Lemma 3. *Let $\Sigma = \sigma_1\sigma_2\ldots$ and $\mathrm{P} = \rho_1\rho_2\ldots$ be two discrete sequences such that for every $i \geq 1$, $\sigma_i \cap B = \rho_i \cap B$ (the sequences agrees on the propositional letters in B) and $\sigma_i \subseteq \rho_i$ (every flow constraint that is true in Σ is true also in P). Then, for every LTL formula γ where flow constraints appear only in positive form and index $j \geq 1$, if $\Sigma, j \Vdash \gamma$ then $\mathrm{P}, j \Vdash \gamma$.*

Proof. Suppose $\Sigma, j \Vdash \gamma$. We prove the claim by induction on the structure of γ.

- If $\gamma = b_k$ or $\gamma = \neg b_k$, for some $b_k \in B$, we have that the claim follows immediately by the fact that $\sigma_j \cap B = \rho_j \cap B$;
- If $\gamma = f$ for some $f \in FC$, by the semantics of LTL we have that $f \in \sigma_j$. By hypothesis $\sigma_j \subseteq \rho_j$ and this implies that $\mathrm{P}, j \Vdash f$;
- The remaining cases can be easily proved from the inductive hypothesis and the semantics of LTL. □

6.2 Building the Büchi automaton $\mathcal{A}_{\gamma(\varphi)}$

Since the seminal work of Vardi and Wolper [19], translation of LTL formulas into equivalent Büchi automata plays an important role in many model checking and satisfiability checking algorithms. This led to the development of many translation algorithms exploiting several heuristics and optimization techniques. According to the experiments in [18], two leading tools are LTL2BA [10] and SPOT [7]. A new version of the former, called LTL3BA, has been recently introduced [3]. According to the authors, it is faster and it produces smaller automata than LTL2BA, while it produces automata of similar quality with respect to SPOT, being usually faster.

We choose to use LTL3BA as the tool for translating the formula $\gamma(\varphi)$ into the Büchi automaton $\mathcal{A}_{\gamma(\varphi)}$, since it is a state-of-the-art tool that is freely available under an open source license. Nevertheless, the high level HyLTL$^+$ translation algorithm is independent from the specific tool used to build $\mathcal{A}_{\gamma(\varphi)}$, and can be easily adapted to use other tools.

The output of LTL3BA is a Büchi automaton $\mathcal{A}_{\gamma(\varphi)}$ of the form $\langle Q, q_0, \delta, F\rangle$, where Q is the set of states, q_0 is the unique initial state, δ is the transition relation and F is the set of final states. To merge many transitions into a single one, the transitions are labelled with conjunctions of atomic propositions from AP: the automaton can fire a transition (q, β, q') whenever it reads a symbol σ_j of the discrete sequence that satisfies the boolean formula β. Since $\gamma(\varphi)$ is a formula where flow constraints appear only positively, Lemma 3 guarantees that we can assume, without loss of generality, that in the boolean formulas labeling the transitions of $\mathcal{A}_{\gamma(\varphi)}$ flow constraints appear only positively. The following lemma connects the language of $\mathcal{A}_{\gamma(\varphi)}$ with the set of hybrid traces satisfying φ.

Algorithm 1: how to build the BHA equivalent to φ

```
Input: $\mathcal{A}_{\gamma(\varphi)} = \langle Q, q_0, \delta, F \rangle$
Output: $\mathcal{H}_\varphi = \langle \text{Loc}, X, A \cup \{T\}, \text{Edg}, \text{Dyn}, \text{Res}, \text{Init}, \mathcal{F} \rangle$
1  Loc $= \emptyset$, Edg $= \emptyset$;
2  $\mathcal{L} = \emptyset$;
3  foreach transition $(q_0, \beta, q) \in \delta$ do
4      if $\beta \to \bigwedge_{i=0}^{n-1} \neg b_i$ then
5          $C = \{f \in FC \mid \beta \to f\}$;
6          add $(q, C)$ to Loc;
7          add $(q, C)$ to Init;
8          set Dyn$(q, C) = C$;
9          add $(q, C)$ to $\mathcal{L}$;
10     end
11 end
12 while the queue $\mathcal{L}$ is not empty do
13     extract an element $(q, C)$ from $\mathcal{L}$;
14     foreach transition $(q, \beta, q') \in \delta$ do
15         $C' = \{f \in FC \mid \beta \to f\}$;
16         if $(q', C') \notin$ Loc then
17             add $(q', C')$ to Loc;
18             set Dyn$(q', C') = C'$;
19             add $(q', C')$ to $\mathcal{L}$;
20         end
21         foreach $a \in A \cup \{T\}$ do
22             if $\beta \to \mathbf{b}(a)$ then
23                 add transition $(q, C, a, q', C')$ to Edg;
24                 set Res$(q, C, a, q', C') = \top$;
25             end
26         end
27     end
28 end
29 $\mathcal{F} = \{(q, C) \in \text{Loc} \mid q \in F\}$;
```

Lemma 4. *Let φ be a HyLTL$^+$ formula, and α a hybrid trace. Then $\alpha, 1 \Vdash \varphi$ if and only if $\Sigma(\alpha)$ is accepted by $\mathcal{A}_{\gamma(\varphi)}$.*

6.3 From $\mathcal{A}_{\gamma(\varphi)}$ to $\mathcal{H}_\varphi$

By Lemma 4, we have that the language of $\mathcal{A}_{\gamma(\varphi)}$ *contains* all the discrete sequences $\Sigma(\alpha)$ such that α satisfies φ. However, $\mathcal{A}_{\gamma(\varphi)}$ may accepts also "spurious" discrete sequences that do not represent a hybrid trace (for instance, sequences where flow constraints are contradictory). Algorithm 1 accepts as input the discrete automaton $\mathcal{A}_{\gamma(\varphi)}$ and build a BHA $\mathcal{H}_\varphi$ that accepts only the hybrid traces satisfying φ.

The following theorem proves that the algorithm is correct.

Theorem 2. *Let φ be a formula of HyLTL$^+$, and let $\mathcal{H}_\varphi$ be the BHA built by Algorithm 1. For every hybrid trace α, we have that $\mathcal{H}_\varphi$ accepts α if and only if $\alpha, 1 \Vdash \varphi$.*

Proof. Let $\alpha = \tau_1 a_1 \tau_2 a_2 \dots$ be a hybrid trace such that $\alpha, 1 \Vdash \varphi$. By Lemma 4, we have that $\mathcal{A}_{\gamma(\varphi)}$ accepts the discrete sequence $\Sigma(\alpha)$. Let $q_0 \xrightarrow{\beta_1} q_1 \xrightarrow{\beta_2} q_2 \xrightarrow{\beta_2} \dots$ be an accepting run of $\mathcal{A}_{\gamma(\varphi)}$ over $\Sigma(\alpha)$. For every $i \geq 1$, let $C_i = \{f \in FC \mid \beta_i \to f\}$, and consider the sequence $(q_1,C_1),(q_2,C_2),(q_3,C_3)\dots$. By Algorithm 1 we have that:

1. every pair (q_i,C_i) of the sequence is a location of $\mathcal{H}_\varphi$;
2. $(q_1,C_1) \in \text{Init}$;
3. every set of flow constraints C_i is such that $\text{Dyn}(q_i,C_i) = C_i$;
4. the transition $(q_i,C_i,a_i,q_{i+1},C_{i+1}) \in \text{Edg}$ with reset condition $\top$.

By definition of $\Sigma(\alpha)$ we have that $\tau_i \vdash C_i$, and thus we can conclude that for every $i \geq 1$, both $(q_i,C_i,\tau_i.\mathit{fstate}) \xrightarrow{\tau_i} (q_i,C_i,\tau_i.\mathit{lstate})$ and $(q_i,C_i,\tau_i.\mathit{lstate}) \xrightarrow{a_i} (q_{i+1},C_{i+1},\tau_{i+1}.\mathit{fstate})$ are valid transitions of $\mathcal{H}_\varphi$. This means that α is generated by $\mathcal{H}_\varphi$. Since $\Sigma(\alpha)$ is accepted by the discrete automaton $\mathcal{A}_{\gamma(\varphi)}$ is possible to find a location $(q_f,C_f) \in \mathcal{F}$ that occurs infinitely often in the sequence. This proves that α is accepted by $\mathcal{H}_\varphi$.

To conclude the proof, consider a hybrid trace $\alpha = \tau_1 a_1 \tau_2 a_2 \dots$ that is accepted by $\mathcal{H}_\varphi$, and let $\Sigma(\alpha) = \sigma_1 \sigma_2 \dots$ be the corresponding discrete sequence. By the semantics of BHA, it is possible to find an accepting sequence of locations $(q_1,C_1),(q_2,C_2),(q_3,C_3)\dots$ such that $(q_i,C_i,\tau_i.\mathit{fstate}) \xrightarrow{\tau_i} (q_i,C_i,\tau_i.\mathit{lstate})$ and $(q_i,C_i,\tau_i.\mathit{lstate}) \xrightarrow{a_i} (q_{i+1},C_{i+1},\tau_{i+1}.\mathit{fstate})$ for every $i \geq 1$. By Algorithm 1 we have that there exists an accepting run $q_0 \xrightarrow{\rho_1} q_1 \xrightarrow{\rho_2} q_2 \xrightarrow{\rho_3} \dots$ of the discrete automaton $\mathcal{A}_{\gamma(\varphi)}$ over the discrete sequence $\mathrm{P} = \rho_1 \rho_2 \dots$ where $\rho_i = C_i \cup \{b_j \in B \mid b_j \text{ holds true in } \mathbf{b}(a_{i-1})\}$ for every $i \geq 1$. Since every location (q_i,C_i) is such that $\text{Dyn}(q_i,C_i) = C_i$ we have that for every $f \in C_i$, $\tau_i \vdash f$ and thus that $\rho_i \subseteq \sigma_i$. From Lemma 3 we can conclude that, since $\mathcal{A}_{\gamma(\varphi)}$ accepts P then $\mathcal{A}_{\gamma(\varphi)}$ accepts also $\Sigma(\alpha)$. By Lemma 4 we can conclude that $\alpha, 1 \Vdash \varphi$. □

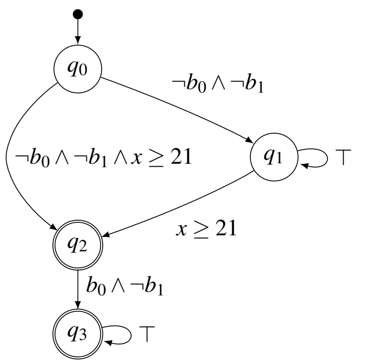
7 The improved algorithm at work

In [5] feasibility of the automaton-based model checking approach has been tested by verifying the well-known Thermostat example against the HyLTL formula $\varphi_{hyb} = \neg\mathbf{F}(x \geq 21 \wedge \mathbf{X} on)$ corresponding to the property that *"it is not possible that the heater turns on when the temperature is above 21 degrees"*.

To verify the example it is necessary to build the automaton for $\neg\varphi_{hyb} = \mathbf{F}(x \geq 21 \wedge \mathbf{X} on) = \top\,\mathbf{U}\,(x \geq 21 \wedge \mathbf{X} on)$. The original declarative construction builds a BHA with 18 locations. In this section we will apply the new algorithm to the formula and we will show that the resulting BHA is much smaller that the previous one. Notice that the formula $\neg\varphi_{hyb}$ is a formula where flow constraints appears only in positive form. Hence, it is not necessary to apply the translation π of Table 1 to obtain a formula of HyLTL$^+$. The first step of the translation algorithm is thus the application of function γ (Table 2) to obtain the following formula of discrete LTL:

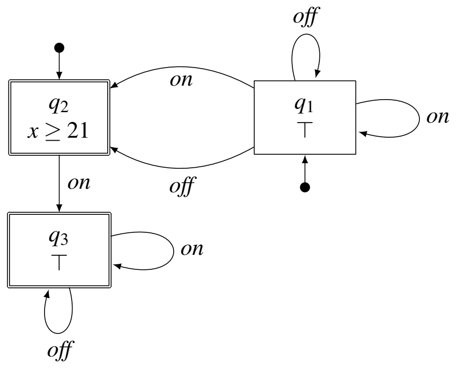
$$\gamma(\neg\varphi_{hyb}) = \neg b_0 \wedge \neg b_1 \wedge \top\,\mathfrak{U}\,(x \geq 21 \wedge \mathfrak{X}(b_0 \wedge \neg b_1))\,,$$

where we assume that $\mathbf{b}(on) = b_0 \wedge \neg b_1$. By using the tool LTL3BA we obtain the Büchi automaton $\mathcal{A}_{\gamma(\neg\varphi_{hyb})}$ depicted in Figure 1a. Then, by applying Algorithm 1 we can build the BHA depicted in Figure 1b. In both pictures initial states/locations are identified by a bullet-arrow while the final states/locations have a double border. The final BHA obtained by the new construction algorithm is made of only 3 location, with a great improvement over the original declarative construction.



q_0
$\neg b_0 \wedge \neg b_1$
$\neg b_0 \wedge \neg b_1 \wedge x \geq 21$
q_1
$\top$
$x \geq 21$
q_2
$b_0 \wedge \neg b_1$
q_3
$\top$

(a) Büchi automaton $\mathcal{A}_{\gamma(\neg\varphi_{hyb})}$.



(b) Hybrid automaton $\mathcal{H}_{\neg\varphi_{hyb}}$.

Figure 1: The discrete and hybrid automata for $\neg\varphi_{hyb}$.

As a second example, consider the globally-eventually formula $\varphi_{liv} = \mathbf{G}(\neg x \geq 18 \rightarrow \mathbf{X}\,\mathbf{F}\,on)$ expressing the liveness property to *"eventually switch the heater on if the temperature falls below* 18 *degrees"*. In this case the negation of the property is the formula $\neg\varphi_{liv} = \mathbf{F}(\neg x \geq 18 \wedge \mathbf{X}\,\mathbf{G}\,\neg on) = \top\,\mathbf{U}\,(\neg x \geq 18 \wedge \mathbf{X}(\bot\,\mathbf{R}\,\neg on))$, that do not belongs to the language of HyLTL$^+$. Hence, it is necessary to apply the translation function π to obtain the following equivalent formula:

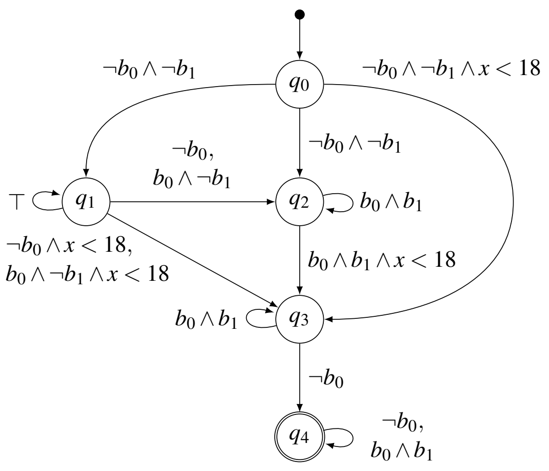
$$\begin{aligned}
\overline{\varphi}_{liv} &= \pi\Big(\top\,\mathbf{U}\,\big(\neg x \geq 18 \wedge \mathbf{X}(\bot\,\mathbf{R}\,\neg on)\big)\Big) = (T \vee \top)\,\mathbf{U}\,\Big(\neg T \wedge \pi\big(\neg x \geq 18 \wedge \mathbf{X}(\bot\,\mathbf{R}\,\neg on)\big)\Big) \\
&= \top\,\mathbf{U}\,\Big(\neg T \wedge \pi(\neg x \geq 18) \wedge \pi\big(\mathbf{X}\,(\bot\,\mathbf{R}\,\neg on)\big)\Big) \\
&= \top\,\mathbf{U}\,\Big(\neg T \wedge \Big(x < 18 \vee \mathbf{X}\big(T\,\mathbf{U}\,(T \wedge x < 18)\big)\Big) \wedge \mathbf{X}\Big(T\,\mathbf{U}\,\big(\neg T \wedge \pi(\bot\,\mathbf{R}\,\neg on)\big)\Big)\Big) \\
&= \top\,\mathbf{U}\,\Big(\neg T \wedge \Big(x < 18 \vee \mathbf{X}\big(T\,\mathbf{U}\,(T \wedge x < 18)\big)\Big) \wedge \mathbf{X}\Big(T\,\mathbf{U}\,\big(\neg T \wedge \bot\,\mathbf{R}\,(T \vee \neg on)\big)\Big)\Big)
\end{aligned}$$

The input formula for LTL3BA is thus

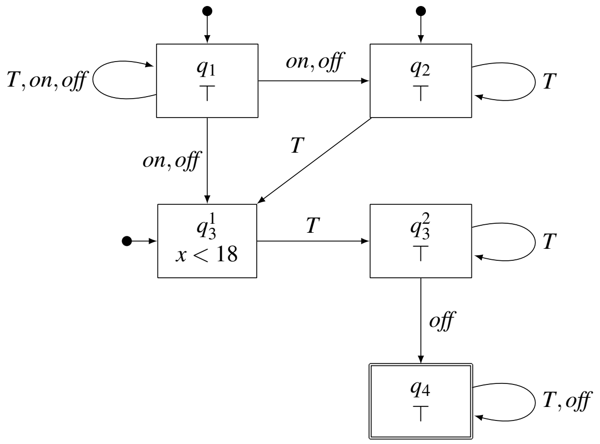
$$\begin{aligned}
\gamma(\overline{\varphi}_{safe}) = \neg b_0 \wedge \neg b_1 \wedge \top\,\mathfrak{U}\,\Bigg(&\neg(b_0 \wedge b_1) \wedge \Big(x < 18 \vee \mathfrak{X}\big((b_0 \wedge b_1)\,\mathfrak{U}\,(b_0 \wedge b_1 \wedge x < 18)\big)\Big) \\
&\wedge \mathfrak{X}\Big((b_0 \wedge b_1)\,\mathfrak{U}\,\Big(\neg(b_0 \wedge b_1) \wedge \bot\,\mathfrak{R}\,\big((b_0 \wedge b_1) \vee \neg(b_0 \wedge \neg b_1)\big)\Big)\Big)\Bigg)
\end{aligned}$$

while the resulting discrete Büchi automaton is depicted in Figure 2a. Algorithm 1 transforms it into the BHA with 5 locations shown in Figure 2b. Notice that, despite the increased complexity of the formula due to the translation into HyLTL$^+$ the final result is still of very small size.

We have verified that the thermostat example given in [5] respects the two example properties φ_{hyb} and φ_{liv} using the software package PhaVer [8]. Since the system and the automata for the properties are very simple, the computation time was almost instantaneous: less than $0.1s$ for both formulas on an Intel Core 2 Duo 2.4 GHz iMac with 4 Gb of RAM.


q_0
q_1
q_2
q_3
q_4
$\neg b_0 \wedge \neg b_1$
$\neg b_0 \wedge \neg b_1 \wedge x < 18$
$\neg b_0,$
$b_0 \wedge \neg b_1$
$\top$
$b_0 \wedge b_1$
$\neg b_0 \wedge x < 18,$
$b_0 \wedge \neg b_1 \wedge x < 18$
$b_0 \wedge b_1 \wedge x < 18$
$\neg b_0$
$\neg b_0,$
$b_0 \wedge b_1$

(a) Büchi automaton $\mathcal{A}_{\gamma(\overline{\varphi}_{liv})}$.



(b) Hybrid automaton $\mathcal{H}_{\overline{\varphi}_{liv}}$.

Figure 2: The discrete and hybrid automata for $\neg\varphi_{liv}$.

8 Conclusion

In this paper we extended the current research on HyLTL, a logic that is able to express properties of hybrid traces, and that can be used to verify hybrid systems. We identified the fragment of HyLTL that can be transformed into hybrid automata, that is, the positive flow constraints fragment HyLTL$^+$. Then, we have shown that every property definable in the full language is also definable by HyLTL$^+$. Finally, we developed a new algorithm to translate formulas into hybrid automata, that turned out to be much more efficient than the original declarative algorithm.

This work can be extended in many directions. The expressivity of the logic can be extended by adding jump predicates to the language, to express properties on the reset functions of the system. A comprehensive tool support for the logic is currently missing: an implementation of the complete model checking algorithm into the software package Ariadne [4] is under development.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. h. Ho, X. Nicollin, A. Olivero, J. Sifakis & S. Yovine (1995): *The Algorithmic Analysis of Hybrid Systems*. Theoretical Computer Science 138, pp. 3–34, doi:10.1016/0304-3975(94)00202-T.

[2] R. Alur & D. L. Dill (1994): *A Theory of Timed Automata*. J. of Theor. Computer Science 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[3] T. Babiak, M. Kretínský, V. Rehák & J. Strejcek (2012): *LTL to Büchi Automata Translation: Fast and More Deterministic*. In: Proc. of the 18th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012), LNCS 7214, pp. 95–109, doi:10.1007/978-3-642-28756-5_8.

[4] L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti & T. Villa (2012): *Assume-guarantee verification of nonlinear hybrid systems with ARIADNE*. Int. J. Robust Nonlinear Control, doi:10.1002/rnc.2914.

[5] D. Bresolin (2013): *HyLTL: a temporal logic for model checking hybrid systems*. In: Proc. of the 3rd International Workshop on Hybrid Autonomous Systems (HAS 2013), EPTCS 118, pp. 64–75. To appear.

[6] A. Cimatti, M. Roveri & S. Tonetta (2009): *Requirements Validation for Hybrid Systems*. In: CAV, LNCS 5643, pp. 188–203, doi:10.1007/978-3-642-02658-4_17.

[7] A. Duret-Lutz (2011): *LTL translation improvements in SPOT*. In: *Proc. of the 5th Int. Conf. on Verification and Evaluation of Computer and Communication Systems (VECoS'11)*, British Computer Society, pp. 72–83.

[8] G. Frehse (2008): *PHAVer: algorithmic verification of hybrid systems past HyTech*. *International Journal on Software Tools for Technology Transfer (STTT)* 10, pp. 263–279, doi:10.1007/s10009-007-0062-x.

[9] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang & O. Maler (2011): *SpaceEx: Scalable Verification of Hybrid Systems*. In: *Proc. 23rd International Conference on Computer Aided Verification (CAV 2011)*, *LNCS* 6806, Springer Berlin / Heidelberg, pp. 379–395, doi:10.1007/978-3-642-22110-1_30.

[10] P. Gastin & D. Oddoux (2001): *Fast LTL to Büchi Automata Translation*. In: *Proc. of the 13th Int. Conf. on Computer Aided Verification (CAV 2001)*, *LNCS* 2102, Springer, pp. 53–65, doi:10.1007/3-540-44585-4_6.

[11] T. A. Henzinger, P. W. Kopke, A. Puri & P. Varaiya (1998): *What's Decidable about Hybrid Automata?* *Journal of Computer and System Sciences* 57(1), pp. 94 – 124, doi:10.1006/jcss.1998.1581.

[12] L. Lamport (1993): *Hybrid systems in TLA+*. In: *Hybrid Systems*, *LNCS* 736, Springer, pp. 77–102, doi:10.1007/3-540-57318-6_25.

[13] K. G. Larsen, P. Pettersson & W. Yi (1997): *UPPAAL in a nutshell*. *Int. J. on Software Tools for Technology Transfer* 1(1–2), pp. 134–152, doi:10.1007/s100090050010.

[14] O. Maler, Z. Manna & A. Pnueli (1991): *From Timed to Hybrid Systems*. In: *Real-Time: Theory in Practice*, *LNCS* 600, Springer-Verlag, pp. 447–484, doi:10.1007/BFb0032003.

[15] O. Maler & D. Nickovic (2004): *Monitoring Temporal Properties of Continuous Signals*. In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, *LNCS* 3253, Springer, pp. 152–166, doi:10.1007/978-3-540-30206-3_12.

[16] A. Platzer & J.-D. Quesel (2008): *KeYmaera: A Hybrid Theorem Prover for Hybrid Systems*. In: *Proc. of the 3rd International Joint Conference on Automated Reasoning (IJCAR 2008)*, *LNCS* 5195, Springer, pp. 171–178, doi:10.1007/978-3-540-71070-7_15.

[17] S. Ratschan & Z. She (2007): *Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement*. *ACM Trans. in Embedded Computing Systems* 6(1), doi:10.1145/1210268.1210276.

[18] K. Y. Rozier & M. Y. Vardi (2010): *LTL satisfiability checking*. *Int. J. on Software Tools for Technology Transfer* 12(2), pp. 123–137, doi:10.1007/s10009-010-0140-3.

[19] M. Y. Vardi & P. Wolper (1986): *An Automata-Theoretic Approach to Automatic Program Verification*. In: *Proc. of the 1st Symposium on Logic in Computer Science (LICS'86)*, IEEE Computer Society, pp. 332–344.

[20] S. Yovine (1997): *Kronos: a verification tool for real-time systems*. *Int. J. on Software Tools for Technology Transfer* 1(1–2), pp. 123–133, doi:10.1007/s100090050009.

Upwards Closed Dependencies in Team Semantics

Pietro Galliani[*]
Department of Mathematics and Statistics
Helsinki, Finland
pgallian@gmail.com

We prove that adding upwards closed first-order dependency atoms to first-order logic with team semantics does not increase its expressive power (with respect to sentences), and that the same remains true if we also add constancy atoms. As a consequence, the negations of functional dependence, conditional independence, inclusion and exclusion atoms can all be added to first-order logic without increasing its expressive power.

Furthermore, we define a class of bounded upwards closed dependencies and we prove that unbounded dependencies cannot be defined in terms of bounded ones.

1 Introduction

Team semantics is a generalization of Tarski's semantics in which formulas are satisfied or not satisfied by sets of assignments, called *teams*, rather than by single assignments. It was originally developed by Hodges, in [14], as a compositional alternative to the imperfect-information *game theoretic semantics* for independence friendly logic [13, 18].

Over the past few years team semantics has been used to specify and study many other extensions of first-order logic. In particular, since a team describes a relation between the elements of its model team semantics offers a natural way to add to first-order logic atoms corresponding to database-theoretic *dependency notions*.

This line of thought led first to the development of *dependence logic* [19], and later to that of *independence logic* [12] and *inclusion and exclusion logics* [8].[1] By now there are many results in the literature concerning the properties of these logics, and in Section 2 we recall some of the principal ones.

One common characteristic of all these logics is that they are much stronger than first-order logic proper, even though they merely add *first-order definable* dependency conditions to its language. Indeed, the rules of team semantics straddle the line between first and second order, since they evaluate first-order connectives by means of second-order machinery: and, while in the case of first-order logic formulas team semantics can be reduced to Tarski's semantics, if we add to our language atoms corresponding to further conditions the second-order nature of team semantics can take over.

The purpose of the present paper is to investigate the boundary between first and second order "from below", so to say, taking first-order logic with team semantics and trying to find out how much we can add to it while preserving first-orderness. In Section 3 we define a fairly general family of classes of first-order definable dependency conditions and prove they can be safely added to first-order logic; then in Section 4 we expand this family, and in Section 5 we show that, as a consequence, the negations of all the main dependency atoms studied in team semantics do not "blow up" first-order logic into a higher

[*]Research supported by Grant 264917 of the Academy of Finland.

[1]The literature contains many other extensions of first-order logic with team semantics, but we do not examine them in this work.

order one. Finally, in Section 6 we introduce a notion of *boundedness* for dependencies and use it to demonstrate some *non-definability results*.

2 Preliminaries

In this section we will recall some fundamental definitions and results concerning team semantics.

Definition 1 (Team) *Let M be a first-order model and let $\mathtt{Dom}(M)$ be the set of its elements.*[2] *Furthermore, let V be a finite set of variables. Then a* team *X over M with* domain *$\mathtt{Dom}(X) = V$ is a set of assignments s from V to $\mathtt{Dom}(M)$.*

Given a team X and a tuple of variables $\vec{v}$ contained in the domain of X, we write $X \upharpoonright \vec{v}$ for the team obtained by restricting all assignments of X to the variables of $\vec{v}$ and $X(\vec{v})$ for the relation $\{s(\vec{v}) : s \in X\} \subseteq \mathtt{Dom}(M)^{|\vec{v}|}$.

As it is common when working with team semantics, we will assume that all our expressions are in negation normal form.

Definition 2 (Team Semantics for First-Order Logic) *Let $\phi(\vec{x})$ be a first-order formula in negation normal form with free variables in $\vec{x}$. Furthermore, let M be a first-order model whose signature contains the signature of ϕ and let X be a team over it whose domain contains $\vec{x}$. Then we say that X* satisfies *ϕ in M, and we write $M \models_X \phi$, if and only if this follows from these rules:*[3]

TS-lit: *For all first-order literals α, $M \models_X \alpha$ if and only if for all $s \in X$, $M \models_s \alpha$ according to the usual Tarski semantics;*

TS-$\vee$: *For all ψ and θ, $M \models_X \psi \vee \theta$ if and only if $X = Y \cup Z$ for two subteams Y and Z such that $M \models_Y \psi$ and $M \models_Z \theta$;*

TS-$\wedge$: *For all ψ and θ, $M \models_X \psi \wedge \theta$ if and only if $M \models_X \psi$ and $M \models_X \theta$;*

TS-$\exists$: *For all ψ and all variables v, $M \models_X \exists v \psi$ if and only if there exists a function*

$$H : X \to \mathscr{P}(\mathtt{Dom}(M)) \backslash \{\emptyset\}$$

such that $M \models_{X[H/v]} \psi$, where $X[H/v] = \{s[m/v] : s \in X, m \in H(s)\}$ and $\mathscr{P}(\mathtt{Dom}(M))$ is the powerset of $\mathtt{Dom}(M)$;

TS-$\forall$: *For all ψ and all variables v, $M \models_X \forall v \psi$ if and only if $M \models_{X[M/v]} \psi$, where $X[M/v] = \{s[m/v] : s \in X, m \in M\}$.*

Given a sentence (that is, a formula with no free variables) ϕ and a model M over its signature, we say that ϕ is true *in M and we write $M \models \phi$ if and only if $M \models_{\{\emptyset\}} \phi$.*[4]

The following is a useful and easily derived rule:

Lemma 3 *Let $\vec{v} = v_1 \ldots v_n$ be a tuple of n variables and let $\exists \vec{v} \psi$ be a shorthand for $\exists v_1 \ldots \exists v_n \psi$. Then $M \models_X \exists \vec{v} \psi$ if and only if there exists a function $H : X \to \mathscr{P}(\mathtt{Dom}(M)^n) \backslash \{\emptyset\}$ such that $M \models_{X[H/\vec{v}]} \psi$, where $X[H/\vec{v}] = \{s[\vec{m}/\vec{v}] : s \in X, \vec{m} \in H(s)\}$.*

[2]We always assume that models have at least two elements in their domain.

[3]What we give here is the so-called *lax* version of team semantics. There also exists a *strict* version, with slightly different rules for disjunction and existential quantification; but as pointed out in [8], *locality* – in the sense of Theorem 8 here – fails in strict team semantics for some of the logics we are interested in. Therefore, in this work we will only deal with lax team semantics.

[4]Of course, one should not confuse the team $\{\emptyset\}$, which contains only the empty assignment, with the *empty team* $\emptyset$, which contains no assignments at all.

With respect to first-order formulas, team semantics can be reduced to Tarski's semantics. Indeed,

Proposition 4 ([14, 19]) *Let $\phi(\vec{x})$ be a first-order formula in negation normal form with free variables in $\vec{x}$. Furthermore, let M be a first-order model whose signature contains that of ϕ, and let X be a team over M whose domain contains $\vec{x}$. Then $M \models_X \phi$ if and only if, for all $s \in X$, $M \models_s \phi$ with respect to Tarski's semantics.*

In particular, a first-order sentence ϕ is true in a model M with respect to team semantics if and only if it is true in M with respect to Tarski's semantics.

Therefore, not all first-order definable properties of relations correspond to the satisfaction conditions of first-order formulas: for example, the non-emptiness of a relation R is definable by $\exists \vec{x} R \vec{x}$, but there is no first order ϕ such that $M \models_X \phi$ if and only if $X \neq \emptyset$. More in general, let $\phi^*(R)$ be a first-order sentence specifying a property of the k-ary relation R and let $\vec{x} = x_1 \dots x_k$ be a tuple of new variables: then, as it follows easily from the above proposition, there exists a first-order formula $\phi(\vec{x})$ such that

$$M \models_X \phi(\vec{x}) \Leftrightarrow M, X(\vec{x}) \models \phi^*(R)$$

if and only if $\phi^*(R)$ can be put in the form $\forall \vec{x}(R\vec{x} \rightarrow \theta(\vec{x}))$ for some θ in which R does not occur.[5]

It is hence possible to extend first-order logic (with team semantics) by introducing new atoms corresponding to further properties of relations. Database theory is a most natural choice as a source for such properties; and, in the rest of this section, we will recall the fundamental database-theoretic extensions of first-order logic with team semantics and some of their properties.

Dependence logic $\mathrm{FO}(=(\cdot,\cdot))$, from [19], adds to first-order logic *functional dependence atoms* $=(\vec{x},\vec{y})$ based on database-theoretic *functional dependencies* ([2]). Their rule in team semantics is

TS-fdep: $M \models_X =(\vec{x},\vec{y})$ if and only if for all $s, s' \in X$, $s(\vec{x}) = s'(\vec{x}) \Rightarrow s(\vec{y}) = s'(\vec{y})$.

This atom, and dependence logic as a whole, is *downwards closed*: for all dependence logic formulas ϕ, models M and teams X, if $M \models_X \phi$ then $M \models_Y \phi$ for all $Y \subseteq X$. It is not however *union closed*: if $M \models_X \phi$ and $M \models_Y \phi$ then we cannot in general conclude that $M \models_{X \cup Y} \phi$.

Dependence logic is equivalent to existential second-order logic over sentences:

Theorem 5 ([19]) *Every dependence logic sentence ϕ is logically equivalent to some ESO sentence ϕ^*, and vice versa.*

Constancy logic $\mathrm{FO}(=(\cdot))$ is the fragment of dependence logic which only allows functional dependence atoms of the form $=(\emptyset,\vec{x})$, which we will abbreviate as $=(\vec{x})$ and call *constancy atoms*. Clearly we have that

TS-const: $M \models_X =(\vec{x})$ if and only if for all $s, s' \in X$, $s(\vec{x}) = s'(\vec{x})$.

As proved in [8], every constancy logic sentence is equivalent to some first-order sentence: therefore, constancy logic is strictly weaker than dependence logic. Nonetheless, constancy logic is more expressive than first-order logic with respect to the second-order relations generated by the satisfaction conditions of formulas: indeed, it is an easy consequence of Proposition 4 that no first-order formula is logically equivalent to the constancy atom $=(x)$.

Exclusion logic $\mathrm{FO}(|)$, from [8], adds to first-order logic *exclusion atoms* $\vec{x} \mid \vec{y}$, where $\vec{x}$ and $\vec{y}$ are tuples of variables of the same length. Just as functional dependence atoms correspond to functional database-theoretic dependencies, exclusion atoms correspond to *exclusion dependencies* [3]; and their satisfaction rule is

[5]That is, according to the terminology of [19], if and only if $\phi^*(R)$ is *flat*.

TS-excl: $M \models_X \vec{x} \mid \vec{y}$ if and only if $X(\vec{x}) \cap X(\vec{y}) = \emptyset$.

As proved in [8], exclusion logic is entirely equivalent to dependence logic: every exclusion logic formula is logically equivalent to some dependence logic formula, and vice versa.

Inclusion logic FO$(\subseteq)$, also from [8], adds instead to first-order logic *inclusion atoms* $\vec{x} \subseteq \vec{y}$ based on database-theoretic *inclusion dependencies* [6]. The corresponding rule is

TS-inc: $M \models_X \vec{x} \subseteq \vec{y}$ if and only if $X(\vec{x}) \subseteq X(\vec{y})$.

Inclusion logic is stronger than first-order logic, but weaker than existential second-order logic: indeed, as shown in [9], sentence-wise it is equivalent to positive greatest fixed point logic GFP$^+$. Formula-wise, it is incomparable with constancy, dependence or exclusion logic, since its formulas are union closed but not downwards closed.

Independence logic FO$(\perp)$, from [12], adds to first-order logic *independence atoms* $\vec{x} \perp \vec{y}$ with the intended meaning of "the values of $\vec{x}$ and $\vec{y}$ are informationally independent". More formally,

TS-ind: $M \models_X \vec{x} \perp \vec{y}$ if and only if $X(\vec{x}\vec{y}) = X(\vec{x}) \times X(\vec{y})$.

This notion of informational independence has a long history: see for example [11] for an analysis of this concept from a probabilistic perspective.

The *conditional* independence atoms $\vec{x} \perp_{\vec{z}} \vec{y}$, also from [12], relativize the independence of $\vec{x}$ and $\vec{y}$ to all *fixed* value of $\vec{z}$. Their semantics is

TS-c-ind: $M \models_X \vec{x} \perp_{\vec{z}} \vec{y}$ if and only if for all tuples $\vec{m} \in \texttt{Dom}(M)^{|\vec{z}|}$ and for $X_{\vec{z}=\vec{m}} = \{s \in X : s(\vec{z}) = \vec{m}\}$ it holds that $X_{\vec{z}=\vec{m}}(\vec{x}\vec{y}) = X_{\vec{z}=\vec{m}}(\vec{x}) \times X_{\vec{z}=\vec{m}}(\vec{y})$.

As pointed out in [4], the rule for $\vec{x} \perp_{\vec{z}} \vec{y}$ corresponds precisely to the database-theoretic *embedded multivalued dependency* [5] $(\vec{z} \twoheadrightarrow \vec{x}|\vec{y})$.

In [12] it was shown that every dependence logic formula is equivalent to some $FO(\perp_c)$ (conditional independence logic) formula, but not vice versa; and sentence-wise, both of these logics are equivalent to each other (and to ESO). Furthermore, in [8] it was proved that FO$(\perp_c)$ is equivalent to **inclusion/exclusion logic**[6] FO$(\subseteq, |)$, even with respect to open formulas, and that this is, roughly speaking, the most general logic obtainable by adding first-order (or even existential second-order) definable dependency conditions to first-order logic.[7] More recently, in [10], it was shown that FO$(\perp)$ and FO$(\perp_c)$ are also equivalent.

We conclude this section with Figure 1, which depicts the relations between the logics we discussed so far.

3 Upwards Closed Dependencies

In this work we will study the properties of the logics obtained by adding families of *dependency conditions* to the language of first-order logic. But what is a dependency condition, in a general sense? The following definition is based on the *generalized atoms* of [17]:

Definition 6 *Let $n \in \mathbb{N}$. A* dependency *of arity n is a class* **D**, *closed under isomorphisms, of models over the signature $\{R\}$ where R is a n-ary relation symbol. If $\vec{x}$ is a tuple of n variables (possibly with repetitions), M is a first-order model and X is a team over it whose domain contains all variables of $\vec{x}$ then*

[6]That is, to first-order logic plus inclusion *and* exclusion atoms.

[7]To be more precise, for every ESO formula $\phi^*(R)$ there exists a FO$(\perp_c)$ formula $\phi(\vec{x})$ such that, for all suitable models M and nonempty teams X, $M \models_X \phi(\vec{x})$ if and only if $M, X(\vec{x}) \models \phi^*(R)$.

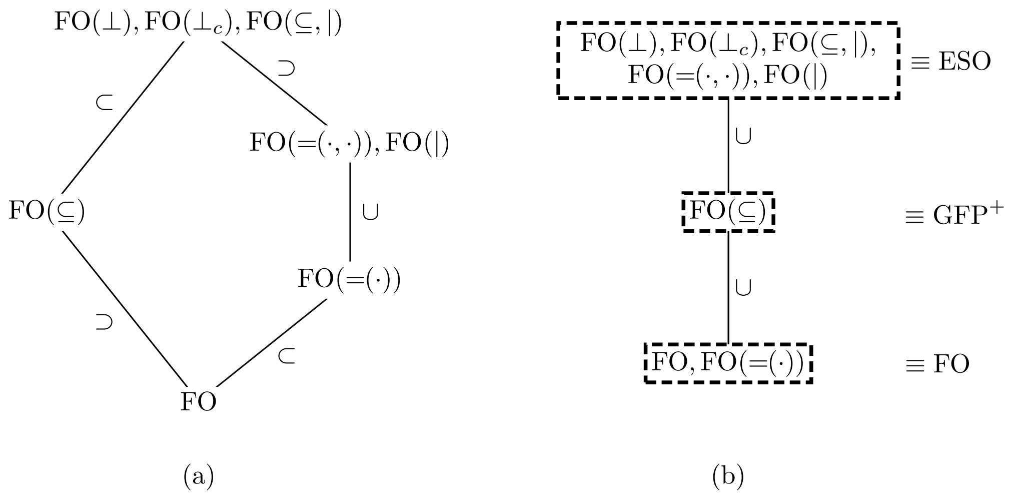


Figure 1: Relations between logics wrt formulas (a) and sentences (b).

TS-D: *$M \models_X \mathbf{D}\vec{x}$ if and only if $(\mathtt{Dom}(M), X(\vec{x})) \in \mathbf{D}$.*

Definition 7 *Let $\mathscr{D} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots\}$ be a family of dependencies. Then we write FO($\mathscr{D}$) for the logic obtained by adding to the language of first-order logic all* dependency atoms *$\mathbf{D}\vec{x}$, where $\mathbf{D} \in \mathscr{D}$ and $\vec{x}$ is a tuple of variables of the arity of $\mathbf{D}$.*

It is not difficult to represent the logics of Section 2 in this notation. For example, dependence logic is $\mathrm{FO}(=(\cdot,\cdot))$ for $=(\cdot,\cdot) = \{=(n,m) : n,m \in \mathbb{N}\}$, where $(\mathtt{Dom}(M), R) \in \ =(n,m)$ if and only if

$$\vec{a}\vec{b}, \vec{a}\vec{c} \in R \Rightarrow \vec{b} = \vec{c}$$

for all tuples of elements $\vec{a} = a_1 \ldots a_n$, $\vec{b} = b_1 \ldots b_m$, $\vec{c} = c_1 \ldots c_m \in \mathtt{Dom}(M)$.

The following property can be easily verified, by induction on the formulas ϕ:[8]

Theorem 8 (Locality) *Let $\mathscr{D}$ be a family of dependencies and let $\phi(\vec{x})$ be a formula of FO($\mathscr{D}$) with free variables in $\vec{x}$. Then for all models M and all teams X over it whose domain contains $\vec{x}$, $M \models_X \phi(\vec{x})$ if and only if $M \models_{X \upharpoonright \vec{x}} \phi(\vec{x})$.*

In this work, we will be mainly interested in dependencies which correspond to first-order definable properties of relations:

Definition 9 *A dependency notion $\mathbf{D}$ is* first-order definable *if there exists a first-order sentence $\mathbf{D}^*(R)$ over the signature $\{R\}$, where R is a new relation symbol, such that*

$$M \in \mathbf{D} \Leftrightarrow M \models \mathbf{D}^*(R)$$

for all models $M = (\mathtt{Dom}(M), R)$.

[8]For the sake of reference, we mention Theorem 4.22 of [8] in which the same result is proved in detail for (conditional) independence logic. The only new case here is the one in which $\phi(\vec{x}) = \mathbf{D}\vec{y}$ for some $\mathbf{D} \in \mathscr{D}$ and $\vec{y}$ is contained in $\vec{x}$; and for it the result follows at once from condition **TS-D** and from the fact that $X(\vec{y}) = (X \upharpoonright \vec{x})(\vec{y})$.

It is not necessarily the case that if $\mathbf{D}$ is first-order definable then FO($\mathbf{D}$) and FO are equivalent with respect to sentences. For example $=(n,m)^*(R)$ is $\forall \vec{x}\vec{y}\vec{z}(R\vec{x}\vec{y} \wedge R\vec{x}\vec{z} \rightarrow \vec{y} = \vec{z})$, where $\vec{x}$ has length n and $\vec{y}, \vec{z}$ have length m; but as we said in Section 2, dependence logic is stronger than first-order logic.

When is then the case that dependency conditions can be added safely to first-order logic, without increasing the expressive power? The following definition will provide us a partial answer:

Definition 10 *A dependency notion* $\mathbf{D}$ *is* upwards closed *if*

$$(\mathtt{Dom}(M), R) \in \mathbf{D}, R \subseteq S \Rightarrow (\mathtt{Dom}(M), S) \in \mathbf{D}$$

for all models $(\mathtt{Dom}(M), R)$ *and all relations* S *over* $\mathtt{Dom}(M)$ *of the same arity of* R.

It is easy to see that upwards closed dependencies induce upwards closed satisfaction rules: if $\mathbf{D}$ is upwards closed, $M \models_X \mathbf{D}\vec{x}$ and $X \subseteq Y$ then it is always the case that $M \models_Y \mathbf{D}\vec{x}$. However, differently from the case of downwards or union closure, upwards closure is not preserved by team semantics: if $\mathbf{D}$ is upwards closed, $\phi \in$ FO($\mathbf{D}$) and $M \models_X \phi$ then it is not in general true that $M \models_Y \phi$ for all $Y \supseteq X$ (for example, let ϕ be a nontrivial first-order literal and recall Rule **TS-lit**).

Some examples of upwards closed dependencies follow:

Non-emptiness: $M \models_X \mathtt{NE}$ if and only if $X \neq \emptyset$;

Intersection: $M \models_X \Diamond(\vec{x} = \vec{y})$ if and only if there exists a $s \in X$ with $s(\vec{x}) = s(\vec{y})$;

Inconstancy: $M \models_X \neq\!(\vec{x})$ if and only if $|X(\vec{x})| > 1$;

n-**bigness:** For all $n \in \mathbb{N}$, $M \models_X |\vec{x}| \geq n$ if and only if $|X(\vec{x})| \geq n$;

Totality: $M \models_X \mathtt{All}(\vec{x})$ if and only if $X(\vec{x}) = \mathtt{Dom}(M)^{|\vec{x}|}$;

Non-dependence: $M \models_X \neq\!(\vec{x}, \vec{y})$ if and only if there exist $s, s' \in X$ with $s(\vec{x}) = s'(\vec{x})$ but $s(\vec{y}) \neq s'(\vec{y})$;[9]

Non-exclusion: $M \models_X \vec{x} \nmid \vec{y}$ if and only if there exist $s, s' \in X$ with $s(\vec{x}) = s'(\vec{y})$;

Infinity: $M \models_X |\vec{x}| \geq \omega$ if and only if $X(\vec{x})$ is infinite;

κ-**bigness:** For all cardinals κ, $M \models_X |\vec{x}| \geq \kappa$ if and only if $|X(\vec{x})| \geq \kappa$.

All the above examples except infinity and κ-bigness are first-order definable. The NE atom is the adaptation to first-order team semantics of the non-emptiness atom introduced in [20] for the propositional version of dependence logic, and the totality atom All is due to Abramsky and Väänänen ([1]).

The main result of this section is the following:

Theorem 11 *Let* $\mathscr{D}$ *be a collection of upwards closed first-order definable dependency conditions. Then for every formula* $\phi(\vec{x})$ *of* $FO(\mathscr{D})$ *with free variables in* $\vec{x}$ *there exists a first-order sentence* $\phi^*(R)$*, where* R *is a new* $|\vec{x}|$*-ary relation symbol, such that*

$$M \models_X \phi(\vec{x}) \Leftrightarrow M, X(\vec{x}) \models \phi^*(R)$$

for all models M *over the signature of* ϕ *and all teams* X.

In particular, every sentence of $FO(\mathscr{D})$ *is equivalent to some first-order sentence.*

Let us begin by adapting the notion of *flattening* of [19] to the case of an arbitrary logic FO($\mathscr{D}$):

[9]The same symbol $\neq\!(\vec{x}, \vec{y})$ has been used in [7] to describe a different non-dependence notion, stating that for *every* $s \in X$ there exists a $s' \in X$ with $s(\vec{x}) = s'(\vec{x}), s(\vec{y}) \neq s'(\vec{y})$. In that thesis it was proved that the resulting "non-dependence logic" is equivalent to inclusion logic. As we will see, this is not the case for the non-dependence notion of this paper.

Definition 12 *Let $\mathscr{D}$ be any set of dependency conditions and let ϕ be a $FO(\mathscr{D})$ formula. Then its* flattening ϕ^f *is the first-order formula obtained by replacing any non-first-order atom with $\top$, where $\top$ is the trivially true atom.*

It is trivial to see, by induction on ϕ, that

Lemma 13 *For all $\mathscr{D}$, all $\phi \in FO(\mathscr{D})$, all models M and all teams X over M, if $M \models_X \phi$ then $M \models_X \phi^f$.*

As we said, even if $\mathscr{D}$ contains only upwards closed dependency conditions it is not true that all formulas of FO($\mathscr{D}$) are upwards closed. However, the following restricted variant of upwards closure is preserved:

Theorem 14 *Let ϕ be a $FO(\mathscr{D})$ formula, where $\mathscr{D}$ contains only upwards closed dependencies. Let M be a first-order model, and let X, Y be teams such that $X \subseteq Y$, $M \models_X \phi$, and $M \models_Y \phi^f$. Then $M \models_Y \phi$.*

Proof:
The proof is by structural induction on ϕ.

1. If ϕ is a first-order literal, $\phi^f = \phi$ and there is nothing to prove;
2. If ϕ is of the form $\mathbf{D}\vec{x}$ for some $\mathbf{D} \in \mathscr{D}$, $M \models_X \phi$ and $X \subseteq Y$, then by upwards closure $M \models_Y \phi$;
3. Suppose that $M \models_X \phi_1 \vee \phi_2$ and $M \models_Y \phi_1^f \vee \phi_2^f$. Now $X = X_1 \cup X_2$ for two X_1, X_2 such that $M \models_{X_1} \phi_1$ and $M \models_{X_2} \phi_2$, and therefore by Lemma 13 $M \models_{X_1} \phi_1^f$ and $M \models_{X_2} \phi_2^f$. Furthermore, $Y = Y_1 \cup Y_2$ for two Y_1, Y_2 such that $M \models_{Y_1} \phi^f$ and $M \models_{Y_2} \phi_2^f$. Let $Z_1 = X_1 \cup Y_1$ and $Z_2 = X_2 \cup Y_2$; then $Z_1 \cup Z_2 = X \cup Y = Y$, and by Proposition 4 $M \models_{Z_1} \phi_1^f$ and $M \models_{Z_2} \phi_2^f$. But $M \models_{X_1} \phi_1$ and $X_1 \subseteq Z_1$, so by induction hypothesis $M \models_{Z_1} \phi_1$; and similarly, $M \models_{X_2} \phi_2$ and $X_2 \subseteq Z_2$, so $M \models_{Z_2} \phi_2$. Therefore $M \models_Y \phi_1 \vee \phi_2$, as required.
4. If $M \models_X \phi_1 \wedge \phi_2$ then $M \models_X \phi_1$ and $M \models_X \phi_2$. Then by induction hypothesis, since $M \models_Y \phi_1^f$ and $X \subseteq Y$, $M \models_Y \phi_1$; and similarly, since $M \models_Y \phi_2^f$ and $X \subseteq Y$, $M \models_Y \phi_2$, and therefore $M \models_Y \phi_1 \wedge \phi_2$.
5. If $M \models_X \exists v\phi$ then there is a function $H : X \to \mathscr{P}(\mathtt{Dom}(M))\backslash\{\emptyset\}$ such that $M \models_{X[H/v]} \phi$, and therefore (by Lemma 13) such that $M \models_{X[H/v]} \phi^f$. Similarly, if $M \models_Y \exists v\phi^f$ then for some K we have that $M \models_{Y[K/v]} \phi^f$. Now let $W : Y \to \mathscr{P}(\mathtt{Dom}(M))\backslash\{\emptyset\}$ be such that

$$W(s) = \begin{cases} H(s) \cup K(s) & \text{if } s \in X; \\ K(s) & \text{if } s \in Y\backslash X. \end{cases}$$

   Then $Y[W/v] = X[H/v] \cup Y[K/v]$, and therefore by Proposition 4 $M \models_{Y[W/v]} \phi^f$. Then by induction hypothesis $M \models_{Y[W/v]} \phi$, since $X[H/v]$ satisfies ϕ and is contained in $Y[W/v]$; and therefore $M \models_Y \exists v\phi$, as required.
6. If $M \models_X \forall v\phi$ then $M \models_{X[M/v]} \phi$, and if $M \models_Y \forall v\phi^f$ then $M \models_{Y[M/v]} \phi^f$. Now $X[M/v] \subseteq Y[M/v]$, so by induction hypothesis $M \models_{Y[M/v]} \phi$, and therefore $M \models_Y \forall v\phi$.

□

Definition 15 *If θ is a first-order formula and ϕ is a $FO(\mathscr{D})$ formula we define $(\phi \upharpoonright \theta)$ as $(\neg\theta) \vee (\theta \wedge \phi)$, where $\neg\theta$ is a shorthand for the first-order formula in negation normal form which is equivalent to the negation of θ.*

The following lemma is obvious:

Lemma 16 *For all first order θ and $\phi \in FO(\mathscr{D})$, $M \models_X (\phi \restriction \theta)$ if and only if $M \models_Y \phi$ for $Y = \{s \in X : M \models_s \theta\}$.*

One can observe that $(\phi \restriction \theta)$ is logically equivalent to $\theta \hookrightarrow \phi$, where $\hookrightarrow$ is the *maximal implication* of [16]:

TS-maximp: $M \models_X \theta \hookrightarrow \phi$ if and only if for all maximal $Y \subseteq X$ s.t. $M \models_Y \theta$, $M \models_Y \phi$.

We use the notation $(\phi \restriction \theta)$, instead of $\theta \hookrightarrow \phi$, to make it explicit that θ is first order and that Lemma 16 holds.

The next step of our proof of Theorem 11 is to identify a fragment of our language whose satisfaction conditions do not involve quantification over second-order objects such as teams or functions. We do so by limiting the availability of disjunction and existential quantification:

Definition 17 *A $FO(\mathscr{D})$ formula ϕ is* clean *if*

1. *All its disjunctive subformulas $\psi_1 \vee \psi_2$ are first order or of the form $\psi \restriction \theta$ for some suitable choice of ψ and θ (where θ is first order);*
2. *All its existential subformulas $\exists v \psi$ are first order.*

As the next proposition shows, clean formulas correspond to first-order definable properties of relations.

Proposition 18 *Let $\mathscr{D}$ be a class of first-order definable dependencies and let $\phi(\vec{x}) \in FO(\mathscr{D})$ be a clean formula with free variables in $\vec{x}$. Then there exists some first-order sentence $\phi^*(R)$, where R is a new $|\vec{x}|$-ary relation, such that*

$$M \models_X \phi(\vec{x}) \Leftrightarrow M, X(\vec{x}) \models \phi^*(R). \tag{1}$$

Proof:
By induction over ϕ.

1. If $\phi(\vec{x})$ is a first-order formula (not necessarily just a literal) then let $\phi^*(R) = \forall \vec{x}(R\vec{x} \to \phi(\vec{x}))$. By Proposition 4, (1) holds.
2. If $\phi(\vec{x})$ is a dependency atom $\mathbf{D}\vec{y}$, where $\mathbf{D} \in \mathscr{D}$ and $\vec{y}$ is a tuple (possibly with repetitions) of variables occurring in $\vec{x}$, let $\phi^*(R)$ be obtained from $\mathbf{D}^*(S)$ by replacing every instance $S\vec{z}$ of S in it with $\exists \vec{x}(\vec{z} = \vec{y} \wedge R\vec{x})$. Indeed, $M \models_X \mathbf{D}\vec{y}$ if and only if $M, X(\vec{y}) \models \mathbf{D}^*(S)$, and $\vec{m} \in X(\vec{y})$ if and only if $M, X(\vec{x}) \models \exists \vec{x}(\vec{m} = \vec{y} \wedge R\vec{x})$.
3. If $\phi(\vec{x})$ is of the form $(\psi(\vec{x}) \restriction \theta(\vec{x}))$, let $\phi^*(R)$ be obtained from $\psi^*(R)$ by replacing every instance $R\vec{z}$ of R with $R\vec{z} \wedge \theta(\vec{z})$. Indeed, by Lemma 16 $M \models_X (\psi(\vec{x}) \restriction \theta(\vec{x}))$ if and only if $M \models_Y \psi(\vec{x})$ for $Y = \{s \in X : M \models_s \theta\}$, and $\vec{m} \in Y(\vec{x}) \Leftrightarrow \vec{m} \in X(\vec{x})$ and $M \models \theta(\vec{m})$.
4. If $\phi(\vec{x})$ is of the form $\psi(\vec{x}) \wedge \theta(\vec{x})$ simply let $\phi^*(R) = \psi^*(R) \wedge \theta^*(R)$.
5. If $\phi(\vec{x})$ is of the form $\forall v \psi(\vec{x}, v)$, where we assume without loss of generality that v is distinct from all $x \in \vec{x}$, and $\psi^*(S)$ corresponds to $\psi(\vec{x}, v)$ then let $\phi^*(R)$ be obtained from $\psi^*(S)$ by replacing every $S\vec{z}w$ with $R\vec{z}$. Indeed, $M \models_X \forall v \psi$ if and only if $M \models_{X[M/v]} \psi(\vec{x}, v)$ and $\vec{m}m' \in X[M/v](\vec{x}v)$ if and only if $\vec{m} \in X(\vec{x})$.

□

All that is now left to prove is the following:

Proposition 19 *Let $\mathscr{D}$ be a family of upwards closed dependencies. Then every $FO(\mathscr{D})$ formula is equivalent to some clean $FO(\mathscr{D})$ formula.*

Proof:
It suffices to observe the following facts:

- If $\phi_1(\vec{x})$ and $\phi_2(\vec{x})$ are in FO($\mathscr{D}$) then $\phi_1(\vec{x}) \vee \phi_2(\vec{x})$ is logically equivalent to

$$(\phi_1^f \vee \phi_2^f) \wedge (\phi_1 \upharpoonright \phi_1^f) \wedge (\phi_2 \upharpoonright \phi_2^f).$$

  Indeed, suppose that $M \models_X \phi_1 \vee \phi_2$: then, by Lemma 13, $M \models_X \phi_1^f \vee \phi_2^f$. Furthermore, $X = Y \cup Z$ for two Y and Z such that $M \models_Y \phi_1$ and $M \models_Z \phi_2$. Now let $Y' = \{s \in X : M \models_s \phi_1^f\}$ and $Z' = \{s \in X : M \models_s \phi_2^f\}$: by Lemma 13 and Proposition 4 we have that $Y \subseteq Y'$ and that $Z \subseteq Z'$, and therefore by Theorem 14 $M \models_{Y'} \phi_1$ and $M \models_{Z'} \phi_2$. Thus by Lemma 16 $M \models_X (\phi_1 \upharpoonright \phi_1^f)$ and $M \models_X (\phi_2 \upharpoonright \phi_2^f)$, as required.

  Conversely, suppose that $M \models_X (\phi_1^f \vee \phi_2^f) \wedge (\phi_1 \upharpoonright \phi_1^f) \wedge (\phi_2 \upharpoonright \phi_2^f)$. Then let $Y = \{s \in X : M \models_s \phi_1^f\}$ and $Z = \{s \in X : M \models_s \phi_2^f\}$. By Proposition 4 and since $M \models_X \phi_1^f \vee \phi_2^f$, $X = Y \cup Z$; and by Lemma 16, $M \models_Y \phi_1$ and $M \models_Z \phi_2$. So $M \models_X \phi_1 \vee \phi_2$, as required.

- If $\phi(\vec{x}, v) \in$ FO($\mathscr{D}$) then $\exists v \phi(\vec{x}, v)$ is logically equivalent to

$$(\exists v \phi^f(\vec{x}, v)) \wedge \forall v(\phi(\vec{x}, v) \upharpoonright \phi^f(\vec{x}, v)).$$

  Indeed, suppose that $M \models_X \exists v \phi(\vec{x}, v)$. Then by Lemma 13 $M \models_X \exists v \phi^f(\vec{x}, v)$. Furthermore, for some $H : X \to \mathscr{P}(\texttt{Dom}(M)) \backslash \{\emptyset\}$ and for $Y = X[H/v]$ it holds that $M \models_Y \phi(\vec{x}, v)$. Now let $Z = \{h \in X[M/v] : M \models_h \phi^f(\vec{x}, v)\}$. By Proposition 4, $M \models_Z \phi^f(\vec{x}, v)$; and since $Y \subseteq Z$, by Theorem 14 $M \models_Z \phi(\vec{x}, v)$, and therefore by Lemma 16 $M \models_{X[M/v]} (\phi(\vec{x}, v) \upharpoonright \phi^f(\vec{x}, v))$, as required.

  Conversely, suppose that $M \models_X (\exists v \phi^f(\vec{x}, v)) \wedge \forall v(\phi(\vec{x}, v) \upharpoonright \phi^f(\vec{x}, v))$. Then, for all $s \in X$, let $K(s) = \{m \in \texttt{Dom}(M) : M \models_{s[m/v]} \phi^f(\vec{x}, v)\}$. Since $M \models_X \exists v \phi^f(\vec{x}, v)$, $K(s)$ is nonempty for all $s \in X$, and by construction $X[K/v] = \{s \in X[M/v] : M \models_s \phi^f(\vec{x}, v)\}$. Now $M \models_{X[M/v]} (\phi(\vec{x}, v) \upharpoonright \phi^f(\vec{x}, v))$, so by Lemma 16 $M \models_{X[K/v]} \phi(\vec{x}, v)$ and in conclusion $M \models_X \exists v \phi(\vec{x}, v)$.

Applying inductively these two results to all subformulas of some $\phi \in$ FO($\mathscr{D}$) we can obtain some clean ϕ' to which ϕ is equivalent, and this concludes the proof.
□
Finally, the proof of Theorem 11 follows at once from Propositions 18 and 19.

Since, as we saw, the negations of functional and exclusion dependencies are upwards closed, we obtain at once the following corollary:

Corollary 20 *Any sentence of FO($\neq(\cdot,\cdot)$,$\nmid$) (that is, of first-order logic plus negated functional and exclusion dependencies) is equivalent to some first-order sentence.*

4 Adding Constancy Atoms

As we saw in the previous section, upwards closed dependencies can be added to first-order logic without increasing its expressive power (with respect to sentences); and as mentioned in Section 2, this is also true for the (non upwards-closed) constancy dependencies $=(\vec{x})$.

But what if our logic contains both upwards closed *and* constancy dependencies? As we will now see, the conclusion of Theorem 11 remains valid:

Theorem 21 *Let $\mathscr{D}$ be a collection of upwards closed first-order definable dependency conditions. Then for every formula $\phi(\vec{x})$ of*[10] *$FO(=(\cdot),\mathscr{D})$ with free variables in $\vec{x}$ there exists a first-order sentence $\phi^*(R)$, where R is a new $|\vec{x}|$-ary relation symbol, such that*

$$M \models_X \phi(\vec{x}) \Leftrightarrow M, X(\vec{x}) \models \phi^*(R).$$

In particular, every sentence of $FO(\mathscr{D})$ is equivalent to some first-order sentence.

The main ingredient of our proof will be the following lemma.

Lemma 22 *Let $\mathscr{D}$ be any family of dependencies and let $\phi(\vec{x})$ be a $FO(=(\cdot),\mathscr{D})$ formula. Then $\phi(\vec{x})$ is equivalent to some formula of the form $\exists\vec{v}(=(\vec{v}) \wedge \psi(\vec{x},\vec{v}))$, where $\psi \in FO(\mathscr{D})$ contains exactly the same instances of* **D***-atoms (for all $\mathbf{D} \in \mathscr{D}$) that ϕ does, and in the same number.*

The proof of this lemma is by induction on ϕ, and it is entirely analogous to the corresponding proof from [8].

Now we can prove Theorem 21.

Proof:

Let $\phi(\vec{x})$ be a FO$(=(\cdot),\mathscr{D})$-formula. Then by Lemma 22 $\phi(\vec{x})$ is equivalent to some sentence of the form $\exists\vec{v}(=(\vec{v}) \wedge \psi(\vec{x},\vec{v}))$, where $\psi(\vec{x},\vec{v}) \in$ FO$(\mathscr{D})$. But then by Theorem 11 there exists a first-order formula $\psi^*(S)$ such that $M \models_X \psi(\vec{x},\vec{v})$ if and only if $M, X(\vec{x}\vec{v}) \models \psi^*(S)$. Now let $\theta(R,\vec{v})$ be obtained from $\psi^*(S)$ by replacing any $S\vec{y}\vec{z}$ with $R\vec{y} \wedge \vec{z} = \vec{v}$. Since $X[\vec{m}/\vec{v}](\vec{x}\vec{v}) = \{\vec{a}\vec{m} : \vec{a} \in X(\vec{x})\}$ it is easy to see that $M \models_X \exists\vec{v}(=(\vec{v}) \wedge \psi(\vec{x},\vec{v}))$ if and only if $M, X(\vec{x}) \models \exists v\theta(R,\vec{v})$, and this concludes the proof.

$\Box$

5 Possibility, Negated Inclusion and Negated Conditional Independence

By Corollary 20, the negations of exclusion and functional dependence atoms can be added to first-order logic without increasing its power. But what about the negations of inclusion and (conditional) independence? These are of course first-order definable, but they are not upwards closed: indeed, their semantic rules can be given as

TS-$\not\subseteq$: $M \models_X \vec{x} \not\subseteq \vec{y}$ if and only if there is a $s \in X$ such that for all $s' \in X$, $s(\vec{x}) \neq s'(\vec{y})$;

TS-$\not\perp_c$: $M \models_X \vec{x} \not\perp_{\vec{z}} \vec{y}$ if and only if there are $s, s' \in X$ with $s(\vec{z}) = s'(\vec{z})$ and such that for all $s'' \in X$, $s''(\vec{x}\vec{z}) \neq s(\vec{x}\vec{z})$ or $s''(\vec{y}\vec{z}) \neq s(\vec{y}\vec{z})$.

However, we will now prove that, nonetheless, FO$(\neq(\cdot,\cdot), \not\subseteq, \nmid, \not\perp_c)$ is equivalent to FO on the level of sentences. In order to do so, let us first define the following *possibility operator* and prove that it is uniformly definable in FO$(=(\cdot), \neq(\cdot))$:

Definition 23 *Let ϕ be any $FO(\mathscr{D})$ formula, for any choice of $\mathscr{D}$. Then*

TS-$\Diamond$: *$M \models_X \Diamond\phi$ if there exists a $Y \subseteq X$, $Y \neq \emptyset$, such that $M \models_Y \phi$.*

Lemma 24 *Let ϕ be any $FO(\mathscr{D})$ formula, for any $\mathscr{D}$. Then $\Diamond\phi$ is logically equivalent to*

$$\exists u_0 u_1 \exists v(=(u_0) \wedge =(u_1) \wedge (v = u_0 \vee v = u_1) \wedge (\phi \upharpoonright v = u_1) \wedge \neq(v)). \quad (2)$$

[10] Here $=(\cdot)$ represents the class of all constancy dependencies of all arities. But it is easy to see that the one of arity 1 would suffice: indeed, if $\vec{x}$ is $x_1 \ldots x_n$ then $=(\vec{x})$ is logically equivalent to $=(x_1) \wedge \ldots \wedge =(x_n)$.

Proof:
Suppose that there is a $Y \subseteq X$, $Y \neq \emptyset$, such that $M \models_Y \phi$. Then let $0, 1 \in \mathtt{Dom}(M)$ be such that $0 \neq 1$, let $H : X[01/u_0u_1] \to \mathscr{P}(\mathtt{Dom}(M))\backslash\{\emptyset\}$ be such that

$$H(s[01/u_0u_1]) = \begin{cases} \{0,1\} & \text{if } s \in Y; \\ \{0\} & \text{if } s \in X\backslash Y \end{cases}$$

and let $Z = X[01/u_0u_1][H/v]$. Clearly $M \models_Z =\!(u_0) \wedge =\!(u_1) \wedge (v = u_0 \vee v = u_1) \wedge (\phi \upharpoonright v = u_1)$, and it remains to show that $M \models_Z \neq\!(v)$. But by hypothesis Y is nonempty, and therefore there exists a $s \in Y \subseteq X$ such that $\{s[010/u_0u_1v], s[011/u_0u_1v]\} \subseteq Z$. So v is not constant in Z, as required, and X satisfies (2).

Conversely, suppose that X satisfies (2), let 0 and 1 be our choices for u_0 and u_1, and let H be the choice function for v. Then let $Y = \{s \in X : 1 \in H(s[01/u_0u_1])\}$. By locality, Lemma 16 and the fact that $M \models_{X[01H/u_1u_2v]} (\phi \upharpoonright v = u_1)$ we have that $M \models_Y \phi$; and Y is nonempty, since
$M \models_Z (v = u_0 \vee v = u_1) \wedge \neq\!(v)$.
$\square$
It is now easy to see that the negations of inclusion and conditional independence are in FO$(=\!(\cdot), \neq\!(\cdot))$:

Proposition 25 *For all $\vec{x}, \vec{y}$ with $|\vec{x}| = |\vec{y}|$, $\vec{x} \not\subseteq \vec{y}$ is logically equivalent to*

$$\exists \vec{z}(=\!(\vec{z}) \wedge \Diamond(\vec{z} = \vec{x}) \wedge \vec{z} \neq \vec{y}).$$

Proposition 26 *For all $\vec{x}, \vec{y}$ and $\vec{z}$, $\vec{x} \not\perp_{\vec{z}} \vec{y}$ is logically equivalent to*

$$\exists \vec{p}\vec{q}\vec{r}(=\!(\vec{p}\vec{q}\vec{r}) \wedge \Diamond(\vec{p}\vec{r} = \vec{x}\vec{z}) \wedge \Diamond(\vec{q}\vec{r} = \vec{y}\vec{z}) \wedge \vec{p}\vec{q}\vec{r} \neq \vec{x}\vec{y}\vec{z}).$$

Corollary 27 *Every sentence of $FO(\neq\!(\cdot,\cdot), \not\subseteq, \nmid, \not\perp_c)$ is equivalent to some sentence of $FO(=\!(\cdot), \neq\!(\cdot,\cdot), \nmid)$, and hence to some first-order sentence.*

6 Bounded Dependencies and Totality

Now that we know something about upwards closed dependencies, it would be useful to classify them in different categories and prove *non-definability* results between the corresponding extensions of first-order logic. As a first such classification, we introduce the following property:

Definition 28 (Boundedness) *Let κ be a (finite or infinite) cardinal. A dependency condition* **D** *is κ-*bounded *if whenever $M \models_X \mathbf{D}\vec{x}$ there exists a $Y \subseteq X$ with $|Y| \leq \kappa$ such that $M \models_Y \mathbf{D}\vec{x}$.*

We say that **D** *is* bounded *if it is κ-bounded for some κ.*[11]

For example, non-emptiness and intersection are 1-bounded; inconstancy and the negations of functional dependence and exclusion are 2-bounded; and for all finite or infinite κ, κ-bigness is κ-bounded. However, totality is not bounded at all. Indeed, for any κ consider a model M of cardinality greater than κ and take the team $X = \{\emptyset\}[M/x]$. Then $M \models_X \mathtt{All}(x)$, but if $Y \subseteq X$ has cardinality $\leq \kappa$ then $Y(x) \subsetneq \mathtt{Dom}(M)$ and $M \not\models_Y \mathtt{All}(x)$.

As we will now see, the property of boundedness is preserved by the connectives of our language.

Definition 29 (Height of a formula) *Let $\mathscr{D}$ be any family of bounded dependencies. Then for all formulas $\phi \in FO(\mathscr{D})$, the* height $\mathtt{ht}(\phi)$ *of ϕ is defined as follows:*

[11] After a fashion, this notion of boundedness may be thought of as a dual of the notion of *coherence* of [15].

1. *If ϕ is a first-order literal then $\mathtt{ht}(\phi) = 0$;*
2. *If ϕ is a functional dependence atom $\mathbf{D}\vec{x}$ then $\mathtt{ht}(\phi)$ is the least cardinal κ such that $\mathbf{D}$ is κ-bounded;*
3. *If ϕ is of the form $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2$ then $\mathtt{ht}(\phi) = \mathtt{ht}(\psi_1) + \mathtt{ht}(\psi_2)$;*
4. *If ϕ is of the form $\exists v\psi$ or $\forall v\psi$. then $\mathtt{ht}(\phi) = \mathtt{ht}(\psi)$.*

In other words, the height of a formula is the sum of the heights of all instances of dependency atoms occurring in it.

Theorem 30 *Let $\mathscr{D}$ be a family of bounded upwards closed dependencies. Then for all formulas $\phi \in FO(\mathscr{D})$*

$$M \models_X \phi \Rightarrow \exists Y \subseteq X \text{ with } |Y| \leq \mathtt{ht}(\phi) \text{ s.t. } M \models_Y \phi.$$

Proof:
The proof is by induction on ϕ.

1. If ϕ is a first-order literal then $\mathtt{ht}(\phi) = 0$ and it is always the case that $M \models_\emptyset \phi$, as required.
2. If ϕ is an atom $\mathbf{D}\vec{x}$ then the statement follows at once from the definitions of boundedness and height.
3. If ϕ is a disjunction $\psi_1 \vee \psi_2$ then $\mathtt{ht}(\phi) = \mathtt{ht}(\psi_1) + \mathtt{ht}(\psi_2)$. Suppose now that $M \models_X \psi_1 \vee \psi_2$: then $X = X_1 \cup X_2$ for two X_1 and X_2 such that $M \models_{X_1} \psi_1$ and $M \models_{X_2} \psi_2$. This implies that there exist $Y_1 \subseteq X_1$, $Y_2 \subseteq X_2$ such that $M \models_{Y_1} \psi_1$ and $M \models_{Y_2} \psi_2$, $|Y_1| \leq \mathtt{ht}(\psi_1)$ and $|Y_2| \leq \mathtt{ht}(\psi_2)$. But then $Y = Y_1 \cup Y_2$ satisfies $\psi_1 \vee \psi_2$ and has at most $\mathtt{ht}(\psi_1) + \mathtt{ht}(\psi_2)$ elements.
4. If ϕ is a conjunction $\psi_1 \wedge \psi_2$ then, again, $\mathtt{ht}(\phi) = \mathtt{ht}(\psi_1) + \mathtt{ht}(\psi_2)$. Suppose that $M \models_X \psi_1 \wedge \psi_2$: then $M \models_X \psi_1$ and $M \models_X \psi_2$, and therefore by Lemma 13 $M \models_X \psi_1^f$ and $M \models_X \psi_2^f$; and, by induction hypothesis, there exist $Y_1, Y_2 \subseteq X$ with $|Y_1| \leq \mathtt{ht}(\psi_1)$, $|Y_2| \leq \mathtt{ht}(\psi_2)$, $M \models_{Y_1} \psi_1$ and $M \models_{Y_2} \psi_2$. Now let $Y = Y_1 \cup Y_2$: since $Y \subseteq X$, by Proposition 4 $M \models_Y \psi_1^f$ and $M \models_Y \psi_2^f$. But $Y_1, Y_2 \subseteq Y$, and therefore by Theorem 14 $M \models_Y \psi_1$ and $M \models_Y \psi_2$, and in conclusion $M \models_Y \psi_1 \wedge \psi_2$.
5. If ϕ is of the form $\exists v\psi$ then $\mathtt{ht}(\phi) = \mathtt{ht}(\psi)$. Suppose that $M \models_X \exists v\psi$: then for some H we have that $M \models_{X[H/v]} \psi$, and therefore by induction hypothesis there exists a $Z \subseteq X[H/v]$ with $|Z| \leq \mathtt{ht}(\psi)$ such that $M \models_Z \psi$. For any $h \in Z$, let $\mathfrak{f}(h)$ be a $s \in X$ such that $h \in s[H/v] = \{s[m/v] : m \in H(s)\}$,[12] and let $Y = \{\mathfrak{f}(h) : h \in Z\}$. Now $Z \subseteq Y[H/v] \subseteq X[H/v]$. Since $M \models_{X[H/v]} \psi^f$ and $Y[H/v] \subseteq X[H/v]$, we have that $M \models_{Y[H/v]} \psi^f$; and since $M \models_Z \psi$, this implies that $M \models_{Y[H/v]} \psi$ and that $M \models_Y \exists v\psi$. Furthermore $|Y| = |Z| \leq \mathtt{ht}(\psi)$, as required.
6. If ϕ is of the form $\forall v\psi$ then, again, $\mathtt{ht}(\phi) = \mathtt{ht}(\psi)$. Suppose that $M \models_{X[M/v]} \psi$: again, by induction hypothesis there is a $Z \subseteq X[M/v]$ with $|Z| \leq \mathtt{ht}(\psi)$ and such that $M \models_Z \psi$. For any $h \in Y$, let $\mathfrak{g}(h)$ pick some $s \in X$ which agrees with h on all variables except v, and let $Y = \{\mathfrak{g}(h) : h \in Z\}$. Similarly to the previous case, $Z \subseteq Y[M/v] \subseteq X[M/v]$: therefore, since $M \models_{X[M/v]} \psi^f$ we have that $M \models_{Y[M/v]} \psi^f$, and since $M \models_Z \psi$ we have that $M \models_{Y[M/v]} \psi$. So in conclusion $M \models_Y \forall v\psi$, as required, and $|Y| = |Z| \leq n$.

□

Even though constancy atoms are not upwards closed, it is possible to extend this result to FO$(=(\cdot), \mathscr{D})$. Indeed, constancy atoms are trivially 0-bounded, since the empty team always satisfies them, and

[12]Since $Z \subseteq X[H/v]$, such a s always exists. Of course, there may be multiple ones; in that case, we pick one arbitrarily.

Corollary 31 *Let $\mathscr{D}$ be a family of upwards closed bounded dependencies. Then for all $\phi \in FO(=(\cdot), \mathscr{D})$*

$$M \models_X \phi \Rightarrow \exists Y \subseteq X \text{ with } |Y| \leq \mathtt{ht}(\phi) \text{ s.t. } M \models_Y \phi.$$

Proof:
Let $\phi \in \text{FO}(=(\cdot), \mathscr{D})$: then by Lemma 22 ϕ is equivalent to some formula of the form $\exists \vec{v}(=(\vec{v}) \wedge \psi)$, where ψ does not contain constancy atoms and $\mathtt{ht}(\psi) = \mathtt{ht}(\phi)$. Now suppose that $M \models_X \phi$: then, for some choice of elements $\vec{m} \in \mathtt{Dom}(M)^{|\vec{v}|}$, $M \models_{X[\vec{m}/\vec{v}]} \psi$. Now by Theorem 30 there exists a $Z \subseteq X[\vec{m}/\vec{v}]$, with $|Z| \leq \mathtt{ht}(\psi)$, such that $M \models_Z \psi$; and Z is necessarily of the form $Y[\vec{m}/\vec{v}]$ for some $Y \subseteq X$ with $|Y| = |Z| \leq \mathtt{ht}(\psi)$. But then $M \models_Y \exists \vec{v}(=(\vec{v}) \wedge \psi)$, as required.
□

This result allows us to prove at once a number of nondefinability results concerning upwards closed dependencies. For example, it is now easy to see that

Corollary 32 *Let $\mathscr{D}$ be a family of upwards closed bounded dependencies. Then the totality dependency $\mathtt{All}$ is not definable in $FO(=(\cdot), \mathscr{D})$. In particular, totality atoms cannot be defined by means of the negations of inclusion, exclusion, functional dependence and independence atoms.*

Corollary 33 *Let $\mathscr{D}$ be a family of κ-bounded upwards closed dependencies and let $\kappa' > \kappa$ be infinite. Then κ'-bigness is not definable in $FO(=(\cdot), \mathscr{D})$.*

Corollary 34 *Let $\mathbf{D}$ be a k-bounded upwards closed dependency, and let $n > k$. If $\phi(\vec{x})$ of $FO(=(\cdot), \mathbf{D})$ characterizes n-bigness, in the sense that for all M and X*

$$M \models_X \phi(\vec{x}) \Leftrightarrow |X(\vec{x})| \geq n,$$

then $\phi(\vec{x})$ contains at least $\lceil \frac{n}{k} \rceil$ instances of $\mathbf{D}$.

7 Conclusions and Further Work

In this work we discovered a surprising asymmetry between downwards closed and upwards closed first-order definable dependency conditions: whereas, as it was known since [19], the former can bring the expressive power of a logic with team semantics beyond the first order, the latter cannot do so by their own or even together with constancy atoms. As a consequence, the negations of the principal dependency notions studied so far in team semantics can all be added to first-order logic without increasing its expressive power.

Our original question was: how much can we get away with adding to the team semantics of first-order logic before ending up in a higher order logic? The answer, it is now apparent, is *quite a lot*. This demonstrates that team semantics is useful not only (as it has been employed so far) as a formalism for the study of very expressive extensions of first-order logic, but also as one for that of more treatable ones.

Much of course remains to be done. The notion of boundedness of Section 6 allowed us to find some non-definability results between our extensions; but the classification of these extensions is far from complete. In particular, it would be interesting to find necessary and sufficient conditions for FO($\mathscr{D}$) to be equivalent to FO over sentences. The complexity-theoretic properties of these logics, or of fragments thereof, also deserve further investigation.

Another open issue concerns the development of sound and complete proof systems for our logics. Of course, one can check whether a theory T implies a formula ϕ simply by using Theorems 11 and 21 to translate everything in first-order logic and then use one of the many well-understood proof systems for it; but nonetheless, it could be very informative to find out directly which logical laws our formalisms obey.

Acknowledgments The author thanks the referees for a number of useful suggestions and corrections.

References

[1] Samson Abramsky & Jouko Väänänen (2013): *Dependence logic, social choice and quantum physics*. In preparation.

[2] William W. Armstrong (1974): *Dependency Structures of Data Base Relationships.* In: *Proc. of IFIP World Computer Congress*, pp. 580–583.

[3] Marco A. Casanova & Vânia M. P. Vidal (1983): *Towards a sound view integration methodology*. In: *Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, PODS '83, ACM, New York, NY, USA, pp. 36–47, doi:`10.1145/588058.588065`.

[4] Fredrik Engström (2012): *Generalized quantifiers in dependence logic*. *Journal of Logic, Language and Information* 21(3), pp. 299–324, doi:`10.1007/s10849-012-9162-4`.

[5] Ronald Fagin (1977): *Multivalued dependencies and a new normal form for relational databases*. *ACM Transactions on Database Systems* 2, pp. 262–278, doi:`10.1145/320557.320571`.

[6] Ronald Fagin (1981): *A normal form for relational databases that is based on domains and keys*. *ACM Transactions on Database Systems* 6, pp. 387–415, doi:`10.1145/319587.319592`.

[7] Pietro Galliani (2012): *The Dynamics of Imperfect Information*. Ph.D. thesis, University of Amsterdam. Available at `http://dare.uva.nl/record/425951`.

[8] Pietro Galliani (2012): *Inclusion and exclusion dependencies in team semantics: On some logics of imperfect information*. *Annals of Pure and Applied Logic* 163(1), pp. 68 – 84, doi:`10.1016/j.apal.2011.08.005`.

[9] Pietro Galliani & Lauri Hella (2013): *Inclusion Logic and Fixed Point Logic*. ArXiv:1304.4267.

[10] Pietro Galliani & Jouko Väänänen (2013): *On Dependence Logic*. ArXiv:1305.5948.

[11] Dan Geiger, Azaria Paz & Judea Pearl (1991): *Axioms and algorithms for inferences involving probabilistic independence*. *Information and Computation* 91(1), pp. 128 – 141, doi:`10.1016/0890-5401(91)90077-F`.

[12] Erich Grädel & Jouko Väänänen (2013): *Dependence and Independence*. *Studia Logica* 101(2), pp. 399–410, doi:`10.1007/s11225-013-9479-2`.

[13] Jaakko Hintikka & Gabriel Sandu (1989): *Informational independence as a semantic phenomenon*. In J.E Fenstad, I.T Frolov & R. Hilpinen, editors: *Logic, methodology and philosophy of science*, Elsevier, pp. 571–589, doi:`10.1016/S0049-237X(08)70066-1`.

[14] Wilfrid Hodges (1997): *Compositional Semantics for a Language of Imperfect Information*. *Journal of the Interest Group in Pure and Applied Logics* 5 (4), pp. 539–563, doi:`10.1093/jigpal/5.4.539`.

[15] Jarmo Kontinen (2013): *Coherence and Computational Complexity of Quantifier-free Dependence Logic Formulas*. *Studia Logica* 101(2), pp. 267–291, doi:`10.1007/s11225-013-9481-8`.

[16] Juha Kontinen & Ville Nurmi (2009): *Team Logic and Second-Order Logic*. In Hiroakira Ono, Makoto Kanazawa & Ruy de Queiroz, editors: *Logic, Language, Information and Computation*, *Lecture Notes in Computer Science* 5514, Springer Berlin / Heidelberg, pp. 230–241, doi:`10.1007/978-3-642-02261-6_19`.

[17] Antti Kuusisto (2013): *Defining a Double Team Semantics for Generalized Quantifiers (Extended Version)*. Available at `https://uta17-kk.lib.helsinki.fi/bitstream/handle/10024/68064/defining_double_team_2013.pdf?sequence=1`. Manuscript.

[18] Allen L. Mann, Gabriel Sandu & Merlijn Sevenster (2011): *Independence-Friendly Logic: A Game-Theoretic Approach*. Cambridge University Press, doi:`10.1017/CBO9780511981418`.

[19] Jouko Väänänen (2007): *Dependence Logic*. Cambridge University Press, doi:`10.1017/CBO9780511611193`.

[20] Jouko Väänänen & Fan Yang (2013): *Propositional dependence logic*. Manuscript.

Profile Trees for Büchi Word Automata, with Application to Determinization*

Seth Fogarty
Computer Science Department
Trinity University

Orna Kupferman
School of Computer Science and Engineering
Hebrew University of Jerusalem

Moshe Y. Vardi
Department of Computer Science
Rice University

Thomas Wilke
Institut für Informatik
Christian-Albrechts-Universität zu Kiel

The determinization of Büchi automata is a celebrated problem, with applications in synthesis, probabilistic verification, and multi-agent systems. Since the 1960s, there has been a steady progress of constructions: by McNaughton, Safra, Piterman, Schewe, and others. Despite the proliferation of solutions, they are all essentially ad-hoc constructions, with little theory behind them other than proofs of correctness. Since Safra, all optimal constructions employ trees as states of the deterministic automaton, and transitions between states are defined operationally over these trees. The operational nature of these constructions complicates understanding, implementing, and reasoning about them, and should be contrasted with complementation, where a solid theory in terms of automata run DAGs underlies modern constructions.

In 2010, we described a *profile*-based approach to Büchi complementation, where a profile is simply the history of visits to accepting states. We developed a structural theory of profiles and used it to describe a complementation construction that is deterministic in the limit. Here we extend the theory of profiles to prove that every run DAG contains a *profile tree* with at most a finite number of infinite branches. We then show that this property provides a theoretical grounding for a new determinization construction where macrostates are doubly preordered sets of states. In contrast to extant determinization constructions, transitions in the new construction are described declaratively rather than operationally.

1 Introduction

Büchi automata were introduced in the context of decision problems for second-order arithmetic [3]. These automata constitute a natural generalization of automata over finite words to languages of infinite words. Whereas a run of an automaton on finite words is accepting if the run ends in an accepting state, a run of a Büchi automaton is accepting if it visits an accepting state infinitely often.

Determinization of nondeterministic automata is a fundamental problem in automata theory, going back to [19]. Determinization of Büchi automata is employed in many applications, including synthesis of reactive systems [18], verification of probabilistic systems [4, 25], and reasoning about multi-agent systems [2]. Nondeterministic automata over finite words can be determinized with a simple, although exponential, *subset construction* [19], where a state in the determinized automaton is a set of states of the input automaton. Nondeterministic Büchi automata, on the other hand, are not closed under determinization, as deterministic Büchi automata are strictly less expressive than their nondeterministic

*Work supported in part by NSF grants CNS 1049862 and CCF-1139011, by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering", by a gift from Intel, by BSF grant 9800096, and by a stipend from Trinity University. A full version, with appendices and missing proofs, is available at `http://www.cs.trinity.edu/~sfogarty/papers/gandalf13rj.pdf`

Gabriele Puppis, Tiziano Villa (Eds.): Fourth International Symposium on Games, Automata, Logics and Formal Verification
EPTCS 119, 2013, pp. 107–121, doi:10.4204/EPTCS.119.11

counterparts [13]. Thus, a determinization construction for Büchi automata must result in automata with a more powerful acceptance condition, such as Muller [15], Rabin [20], or parity conditions [9, 17].

The first determinization construction for Büchi automata was presented by McNaughton, with a doubly-exponential blowup [15]. In 1988, Safra introduced a singly exponential construction [20], matching the lower bound of $n^{O(n)}$ [14]. Safra's construction encodes a state of the determinized automaton as a labeled tree, now called a *Safra tree*, of sets of states of the input Büchi automaton. Subsequently, Safra's construction was improved by Piterman, who simplified the use of tree-node labels [17], and by Schewe, who moved the acceptance conditions from states to edges [22]. In a separate line of work, Muller and Schupp proposed in 1995 a different singly exponential determinization construction, based on *Muller-Schupp trees* [16], which was subsequently simplified by Kähler and Wilke [9].

Despite the proliferation of Büchi determinization constructions, even in their improved and simplified forms all constructions are essentially ad-hoc, with little theory behind them other than correctness proofs. These constructions rely on the encoding of determinized-automaton states as finite trees. They are operational in nature, with transitions between determinized-automaton states defined "horticulturally," as a sequence of operations that grow trees and then prune them in various ways. The operational nature of these constructions complicates understanding, implementing, and reasoning about them [1, 23], and should be contrasted with complementation, where an elegant theory in terms of automata run DAGs underlies modern constructions [8, 11, 21]. In fact, the difficulty of determinization has motivated attempts to find determinization-free decision procedures [12] and works on determinization of fragments of LTL [10].

In a recent work [6], we introduced the notion of *profiles* for nodes in the run DAG. We began by labeling accepting nodes of the DAG by 1 and non-accepting nodes by 0, essentially recording visits to accepting states. The profile of a node is the lexicographically *maximal* sequence of labels along paths of the run DAG that lead to that node. Once profiles and a lexicographic order over profiles were defined, we removed from the run DAG edges that do not contribute to profiles. In the pruned run DAG, we focused on lexicographically maximal runs. This enabled us to define a novel, profile-based Büchi complementation construction that yields *deterministic-in-the-limit* automata: one in which every accepting run of the complementing automaton is eventually deterministic [6] A state in the complementary automaton is a set of states of the input nondeterministic automaton, augmented with the preorder induced by profiles. Thus, this construction can be viewed as an augmented subset construction.

In this paper, we develop the theory of profiles further, and consider the equivalence classes of nodes induced by profiles, in which two nodes are in the same class if they have the same profile. We show that profiles turn the run DAG into a *profile tree*: a binary tree of bounded width over the equivalence classes. The profile tree affords us a novel singly exponential Büchi determinization construction. In this profile-based determinization construction, a state of the determinized automaton is a set of states of the input automaton, augmented with *two* preorders induced by profiles. Note that while a Safra tree is finite and encodes a single level of the run DAG, our profile tree is infinite and encodes the entire run DAG, capturing the accepting or rejecting nature of all paths. Thus, while a state in a traditional determinization construction corresponds to a Safra tree, a state in our deterministic automaton corresponds to a single level in the profile tree.

Unlike previous Büchi determinization constructions, transitions between states of the determinized automaton are defined declaratively rather than operationally. We believe that the declarative character of the new construction will open new lines of research on Büchi determinization. For Büchi complementation, the theory of run DAGs [11] led not only to tighter constructions [8, 21], but also to a rich body of work on heuristics and optimizations [5, 7]. We foresee analogous developments in research on Büchi determinization.

2 Preliminaries

This section introduces the notations and definitions employed in our analysis.

2.1 Relations on Sets

Given a set R, a binary relation $\leq$ over R is a *preorder* if $\leq$ is reflexive and transitive. A *linear preorder* relates every two elements: for every $r_1, r_2 \in R$ either $r_1 \leq r_2$, or $r_2 \leq r_1$, or both. A relation is *antisymmetric* if $r_1 \leq r_2$ and $r_2 \leq r_1$ implies $r_1 = r_2$. A preorder that is antisymmetric is a *partial order*. A linear partial order is a *total order*. Consider a partial order $\leq$. If for every $r \in R$, the set $\{r' \mid r' \leq r\}$ of smaller elements is totally ordered by $\leq$, then we say that $\leq$ is a *tree order*. The equivalence class of $r \in R$ under $\leq$, written $[r]$, is $\{r' \mid r' \leq r \text{ and } r \leq r'\}$. The equivalence classes under a linear preorder form a totally ordered partition of R. Given a set R and linear preorder $\leq$ over R, define the minimal elements of R as $\mathtt{min}_{\leq}(R) = \{r_1 \in R \mid r_1 \leq r_2 \text{ for all } r_2 \in R\}$. Note that $\mathtt{min}_{\leq}(R)$ is either empty or an equivalence class under $\leq$. Given a non-empty set R and a total order $\leq$, we instead define $\mathtt{min}_{\leq}(R)$ as the the unique minimal element of R.

Given two finite sets R and R' where $|R| \leq |R'|$, a linear preorder $\leq$ over R, and a total order $<'$ over R', define the $\langle \leq, <' \rangle$*-minjection* from R to R' to be the function $\mathtt{mj}$ that maps all the elements in the k-th equivalence class of R to the k-th element of R'. The number of equivalence classes is at most $|R|$, and thus at most $|R'|$. If $\leq$ is also a total order, than the $\langle \leq, <' \rangle$-minjection is also an injection.

Example 2.1. Let $R = \mathbb{Q}$ and $R' = \mathbb{Z}$ be the sets of rational numbers and integers, respectively. Define the linear preorder $\leq_1$ over $\mathbb{Q}$ by $x \leq_1 x'$ iff $\lfloor x \rfloor \leq \lfloor x' \rfloor$, and the total order $<_2$ over $\mathbb{Z}$ by $x <_2 x'$ if $x < x'$. Then, the $\langle \leq_1, <_2 \rangle$-minjection from $\mathbb{Q}$ to $\mathbb{Z}$ maps a rational number x to $\lfloor x \rfloor$.

2.2 ω-Automata

A *nondeterministic ω-automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, Q^{in}, \rho, \alpha \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $Q^{in} \subseteq Q$ is a set of initial states, $\rho \colon Q \times \Sigma \to 2^Q$ is a nondeterministic transition relation, and α is an acceptance condition defined below. An automaton is *deterministic* if $|Q^{in}| = 1$ and, for every $q \in Q$ and $\sigma \in \Sigma$, we have $|\rho(q, \sigma)| = 1$. For a function $\delta \colon Q \times \Sigma \to 2^Q$, we lift δ to sets R of states in the usual fashion: $\delta(R, \sigma) = \bigcup_{r \in R} \delta(r, \sigma)$. Further, we define the inverse of δ, written δ^{-1}, to be $\delta^{-1}(r, \sigma) = \{q \mid r \in \delta(q, \sigma)\}$.

A *run* of an ω-automaton $\mathcal{A}$ on a word $w = \sigma_0 \sigma_1 \cdots \in \Sigma^\omega$ is an infinite sequence of states $q_0, q_1, \ldots \in Q^\omega$ such that $q_0 \in Q^{in}$ and, for every $i \geq 0$, we have that $q_{i+1} \in \rho(q_i, \sigma_i)$. Correspondingly, a *finite run* of $\mathcal{A}$ to q on $w = \sigma_0 \cdots \sigma_{n-1} \in \Sigma^*$ is a finite sequence of states $p_0, \ldots, p_n$ such that $p_0 \in Q^{in}$, $p_n = q$, and for every $0 \leq i < n$ we have $p_{i+1} \in \rho(p_i, \sigma_i)$.

The acceptance condition α determines if a run is *accepting*. If a run is not accepting, we say it is *rejecting*. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on w. The words accepted by $\mathcal{A}$ form the *language* of $\mathcal{A}$, denoted by $L(\mathcal{A})$. For a *Büchi automaton*, the acceptance condition is a set of states $F \subseteq Q$, and a run $q_0, q_1, \ldots$ is accepting iff $q_i \in F$ for infinitely many i's. For convenience, we assume $Q^{in} \cap F = \emptyset$. For a *Rabin automaton*, the acceptance condition is a sequence $\langle G_0, B_0 \rangle, \ldots, \langle G_k, B_k \rangle$ of pairs of sets of states. Intuitively, the sets G are "good" conditions, and the sets B are "bad" conditions. A run $q_o, q_1, \ldots$ is accepting iff there exists $0 \leq j \leq k$ so that $q_i \in G_j$ for infinitely many i's, while $q_i \in B_j$ for only finitely many i's. Our focus in this paper is on nondeterministic Büchi automata on words (NBW) and deterministic Rabin automata on words (DRW).

2.3 Safra's Determinization Construction

This section presents Safra's determinization construction, using the exposition in [17]. Safra's construction takes an NBW and constructs an equivalent DRW. Intuitively, a state in this construction is a tree of subsets. Every node in the tree is labeled by the states it follows. The label of a node is a strict superset of the union of labels of its descendants, and the labels of siblings are disjoint. Children of a node are ordered by "age". Let $\mathcal{A} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$ be an NBW, $n = |Q|$, and $V = \{0, \ldots, n-1\}$.

Definition 2.2. [17] A *Safra tree* over $\mathcal{A}$ is a tuple $t = \langle N, r, p, \psi, l, G, B \rangle$ where:

- $N \subseteq V$ is a set of nodes.
- $r \in N$ is the root node.
- $p : (N \setminus \{r\}) \to N$ is the parent function over $N \setminus \{r\}$.
- ψ is a partial order defining 'older than' over siblings.
- $l : N \to 2^Q$ is a labeling function from nodes to non-empty sets of states. The label of every node is a proper superset of the union of the labels of its sons. The labels of two siblings are disjoint.
- $G, B \subseteq V$ are two disjoint subsets of V.

The only way to move from one Safra tree to the next is through a sequence of "horticultural" operations, growing the tree and then pruning it to ensure that the above invariants hold.

Definition 2.3. Define the DRW $D^S(\mathcal{A}) = \langle \Sigma, Q_S, \rho_S, t_0, \alpha \rangle$ where:

- Q_S is the set of Safra trees over $\mathcal{A}$.
- $t_0 = \langle \{0\}, 0, \emptyset, \emptyset, l_0, \emptyset, \{1, \ldots, n-1\} \rangle$ where $l_0(0) = Q^{in}$
- For $t = \langle N, r, p, \psi, l, G, B \rangle \in Q_S$ and $\sigma \in \Sigma$, the tree $t' = \rho_S(t, \sigma)$ is the result of the following sequence of operations. We temporarily use a set V' of names disjoint from V. Initially, let $t' = \langle N', r', p', \psi', l', G', B' \rangle$ where $N' = N$, $r' = r$, $p' = p$, $\psi' = \psi$, l' is undefined, and $G' = B' = \emptyset$.
  1. For every $v \in N'$, let $l'(v) = \rho(l(v), \sigma)$.
  2. For every $v \in N'$ such that $l'(v) \cap F \neq \emptyset$, create a new node $v' \in V'$ where: $p(v') = v$; $l'(v') = l'(v) \cap F$; and for every $w' \in V'$ where $p(w') = v$ add (w', v') to ψ.
  3. For every $v \in N'$ and $q \in l'(v)$, if there is a $w \in N'$ such that $(w, v) \in \psi$ and $q \in l'(w)$, then remove q from $l'(v)$ and, for every descendant v' of v, remove q from $l'(v')$.
  4. Remove all nodes with empty labels.
  5. For every $v \in N'$, if $l'(v) = \bigcup \{l'(v') \mid p'(v') = v\}$ remove all children of v, add v to G.
  6. Add all nodes in $V \setminus N'$ to B.
  7. Change the nodes in V' to unused nodes in V.
- $\alpha = \{\langle G_0, B_0 \rangle, \ldots, \langle G_{n-1}, B_{n-1} \rangle\}$, where:
  - $G_i = \{\langle N, r, p, \psi, l, G, B \rangle \in Q_S \mid i \in G\}$
  - $B_i = \{\langle N, r, p, \psi, l, G, B \rangle \in Q_S \mid i \in B\}$

Theorem 2.4. [20] *For an NBW $\mathcal{A}$ with n states, $L(D^S(\mathcal{A})) = L(\mathcal{A})$ and $D^S(\mathcal{A})$ has $n^{O(n)}$ states.*

3 From Run DAGs to Profile Trees

In this section, we present a framework for simultaneously reasoning about all runs of a Büchi automaton on a word. We use a DAG to encode all possible runs, and give each node in this DAG a profile based on its history. The lexicographic order over profiles induces a preorder $\preceq_i$ over the nodes on level i of the run DAG. Using $\preceq_i$, we prune the edges of the run DAG, and derive a binary tree of bounded width. Throughout this paper we fix an NBW $\mathcal{A} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$ and an infinite word $w = \sigma_0 \sigma_1 \cdots$.

3.1 Run DAGs and Profiles

The runs of $\mathcal{A}$ on w can be arranged in an infinite DAG $G = \langle V, E\rangle$, where

- $V \subseteq Q \times \mathbb{N}$ is such that $\langle q, i\rangle \in V$ iff there is a finite run of $\mathcal{A}$ to q on $\sigma_0 \cdots \sigma_{i-1}$.
- $E \subseteq \bigcup_{i\geq 0}(Q\times\{i\})\times(Q\times\{i+1\})$ is such that $E(\langle q,i\rangle, \langle q', i+1\rangle)$ iff $\langle q,i\rangle \in V$ and $q' \in \rho(q, \sigma_i)$.

The DAG G, called the *run* DAG *of $\mathcal{A}$ on w*, embodies all possible runs of $\mathcal{A}$ on w. We are primarily concerned with *initial paths* in G: paths that start in $Q^{in} \times \{0\}$. A node $\langle q,i\rangle$ is an F-node if $q \in F$, and a path in G is *accepting* if it is both initial and contains infinitely many F-nodes. An accepting path in G corresponds to an accepting run of $\mathcal{A}$ on w. If G contains an accepting path, we say that G is *accepting*; otherwise it is *rejecting*. Let G' be a sub-DAG of G. For $i \geq 0$, we refer to the nodes in $Q \times \{i\}$ as *level i* of G'. Note that a node on level $i+1$ has edges only from nodes on level i. We say that G' has *bounded width of degree c* if every level in G' has at most c nodes. By construction, G has bounded width of degree $|Q|$.

Consider the run DAG $G = \langle V, E\rangle$ of $\mathcal{A}$ on w. Let $f : V \to \{0,1\}$ be such that $f(\langle q,i\rangle) = 1$ if $q \in F$ and $f(\langle q,i\rangle) = 0$ otherwise. Thus, f labels F-nodes by 1 and all other nodes by 0. The *profile* of a path in G is the sequence of labels of nodes in the path. We define the profile of a node to be the lexicographically maximal profile of all initial paths to that node. Formally, the profile of a finite path $b = v_0, v_1, \ldots, v_n$ in G, written h_b, is $f(v_0)f(v_1)\cdots f(v_n)$, and the profile of an infinite path $b = v_0, v_1, \ldots$ is $h_b = f(v_0)f(v_1)\cdots$. Finally, the profile of a node v, written h_v, is the lexicographically maximal element of $\{h_b \mid b \text{ is an initial path to } v\}$.

The lexicographic order of profiles induces a linear preorder over nodes on every level of G. We define a sequence of linear preorders $\preceq_i$ over the nodes on level i of G as follows. For nodes u and v on level i, let $u \prec_i v$ if $h_u < h_v$, and $u \approx_i v$ if $h_u = h_v$. We group nodes by their equivalence classes under $\preceq_i$. Since the final element of a node's profile is 1 if and only if the node is an F-node, all nodes in an equivalence class agree on membership in F. Call an equivalence class an F-class when all members are F-nodes, and a non-F-class when none of its members are F-nodes. When a state can be reached by two finite runs, a node will have multiple incoming edges in G. We now remove from G all edges that do not contribute to profiles. Formally, define the pruned run DAG $G' = \langle V, E'\rangle$ where $E' = \{\langle u,v\rangle \in E \mid \text{for every } u' \in V, \text{ if } \langle u',v\rangle \in E \text{ then } u' \preceq_{|u|} u\}$. Note that the set of nodes in G and G' are the same, and that an edge is removed from E' only when there is another edge to its destination.

Lemma 3.1 states that, as we have removed only edges that do not contribute to profiles, nodes derive their profiles from their parents in G'.

Lemma 3.1. [6] *For two nodes u and u' in V, if $\langle u,u'\rangle \in E'$, then $h_{u'} = h_u 0$ or $h_{u'} = h_u 1$.*

While nodes with different profiles can share a child in G, Lemma 3.2 precludes this in G'.

Lemma 3.2. *Consider nodes u and v on level i of G' and nodes u' and v' on level $i+1$ of G'. If $\langle u,u'\rangle \in E'$, $\langle v,v'\rangle \in E'$, and $u' \approx_{i+1} v'$, then $u \approx_i v$.*

Proof: Since $u' \approx_{i+1} v'$, we have $h_{u'} = h_{v'}$. If u' is an F-node, then v' is an F-node and the last letter in both $h_{u'}$ and $h_{v'}$ is 1. By Lemma 3.1 we have $h_u 1 = h_{u'} = h_{v'} = h_v 1$. If u' and v' are non-F-nodes, then we have $h_u 0 = h_{u'} = h_{v'} = h_v 0$. In either case, $h_u = h_v$ and $u \approx_i v$. ☐

Finally, we have that G' captures the accepting or rejecting nature of G. This result was employed to provide deterministic-in-the-limit complementation in [6]

Theorem 3.3. [6] *The pruned run* DAG *G' of an NBW $\mathcal{A}$ on a word w is accepting iff $\mathcal{A}$ accepts w.*

3.2 The Profile Tree

Using profiles, we define the *profile tree* T, which we show to be a binary tree of bounded width that captures the accepting or rejecting nature of the pruned run DAG G'. The nodes of T are the equivalence classes $\{[u] \mid u \in V\}$ of $G' = \langle V, E' \rangle$. To remove confusion, we refer to the nodes of T as *classes* and use and U and W for classes in T, while reserving u and v for nodes in G or G'. The edges in T are induced by those in G' as expected: for an edge $\langle u, v \rangle \in E'$, the class $[v]$ is the child of $[u]$ in T. A class W is a *descendant* of a class U if there is a, possibly empty, path from U to W.

Theorem 3.4. *The profile tree T of an n-state NBW $\mathcal{A}$ on an infinite word w is a binary tree whose width is bounded by n.*

Proof: That T has bounded width follows from the fact that a class on level i contains at least one node on level i of G, and G is of bounded width of degree $|Q|$. To prove that every class has one parent, for a class W let $U = \{u \mid \text{there is } v \in W \text{ such that } \langle u, v \rangle \in E'\}$. Lemma 3.2 implies that U is an equivalence class, and is the sole parent of W. To show that T has a root, note that as $Q^{in} \cap F = \emptyset$, all nodes on the first level of G have profile 0, and every class descends from this class of nodes with profile 0. Finally, as noted Lemma 3.1 entails that a class U can have at most two children: the class with profile $h_U 1$, and the class with profile $h_U 0$. Thus T is binary. □

A *branch* of T is a finite or infinite initial path in T. Since T is a tree, two branches share a prefix until they *split*. An infinite branch is *accepting* if it contains infinitely many F-classes, and *rejecting* otherwise. An infinite rejecting branch must reach a suffix consisting only of non-F-classes. A class U is called *finite* if it has finitely many descendants, and a finite class U *dies out* on level k if it has a descendant on level $k-1$, but none on level k. Say T is *accepting* if it contains an accepting branch, and *rejecting* if all branches are rejecting.

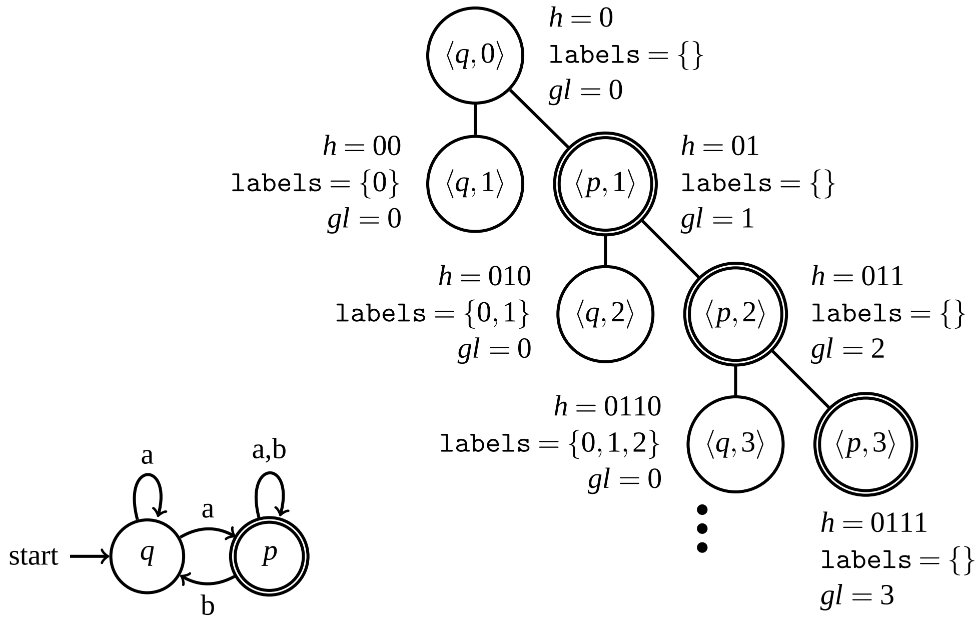
As all members of a class share a profile, we define the profile h_U of a class U to be h_u for some node $u \in U$. We extend the function f to classes, so that $f(U) = 1$ if U is an F-class, and $f(U) = 0$ otherwise. We can then define the profile of an infinite branch $b = U_0, U_1, \ldots$ to be $h_b = f(U_0) f(U_1) \cdots$. For two classes U and W on level i, we say that $U \prec_i W$ if $h_U < h_W$. For two infinite branches b and b', we say that $b \prec b'$ if $h_b < h_{b'}$. Note that $\prec_i$ is a total order over the classes on level i, and that $\prec$ is a total order over the set of infinite branches.

As proven above, a class U has at most two children: the class of F-nodes with profile $h_U 1$, and the class of non-F-nodes with profile $h_U 0$. We call the first class the F-child of U, and the second class the non-F-child of U. While the DAG G' can have infinitely many infinite branches, bounding the width of a tree also bounds the number of infinite branches it may have.

Corollary 3.5. *The profile tree T of an NBW $\mathcal{A}$ on an infinite word w has a finite number of infinite branches.*

Example 3.6. Consider, for example, the NBW in Figure 1.(a) and the first four levels of a tree of equivalence classes in Figure 1.(b). This tree corresponds to all runs of the NBW on the word ab^{ω}. There is only one infinite branch, $\{\langle q, 0 \rangle\}, \{\langle p, 1 \rangle\}, \{\langle p, 2 \rangle\}, \ldots$, which is accepting. The set of labels and the global labeling gl are explained below, in Section 4.1.

We conclude this section with Theorem 3.7, which enables us to reduce the search for an accepting path in G' to a search for an accepting branch in T.



(a) An automaton **(b)** T for automaton (a) on ab^ω.

Figure 1: An automaton and tree of classes. Each class is a singleton set, brackets are omitted for brevity. F-classes are circled twice. Each class is labeled with its profile h, as well as the set `labels` and the global label gl as defined in Section 4.1.

Theorem 3.7. *The profile tree T of an NBW $\mathcal{A}$ on an infinite word w is accepting iff $\mathcal{A}$ accepts w.*

Proof: If $w \in L(\mathcal{A})$, then by Theorem 3.3 we have that G' contains an accepting path $u_0, u_1, \ldots$. This path gives rise to an accepting branch $[u_0], [u_1], \ldots$ in T. In the other direction, if T has an accepting branch $U_0, U_1, \ldots$, consider the infinite subgraph of G' consisting only of the nodes in U_i, for $i > 0$. For every $i > 0$ there exists $u_i \in U_i$ and $u_{i+1} \in U_{i+1}$ so that $E'(u_i, u_{i+1})$. Because no node is orphaned in G', Lemma 3.2 implies that every node in U_{i+1} has a parent in U_i, thus this subgraph is connected. As each node has degree of as most n, König's Lemma implies that there is an infinite initial path $u_0, u_1, \ldots$ through this subgraph. Further, at every level i where U_i is an F-class, we have that $u_i \in F$, and thus this path is accepting and $w \in L(\mathcal{A})$. $\square$

4 Labeling

In this section we present a method of deterministically labeling the classes in T with integers, so we can determine if T is accepting by examining the labels. Each label m represents the proposition that the lexicographically minimal infinite branch through the first class labeled with m is accepting. On each level we give the label m to the lexicographically minimal descendant, on any branch, of this first class labeled with m. We initially allow the use of global information about T and an unbounded number of labels. We then show how to determine the labeling using bounded information about each level of T, and how to use a fixed set of labels.

4.1 Labeling T

We first present a labeling that uses an unbounded number of labels and global information about T. We call this labeling the *global labeling*, and denote it with gl. For a class U on level i of T, and a class W on level j, we say that W is *before* U if $j < i$ or $j = i$ and $W \prec_i U$. For each label m, we refer to the first class labeled m as $\texttt{first}(m)$. Formally, $U = \texttt{first}(m)$ if U is labeled m and, for all classes W before U, the label of W is not m. We define the labeling function gl inductively over the nodes of T. For the initial class $U_0 = \{\langle q,0\rangle \mid q \in Q^{in}\}$ with profile 0, let $gl(U_0) = 0$.

Each label m follows the lexicographically minimal child of $\texttt{first}(m)$ on every level. When a class with label m has two children, we are not certain which, if either, is part of an infinite branch. We are thus conservative, and follow the non-F-child. If the non-F-child dies out, we revise our guess and move to a descendant of the F-child. For a label m and level i, let the *lexicographically minimal descendant* of m on level i, written $\texttt{lmd}(m,i)$, be $\texttt{min}_{\preceq}(\{W \mid W \text{ is a descendant of } \texttt{first}(m) \text{ on level } i\})$: the class with the minimal profile among all the descendants of $\texttt{first}(m)$ on level i. For a class U on level i, define $\texttt{labels}(U) = \{m \mid U = \texttt{lmd}(m,i)\}$ as the set of valid labels for U. When labelling U, if U has more than one valid label, we give it the smallest label, which corresponds to the earliest ancestor. If $\texttt{labels}(U)$ is empty, U is given an unused label one greater than the maximum label occurring earlier in T.

Definition 4.1. $gl(U) = \begin{cases} \texttt{min}(\texttt{labels}(U)) & \text{if } \texttt{labels}(U) \neq \emptyset, \\ \texttt{max}(\{gl(W) \mid W \text{ is before } U\}) + 1 & \text{if } \texttt{labels}(U) = \emptyset. \end{cases}$

Lemma 4.2 demonstrates that every class on a level gets a unique label, and that despite moving between nephews the labeling adheres to branches in the tree.

Lemma 4.2. *For classes U and W on level i of T, it holds that:*

(1) If $U \neq W$ then $gl(U) \neq gl(W)$.
(2) U is a descendant of $\texttt{first}(gl(U))$.
(3) If U is a descendant of $\texttt{first}(gl(W))$, then $W \preceq_i U$. Consequently, if $U \prec_i W$, then U is not a descendant of $\texttt{first}(gl(W))$.
(4) $\texttt{first}(gl(U))$ is the root or an F-class with a sibling.
(5) If $U \neq \texttt{first}(gl(U))$, then there is a class on level $i-1$ that has label $gl(U)$.
(6) If $gl(U) < gl(W)$ then $\texttt{first}(gl(U))$ is before $\texttt{first}(gl(W))$.

As stated above, the label m represents the proposition that the lexicographically minimal *infinite* branch going through $\texttt{first}(m)$ is accepting. Every time we pass through an F-child, this is evidence towards this proposition. Recall that when a class with label m has two children, we initially follow the non-F-child. If the non-F-child dies out, we revise our guess and move to a descendant of the F-child. Thus revising our guess indicates that at an earlier point the branch did visit an F-child, and also provides evidence towards this proposition. Formally, we say that a label m is *successful on level i* if there is a class U on level $i-1$ and a class U' on level i such that $gl(U) = gl(U') = m$, and either U' is the F-child of U, or U' is not a child of U at all.

Example 4.3. In Figure 1.(b), the only infinite branch $\{\langle q,0\rangle\},\{\langle p,1\rangle\},\ldots$ is accepting. At level 0 this branch is labeled with 0. At each level $i > 0$, we conservatively assume that the infinite branch beginning with $\langle q,0\rangle$ goes through $\{\langle q,i\rangle\}$, and thus label $\{\langle q,i\rangle\}$ by 0. As $\{\langle q,i\rangle\}$ is proven finite on level $i+1$, we revise our assumption and continue to follow the path through $\{\langle p,i\rangle\}$. Since $\{\langle p,i\rangle\}$ is an F-class, the label 0 is successful on every level $i+1$. Although the infinite branch is not labeled 0 after the first level, the label 0 asymptotically approaches the infinite branch, checking along the way that the branch is lexicographically minimal among the infinite branches through the root.

Theorem 4.4 demonstrates that the global labeling captures the accepting or rejecting nature of T. Intuitively, at each level the class U with label m is on the lexicographically minimal branch from $\texttt{first}(m)$. If U is on the lexicographically minimal *infinite* branch from $\texttt{first}(m)$, the label m is waiting for the branch to next reach an F-class. If U is not on the lexicographically minimal infinite branch from $\texttt{first}(m)$, then U is finite and m is waiting for U to die out.

Theorem 4.4. *A profile tree T is accepting iff there is a label m that is successful infinitely often.*

Proof: In one direction, assume there is a label m that is successful infinitely often. The label m can be successful only when it occurs, and thus m occurs infinitely often, $\texttt{first}(m)$ has infinitely many descendants, and there is at least one infinite branch through $\texttt{first}(m)$. Let $b = U_0, U_1, \ldots$ be the lexicographically minimal infinite branch that goes through $\texttt{first}(m)$. We demonstrate that b cannot have a suffix consisting solely of non-F-classes, and therefore is an accepting branch. By way of contradiction, assume there is an index j so that for every $k > j$, the class U_k is a non-F-class. By Lemma 4.2.(4), $\texttt{first}(m)$ is an F-class or the root and thus occurs before level j.

Let $\mathcal{U} = \{W \mid W \prec_j U_j,\ W \text{ is a descendant of } \texttt{first}(m)\}$ be the set of descendants of $\texttt{first}(m)$, on level j, that are lexicographically smaller than U_j. Since b is the lexicographically minimal infinite branch through $\texttt{first}(m)$, every class in $\mathcal{U}$ must be finite. Let $j' \geq j$ be the level at which the last class in $\mathcal{U}$ dies out. At this point, $U_{j'}$ is the lexicographically minimal descendant of $\texttt{first}(m)$. If $gl(U_{j'}) \neq m$, then there is no class on level j' with label m, and, by Lemma 4.2.(5), m would not occur after level j'. Since m occurs infinitely often, it must be that $gl(U_{j'}) = m$. On every level $k > j'$, the class U_k is a non-F-child, and thus U_k is the lexicographically minimal descendant of $U_{j'}$ on level k and so $gl(U_k) = m$. This entails m cannot be not successful after level j', and we have reached a contradiction. Therefore, there is no such rejecting suffix of b, and b must be an accepting branch.

In the other direction, if there is an infinite accepting branch, then let $b = U_0, U_1, \ldots$ be the lexicographically minimal infinite accepting branch. Let B' be the set of infinite branches that are lexicographically smaller than b. Every branch in B' must be rejecting, or b would not be the minimal infinite accepting branch. Let j be the first index after which the last branch in B' splits from b. Note that either $j = 0$, or U_{j-1} is part of an infinite rejecting branch $U_0, \ldots, U_{j-1}, W_j, W_{j+1}, \ldots$ smaller than b. In both cases, we show that U_j is the first class for a new label m that occurs on every level $k > j$ of T.

If $j = 0$, then let $m = 0$. As m is the smallest label, and there is a descendant of U_j on every level of T, it holds that m will occur on every level. In the second case, where $j > 0$, then W_j must be the non-F-child of U_{j-1}, and so U_j is the F-child. Thus, U_j is given a new label m where $U_j = \texttt{first}(m)$. For every label $m' < m$ and level $k > j$, since for every descendant U' of U_j it holds that $W_k \preceq_k U'$, it cannot be that $\texttt{lmd}(m', k)$ is a descendant of U_j. Thus, on every level $k > j$, the lexicographically minimal descendant of U_j will be labeled m, and m occurs on every level of T.

We show that m is successful infinitely often by defining an infinite sequence of levels, $j_0, j_1, j_2, \ldots$ so that m is successful on j_i for all $i > 0$. As a base case, let $j_0 = j$. Inductively, at level j_i, let U' be the class on level j_i labeled with m. We have two cases. If $U' \neq U_{j_i}$, then as all infinite branches smaller than b have already split from b, U' must be finite in T. Let j_{i+1} be the level at which U' dies out. At level j_{i+1}, m will return to a descendant of U_{j_0}, and m will be successful. In the second case, $U' = U_{j_i}$. Take the first $k > j_i$ so that U_k is an F-class. As b is an accepting branch, such a k must exist. As every class between U_j and U_k is a non-F-class, $gl(U_{k-1}) = m$. If U_k is the only child of U_{k-1} then let $j_{i+1} = k$: since $gl(U_k) = m$ and U_k is not the non-F-child of U_{k-1}, it holds that m is successful on level k. Otherwise let U'_k be the non-F-child of U_{k-1}, so that $gl(U'_k) = m$. Again, U'_k is finite. Let j_{i+1} be the level at which U'_k dies out. At level j_{i+1}, the label m will return to a descendant of U_k, and m will be successful. □

4.2 Determining Lexicographically Minimal Descendants

Recall that the definition of the labeling gl involves the computation of $\mathtt{lmd}(m,i)$, the class with the minimal profile among all the descendants of $\mathtt{first}(m)$ on level i. Finding $\mathtt{lmd}(m,i)$ requires knowing the descendants of $\mathtt{first}(m)$ on level i. We show how to store this information with a partial order, denoted $\lessdot_i$, over classes that tracks which classes are minimal cousins of other classes. Using this partial order, we can determine the class $\mathtt{lmd}(m,i+1)$ for every label m that occurs on level i, using only information about levels i and $i+1$ of T. Lemma 4.2.(5) implies that we can safely restrict ourselves to labels that occur on level i.

Definition 4.5. For two classes U and W on level i of T, say that U is a *minimal cousin* of W, written $U \lessdot_i W$, iff W is a descendant of $\mathtt{first}(gl(U))$. Say $U \lessdot_i W$ when $U \lessdot_i W$ and $U \neq W$.

For a label m and level i, we can determine $\mathtt{lmd}(m,i+1)$ given only the classes on levels i and $i+1$ and the partial order $\lessdot_i$. Let U be a class U on level i. Because labels can move between branches, the minimal descendant of $\mathtt{first}(gl(U))$ on level $i+1$ may be a nephew of U, not necessarily a direct descendant. Define the $\lessdot_i$-nephew of U as $\mathtt{neph}_i(U) = \mathtt{min}_{\preceq_{i+1}}(\{W' \mid W \text{ is the parent of } W' \text{ and } U \lessdot_i W\})$.

Lemma 4.6. *For a class U on level i of T, it holds that* $\mathtt{lmd}(gl(U),i+1) = \mathtt{neph}_i(U)$.

Proof: We prove that $\{W' \mid W \text{ is the parent of } W' \text{ and } U \lessdot_i W\}$ contains every descendant of $\mathtt{first}(gl(U))$ on level $i+1$, and thus that its minimal element is $\mathtt{lmd}(gl(U),i+1)$. Let W' be a class on level $i+1$, with parent W on level i. If $U \lessdot_i W$, then W is a descendant of $\mathtt{first}(gl(U))$ and W' is likewise a descendant of $\mathtt{first}(gl(U))$. Conversely, as $gl(U)$ exists on level i, if W' is a descendant of $\mathtt{first}(gl(U))$, then its parent W must also be a descendant of $\mathtt{first}(gl(U))$ and $U \lessdot_i W$. □

By using $\mathtt{neph}_i$, we can in turn define the set of valid labels for a class U' on level $i+1$. Formally, define the $\lessdot_i$-uncles of U' as $\mathtt{unc}_i(U') = \{U \mid U' = \mathtt{neph}_i(U)\}$. Lemma 4.7 demonstrates how $\mathtt{unc}_i$ corresponds to $\mathtt{labels}$.

Lemma 4.7. *Consider a class U' on level $i+1$. The following hold:*

(1) $\mathtt{labels}(U') \cap \{gl(W) \mid W \text{ on level } i\} = \{gl(U) \mid U \in \mathtt{unc}_i(U')\}$.

(2) $\mathtt{labels}(U') = \emptyset$ *iff* $\mathtt{unc}_i(U') = \emptyset$.

Proof:

(1) Let U be a class on level i. By definition, $gl(U) \in \mathtt{labels}(U')$ iff $U' = \mathtt{lmd}(gl(U),i+1)$. By Lemma 4.6, it holds that $\mathtt{lmd}(gl(U),i+1) = \mathtt{neph}_i(U)$. By the definition of $\mathtt{unc}_i$, we have that $U' = \mathtt{neph}_i(U)$ iff $U \in \mathtt{unc}_i(U')$. Thus every label in $\mathtt{labels}(U')$ that occurs on level i labels some node in $\mathtt{unc}_i(U')$.

(2) If $\mathtt{unc}_i(U') \neq \emptyset$, then part (1) implies $\mathtt{labels}(U') \neq \emptyset$. In other direction, let $m = \mathtt{min}(\mathtt{labels}(U'))$. By Lemma 4.2.(5), there is a U on level i so that $gl(U) = m$, and by part (1) $U \in \mathtt{unc}_i(U')$.

□

Finally, we demonstrate how to compute $\lessdot_{i+1}$ only using information about the level i of T and the labeling for level $i+1$. As the labeling depends only on $\lessdot_i$, this removes the final piece of global information used in defining gl.

Lemma 4.8. *Let U' and W' be two classes on level $i+1$ of T, where $U' \neq W'$. Let W be the parent of W'. We have that $U' \lessdot_{i+1} W'$ iff there exists a class U on level i so that $gl(U) = gl(U')$ and $U \lessdot_i W$.*

Proof: If there is no class U on level i so that $gl(U) = gl(U')$, then $U' = \mathtt{first}(gl(U'))$. Since W' is not a descendant of U', it cannot be that $U' \lessdot_{i+1} W'$. If such a class U exists, then $U \lessdot_i W$ iff W is a descendant of $\mathtt{first}(gl(U))$, which is true iff W' is a descendant of $\mathtt{first}(gl(U'))$: the definition of $U' \lessdot_{i+1} W'$. □

4.3 Reusing Labels

As defined, the labeling function gl uses an unbounded number of labels. However, as there are at most $|Q|$ classes on a level, there are at most $|Q|$ labels in use on a level. We can thus use a fixed set of labels by reusing dead labels. For convenience, we use $2|Q|$ labels, so that we never need reuse a label that was in use on the previous level. The full version demonstrates how to use $|Q|-1$ labels. There are two barriers to reusing labelings. First, we can no longer take the numerically minimal element of $\texttt{labels}(U)$ as the label of U. Instead, we calculate which label is the oldest through $\preceq$. Second, we must ensure that a label that is good infinitely often is not reused infinitely often. To do this, we introduce a Rabin condition to reset each label before we reuse it.

We inductively define a sequence of labelings, l_i, each from the ith level of T to $\{0,\ldots,2|Q|\}$. As a base case, there is only one equivalence class U on level 0 of T, and define $l_0(U)=0$. Inductively, given the set of classes $\mathcal{U}_i$ on level i, the function l_i, and the set of classes $\mathcal{U}_{i+1}$ on level $i+1$, we define l_{i+1} as follows. Define the set of unused labels $\mathrm{FL}(l_i)$ to be $\{m \mid m \text{ is not in the range of } l_i\}$. As T has bounded width $|Q|$, we have that $|Q| \leq |\mathrm{FL}(l_i)|$. Let $\texttt{mj}_{i+1}$ be the $\langle \preceq_{i+1}, < \rangle$-minjection from $\{U' \text{ on level i+1} \mid \texttt{unc}_i(U')=\emptyset\}$ to $\mathrm{FL}(l_i)$. Finally, define the labeling l_{i+1} as

$$l_{i+1}(U') = \begin{cases} l_i(\texttt{min}_{\preceq_i}(\texttt{unc}_i(U'))) & \text{if } \texttt{unc}_i(U') \neq \emptyset, \\ \texttt{mj}_{i+1}(U') & \text{if } \texttt{unc}_i(U') = \emptyset. \end{cases}$$

Because we are reusing labels, we need to ensure that a label that is good infinitely often is not reused infinitely often. Say that a label m is *bad in l_i* if $m \notin \mathrm{FL}(l_{i-1})$, but $m \in \mathrm{FL}(l_i)$. We say that a label m is *good in l_i* if there is a class U on level $i-1$ and a class U' on level i such that $l_{i-1}(U) = l_i(U') = m$ and U' is either the F-child of U or is not a child of U at all.

Theorem 4.9 demonstrates that the Rabin condition of a label being good infinitely often, but bad only finitely often, is a necessary and sufficient condition to T being accepting. The proof, ommitted for brevity, associates each label m in gl with the label $l_i(\texttt{first}(m))$.

Theorem 4.9. *A profile tree T is accepting iff there is a label m where $\{i \mid m \text{ is bad in } l_i\}$ is finite, and $\{i \mid m \text{ is good in } l_i\}$ is infinite.*

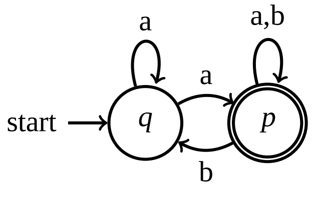
5 A New Determinization Construction for Büchi Automata

In this section we present a determinization construction for $\mathcal{A}$ based on the profile tree T. For clarity, we call the states of our deterministic automaton *macrostates*.

Definition 5.1. Macrostates over $\mathcal{A}$ are six-tuples $\langle S, \preceq, l, \lessdot, G, B \rangle$ where:

- $S \subseteq Q$ is a set of states.
- $\preceq$ is a linear preorder over S.
- $l : S \to \{0,\ldots,2|Q|\}$ is a labeling.
- $\lessdot \subseteq \preceq$ is another preorder over S.
- G, B are sets of good and bad labels used for the Rabin condition.

For two states q and r in Q, we say that $q \approx r$ if $q \preceq r$ and $r \preceq q$. We constrain the labeling l so that it characterizes the equivalence classes of S under $\preceq$, and the preorder $\lessdot$ to be a partial order over the equivalence classes of $\preceq$. Let $\mathbf{Q}$ be the set of macrostates.



(a) An automaton $\mathcal{B}$

$$\mathbf{q}_0 = \boxed{\langle\{q\}^0\rangle,\ \emptyset,\ G=\emptyset,\ B=\emptyset}$$

$$\mathbf{q}_1 = \boxed{\langle\{q\}^0 \prec \{p\}^1\rangle,\ q \lessdot p,\ G=\emptyset,\ B=\emptyset}$$

$$\mathbf{q}_2 = \boxed{\langle\{q\}^0 \prec \{p\}^2\rangle,\ q \lessdot p,\ G=\{0\},\ B=\{1\}}$$

$$\mathbf{q}_3 = \boxed{\langle\{q\}^0 \prec \{p\}^1\rangle,\ q \lessdot p,\ G=\{0\},\ B=\{2\}}$$

(b) The first four macrostates in the run of $D^R(\mathcal{B})$ on ab^ω.

Figure 2: An automaton and four macrostates. For each macrostate $\langle S, \preceq, l, \lessdot, G, B\rangle$, we first display the equivalence classes of S under $\preceq$ in angle brackets, superscripted with the labels of l. We then display the $\lessdot$ relation, and finally the sets G and B.

Before defining transitions between macrostates, we reproduce the pruning of edges from G' by restricting the transition function ρ with respect to S and $\preceq$. For a state $q \in S$ and $\sigma \in \Sigma$, let $\rho_{S,\preceq}(q,\sigma) = \{q' \in \rho(q,\sigma) \mid \text{for every } r \in \rho^{-1}(q',\sigma) \cap S,\ r \preceq q\}$. Thus, when a state has multiple incoming σ-transitions from S, the function $\rho_{S,\preceq}$ keeps only the transitions from states maximal under the $\preceq$ relation. For every state $q' \in \rho(S,\sigma)$, the set $\rho^{-1}_{S,\preceq}(q',\sigma) \cap S$ is an equivalence class under $\preceq$. We note that $\rho(S,\sigma) = \rho_{S,\preceq}(S,\sigma)$.

Example 5.2. Figure 2 displays the first four macrostates in a run of this determinization construction. Consider the state $\mathbf{q}_1 = \langle\{q,p\}, \preceq, l, \lessdot, \emptyset, \emptyset\rangle$ where $q \prec p$, $q \lessdot p$, $l(q) = 0$, and $l(p) = 1$. We have $\rho(q,a) = \{p,q\}$. However, $p \in \rho(p,a)$ and $q \prec p$. Thus we discard the transition from q to p, and $\rho_{S,\preceq}(q,a) = \{q\}$. In contrast, $\rho_{S,\preceq}(p,a) = \rho(p,a) = \{p\}$, because while $p \in \rho(q,a)$, it holds that $q \prec p$.

For $\sigma \in \Sigma$, we define the σ-successor of $\langle S, \preceq, l, \lessdot, G, B\rangle$ to be $\langle S', \preceq', l', \lessdot', G', B'\rangle$ as follows. First, $S' = \rho(S,\sigma)$. Second, define $\preceq'$ as follows. For states $q', r' \in S'$, let $q \in \rho^{-1}_{S,\preceq}(q',\sigma)$ and $r \in \rho^{-1}_{S,\preceq}(r',\sigma)$. As the parents of q' and r' under $\rho_{S,\preceq}$ are equivalence classes the choice of q and r is arbitrary.

- If $q \prec r$, then $q' \prec' r'$.
- If $q \approx r$ and $q' \in F$ iff $r' \in F$, then $q' \approx' r'$.
- If $q \approx r$, $q' \notin F$, and $r' \in F$, then $q' \prec' r'$.

Example 5.3. As a running example we detail the transition from $\mathbf{q}_1 = \langle\{q,p\}, \preceq, l, \lessdot, \emptyset, \emptyset\rangle$ to $\mathbf{q}_2 = \langle S', \preceq', l', \lessdot', G', B'\rangle$ on b. We have $S' = \rho(\{q,p\}, b) = \{q,p\}$. To determine $\preceq'$, we note that $p \in S$ is the parent of both $q \in S'$ and $p \in S'$. Since $q \notin F$, and $p \in F$, we have $q \prec' p$.

Third, we define the labeling l' as follows. As in the profile tree T, on each level we give the label m to the minimal descendants, under the $\preceq$ relation, of the first equivalence class to be labeled m. For a state $q \in S$, define the *nephews of* q to be $\mathtt{neph}(q,\sigma) = \mathtt{min}_{\preceq'}(\rho_{S,\preceq}(\{r \in S \mid q \lessdot r\}, \sigma))$. Conversely, for a state $r' \in S'$ we define the *uncles of* r' to be be $\mathtt{unc}(r',\sigma) = \{q \mid r' \in \mathtt{neph}(q,\sigma)\}$.

Each state $r' \in S'$ inherits the oldest label from its uncles. If r' has no uncles, it gets a fresh label. Let $\mathrm{FL}(l) = \{m \mid m \text{ not in the range of } l\}$ be the free labels in l, and let $\mathtt{mj}$ be the $\langle \preceq', < \rangle$-minjection from $\{r' \in S' \mid \mathtt{unc}(r',\sigma) = \emptyset\}$ to $\mathrm{FL}(l)$, where $<$ is the standard order on $\{0, \ldots, 2|Q|\}$. Let

$$l'(r') = \begin{cases} l(q), \ \text{ for some } q \in \mathtt{min}_{\preceq}(\mathtt{unc}(r',\sigma)) & \text{if } \mathtt{unc}(r',\sigma) \neq \emptyset, \\ \mathtt{mj}(r') & \text{if } \mathtt{unc}(r',\sigma) = \emptyset. \end{cases}$$

Example 5.4. The nephews of $q \in S$ is the $\preceq'$-minimal subset of the set $\rho_{S,\preceq}(\{r \in S \mid q \lessdot r\}, \sigma)$. Since $q \lessdot q$ and $q \lessdot p$, we have that $\texttt{neph}(q,b) = \texttt{min}_{\preceq'}(\{q,p\}) = \{q\}$. Similarly, for $p \in S$ we have $p \lessdot p$ and $\texttt{neph}(p,b) = \texttt{min}_{\preceq'}(\{p,q\}) = \{q\}$. Thus for $q \in S'$, we have $\texttt{min}_{\preceq}(\texttt{unc}(q,b)) = \texttt{min}_{\preceq}(\{p,q\}) = \{q\}$ and we set $l'(q) = l(q) = 0$. For $p \in S'$, we have $\texttt{unc}(p,b) = \emptyset$ and $l'(p)$ is the first unused label: $l'(p) = 2$.

Fourth, define the preorder $\lessdot'$ as follows. For states $q', r' \in S'$, define $q' \lessdot' r'$ iff $q' \approx' r'$ or there exist $q, r \in S$ so that: $r' \in \rho_{S,\preceq}(r,\sigma)$; $q \in \texttt{unc}(q',\sigma)$; and $q \lessdot r$. The labeling l' depends on recalling which states descend from the first equivalence class with a given label, and $\lessdot'$ tracks these descendants.

Finally, for a label m let $S_m = \{r \in S \mid l(r) = m\}$ and $S'_m = \{r' \in S' \mid l'(r') = m\}$ be the states in S, resp S', labeled with m. Recall that a label m is good either when the branch it is following visits F-states, or the branch dies and it moves to another branch. Thus say m is *good* when: $S_m \neq \emptyset$; $S'_m \neq \emptyset$; and either $S'_m \subseteq F$ or $\rho_{S,\preceq}(S_m,\sigma) \cap S'_m = \emptyset$. G' is then $\{m \mid m \text{ is good}\}$. Conversely, a label is bad when it occurs in S, but not in S'. Thus the set of *bad* labels is $B' = \{m \mid S_m \neq \emptyset,\ S'_m = \emptyset\}$.

Example 5.5. As $p \in \rho_{S,\preceq}(p,b)$; $q \in \texttt{unc}(q,b)$; and $q \lessdot p$, we have $q \lessdot' p$. Since $l(q) = 0$ and $l'(q) = 0$, but $q \notin \rho_{S,\preceq}(q,b)$, we have $0 \in G'$, and as nothing is labeled 1 in l', we have $1 \in B'$.

Lemma 5.6, proven in the full version, states that $\langle S', \preceq', l', \lessdot', G', B' \rangle$ is a valid macrostate.

Lemma 5.6. *For a macrostate $\mathbf{q} \in \mathbf{Q}$ and $\sigma \in \Sigma$, the σ-successor of $\mathbf{q}$ is a macrostate.*

Definition 5.7. Define the DRW automaton $D^R(\mathcal{A})$ to be $\langle \Sigma, \mathbf{Q}, \mathbf{Q}^{in}, \rho_{\mathbf{Q}}, \alpha \rangle$, where:

- $\mathbf{Q}^{in} = \{\langle Q^{in}, \preceq_0, l_0, \lessdot_0, \emptyset, \emptyset \rangle\}$, where:
  - $\preceq_0 = \lessdot_0 = Q^{in} \times Q^{in}$
  - $l_0(q) = 0$ for all $q \in Q^{in}$
- For $\mathbf{q} \in \mathbf{Q}$ and $\sigma \in \Sigma$, let $\rho_{\mathbf{Q}}(\mathbf{q},\sigma) = \{\mathbf{q}'\}$, where $\mathbf{q}'$ is the σ-successor of $\mathbf{q}$
- $\alpha = \langle G_0, B_0 \rangle, \ldots, \langle G_{2|Q|}, B_{2|Q|} \rangle$, where for a label $m \in \{0, \ldots, 2|Q|\}$:
  - $G_m = \{\langle S, \preceq, l, \lessdot, G, B \rangle \mid m \in G\}$
  - $B_m = \{\langle S, \preceq, l, \lessdot, G, B \rangle \mid m \in B\}$

Theorem 5.8, proven in the full version, asserts the correctness of the construction and says that its blowup is comparable with known determinization constructions.

Theorem 5.8. *For an NBW $\mathcal{A}$ with n states, $L(D^R(\mathcal{A})) = L(\mathcal{A})$ and $D^R(\mathcal{A})$ has $n^{O(n)}$ states.*

There are two simple improvements to the new construction, detailed in the full version. First, we do not need $2|Q|$ labels: it is sufficient to use $|Q| - 1$ labels. Second, Piterman's technique of dynamic renaming can reduce the Rabin condition to a parity condition.

6 Discussion

In this paper we extended the notion of profiles from [6] and developed a theory of profile trees. This theory affords a novel determinization construction, where determinized-automaton states are sets of input-automaton states augmented with two preorders. In the future, a more thorough analysis could likely improve the upper bound on the size of our construction. We hope to see heuristic optimization techniques developed for this construction, just as heuristic optimization techniques were developed for Safra's construction [24].

More significantly, profile trees afford us the first theoretical underpinnings for determinization. Decades of research on Büchi determinization have resulted in a plethora of constructions, but a paucity of mathematical structures underlying their correctness. This is the first new major line of research in Büchi determinization since [16], and we expect it to lead to further research in this important area.

One important question is to understand better the connection between profile trees and Safra's construction. A key step in the transition between Safra trees is to remove states if they appear in more than one node. This seems analogous to the pruning of edges from G'. The second preorder in our construction, namely the relation $\lesssim_i$, seems to encodes the order information embedded in Safra trees. Perhaps our approach could lead to declarative definition of constructions based on Safra and Muller-Schupp trees. In any case, it is our hope that profile trees will encourage the development of new methods to analyze and optimize determinization constructions.

References

[1] C.S. Althoff, W. Thomas & N. Wallmeier (2005): *Observations on determinization of Büchi automata*. In: *ICALP*, doi:10.1016/j.tcs.2006.07.026.

[2] R. Alur, T. A. Henzinger & O. Kupferman (2002): *Alternating-time temporal logic*. *J. ACM*, doi:10.1145/585265.585270.

[3] J.R. Büchi (1962): *On a decision method in restricted second order arithmetic*. In: *ICLMPS*.

[4] C. Courcoubetis & M. Yannakakis (1995): *The complexity of probabilistic verification*. *J. ACM*, doi:10.1145/210332.210339.

[5] L. Doyen & J.-F. Raskin (2007): *Improved algorithms for the automata-based approach to model-checking*. In: *TACAS*, doi:10.1007/978-3-540-71209-1_34.

[6] S. Fogarty, O. Kupferman, M.Y. Vardi & Th. Wilke (2011): *Unifying Büchi complementation constructions*. In: *CSL*, doi:10.4230/LIPIcs.CSL.2011.248.

[7] S. Fogarty & M.Y. Vardi (2010): *Efficient Büchi universality checking*. In: *TACAS*, doi:10.1007/978-3-642-12002-2_17.

[8] E. Friedgut, O. Kupferman & M.Y. Vardi (2006): *Büchi complementation made tighter*. *IJFCS*, doi:10.1142/S0129054106004145.

[9] D. Kähler & Th. Wilke (2008): *Complementation, disambiguation, and determinization of Büchi automata unified*. In: *ICALP*, doi:10.1007/978-3-540-70575-8_59.

[10] J. Kretínský & J. Esparza (2012): *Deterministic automata for the (F, G)-fragment of LTL*. In: *CAV*, doi:10.1007/978-3-642-31424-7_7.

[11] O. Kupferman & M.Y. Vardi (2001): *Weak alternating automata are not that weak*. *TOCL*, doi:10.1145/377978.377993.

[12] O. Kupferman & M.Y. Vardi (2005): *Safraless decision procedures*. In: *FOCS*, doi:10.1109/SFCS.2005.66.

[13] L.H. Landweber (1969): *Decision problems for ω–automata*. *MST*, doi:10.1007/BF01691063.

[14] C. Löding (1999): *Optimal bounds for the transformation of omega-automata*. In: *FSTTCS*, doi:10.1007/3-540-46691-6_8.

[15] R. McNaughton (1966): *Testing and generating infinite sequences by a finite automaton*. *ICONT*, doi:10.1016/S0019-9958(66)80013-X.

[16] D.E. Muller & P.E. Schupp (1995): *Simulating alternating tree automata by nondeterministic automata: new results and new proofs of theorems of Rabin, McNaughton and Safra*. *TCS*, doi:10.1016/0304-3975(94)00214-4.

[17] N. Piterman (2006): *From nondeterministic Büchi and Streett automata to deterministic parity automata*. In: *LICS*, doi:10.2168/LMCS-3(3:5)2007.

[18] A. Pnueli & R. Rosner (1989): *On the synthesis of a reactive module*. In: *POPL*, doi:10.1145/75277.75293.

[19] M.O. Rabin & D. Scott (1959): *Finite automata and their decision problems*. *IBM JRD*, doi:10.1147/rd.32.0114.

[20] S. Safra (1988): *On the complexity of ω-automata*. In: *FOCS*, doi:10.1109/SFCS.1988.21948.

[21] S. Schewe (2009): *Büchi complementation made tight*. In: *STACS*, doi:10.4230/LIPIcs.STACS.2009.1854.

[22] S. Schewe (2009): *Tighter bounds for the determinisation of Büchi automata*. In: *FOSSACS*, doi:10.1007/978-3-642-00596-1_13.

[23] S. Tasiran, R. Hojati & R.K. Brayton (1995): *Language containment of non-deterministic omega-automata*. In: *CHARME*, doi:10.1007/3-540-60385-9_16.

[24] M.-H. Tsai, S. Fogarty, M. Y. Vardi & Y.-K. Tsay (2010): *State of Büchi complementation*. In: *CIAA*, doi:10.1007/978-3-642-18098-9_28.

[25] M.Y. Vardi (1985): *Automatic verification of probabilistic concurrent finite-state programs*. In: *FOCS*, doi:10.1109/SFCS.1985.12.

Weighted Automata and Monadic Second Order Logic

Nadia Labai
Faculty of Computer Science
Technion–Israel Institute of Technology
nadia@cs.technion.ac.il

Johann A. Makowsky
Faculty of Computer Science
Technion–Israel Institute of Technology
janos@cs.technion.ac.il

Let $\mathscr{S}$ be a commutative semiring. M. Droste and P. Gastin have introduced in 2005 weighted monadic second order logic **WMSOL** with weights in $\mathscr{S}$. They use a syntactic fragment **RMSOL** of **WMSOL** to characterize word functions (power series) recognizable by weighted automata, where the semantics of quantifiers is used both as arithmetical operations and, in the boolean case, as quantification.

Already in 2001, B. Courcelle, J.Makowsky and U. Rotics have introduced a formalism for graph parameters definable in Monadic Second order Logic, here called **MSOLEVAL** with values in a ring $\mathscr{R}$. Their framework can be easily adapted to semirings $\mathscr{S}$. This formalism clearly separates the logical part from the arithmetical part and also applies to word functions.

In this paper we give two proofs that **RMSOL** and **MSOLEVAL** with values in $\mathscr{S}$ have the same expressive power over words. One proof shows directly that **MSOLEVAL** captures the functions recognizable by weighted automata. The other proof shows how to translate the formalisms from one into the other.

1 Introduction

Let f be a function from relational structures of a fixed relational vocabulary τ into some field, ring, or a commutative semiring $\mathscr{S}$ which is invariant under τ-isomorphisms. $\mathscr{S}$ is called a *weight structure*. In the case where the structures are graphs, such a function is called a graph parameter, or, if $\mathscr{S}$ is a polynomial ring, a graph polynomial. In the case where the structures are words, it is called a word function.

The study of definability of graph parameters and graph polynomials in Monadic Second Order Logic **MSOL** was initiated in [6] and further developed in [22, 20]. For a weight structure $\mathscr{S}$ we denote the set of functions of τ-structures definable in **MSOL** by $\mathbf{MSOLEVAL}(\tau)_{\mathscr{S}}$, or if the context is clear, just by $\mathbf{MSOLEVAL}_{\mathscr{S}}$. The original purpose for studying functions in $\mathbf{MSOLEVAL}_{\mathscr{S}}$ was to prove an analogue to Courcelle's celebrated theorem for polynomial rings as weight structures, which states that graph parameters $f \in \mathbf{MSOLEVAL}_{\mathscr{S}}$ are computable in linear time for graphs of fixed tree-width, [6], and various generalizations thereof. **MSOLEVAL** can be seen as an analogue of the *Skolem elementary functions* aka *lower elementary functions*, [25, 26], adapted to the framework of *meta-finite model theory* as defined in [15].

In [8] a different formalism to define $\mathscr{S}$-valued word functions was introduced, which the authors called *weighted monadic second order logic WMSOL*, and used a fragment, **RMSOL**, of it to prove that a word function is recognized by a weighted automaton iff it is definable in **RMSOL**. This can be seen as an analogue of the Büchi-Elgot-Trakhtenbrot Theorem characterizing regular languages for the case of weighted (aka multiplicity) automata.

Main results

Our main results explore various features of the two formalisms **MSOLEVAL** and **RMSOL** for word functions with values in a semiring $\mathcal{S}$. In the study of **MSOLEVAL** we show how *model theoretic tools* can be used to characterize the word functions in **MSOLEVAL** as the fuctions recognizable by weigthed automata. This complements the automata theoretic approach used in the study of weighted automata, [9, 11]. In particular, we give two proofs that **RMSOL** and **MSOLEVAL** with values in a semiring $\mathcal{S}$ have the same expressive power over words. To see this we show the following for a word function f with values in $\mathcal{S}$:

(i) If f is definable in **MSOLEVAL**, it is contained in a finitely generated stable semimodule of word functions, Theorem 11.

(ii) If f is recognizable by some weighted automaton, it is definable in **MSOLEVAL**, the "if" direction of Theorem 8.

(iii) If f is definable in **RMSOL**, we can translate it, using Lemma 15, into an expression in **MSOLEVAL**, Theorem 16.

(iv) If f is definable in **MSOLEVAL**, we can, again using Lemma 15, translate it into an expression in **RMSOL**, Theorem 17.

Items (i) and (ii) together with a classical characterization of recognizable word functions in terms of finitely generated stable semimodules, Theorem 10, cf. [1, 17, 13], give us a direct proof that **MSOLEVAL** captures the functions recognizable by weighted automata. To prove item (i) we rely on and extend results about **MSOLEVAL** from [22, 14, 18].

Items (iii) and (iv) together show how to translate the formalisms **RMSOL** and **MSOLEVAL** into each other. Lemma 15 also shows how the fragment **RMSOL** of the weighted logic **WMSOL** comes into play.

The point of separating (i) and (ii) from (ii) and (iv) and giving *two* proofs of Theorem 8 is to show that the model theoretic methods developed in the 1950ties and further developed in [22] suffice to characterize the functions recognized by weighted automata.

Background and outline of the paper

We assume the reader is familiar with Monadic Second Order Logic and Automata Theory as described in [12, 1] or similar references. In Section 2 we introduce **MSOLEVAL** by example, which suffices for our purposes. A full definition is given in Appendix 2.2. In Section 3 we show that the word functions which are recognizable by a weighted automaton are exactly the word functions definable in **MSOLEVAL**. In Section 4 we give the exact definitions of **WMSOL** and **RMSOL**, and present translations between **MSOLEVAL** and **RMSOL** in both directions. In Section 5 we draw our conclusions.

2 Definable word functions

Let $\mathcal{S}$ be a commutative semiring. We denote structures over a finite relational signature (aka vocabulary) τ by $\mathcal{A}$ and their underlying universe by A. The class of functions in **MSOLEVAL**$_{\mathcal{S}}$ consists of the functions which map relational structures into $\mathcal{S}$, and which are definable in Monadic Second Order Logic **MSOL**. The functions in **MSOLEVAL**$_{\mathcal{S}}$ are represented as terms associating with each τ-structure $\mathcal{A}$ a polynomial $p(\mathcal{A}, \bar{X}) \in \mathcal{S}[\bar{X}]$. The class of such polynomials is defined inductively where monomials are products of constants in $\mathscr{S}$ and indeterminates in $\bar{X}$ and the product ranges over elements a of A which satisfy an **MSOL**-formula $\phi(a)$. The polynomials are then defined as sums of monomials where the sum ranges over *unary* relations $U \subseteq A$ satisfying an **MSOL**-formula $\psi(U)$. The word functions are obtained by substituting elements of $\mathscr{S}$ for the indeterminates. The details of the definition of **MSOLEVAL**$_{\mathscr{S}}$ are given at the end of this section. We first explain the idea of **MSOLEVAL**$_{\mathscr{S}}$ by examples for the case where structures represent words over a fixed alphabet Σ.

2.1 Guiding examples

Let $f : \Sigma^\star \to \mathscr{S}$ be an $\mathscr{S}$-valued function on words over the alphabet Σ and let w be a word in $\Sigma^\star$. We call such functions *word functions*, following [3, 4]. They are also called *formal power series* in [1], where the indeterminates are indexed by words and the coefficient of X_w is $f(w)$.

We denote by $w[i]$ the letter at position i in w, and by $w[U]$ the word induced by U, for U a set of positions in w. We denote the length of a word w by $\ell(w)$ and the concatenation of two words $u, v \in \Sigma^\star$ by $u \circ v$. We denote by $[n]$ the set $\{1, 2, \ldots, n\}$.

We will freely pass between words and structures representing words. For the sequel, let $\Sigma = \{0, 1\}$ and $w \in \{0, 1\}^\star$ be represented by the structure

$$\mathscr{A}_w = \langle \{0\} \cup [\ell(w)], <^w, P_0^w, P_1^w \rangle.$$

$P_0^w, P_1^w \subseteq [\ell(w)]$ and $P_0^w \cap P_1^w = \emptyset$ and $P_0^w \cup P_1^w = [\ell]$.

As structures are always non-empty, the universe of a word w is represented by a structure containing the zero position $[n] \cup \{0\} = \{0, 1, \ldots, n\}$. So strictly speaking the size of the structure of the empty word is one, and of a word of length n it is $n+1$. The zero position, represented by 0, has no letter attached to it, and the elements of the structure different from 0 represent positions in the word which carry letters. The positions in P_0^w carry the letter 0 and the positions in P_1^w carry the letter 1.

Examples 1. *In the following examples the functions are word functions with values in the ring $\mathbb{Z}$ or the polynomial ring $\mathbb{Z}[X]$.*

(i) The function $\sharp_1(w)$ counts the number of occurrences of 1 in a word w and can be written as

$$\sharp_1(w) = \sum_{i \in [n] : P_1(i)} 1.$$

(ii) The polynomial $X^{\sharp_1(w)}$ can be written as

$$X^{\sharp_1(w)} = \prod_{i \in [n] : P_1(i)} X.$$

(iii) Let L be a regular language defined by the **MSOL***-formula ϕ_L. The generating function of the number of (contiguous) occurrences of words $u \in L$ in a word w, can be written as*

$$\sharp_L(w) = \sum_{U \subseteq [n] : w[U] \models \psi_L} \prod_{i \in U} X,$$

where $\psi_L(U)$ says that U is an interval and ϕ_L^U, the relativization of ϕ_L to U, holds.

(iv) The functions $\mathrm{sq}(w) = 2^{\ell(w)^2}$ and $\mathrm{dexp}(w) = 2^{2^{\ell(w)}}$ are not representable in **MSOLEVAL**$_{\mathscr{F}}$.

The *tropical semiring* $\mathcal{T}_{min}$ is the semiring with universe $\mathbb{R} \cup \{\infty\}$, consisting of the real numbers augmented by an additional element ∞, and *min* as addition with ∞ as neutral element and real addition $+$ as multiplication with 0 as neutral element. The tropical semiring $\mathcal{T}_{max}$, also sometimes called *arctic semiring*, is defined analogously, where ∞ is replaced by $-\infty$ and *min* by *max*. The choice of the commutative semiring $\mathcal{S}$ makes quite a difference as illustrated by the following:

Examples 2. *In the next examples the word functions take values in the ring $\mathbb{Z}$ with addition and multiplication, or in the subsemiring of $\mathcal{T}_{max}$ generated by $\mathbb{Z}$. A* block of 1's in a word $w \in \{0,1\}^\star$ *is a maximal set of consecutive positions $i \in [\ell(w)]$ in the word w with $P_1(i)$.*

(i) The function $b_1(w)$ counts the number of blocks of 1's in w. $b_1(w)$ can be written as

$$b_1(w) = \sum_{B \subseteq [\ell(w)]: B \text{ is a block of 1's}} 1$$

which is in $\mathbf{MSOLEVAL}_{\mathbb{Z}}$. Alternatively, it can be written as

$$b_1(w) = \sum_{v \in [\ell(w)]: First-in-Block(v)} 1, \tag{1}$$

where $First-in-Block(v)$ is the formula in **MSOL** *which says that v is a first position in a block of 1's. Equation (1) can be expressed in $\mathbf{MSOLEVAL}_{\mathbb{Z}}$ and also in both $\mathbf{MSOLEVAL}_{\mathcal{T}_{min}}$ and $\mathbf{MSOLEVAL}_{\mathcal{T}_{max}}$.*

(ii) Let $mb_1^{max}(w)$ be the function which assigns to the word w the maximum of the sizes of blocks of 1's, and $mb_1^{min}(w)$ be the function which assigns to the word w the minimum of the sizes of blocks of 1's. One can show, see Remark 3, that mb_1^{max} and mb_1^{min} are not definable over the ring $\mathbb{Z}$. However, they are definable over $\mathcal{T}_{max}$, respectively over $\mathcal{T}_{min}$, by writing

$$mb_1^{max} = \max_{B:B \text{ is a block of 1's}} \sum_{v:v \in B} 1$$

and

$$mb_1^{min} = \min_{B:B \text{ is a block of 1's}} \sum_{v:v \in B} 1$$

(iii) The function $b_1(w)^2$ is definable in $\mathbf{MSOLEVAL}_{\mathbb{Z}}$ because $\mathbf{MSOLEVAL}_{\mathbb{Z}}$ is closed under the usual product, cf. Proposition 7. However, it is not definable over either of the two tropical semirings. To see this one notes that polynomials in a tropical semiring are piecewise linear.

Remark 3. *Let f be a word function which takes values in a field $\mathcal{F}$. The Hankel matrix $\mathcal{H}(f)$ is the infinite matrix where rows and columns are labeled by words u, v and the entry $\mathcal{H}(f)_{u,v} = f(u \circ v)$. It is shown in [14] that for word functions f in $\mathbf{MSOLEVAL}_{\mathcal{F}}$ the Hankel matrix $\mathcal{H}(f)$ has finite rank. To show non-definability of f it suffices to show that $\mathcal{H}(f)$ has infinite rank over a field $\mathcal{F}$ extending $\mathbb{Z}$.*

2.2 Formal definition of MSOLEVAL

Let $\mathcal{S}$ be a commutative semiring, which contains the semiring of natural numbers $\mathbb{N}$. We first define **MSOL**-polynomials, which are multivariate polynomials. The functions in **MSOLEVAL** are obtained from **MSOL**-polynomials by substituting values from $\mathcal{S}$ for the indeterminates.

MSOL-polynomials have a fixed finite set of variables (indeterminates, if we distinguish them from the variables of **SOL**), **X**. We denote by $card_{M,v}(\varphi(v))$ the number of elements v in the universe that satisfy φ. We assume τ contains a relation symbol $\mathbf{R}_{\leq}$ which is always interpreted as a linear ordering of the universe.

Let $\mathfrak{M}$ be a τ-structure. We first define the $\mathbf{MSOL}(\tau)$-*monomials* inductively.

Definition 4 ($\mathbf{MSOL}$-monomials)**.**

(i) Let $\phi(v)$ be a formula in $\mathbf{MSOL}(\tau)$, where v is a first order variable. Let $r \in \mathbf{X} \cup (\mathscr{S} - \{0\})$ be either an indeterminate or an integer. Then

$$r^{card_{M,v}(\phi(v))}$$

is a standard $\mathbf{MSOL}(\tau)$-monomial (whose value depends on $card_{M,v}(\phi(v))$.

(ii) Finite products of $\mathbf{MSOL}(\tau)$-monomials are $\mathbf{MSOL}(\tau)$-monomials.

Even if r is an integer, and $r^{card_{M,v}(\phi(v))}$ does not depend on $\mathfrak{M}$, the monomial stands as it is, and is not evaluated.

Note the degree of a monomial is polynomially bounded by the cardinality of $\mathfrak{M}$.

Definition 5 ($\mathbf{MSOL}$-polynomials)**.** *The polynomials definable in $\mathbf{MSOL}(\tau)$ are defined inductively:*

(i) $\mathbf{MSOL}(\tau)$-monomials are $\mathbf{MSOL}(\tau)$-polynomials.

(ii) Let ϕ be a $\tau \cup \{\bar{\mathbf{R}}\}$-formula in $\mathbf{MSOL}$ where $\bar{\mathbf{R}} = (\mathbf{R}_1, \dots, \mathbf{R}_m)$ is a finite sequence of unary *relation symbols not in τ. Let t be a $\mathbf{MSOL}(\tau \cup \{\bar{\mathbf{R}}\})$-polynomial. Then*

$$\sum_{\bar{R}:\langle \mathfrak{M},\bar{R}\rangle \models \phi(\bar{R})} t$$

is a $\mathbf{MSOL}(\tau)$-polynomial.

For simplicity we refer to $\mathbf{MSOL}(\tau)$-polynomials as $\mathbf{MSOL}$-polynomials when τ is clear from the context.

We shall use the following properties of $\mathbf{MSOL}$-polynomials. The proofs can be found in [21].

Lemma 6.

(i) Every indeterminate $x \in \mathbf{X}$ can be written as an $\mathbf{MSOL}$-monomial.

(ii) Every integer c can be written as an $\mathbf{MSOL}$-monomial.

Proposition 7. *The pointwise product of two $\mathbf{MSOL}$-polynomials is again an $\mathbf{MSOL}$-polynomial.*

3 MSOLEVAL$_{\mathscr{S}}$ and Weighted Automata

Let $\mathscr{S}$ be a commutative semiring and Σ a finite alphabet. A weighted automaton A of size r over $\mathscr{S}$ is given by:

(i) Two vectors $\alpha, \gamma \in \mathscr{S}^r$, and

(ii) for each $\sigma \in \Sigma$ a matrix $\mu_\sigma \in \mathscr{S}^{r\times r}$.

For a matrix or vector M we denote by M^T the transpose of M.

For a word $w = \sigma_1\sigma_2 \dots \sigma_{\ell(w)}$ the automaton A defines the function

$$f_A(w) = \alpha \cdot \mu_{\sigma_1} \cdot \ldots \cdot \mu_{\sigma_{\ell(w)}} \cdot \gamma^T.$$

A word function $f : \Sigma^\star \to \mathscr{S}$ is recognized by an automaton A if $f = f_A$. f is recognizable if there exists a weighted automaton A which recognizes it.

Theorem 8. *Let f be a word function with values in a commutative semiring $\mathcal{S}$. Then $f \in$ **MSOLEVAL**$_{\mathcal{S}}$ iff f is recognized by some weigthed automaton A over $\mathcal{S}$.*

In this section we prove Theorem 8 using model theoretic tools, without going through weighted logic. We need a few definitions.

The *quantifier rank* $qr(f)$ of a word function f in **MSOLEVAL**$_{\mathcal{S}}$ is defined as the maximal quantifier rank of the formulas which appear in the definition of f. It somehow measures the complexity of f, but we do not need the technical details in this paper. Quantifier ranks of formulas in **MSOL** are defines as usual, cf. [12].

We denote by $\mathcal{S}^{\Sigma^\star}$ the set of word functions $\Sigma^\star \to \mathcal{S}$. A *semimodule* $\mathcal{M}$ is a subset of $\mathcal{S}^{\Sigma^\star}$ closed under point-wise addition of word functions in $\mathcal{M}$, and point-wise multiplication with elements of $\mathcal{S}$. Note that $\mathcal{S}^{\Sigma^\star}$ itself is a semimodule.

$M \subseteq \mathcal{S}^{\Sigma^\star}$ *is finitely generated* if there is a finite set $F \subseteq \mathcal{S}^{\Sigma^\star}$ such that each $f \in M$ can be written as a (semiring) linear combination of elements in F. Let w be a word and f a word function. Then we denote by $w^{-1}f$ the word function g defined by

$$g(u) = (w^{-1}f)(u) = f(w \circ u)$$

M is *stable* if for all words $w \in \Sigma^\star$ and for all $f \in \mathcal{M}$ the word function $w^{-1}f$ is also in M.

3.1 Word functions in MSOLEVAL$_{\mathcal{S}}$ are recognizable

To prove the "only if" direction of Theorem 8 we use the following two theorems.

For a commutative semiring $\mathcal{S}$ and a sequence of indeterminates $\bar{X} = (X_1, \dots, X_t)$ we denote by $\mathcal{S}[\bar{X}]$ the commutative semiring of polynomials with indeterminates $\bar{X}$ and coefficients in $\mathcal{S}$. The first theorem is from [22].

Theorem 9 (Bilinear Decomposition Theorem for Word Functions).
*Let $\mathcal{S}$ be a commutative semiring. Let $f \in$ **MSOLEVAL**$_{\mathcal{S}}$ be a word function $\Sigma^+ \to \mathcal{S}$ of quantifier rank $qr(f)$. There are:*

(i) a function $\beta : \mathbb{N} \to \mathbb{N}$,

*(ii) a finite vector $F = (g_1, \dots, g_{\beta(qr(f))})$ of functions in **MSOLEVAL**$_{\mathcal{S}}$ of length $\beta(qr(f))$, with $f = g_i$ for some $i \leq \beta(qr(f))$,*

(iii) and for each $g_i \in F$, a matrix $M^{(i)} \in \mathcal{S}^{\beta(qr(f)) \times \beta(qr(f))}$

such that

$$g_i(u \circ v) = F(u) \cdot M^{(i)} F(v)^T.$$

The other theorem was first proved by G. Jacob, [17, 1].

Theorem 10 (G. Jacob 1975)**.** *Let f be a word function $f : \Sigma^\star \to \mathcal{S}$. Then f is recognizable by a weighted automaton over $\mathcal{S}$ iff there exists a finitely generated stable semimodule $\mathcal{M} \subseteq \mathcal{S}^{\Sigma^\star}$ which contains f.*

In order to prove the "only if" direction of Theorem 8 we reformulate it.

Theorem 11 (Stable Semimodule Theorem)**.** *Let $\mathcal{S}$ be a commutative semiring and let $f \in$ **MSOLEVAL**$_{\mathcal{S}}$ be a word function of quantifier rank $qr(f)$.*
There are:

(i) a function $\beta : \mathbb{N} \to \mathbb{N}$,

(ii) a finite vector $F = (g_1, \ldots, g_{\beta(qr(f))})$ of functions in **MSOLEVAL**$_{\mathscr{S}}$ *of length $\beta(qr(f))$, with $f = g_i$ for some $i \leq \beta(qr(f))$,*

such that the semimodule $\mathscr{M}[F]$ generated by F is stable.

Proof. We take F and the matrices $M^{(i)}$ from Theorem 9 stated in the introduction.

We have to show that for every fixed word w and $f \in \mathscr{M}[F]$ the function $w^{-1}f \in \mathscr{M}[F]$. As $f \in \mathscr{M}[F]$ there is a vector $A = (a_1, \ldots, a_{\beta(qr(f))}) \in \mathscr{S}^{\beta(qr(f))}$ such that

$$f(w) = A \cdot F^T(w)$$

for every fixed word w. Here $F(w)$ is shorthand for $(g_1(w), \ldots, g_{\beta(qr(f))}(w))$.

Let u be a word. We compute $(w^{-1}f)(u)$.

$$(w^{-1}f)(u) = f(w \circ u) = A \cdot F^T(w \circ u) =$$
$$\sum_{i=1}^{\beta(qr(f))} a_i g_i(w \circ u) = \sum_{i=1}^{\beta(qr(f))} a_i F(w) M^{(i)} F^T(u)$$

We put $B_i = a_i F(w) M^{(i)}$ and observe that $B_i \in \mathscr{S}^{\beta(qr(f))}$. If we take $B = \sum_i^{\beta(qr(f))} B_i$ we get that $(w^{-1}f)(u) = B \cdot F^T(u)$, hence $w^{-1}f \in \mathscr{M}[F]$. □

3.2 Recognizable word functions are definable in MSOLEVAL$_{\mathscr{S}}$

For the "if" direction we proceed as follows:

Proof. Let A be a weighted automaton of size r over $\mathscr{S}$ for words in $\Sigma^\star$. For a word w with $\ell(w) = n$, given as a function $w : [n] \to \Sigma$, the automaton A defines the function

$$f_A(w) = \alpha \cdot \mu_{w(1)} \cdot \ldots \cdot \mu_{w(n)} \cdot \gamma^T. \tag{2}$$

We have to show that $f_A \in$ **MSOLEVAL**$_{\mathscr{S}}$.

To unify notation we define

$$M^a_{i,j} = (\mu_a)_{i,j}.$$

Equation (2) is a product of n matrices and two vectors.

Let P be the product of these matrices,

$$P = \prod_{k=1}^{n} \mu_{w(k)}.$$

Using matrix algebra we get for the entry $P_{a,b}$ of P:

$$P_{a,b} = \sum_{i_{n-1}=1}^{r} \left(\sum_{i_{n-2}=1}^{r} \left(\ldots \left(\sum_{i_1=1}^{r} M^{w(1)}_{a,i_1} \cdot M^{w(2)}_{i_1,i_2} \right) M^{w(3)}_{i_2,i_3} \right) \ldots \right) M^{w(n)}_{i_{n-1},b}$$
$$= \sum_{i_1,\ldots i_{n-1} \leq r} \left(M^{w(1)}_{a,i_1} \cdot M^{w(2)}_{i_1,i_2} \cdot \ldots \cdot M^{w(n)}_{i_{n-1},b} \right)$$

Let $\pi : [n-1] \to [r]$ be the function with $\pi(k) = i_k$. We rewrite $P_{a,b}$ as:

$$P_{a,b} = \sum_{\pi:[n-1]\to[r]} \left(M^{w(1)}_{a,\pi(1)} \cdot M^{w(2)}_{\pi(1),\pi(2)} \cdot \ldots M^{w(n)}_{\pi(n-1),b} \right) \tag{3}$$

Next we compute the b coordinate of the vector $\alpha \cdot P$:

$$(\alpha \cdot P)_b = \sum_{i=1}^{r} \alpha_i \cdot P_{i,b}$$

Therefore

$$f_A(w) = \alpha \cdot P \cdot \gamma = \sum_{b=1}^{r} (\alpha \cdot P)_b \cdot \gamma_b$$
$$= \sum_{b=1}^{r} \left(\sum_{a=1}^{r} \alpha_a \cdot P_{a,b} \right) \cdot \gamma_b = \sum_{a,b \le r} \alpha_a \cdot P_{a,b} \cdot \gamma_b$$

and by using Equation (3) for $P_{a,b}$ we get:

$$\sum_{a,b\le r} \alpha_a \cdot \left(\sum_{\pi:[n-1]\to[r]} \left(M^{w(1)}_{a,\pi(1)} \cdot M^{w(2)}_{\pi(1),\pi(2)} \cdot \ldots M^{w(n)}_{\pi(n-1),b} \right) \right) \cdot \gamma_b$$

Now let $\pi' : [n] \cup \{0\} \to [r]$ be the function for which $\pi'(0) = a, \pi'(n) = b$ and $\pi'(k) = \pi(k) = i_k$ for $1 \le k \le n-1$. Then we get

$$f_A(w) =$$
$$\sum_{\pi':[n]\cup\{0\}\to[r]} \alpha_{\pi'(0)} \cdot \left[M^{w(1)}_{\pi'(0),\pi'(1)} \cdot \ldots \cdot M^{w(n)}_{\pi'(n-1),\pi'(n)} \right] \cdot \gamma_{\pi'(n)} =$$
$$\sum_{\pi':[n]\cup\{0\}\to[r]} \alpha_{\pi'(0)} \cdot \left(\prod_{k\in[n]} M^{w(k)}_{\pi'(k-1),\pi'(k)} \right) \cdot \gamma_{\pi'(n)} \tag{4}$$

To convert Equation (4) into an expression in $\mathbf{MSOLEVAL}_{\mathscr{S}}$ we use a few lemmas:

First, let S be any set and $\pi : S \to [r]$ be any function. π induces a partition of S into sets $U^{\pi}_1, \ldots, U^{\pi}_r$ by $U^{\pi}_i = \{s \in S : \pi(s) = i\}$. Conversely, every partition $\mathscr{U} = (U_1, \ldots, U_r)$ of S induces a function $\pi_{\mathscr{U}}$ by setting $\pi_{\mathscr{U}}(s) = i$ for $s \in U_i$. To pass between functions π with finite range $[r]$ and partitions into r-sets we use the following lemma:

Lemma 12. *Let $E(\pi)$ be any expression depending on π.*

$$\sum_{\pi:S\to[r]} E(\pi) = \sum_{\mathscr{U}} E(\pi_{\mathscr{U}}) = \sum_{U_1,\ldots U_r:Partition(U_1,\ldots,U_r)} E(\pi_{\mathscr{U}})$$

where $\mathscr{U}$ ranges over all partitions of S into r sets $U_i : i \in [r]$. Clearly, $Partition(U_1, \ldots, U_r)$ can be written in **MSOL**.

Second, to convert the factors $\alpha_{\pi'(0)}$ and $\gamma_{\pi'(n)}$ we proceed as follows:

Lemma 13. *Let α_i be the unique value of the coordinate of α such that $0 \in U_i$. Similarly, let γ_i be the unique value of the coordinate of γ such that $n \in U_i$.*

$$\alpha_{\pi'(0)} = \prod_{i=1}^{r} \prod_{0 \in U_i} \alpha_i$$
$$\gamma_{\pi'(n)} = \prod_{i=1}^{r} \prod_{n \in U_i} \gamma_i$$

Proof. First we note that, as $\mathscr{U}$ is the partition induced by π', the restriction of π' to U_i is constant for all $i \in [r]$. Next we note that the product ranging over the empty set gives the value 1. ☐

Similarly, to convert the factor $\prod_{k\in[n]} M^{w(k)}_{\pi'(k-1),\pi'(k)}$ use the following lemma:

Lemma 14. *Let $m_{i,j,w(v)}$ be the unique value of the (i,j)-entry of the matrix $\mu_{w(v)}$ such that $v \in U_i$ and $v+1 \in U_j$.*

$$\prod_{k\in[n]} M^{w(k)}_{\pi'(k-1),\pi'(k)} = \prod_{i,j=1}^{r} \left(\prod_{v-1\in U_i, v\in U_j} m_{i,j,w(v)} \right)$$

Using the fact that every element which is the interpretation of a term in $\mathscr{S}$ can be written as an expression in $\mathbf{MSOLEVAL}_{\mathscr{S}}$, Lemma 6 in Section 2.2, we can write $U_i(v)$ instead of $v \in U_i$, and see that the monomials of Lemmas 12, 13 and 14 are indeed in $\mathbf{MSOLEVAL}_{\mathscr{S}}$. Now we apply the fact that the pointwise product of two word functions in $\mathbf{MSOLEVAL}_{\mathscr{S}}$ is again a function in $\mathbf{MSOLEVAL}_{\mathscr{S}}$, Proposition 7 in Section 2.2,

to Lemmas 12, 13 and 14 and complete the proof of Theorem 8. ☐

4 Weighted MSOL and MSOLEVAL

In this section we compare the formalism of weighted **MSOL**, **WMSOL**, with our $\mathbf{MSOLEVAL}_{\mathscr{S}}$ for arbitrary commutative semirings. In [7, 8] and [2] two fragments of weighted **MSOL** are discussed. One is based on *unambiguous* formulas (a semantic concept), the other on *step formulas* based on the Boolean fragment of weighted **MSOL** (a syntactic definition). The two fragments have equal expressive power, as stated in [2], and characterize the functions recognizable by weighted automata. We denote both versions by **RMSOL**.

4.1 Syntax of WMSOL, the weighted version of MSOL

The definitions and properties of **WMSOL** and its fragments are taken literally from [2]. The syntax of formulas ϕ of weighted **MSOL**, denoted by **WMSOL**, is given inductively in Backus–Naur form by

$$\phi ::= k \mid P_a(x) \mid \neg P_a(x) \mid x \le y \mid \neg x \le y \mid x \in X \mid x \notin X$$
$$\mid \phi \vee \psi \mid \phi \wedge \psi \mid \exists x.\phi \mid \exists X.\phi \mid \forall x.\phi \mid \forall X.\phi$$

where $k \in \mathscr{S}$, $a \in \Sigma$. The set of weighted **MSOL**-formulas over the field $\mathscr{S}$ and the alphabet Σ is denoted by $\mathbf{MSOL}(\mathscr{S},\Sigma)$. **bMSOL** *formulas* and **bMSOL**-*step formulas* are defined below. **bMSOL** is the Boolean fragment of **WMSOL**, and its name is justified by Lemma 15. **RMSOL** is the fragment of

WMSOL where universal second order quantification is restricted to **bMSOL** and first order universal quantification is restricted to **bMSOL**-step formulas.

The syntax of weighted **bMSOL** is given by

$$\phi ::= 0 \mid 1 \mid P_a(x) \mid x \leq y \mid x \in X \mid \neg\phi \mid \phi \wedge \psi \mid \forall x.\phi \mid \forall X.\phi$$

where $a \in \Sigma$.

The set of weighted **MSOL**-formulas over the commutative semiring $\mathscr{S}$ and the alphabet Σ is denoted by **WMSOL**$(\mathscr{S},\Sigma)$.

Instead of defining step-formulas as in [2] we use Lemma 3 from [2] as our definition.

A **bMSOL**-step formula ψ is a formula of the form

$$\psi = \bigvee_{i \in I} (\phi_i \wedge k_i) \tag{5}$$

where I is a finite set, $\phi_i \in$ **bMSOL** and $k_i \in \mathscr{S}$.

4.2 Semantics of WMSOL, and translation of RMSOL into MSOLEVAL$_{\mathscr{S}}$

Next we define the semantics of **WMSOL** and, where it is straightforward, simultaneously also its translations into **MSOLEVAL**$_{\mathscr{S}}$.

The evaluations of weighted formulas $\phi \in$ **WMSOL**$(\mathscr{S},\Sigma)$ on a word w are denoted by $WE(\phi,w,\sigma)$, where σ is an assignment of the variables of ϕ to positions, respectively sets of positions, in w.

We denote the evaluation of term t of **MSOLEVAL**$_{\mathscr{S}}$ for a word w and an assignment for the free variables σ by $E(t,w,\sigma)$. $\mathrm{tv}(\phi)$ stands for the truth value of ϕ (subject to an assignment for the free variables), i.e., $E(\mathrm{tv}(\phi),w,\sigma) = 0 \in \mathscr{S}$ for false and $E(\mathrm{tv}(\phi),w,\sigma) = 1 \in \mathscr{S}$ for true. The term $\mathrm{tv}(\phi)$ is used as an abbreviation for

$$\mathrm{tv}(\phi) = \sum_{U: U = A \wedge \phi} 1$$

where $U = A$ stands for $\forall x(U(x) \leftrightarrow x = x)$ and U does not occur freely in ϕ. Indeed, we have

$$E(\mathrm{tv}(\phi),w,\sigma) = \begin{cases} 1 & (w,\sigma) \models \phi \\ 0 & \text{else} \end{cases}$$

We denote by $TRUE(x)$ the formula $x = x$ with free first order variable x. Similarly, $TRUE(X)$ denotes the formula $\exists y \in X \vee \neg\exists y \in X$ with free set variable X.

The evaluations of formulas $\phi \in$ **WMSOL** and their translations are now defined inductively.

(i) For $k \in \mathscr{S}$ we have $tr(k) = k$ and $WE(k,w,\sigma)) = E(tr(k),w,\sigma)) = k$.

(ii) For atomic formulas θ we have $tr(\theta) = \mathrm{tv}(\theta)$ and

$$WE(\theta,w,\sigma) = E(tr(\theta),w,\sigma) = E(\mathrm{tv}(\theta),w,\sigma)$$

(iii) For negated atomic formulas we have

$$tr(\neg\theta) = 1 - tr(\theta) = 1 - \mathrm{tv}(\theta)$$

and

$$WE(\neg\theta,w,\sigma) = 1 - E(\mathrm{tv}(\theta),w,\sigma).$$

(iv) $tr(\phi_1 \vee \phi_2) = tr(\phi_1) + tr(\phi_2)$ and

$$WE(\phi_1 \vee \phi_2, w, \sigma) = E(tr(\phi_1) + tr(\phi_2), w, \sigma) = E(tr(\phi_1), w, \sigma) + E(tr(\phi_2), w, \sigma).$$

(v) $tr(\exists x.\phi) = \sum_{x:TRUE(x)} tr(\phi)$ and

$$WE(\exists x.\phi, w, \sigma) = E(\sum_{x:TRUE(x)} tr(\phi, w, \sigma)) = \sum_{x:TRUE(x)} E(tr(\phi, w, \sigma)).$$

(vi) $tr(\exists X.\phi) = \sum_{X:TRUE(X)} tr(\phi)$ and

$$WE(\exists X.\phi, w, \sigma) = E(\sum_{X:TRUE(X)} tr(\phi, w, \sigma)) = \sum_{X:TRUE(X)} E(tr(\phi, w, \sigma)).$$

(vii) $tr(\phi_1 \wedge \phi_2) = tr(\phi_1) \cdot tr(\phi_2)$ and

$$WE(\phi_1 \wedge \phi_2, w, \sigma) = E(tr(\phi_1) \cdot tr(\phi_2), w, \sigma) = E(tr(\phi_1), w, \sigma) \cdot E(tr(\phi_2), w\sigma).$$

So far the definition of WE was given using the evaluation function E and the translation was straightforward. Problems arise with the universal quantifiers.

The unrestricted definition of WE for **WMSOL** given below gives us functions which are not recognizable by weighted automata, and the straightforward translation defined below gives us expressions which are not in **MSOLEVAL**$_{\mathscr{S}}$:

(viii) $tr(\forall x.\phi) = \prod_{x:TRUE(x)} tr(\phi)$ and

$$WE(\forall x.\phi, w, \sigma) = E(\prod_{x:TRUE(x)} tr(\phi, w, \sigma)) = \prod_{x:TRUE(x)} E(tr(\phi, w, \sigma)).$$

The formula $\phi_{sq} = \forall x.\forall y.2$ gives the function $2^{\ell(w)^2}$ and is not a **bMSOL**-step formula. The straightforward translation tr gives the term

$$\prod_{x:TRUE(x)} \left(\prod_{y:TRUE(y)} 2 \right) = \prod_{(x,y):TRUE(x,y)} 2,$$

which is a product over the tuples of a binary relation, hence not in **MSOLEVAL**$_{\mathscr{S}}$.

(ix) $tr(\forall X.\phi) = \prod_{X:TRUE(X)} tr(\phi)$ and

$$WE(\forall X.\phi, w, \sigma) = E(\prod_{X:TRUE(X)} tr(\phi, w, \sigma)) = \prod_{X:TRUE(X)} E(tr(\phi, w, \sigma)).$$

Here the translation gives a product $\prod_{X:TRUE(X)}$ ranging over subsets, which is not an expression in **MSOLEVAL**$_{\mathscr{S}}$.

In **RMSOL**, universal second order quantification is restricted to formulas of **bMSOL**, and first order universal quantification is restricted to **bMSOL**-step formulas.

In [2, page 590], after Figure 1, the following is stated:

Lemma 15. *The evaluation WE of a* **bMSOL**-*formula ϕ assumes values in $\{0, 1\}$ and coincides with the standard semantics of ϕ as an unweighted* **MSOL**-*formula.*

Because the translation of universal quantifiers using tr leads outside of $\mathbf{MSOLEVAL}_{\mathcal{S}}$, we define a proper translation $tr' : \mathbf{RMSOL} \to \mathbf{MSOLEVAL}_{\mathcal{S}}$.

Using Lemma 15 we set $tr'(\phi) = \mathrm{tv}(\phi)$, for ϕ a **bMSOL**-formula.

For universal first order quantification of **bMSOL**-step formulas

$$\psi = \bigvee_{i \in I} (\phi_i \wedge k_i) \tag{6}$$

we compute $WE(\forall x.\psi, w, \sigma)$ and $E(tr(\forall x.\psi), w, \sigma)$ as follows, leaving the steps for the translation of $tr(\forall x.\psi)$ to the reader.

$$\begin{gathered}
WE((\forall x.\psi), w, \sigma) = E(tr(\forall x.\psi), w, \sigma) = \\
E(tr(\forall x. \bigvee_{i \in I} (\phi_i \wedge k_i)), w, \sigma) = \\
\prod_{x:TRUE(x)} E(tr(\bigvee_{i \in I} (\phi_i \wedge k_i))), w, \sigma) = \\
\prod_{x:TRUE(x)} (\sum_{i \in I} (E(tr'(\phi_i)) \cdot k_i), w, \sigma)) = \\
\prod_{x:TRUE(x)} (\sum_{i \in I} (E(\mathrm{tv}(\phi_i), w, \sigma) \cdot k_i)))
\end{gathered}$$

Clearly, the formula of the last line, $\prod_{x:TRUE(x)} (\sum_{i \in I} (\mathrm{tv}(\phi_i)) \cdot k_i))$ is an expression in $\mathbf{MSOLEVAL}_{\mathcal{S}}$.

For universal second order quantification of **bMSOL**-formulas ψ we use Lemma 15 and get

$$WE(\forall X.\psi, w, \sigma) = E(tr'(\forall X \psi), w, \sigma) = E(\mathrm{tv}(\forall X \psi), w, \sigma)$$

Clearly, the expression $\mathrm{tv}(\forall X \psi)$ is an expression in $\mathbf{MSOLEVAL}_{\mathcal{S}}$. Thus we have proved:

Theorem 16. *Let $\mathcal{S}$ be a commutative semiring. For every expression $\phi \in$ **RMSOL** there is an expression $tr'(\phi) \in \mathbf{MSOLEVAL}_{\mathcal{S}}$ such that $WE(\phi, w, \sigma) = E(tr'(\phi), w, \sigma)$, i.e., ϕ and $tr'(\phi)$ define the same word function.*

4.3 Translation from $\mathbf{MSOLEVAL}_{\mathcal{S}}$ to RMSOL

It follows from our Theorem 8 and the characterization in [8] of recognizable word functions as the functions definable in **RMSOL**, that the converse is also true. We now give a direct proof of the converse without using weighted automata.

Theorem 17. *Let $\mathcal{S}$ be a commutative semiring. For every expression $t \in \mathbf{MSOLEVAL}_{\mathcal{S}}$ there is a formula $\phi_t \in$ **RMSOL** such that $WE(\phi_t, w, \sigma) = E(t, w, \sigma)$, i.e., ϕ_t and t define the same word function.*

Proof. (i) Let $t = \prod_{x:\phi(x)} \alpha$ be a $\mathbf{MSOLEVAL}_{\mathcal{S}}$- monomial. We note that

$$\alpha \cdot \mathrm{tv}(\phi) + \mathrm{tv}(\neg\phi) = \begin{cases} \alpha & \text{if } \phi \text{ is true} \\ 1 & \text{else} \end{cases}$$

Furthermore, by Lemma 15 $\phi \in$ **bMSOL**. So we put

$$\phi_t = \forall x.((\phi(x) \wedge \alpha) \vee \neg\phi(x))$$

(ii) Let $t_1 = \sum_{U:\phi(U)} t$ and let ϕ_t be the translation of t. Then

$$\phi_{t_1} = \exists U.(\phi_t \wedge \phi(U))$$

□

5 Conclusions

We have given two proofs that **RMSOL** and **MSOLEVAL** with values in $\mathscr{S}$ have the same expressive power over words. One proof uses model theoretic tools to show directly that **MSOLEVAL** captures the functions recognizable by weighted automata. The other proof shows how to translate the formalisms from one into the other. Adapting the translation proof, it should be possible to extend the result to tree functions as well, cf. [10].

Although in this paper we dealt only with word functions, our formalism **MSOLEVAL**, introduced first fifteen years ago, was originally designed to deal with definability of graph parameters and graph polynomials, [6, 22, 24, 21]. It has been useful, since, in many applications in algorithmic and structural graph theory and descriptive complexity. Its use in characterizing word functions recognizable by weighted automata is new. **MSOLEVAL** can be seen as an analogue of the *Skolem elementary functions* aka *lower elementary functions*, [25, 26], adapted to the framework of *meta-finite model theory* as defined in [15].

The formalism **WMSOL** of weighted logic was first invented in 2005 in [7] and since then used to characterize word and tree functions recognizable by weighted automata, [10]. These characterizations need some syntactic restrictions which lead to the formalisms of **RMSOL**. No such syntactic restrictions are need for the characterization of recognizable word functions using **MSOLEVAL**. The weighted logic **WMSOL** can also be defined for general relational structures. However, it is not immediate which syntactic restrictions are needed, if at all, to obtain algorithmic applications similar to the ones obtained using **MSOLEVAL**, cf. [6, 5, 23].

Acknowledgements We are thankful to Jacques Sakarovitch, Géraud Sénizergues and Amir Shpilka for useful guidence on the subject of weighted automata. We are thankful to Manfred Droste, Tomer Kotek and Elena Ravve and several anonymous readers, for their valuable comments.

References

[1] J. Berstel & C. Reutenauer (1984): *Rational Series and their languages*. *EATCS Monographs on Theoretical Computer Science* 12, Springer.

[2] B. Bollig, P. Gastin, B. Monmege & M. Zeitoun (2010): *Pebble weighted automata and transitive closure logics*. In: *ICALP'10*, *Lecture Notes in Computer Science* 6199, Springer, pp. 587–598, doi:10.1007/978-3-642-11301-7.

[3] J.W. Carlyle & A. Paz (1971): *Realizations by Stochastic Finite Automata*. *J. Comp. Syst. Sc.* 5, pp. 26–40, doi:10.1016/S0022-0000(71)80005-3.

[4] A. Cobham (1978): *Representation of a Word Function as the Sum of Two Functions*. *Mathematical Systems Theory* 11, pp. 373–377, doi:10.1007/BF01768487.

[5] B. Courcelle, J.A. Makowsky & U. Rotics (2000): *Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width*. *Theory of Computing Systems* 33.2, pp. 125–150, doi:10.1007/s002249910009.

[6] B. Courcelle, J.A. Makowsky & U. Rotics (2001): *On the Fixed Parameter Complexity of Graph Enumeration Problems Definable in Monadic Second Order Logic*. *Discrete Applied Mathematics* 108(1-2), pp. 23–52, doi:10.1016/S0166-218X(00)00221-3.

[7] M. Droste & P. Gastin (2005): *Weighted Automata and Weighted Logics*. In: *ICALP 2005*, pp. 513–525, doi:10.1007/11523468_42.

[8] M. Droste & P. Gastin (2007): *Weighted automata and weighted logics*. *Theor. Comput. Sci.* 380(1-2), pp. 69–86, doi:10.1016/j.tcs.2007.02.055.

[9] M. Droste, W. Kuich & H. Vogler, editors (2009): *Handbook of Weighted Automata*. EATCS Monographs on Theoretical Computer Science, Springer.

[10] M. Droste & H. Vogler (2006): *Weighted tree automata and weighted logics*. *Theor. Comput. Sci.* 366, pp. 228–247, doi:10.1016/j.tcs.2006.08.025.

[11] Manfred Droste & Werner Kuich (2013): *Weighted finite automata over semirings*. *Theor. Comput. Sci.* 485, pp. 38–48, doi:10.1016/j.tcs.2013.02.028.

[12] H.-D. Ebbinghaus & J. Flum (1995): *Finite Model Theory*. Perspectives in Mathematical Logic, Springer, doi:10.1007/978-3-662-03182-7.

[13] M. Fliess (1974): *Matrices de Hankel*. *J Maths Pures Appl* 53, pp. 197–222. Erratum in volume 54.

[14] B. Godlin, T. Kotek & J.A. Makowsky (2008): *Evaluation of graph polynomials*. In: *34th International Workshop on Graph-Theoretic Concepts in Computer Science, WG08*, *Lecture Notes in Computer Science* 5344, pp. 183–194, doi:10.1007/978-3-540-92248-3_17.

[15] E. Grädel & Y. Gurevich (1998): *Metafinite Model Theory*. *Information and Computation* 140, pp. 26–81, doi:10.1006/inco.1997.2675.

[16] J. E. Hopcroft & J. D. Ullman (1980): *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Series in Computer Science, Addison-Wesley.

[17] G. Jacob (1975): *Représentations et substitutions matricielles dans la théorie algébrique des transductions*. Ph.D. thesis, Université de Paris, VII.

[18] T. Kotek (March 2012): *Definability of combinatorial functions*. Ph.D. thesis, Technion - Israel Institute of Technology, Haifa, Israel. Submitted.

[19] T. Kotek & J.A. Makowsky (2012): *Connection Matrices and the Definability of Graph Parameters*. In: *CSL 2012*, pp. 411–425, doi:10.4230/LIPIcs.CSL.2012.411.

[20] T. Kotek, J.A. Makowsky & B. Zilber (2008): *On Counting Generalized Colorings*. In: *Computer Science Logic, CSL'08*, *Lecture Notes in Computer Science* 5213, pp. 339–353, doi:10.1007/978-3-540-87531-4_25.

[21] T. Kotek, J.A. Makowsky & B. Zilber (2011): *On Counting Generalized Colorings*. In M. Grohe & J.A. Makowsky, editors: *Model Theoretic Methods in Finite Combinatorics*, *Contemporary Mathematics* 558, American Mathematical Society, pp. 207–242, doi:10.1090/conm/558/11052.

[22] J.A. Makowsky (2004): *Algorithmic uses of the Feferman-Vaught theorem*. *Annals of Pure and Applied Logic* 126.1-3, pp. 159–213, doi:10.1016/j.apal.2003.11.002.

[23] J.A. Makowsky (2005): *Coloured Tutte polynomials and Kauffman brackets for graphs of bounded tree width*. *Discrete Applied Mathematics* 145(2), pp. 276–290, doi:10.1016/j.dam.2004.01.016.

[24] J.A. Makowsky (2008): *From a Zoo to a Zoology: Towards a general theory of graph polynomials*. *Theory of Computing Systems* 43, pp. 542–562, doi:10.1007/s00224-007-9022-9.

[25] Th. Skolem (1962): *Proof of some theorems on recursively enumerable sets*. *Notre Dame Journal of Formal Logic* 3.2, pp. 65–74, doi:10.1305/ndjfl/1093957149.

[26] S.A. Volkov (2010): *On a class of Skolem elementary functions*. *Journal of Applied and Industrial Mathematics* 4.4, pp. 588–599, doi:10.1134/S1990478910040149.

Approximating the minimum cycle mean

Krishnendu Chatterjee[*]
IST Austria
Institute of Science and Technology
Klosterneuburg, Austria

Monika Henzinger,[†] Sebastian Krinninger,[‡] Veronika Loitzenbauer[§]
University of Vienna
Faculty of Computer Science
Vienna, Austria

We consider directed graphs where each edge is labeled with an integer weight and study the fundamental algorithmic question of computing the value of a cycle with minimum mean weight. Our contributions are twofold: (1) First we show that the algorithmic question is reducible in $O(n^2)$ time to the problem of a logarithmic number of *min-plus* matrix multiplications of $n \times n$-matrices, where n is the number of vertices of the graph. (2) Second, when the weights are nonnegative, we present the first $(1+\varepsilon)$-approximation algorithm for the problem and the running time of our algorithm is $\widetilde{O}(n^\omega \log^3 (nW/\varepsilon)/\varepsilon)^1$, where $O(n^\omega)$ is the time required for the *classic* $n \times n$-matrix multiplication and W is the maximum value of the weights.

1 Introduction

Minimum cycle mean problem. We consider a fundamental graph algorithmic problem of computing the value of a minimum mean-weight cycle in a finite directed graph. The input to the problem is a directed graph $G = (V, E, w)$ with a finite set V of n vertices, E of m edges, and a weight function w that assigns an integer weight to every edge. Given a cycle C, the mean weight $\mu(C)$ of the cycle is the ratio of the sum of the weights of the cycle and the number of edges in the cycle. The algorithmic question asks to compute $\mu = \min\{\mu(C) \mid C \text{ is a cycle}\}$: the minimum cycle mean. The minimum cycle mean problem is an important problem in combinatorial optimization and has a long history of algorithmic study. An $O(nm)$-time algorithm for the problem was given by Karp [17]; and the current best known algorithm for the problem, which is over two decades old, by Orlin and Ahuja require $O(m\sqrt{n}\log(nW))$ time [22], where W is the maximum absolute value of the weights.

Applications. The minimum cycle mean problem is a basic combinatorial optimization problem that has numerous applications in network flows [2]. In the context of formal analysis of reactive systems, the performance of systems as well as the average resource consumption of systems is modeled as the minimum cycle mean problem. A reactive system is modeled as a directed graph, where vertices represent states of the system, edges represent transitions, and every edge is assigned a *nonnegative* integer representing the resource consumption (or delay) associated with the transition. The computation of a minimum average

[*]Supported by the Austrian Science Fund (FWF): P23499-N23 and S11407-N23 (RiSE), an ERC Start Grant (279307: Graph Games), and a Microsoft Faculty Fellows Award.

[†]Supported by the Austrian Science Fund (FWF): P23499-N23, the Vienna Science and Technology Fund (WWTF) grant ICT10-002, and the University of Vienna (IK I049-N).

[‡]Supported by the Austrian Science Fund (FWF): P23499-N23 and the University of Vienna (IK I049-N).

[§]Supported by the Vienna Science and Technology Fund (WWTF) grant ICT10-002.

[1]The $\widetilde{O}$-notation hides a polylogarithmic factor.

resource consumption behavior (or minimum average response time) corresponds to the computation of the minimum cycle mean. Several recent works model other quantitative aspects of system analysis (such as robustness) also as the mean-weight problem (also known as *mean-payoff objectives*) [4, 9].

Results. This work contains the following results.

1. *Reduction to min-plus matrix multiplication.* We show that the minimum cycle mean problem is reducible in $O(n^2)$ time to the problem of a logarithmic number of min-plus matrix multiplications of $n \times n$-matrices, where n is the number of vertices of the graph. Our result implies that algorithmic improvements for min-plus matrix multiplication will carry over to the minimum cycle mean problem with a logarithmic multiplicative factor and $O(n^2)$ additive factor in the running time.

2. *Faster approximation algorithm.* When the weights are nonnegative, we present the first $(1+\varepsilon)$-approximation algorithm for the problem that outputs $\hat{\mu}$ such that $\mu \leq \hat{\mu} \leq (1+\varepsilon)\mu$ and the running time of our algorithm is $\widetilde{O}(n^{\omega} \log^3(nW/\varepsilon)/\varepsilon)$. As usual, the $\widetilde{O}$-notation is used to "hide" a polylogarithmic factor, i.e., $\widetilde{O}(T(n,m,W)) = O(T(n,m,W) \cdot \text{polylog}(n))$, and $O(n^{\omega})$ is the time required for the *classic* $n \times n$-matrix multiplication. The current best known bound for ω is $\omega < 2.3727$. The worst case complexity of the current best known algorithm for the minimum cycle mean problem is $O(m\sqrt{n}\log(nW))$ [22], which could be as bad as $O(n^{2.5}\log(nW))$. Thus for $(1+\varepsilon)$-approximation our algorithm provides better dependence in n. Note that in applications related to systems analysis the weights are always nonnegative (they represent resource consumption, delays, etc); and the weights are typically small, whereas the state space of the system is large. Moreover, due to imprecision in modeling, approximations in weights are already introduced during the modeling phase. Hence $(1+\varepsilon)$-approximation of the minimum cycle mean problem with small weights and large graphs is a very relevant algorithmic problem for reactive system analysis, and we improve the long-standing complexity of the problem.

   The key technique that we use to obtain the approximation algorithm is a combination of the value iteration algorithm for the minimum cycle mean problem, and a technique used for an approximation algorithm for all-pair shortest path problem for directed graphs. Table 1 compares our algorithm with the asymptotically fastest existing algorithms.

Reference	**Running time**	**Approximation**	**Range**
Karp [17]	$O(mn)$	exact	$[-W,W]$
Orlin and Ahuja [22]	$O(m\sqrt{n}\log(nW))$	exact	$[-W,W]\cap\mathbb{Z}$
Sankowski [24] (implicit)	$\widetilde{O}(Wn^{\omega}\log(nW))$	exact	$[-W,W]\cap\mathbb{Z}$
Butkovic and Cuninghame-Green [6]	$O(n^2)$	exact	$\{0,1\}$
This paper	$\widetilde{O}(n^{\omega}\log^3(nW/\varepsilon)/\varepsilon)$	$1+\varepsilon$	$[0,W]\cap\mathbb{Z}$

Table 1: Current fastest asymptotic running times for computing the minimum cycle mean

1.1 Related work

The minimum cycle mean problem is basically equivalent to solving a deterministic Markov decision process (MDP) [31]. The latter can also be seen as a single-player mean-payoff game [10, 13, 31]. We distinguish two types of algorithms: algorithms that are independent of the weights of the graph and algorithms that depend on the weights in some way. By W we denote the maximum absolute edge weight of the graph.

Algorithms independent of weights. The classic algorithm of Karp [17] uses a dynamic programming approach to find the minimum cycle mean and runs in time $O(mn)$. The main drawback of Karp's algorithm is that its best-case and worst-case running times are the same. The algorithms of Hartmann and Orlin [15] and of Dasdan and Gupta [8] address this issue, but also have a worst-case complexity of $O(mn)$. By solving the more general parametric shortest path problem, Karp and Orlin [18] can compute the minimum cycle mean in time $O(mn\log n)$. Young, Tarjan, and Orlin [27] improve this running time to $O(mn+n^2\log n)$.

A well known algorithm for solving MDPs is the value iteration algorithm. In each iteration this algorithm spends time $O(m)$ and in total it performs $O(nW)$ iterations. Madani [20] showed that, for *deterministic* MDPs (i.e., weighted graphs for which we want to find the minimum cycle mean), a certain variant of the value iteration algorithm "converges" to the optimal cycle after $O(n^2)$ iterations which gives a running time of $O(mn^2)$ for computing the minimum cycle mean. Using similar ideas he also obtains a running time of $O(mn)$. Howard's policy iteration algorithm is another well-known algorithm for solving MDPs [16]. The complexity of this algorithm for deterministic MDPs is unresolved. Recently, Hansen and Zwick [14] provided a class of weighted graphs on which Howard's algorithm performs $\Omega(n^2)$ iterations where each iteration takes time $O(m)$.

Algorithms depending on weights. If a graph is complete and has only two different edge weights, then the minimum cycle mean problem problem can be solved in time $O(n^2)$ because the matrix of its weights is bivalent [6].

Another approach is to use the connection to the problem of detecting a negative cycle. Lawler [19] gave a reduction for finding the minimum cycle mean that performs $O(\log(nW))$ calls to a negative cycle detection algorithm. The main idea is to perform binary search on the minimum cycle mean. In each search step the negative cycle detection algorithm is run on a graph with modified edge weights. Orlin and Ahuja [22] extend this idea by the approximate binary search technique [29]. By combining approximate binary search with their scaling algorithm for the assignment problem they can compute the minimum cycle mean in time $O(m\sqrt{n}\log nW)$.

Note that in its full generality the single-source shortest paths problem (SSSP) also demands the detection of a negative cycle reachable from the source vertex.[2] Therefore it is also possible to reduce the minimum cycle mean problem to SSSP. The best time bounds on SSSP are as follows. Goldberg's scaling algorithm [12] solves the SSSP problem (and therefore also the negative cycle detection problem) in time $O(m\sqrt{n}\log W)$. McCormick [21] combines approximate binary search with Goldberg's scaling algorithm to find the minimum cycle mean in time $O(m\sqrt{n}\log nW)$, which matches the result of Orlin and Ahuja [22]. Sankowski's matrix multiplication based algorithm [24] solves the SSSP problem in time $\widetilde{O}(Wn^{\omega})$. By combining binary search with Sankowski's algorithm, the minimum cycle mean problem can be solved in time $\widetilde{O}(Wn^{\omega}\log nW)$

Approximation of minimum cycle mean. To the best of our knowledge, our algorithm is the first approximation algorithm specifically for the minimum cycle mean problem. There are both additive and multiplicative fully polynomial-time approximation schemes for solving mean-payoff games [23, 5], which is a more general problem. Note that in contrast to finding the minimum cycle mean it is not known whether the exact solution to a mean-payoff game can be computed in polynomial time. The results of [23] and [5] are obtained by reductions to a pseudo-polynomial algorithm for solving mean-payoff games. In the case of the minimum cycle mean problem, these reductions do not provide an improvement over the current fastest exact algorithms mentioned above.

[2]Remember that, for example, Dijkstra's algorithm for computing single-source shortest paths requires non-negative edge weights which excludes the possibility of negative cycles.

Min-plus matrix multiplication. Our approach reduces the problem of finding the minimum cycle mean to computing the (approximate) min-plus product of matrices. The naive algorithm for computing the min-plus product of two matrices runs in time $O(n^3)$. To date, no algorithm is known that runs in time $O(n^{3-\alpha})$ for some $\alpha > 0$, so-called *truly subcubic* time. This is in contrast to classic matrix multiplication that can be done in time $O(n^\omega)$ where the current best bound on ω is $\omega < 2.3727$ [25]. Moreover, Williams and Williams [26] showed that computing the min-plus product is computationally equivalent to a series of problems including all-pairs shortest paths and negative triangle detection. This provides evidence for the hardness of these problems. Still, the running time of $O(n^3)$ for the min-plus product can be improved by logarithmic factors and by assuming small integer entries.

Fredman [11] gave an algorithm for computing the min-plus product with a slightly subcubic running time of $O(n^3(\log\log n)^{1/3}/(\log n)^{1/3})$. This algorithm is "purely combinatorial", i.e., it does not rely on fast algorithms for classic matrix multiplication. After a long line of improvements, the current fastest such algorithm by Chan [7] runs in time $O(n^3(\log\log n)^3/(\log n)^2)$.

A different approach for computing the min-plus product of two integer matrix is to reduce the problem to classic matrix multiplication [28]. In this way, the min-plus product can be computed in time $O(Mn^\omega \log M)$ which is pseudo-polynomial since M is the maximum absolute integer entry [3]. This observation was used by Alon, Galil, and Margalit [3] and Zwick [30] to obtain faster all-pairs shortest paths algorithms in directed graphs for the case of small integer edge weights. Zwick also combines this min-plus matrix multiplication algorithm with an adaptive scaling technique that allows to compute $(1+\varepsilon)$-approximate all-pairs shortest paths in graphs with non-negative edge weights. Our approach of finding the minimum cycle mean extensively uses this technique.

2 Definitions

Throughout this paper we let $G = (V, E, w)$ be a weighted directed graph with a finite set of vertices V and a set of edges E such that every vertex has at least one outgoing edge. The weight function w assigns a nonnegative integer weight to every edge. We denote by n the number of vertices of G and by m the number of edges of G. Note that $m \geq n$ because every vertex has at least one outgoing edge.

A *path* is a finite sequence of edges $P = (e_1, \ldots, e_k)$ such that for all consecutive edges $e_i = (x_i, y_i)$ and $e_{i+1} = (x_{i+1}, y_{i+1})$ of P we have $y_i = x_{i+1}$. Note that edges may be repeated on a path, we *do not* only consider simple paths. A *cycle* is a path in which the start vertex and the end vertex are the same. The *length of a path P* is the number of edges of P. The *weight of a path* $P = (e_1, \ldots, e_k)$, denoted by $w(P)$ is the sum of its edge weights, i.e. $w(P) = \sum_{1 \leq i \leq k} w(e_i)$.

The *minimum cycle mean* of G is the minimum mean weight of any cycle in G. For every vertex x we denote by $\mu(x)$ the value of the minimum mean-weight cycle reachable from x. The minimum cycle mean of G is simply the minimum $\mu(x)$ over all vertices x. For every vertex x and every integer $t \geq 1$ we denote by $\delta_t(x)$ the minimum weight of all paths starting at x that have length t, i.e., consist of exactly t edges. For all pairs of vertices x and y and every integer $t \geq 1$ we denote by $d_t(x,y)$ the minimum weight of all paths of length t from x to y. If no such path exists we set $d_t(x,y) = \infty$.

For every matrix A we denote by $A[i,j]$ the entry at the i-th row and the j-th column of A. We only consider $n \times n$ matrices with integer entries, where n is the size of the graph. We assume that the vertices of G are numbered consecutively from 1 to n, which allows us to use $A[x,y]$ to refer to the entry of A belonging to vertices x and y. The *weight matrix D of G* is the matrix containing the weights of G. For all pairs of vertices x and y we set $D[x,y] = w(x,y)$ if the graph contains the edge (x,y) and $D[x,y] = \infty$ otherwise.

We denote the *min-plus product* of two matrices A and B by $A \otimes B$. The min-plus product is defined as follows. If $C = A \otimes B$, then for all indices $1 \leq i, j \leq n$ we have $C[i, j] = \min_{1 \leq k \leq n}(A[i,k] + B[k, j])$. We denote by A^t the t-th power of the matrix A. Formally, we set $A^1 = A$ and $A^{t+1} = A \otimes A^t$ for $t \geq 1$. We denote by ω the exponent of classic matrix multiplication, i.e., the product of two $n \times n$ matrices can be computed in time $O(n^\omega)$. The current best bound on ω is $\omega < 2.3727$ [25].

3 Reduction of minimum cycle mean to min-plus matrix multiplication

In the following we explain the main idea of our approach which is to use min-plus matrix multiplication to find the minimum cycle mean. The well-known value iteration algorithm uses a dynamic programming approach to compute in each iteration a value for every vertex x from the values of the previous iteration. After t iterations, the value computed by the value iteration algorithm for vertex x is equal to $\delta_t(x)$, the minimum weight of all paths with length t starting at x. We are actually interested in $\mu(x)$, the value of the minimum mean-weight cycle reachable from x. It is well known that $\lim_{t\to\infty} \delta_t(x)/t = \mu(x)$ and that the value of $\mu(x)$ can be computed from $\delta_t(x)$ if t is large enough $(t = O(n^3W))$ [31].[3] Thus, one possibility to determine $\mu(x)$ is the following: first, compute $\delta_t(x)$ for t large enough with the value iteration algorithm and then compute $\mu(x)$ from $\delta_t(x)$. However, using the value iteration algorithm for computing $\delta_t(x)$ is expensive because its running time is linear in t and thus pseudo-polynomial.

Our idea is to compute $\delta_t(x)$ for a large value of t by using fast matrix multiplication instead of the value iteration algorithm. We will compute the matrix D^t, the t-th power of the weight matrix (using min-plus matrix multiplication). The matrix D^t contains the value of the minimum-weight path of length exactly t for all pairs of vertices. Given D^t, we can determine the value $\delta_t(x)$ for every vertex x by finding the minimum entry in the row of D^t corresponding to x.

Proposition 1. *For every $t \geq 1$ and all vertices x and y we have (i) $d_t(x,y) = D^t[x,y]$ and (ii) $\delta_t(x) = \min_{y\in V} D^t[x,y]$.*

Proof. We give the proof for the sake of completeness. The claim $d_t(x,y) = D^t[x,y]$ follows from a simple induction on t. If $t = 1$, then clearly the minimal-weight path of length 1 from x to y is the edge from x to y if it exists, otherwise $d_t(x,y) = \infty$. If $t \geq 1$, then a minimal-weight path of length t from x to y (if it exists) consists of some outgoing edge of $e = (x,z)$ as its first edge and then a minimal-weight path of length $t-1$ from z to y. We therefore have $d_t(x,y) = \min_{(x,z)\in E} w(x,z) + d_{t-1}(z,y)$. By the definition of the weight matrix and the induction hypothesis we get $d_t(x,y) = \min_{z\in V} D[x,z] + D^{t-1}[z,y]$. Therefore the matrix $D \otimes D^{t-1} = D^t$ contains the value of $d_t(x,y)$ for every pair of vertices x and y.

For the second claim, $\delta_t(x) = \min_{y\in V} D^t[x,y]$, observe that by the definition of $\delta_t(x)$ we obviously have $\delta_t(x) = \min_{y\in V} d_t(x,y)$ because the minimal-weight path of length t starting at x has *some* node y as its end point. ☐

Using this approach, the main question is how fast the matrix D^t can be computed. The most important observation is that D^t (and therefore also $\delta_t(x)$) can be computed by repeated squaring with only $O(\log t)$ min-plus matrix multiplications. This is different from the value iteration algorithm, where t iterations are necessary to compute $\delta_t(x)$.

Proposition 2. *For every $t \geq 1$ we have $D^{2t} = D^t \otimes D^t$. Therefore the matrix D^t can be computed with $O(\log t)$ many min-plus matrix multiplications.*

[3]Specifically, for $t = 4n^3W$ the unique number in $(\delta_t(x)/t - 1/[2n(n-1)], \delta_t(x)/t + 1/[2n(n-1)]) \cap \mathbb{Q}$ that has a denominator of at most n is equal to $\mu(x)$ [31].

Proof. We give the proof for the sake of completeness. It can easily be verified that the min-plus matrix product is associative [1] and therefore $D^{2t} = D^t \otimes D^t$. Therefore, if t is a power of two, we can compute D^t with $\log t$ min-plus matrix multiplications. If t is not a power of two, we can decompose D^t into $D^t = D^{t_1} \otimes \ldots \otimes D^{t_k}$ where each $t_i \leq t$ (for $1 \leq i \leq k$) is a power of two and $k \leq \lceil \log t \rceil$. By storing intermediate results, we can compute D^{2^i} for every $0 \leq i \leq \lceil \log t \rceil$ with $\lceil \log t \rceil$ min-plus matrix multiplications. Using the decomposition above, we have to multiply at most $\lceil \log t \rceil$ such matrices to obtain D^t. Therefore the total number of min-plus matrix multiplications needed for computing D^t is $O(\log t)$. □

The running time of this algorithm depends on the time needed for computing the min-plus product of two integer matrices. This running time will usually depend on the two parameters n and M where n is the size of the $n \times n$ matrices to be multiplied (in our case this is equal to the number of vertices of the graph) and the parameter M denotes the maximum absolute integer entry in the matrices to be multiplied. When we multiply the matrix D by itself to obtain D^2, we have $M = W$, where W is the maximum absolute edge weight. However, M increases with every multiplication and in general, we can bound the maximum absolute integer entry of the matrix D^t only by $M = tW$. Note that $O(n^2)$ operations are necessary to extract the minimum cycle mean $\mu(x)$ for all vertices x from the matrix D^t.

Theorem 3. *If the min-plus product of two $n \times n$ matrices with entries in $\{-M, \ldots, -1, 0, 1, \ldots, M, \infty\}$ can be computed in time $T(n, M)$, then the minimum cycle mean problem can be solved in time $T(n, tW) \log t$ where $t = O(n^3 W)$.*[4]

Unfortunately, the approach outlined above does not immediately improve the running time for the minimum cycle mean problem because min-plus matrix multiplication currently cannot be done fast enough. However, our approach is still useful for solving the minimum cycle mean problem *approximately* because approximate min-plus matrix multiplication can be done faster than its exact counterpart.

4 Approximation algorithm

In this section we design an algorithm that computes an approximation of the minimum cycle mean in graphs with nonnegative integer edge weights. It follows the approach of reducing the minimum cycle mean problem to min-plus matrix multiplication outlined in Section 3. The key to our algorithm is a fast procedure for computing the min-plus product of two integer matrices approximately. We will proceed as follows. First, we explain how to compute an approximation F of D^t, the t-th power of the weight matrix D. From this we easily get, for every vertex x, an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$, the minimum-weight of all paths of length t starting at x. We then argue that for t large enough (in particular $t = O(n^2 W / \varepsilon)$), the value $\delta_t(x)/t$ is an approximation of $\mu(x)$, the minimum cycle mean of cycles reachable from x. By combining both approximations we can show that $\hat{\delta}_t(x)/t$ is an approximation of $\mu(x)$. Thus, the main idea of our algorithm is to compute an approximation of D^t for a large enough t.

4.1 Computing an approximation of D^t

Our first goal is to compute an approximation of the matrix D^t, the t-th power of the weight matrix D, given $t \geq 1$. Zwick provides the following algorithm for approximate min-plus matrix multiplication.

Theorem 4 (Zwick [30]). *Let A and B be two $n \times n$ matrices with integer entries in $[0, M]$ and let $C := A \otimes B$. Let $R \geq \log n$ be a power of two. The algorithm* approx-min-plus(A, B, M, R) *computes the*

[4]Note that necessarily $T(n, M) = \Omega(n^2)$ because the result matrix has n^2 entries that have to be written.

approximate min-plus product $\overline{C}$ *of* A *and* B *in time*[5] $O(n^\omega R \log(M) \log^2(R) \log(n))$ *such that for every* $1 \le i, j \le n$ *it holds that* $C[i,j] \le \overline{C}[i,j] \le (1+4/R)C[i,j]$.

We now give a modification (see Algorithm 1) of Zwick's algorithm for approximate shortest paths [30] such that the algorithm computes a $(1+\varepsilon)$-approximation F of D^t when t is a power of two such that for $1 \le i, j \le n$ we have $D^t[i,j] \le F[i,j] \le (1+\varepsilon)D^t[i,j]$. Just as we can compute D^t exactly with $\log t$ min-plus matrix multiplications, the algorithm computes the $(1+\varepsilon)$-approximation of D^t in $\log t$ iterations. However, in each iteration only an approximate min-plus product is computed. Let F_s be the approximation of $D_s := D^{2^s}$. In the s-th iteration we use approx-min-plus$(F_{s-1}, F_{s-1}, tW, R)$ to calculate F_s with R chosen beforehand such that the desired error bound is reached for $F = F_{\log t}$.

Algorithm 1: Approximation of D^t

```
input  : weight matrix $D$, error bound $\varepsilon$, $t$ (a power of 2)
output : $(1+\varepsilon)$-approximation of $D^t$
$F \leftarrow D$
$r \leftarrow 4\log t/\ln(1+\varepsilon)$
$R \leftarrow 2^{\lceil \log r \rceil}$
for $\log t$ times do
    $F \leftarrow$ approx-min-plus$(F, F, 2tW, R)$
end
return $F$
```

Lemma 5. *Given an* $0 < \varepsilon \le 1$ *and a power of two* $t \ge 1$, *Algorithm 1 computes a* $(1+\varepsilon)$*-approximation* F *of* D^t *in time*

$$O\left(n^\omega \cdot \frac{\log^2(t)}{\varepsilon} \cdot \log(tW) \log^2\left(\frac{\log(t)}{\varepsilon}\right) \log(n)\right) = \widetilde{O}\left(n^\omega \cdot \frac{\log^2(t)}{\varepsilon} \cdot \log(tW)\right)$$

such that $D^t[i,j] \le F[i,j] \le (1+\varepsilon)D^t[i,j]$ *for all* $1 \le i, j \le n$.

Proof. The basic idea is as follows. The running time of approx-min-plus depends linearly on R and logarithmically on M, the maximum entry of the input matrices. Algorithm 1 calls approx-min-plus $\log t$ times. Each call increases the error by a factor of $(1+4/R)$. However, as only $\log t$ approximate matrix multiplications are used, setting R to the smallest power of 2 that is larger than $4\log(t)/\ln(1+\varepsilon)$ suffices to bound the approximation error by $(1+\varepsilon)$. We will show that $2tW$ is an upper bound on the entries in the input matrices for approx-min-plus. The stated running time follows directly from these two facts and Theorem 4.

Let F_s be the approximation of $D_s := D^{2^s}$ computed by the algorithm after iteration s. Recall that 2^sW is an upper bound on the maximum entry in D_s. As we will show, all entries in F_s are at most $(1+\varepsilon)$-times the entries in D_s. Since we assume $\varepsilon \le 1$, we have $1+\varepsilon \le 2$. Thus $2^{s+1}W$ is an upper bound on the entries in F_s. Hence $2tW$ is an upper bound on the entries of F_s with $1 \le s < \log t$, i.e., for all input matrices of approx-min-plus in our algorithm.

[5]The running time of approx-min-plus is given by $O(n^\omega \log M)$ times the time needed to multiply two $O(R\log n)$-bit integers. With the Schönhage-Strassen algorithm for large integer multiplication, two k-bit integers can be multiplied in $O(k \log k \log\log k)$ time, which gives a running time of $O(n^\omega R \log(M) \log(n) \log(R\log n) \log\log(R\log n))$. This can be bounded by the running time given in Theorem 4 if $R \ge \log n$, which will always be the case in the following.

This results in an overall running time of

$$\begin{aligned}
&O\left(n^{\omega} R \log(tW) \log(R) \log\log(R) \log(n) \cdot \log(t)\right)\\
&= O\left(n^{\omega} \cdot \frac{\log^2(t)}{\log(1+\varepsilon)} \cdot \log(tW) \log^2\left(\frac{\log(t)}{\log(1+\varepsilon)}\right) \log(n)\right)\\
&= O\left(n^{\omega} \cdot \frac{\log^2(t)}{\varepsilon} \cdot \log(tW) \log^2\left(\frac{\log(t)}{\varepsilon}\right) \log(n)\right).
\end{aligned}$$

The last equation follows from the inequality $1/\ln(1+\varepsilon) \leq (1+\varepsilon)/\varepsilon$ for $\varepsilon > 0$. Since $\varepsilon \leq 1$ it follows that $1/\log(1+\varepsilon) = O(1/\varepsilon)$.

To show the claimed approximation guarantee, we will prove that the inequality

$$D_s[i,j] \leq F_s[i,j] \leq \left(1+\frac{4}{R}\right)^s D_s[i,j].$$

holds after the s-th iteration of Algorithm 1 by induction on s. Note that the $(1+\varepsilon)$-approximation follows from this inequality because the parameter R is chosen such that after the $(\log t)$-th iteration of the algorithm it holds that

$$\left(1+\frac{4}{R}\right)^{\log t} \leq \left(1+\frac{\ln(1+\varepsilon)}{\log t}\right)^{\log t} \leq e^{\ln(1+\varepsilon)} = 1+\varepsilon.$$

For $s=0$ we have $F_s = D_s$ and the inequality holds trivially. Assume the inequality holds for s. We will show that it also holds for $s+1$.

First we prove the lower bound on $F_{s+1}[i,j]$. Let C_{s+1} be the exact min-plus product of F_s with itself, i.e., $C_{s+1} = F_s \otimes F_s$. Let k_c be the minimizing index such that $C_{s+1}[i,j] = \min_{1\leq k\leq n}(F_s[i,k]+F_s[k,j]) = F_s[i,k_c]+F_s[k_c,j]$. By the definition of the min-plus product

$$D_{s+1}[i,j] = \min_{1\leq k\leq n}\left(D_s[i,k]+D_s[k,j]\right) \leq D_s[i,k_c]+D_s[k_c,j]. \tag{1}$$

By the induction hypothesis and the definition of k_c we have

$$D_s[i,k_c]+D_s[k_c,j] \leq F_s[i,k_c]+F_s[k_c,j] = C_{s+1}[i,j]. \tag{2}$$

By Theorem 4 the values of F_{s+1} can only be larger than the values in C_{s+1}, i.e.,

$$C_{s+1}[i,j] \leq F_{s+1}[i,j]. \tag{3}$$

Combining Equations (1), (2), and (3) yields the claimed lower bound,

$$D_{s+1}[i,j] \leq F_{s+1}[i,j].$$

Next we prove the upper bound on $F_{s+1}[i,j]$. Let k_d be the minimizing index such that $D_{s+1}[i,j] = D_s[i,k_d]+D_s[k_d,j]$. Theorem 4 gives the error from one call of approx-min-plus, i.e., the error in the entries of F_{s+1} compared to the entries of C_{s+1}. We have

$$F_{s+1}[i,j] \leq \left(1+\frac{4}{R}\right) C_{s+1}[i,j]. \tag{4}$$

By the definition of the min-plus product we know that

$$C_{s+1}[i,j] \le F_s[i,k_d] + F_s[k_d,j]\,. \tag{5}$$

By the induction hypothesis and the definition of k_d we can reformulate the error obtained in the first s iterations of Algorithm 1 as follows:

$$\begin{aligned} F_s[i,k_d] + F_s[k_d,j] &\le \left(1+\frac{4}{R}\right)^s D_s[i,k_d] + \left(1+\frac{4}{R}\right)^s D_s[k_d,j]\,, \\ &= \left(1+\frac{4}{R}\right)^s \left(D_s[i,k_d] + D_s[k_d,j]\right)\,, \\ &= \left(1+\frac{4}{R}\right)^s D_{s+1}[i,j]\,. \end{aligned} \tag{6}$$

Combining Equations (4), (5), and (6) yields the upper bound

$$F_{s+1}[i,j] \le \left(1+\frac{4}{R}\right)^{s+1} D_{s+1}[i,j]\,. \qquad \square$$

Once we have computed an approximation of the matrix D^t, we extract from it the minimal entry of each row to obtain an approximation of $\delta_t(x)$. Here we use the equivalence between the minimum entry of row x of D^t and $\delta_t(x)$ established in Proposition 1. Remember that $\delta_t(x)/t$ approaches $\mu(x)$ for t large enough and later on we want to use the approximation of $\delta_t(x)$ to obtain an approximation of the minimum cycle mean $\mu(x)$.

Lemma 6. *The value* $\hat{\delta}_t(x) := \min_{y\in V} F[x,y]$ *approximates* $\delta_t(x)$ *with* $\delta_t(x) \le \hat{\delta}_t(x) \le (1+\varepsilon)\delta_t(x)$.

Proof. Let y_f and y_d be the indices where the x-th rows of F and D^t obtain their minimal values, respectively, i.e.,

$$y_f := \arg\min_{y\in V} F[x,y] \quad \text{and} \quad y_d := \arg\min_{y\in V} D^t[x,y]\,.$$

By these definitions and Lemma 5 we have

$$\delta_t(x) = D^t[x,y_d] \le D^t[x,y_f] \le F[x,y_f] = \hat{\delta}_t(x)$$

and

$$\hat{\delta}_t(x) = F[x,y_f] \le F[x,y_d] \le (1+\varepsilon)D^t[x,y_d]\,. \qquad \square$$

4.2 Approximating the minimum cycle mean

We now add the next building block to our algorithm. So far, we can obtain an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ for any t that is a power of two. We now show that $\delta_t(x)/t$ is itself an approximation of the minimum cycle mean $\mu(x)$ for t large enough. Then we argue that $\hat{\delta}_t(x)/t$ approximates the minimum cycle mean $\mu(x)$ for t large enough. This value of t bounds the number of iterations of our algorithm. A similar technique was also used in [31] to bound the number of iterations of the value iteration algorithm for the two-player mean-payoff game.

We start by showing that $\delta_t(x)/t$ differs from $\mu(x)$ by at most nW/t *for any* t. Then we will turn this additive error into a multiplicative error by choosing a large enough value of t. A multiplicative error implies that we have to compute the solution exactly for $\mu(x) = 0$. We will use a separate procedure to identify all vertices x with $\mu(x) = 0$ and compute the approximation only for the remaining vertices. Note that $\mu(x) > 0$ implies $\mu(x) \ge 1/n$ because all edge weights are integers.

Lemma 7. *For every $x \in V$ and every integer $t \geq 1$ it holds that*

$$t \cdot \mu(x) - nW \leq \delta_t(x) \leq t \cdot \mu(x) + nW\,.$$

Proof. We first show the lower bound on $\delta_t(x)$. Let P be a path of length t starting at x with weight $\delta_t(x)$. Consider the cycles in P and let E' be the multiset of the edges in P that are in a cycle of P. There can be at most n edges that are not in a cycle of P, thus there are at least $\max(t-n, 0)$ edges in E'. Since $\mu(x)$ is the minimum mean weight of any cycle reachable from x, the sum of the weight of the edges in E' can be bounded below by $\mu(x)$ times the number of edges in E'. Furthermore, the value of $\mu(x)$ can be at most W. As we only allow nonnegative edge weights, the sum of the weights of the edges in E' is a lower bound on $\delta_t(x)$. Thus we have

$$\delta_t(x) \geq \sum_{e \in E'} w(e) \geq (t-n)\mu(x) \geq t \cdot \mu(x) - n \cdot \mu(x) \geq t \cdot \mu(x) - nW\,.$$

Next we prove the upper bound on $\delta_t(x)$. Let l be the length of the shortest path from x to a vertex y in a minimum mean-weight cycle C reachable from x (such that only y is both in the shortest path and in C). Let c be the length of C. Let the path Q be a path of length t that consists of the shortest path from x to y, $\lfloor (t-l)/c \rfloor$ rounds on C, and $t - l - c\lfloor (t-l)/c \rfloor$ additional edges in C. By the definition of $\delta_t(x)$, we have $\delta_t(x) \leq w(Q)$. The sum of the length of the shortest path from x to y and the number of the remaining edges of Q not in a complete round on C can be at most n because in a graph with nonnegative weights no shortest path has a cycle and no vertices in C except y are contained in the shortest path from x to y. Each of these edges has a weight of at most W. The mean weight of C is $\mu(x)$, thus the sum of the weight of the edges in all complete rounds on C is $\mu(x) \cdot c\lfloor (t-l)/c \rfloor \leq \mu(x) \cdot t$. Hence we have

$$\delta_t(x) \leq w(Q) \leq t \cdot \mu(x) + nW\,. \qquad \square$$

In the next step we show that we can use the fact that $\delta_t(x)/t$ is an approximation of $\mu(x)$ to obtain a $(1+\varepsilon)$-approximation $\hat{\mu}(x)$ of $\mu(x)$ even if we only have an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ with $(1+\varepsilon)$-error. We exclude the case $\mu(x) = 0$ for the moment.

Lemma 8. *Assume we have an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ such that $\delta_t(x) \leq \hat{\delta}_t(x) \leq (1+\varepsilon)\delta_t(x)$ for $0 < \varepsilon \leq 1/2$. If*

$$t \geq \frac{n^2 W}{\varepsilon}, \quad \mu(x) \geq \frac{1}{n}, \quad \textit{and} \quad \hat{\mu}(x) := \frac{\hat{\delta}_t(x)}{(1-\varepsilon)t}\,,$$

then

$$\mu(x) \leq \hat{\mu}(x) \leq (1+7\varepsilon)\mu(x)\,.$$

Proof. We first show that $\hat{\mu}(x)$ is at least as large as $\mu(x)$. From Lemma 7 we have $\delta_t(x) \geq t \cdot \mu(x) - nW$. As t is chosen large enough,

$$\frac{\delta_t(x)}{t} \geq \mu(x) - \frac{nW}{t} \geq \mu(x) - \frac{\varepsilon}{n} \geq \mu(x) - \varepsilon\mu(x) \geq (1-\varepsilon)\mu(x)\,.$$

Thus, by the assumption $\delta_t(x) \leq \hat{\delta}_t(x)$ we have

$$\mu(x) \leq \frac{\hat{\delta}_t(x)}{(1-\varepsilon)t} = \hat{\mu}(x)\,.$$

For the upper bound on $\hat{\mu}(x)$ we use the inequality $\delta_t(x) \leq t \cdot \mu(x) + nW$ from Lemma 7. As t is chosen large enough,

$$\frac{\delta_t(x)}{t} \leq \mu(x) + \frac{nW}{t} \leq \mu(x) + \frac{\varepsilon}{n} \leq (1+\varepsilon)\mu(x)\,.$$

With $\hat{\delta}_t(x) \leq (1+\varepsilon)\delta_t(x)$ this gives

$$\hat{\mu}(x) = \frac{\hat{\delta}_t(x)}{(1-\varepsilon)t} \leq \frac{(1+\varepsilon)^2}{(1-\varepsilon)}\mu(x)\,.$$

It can be verified by simple arithmetic that for $\varepsilon > 0$ the inequality $\varepsilon \leq 1/2$ is equivalent to

$$\frac{(1+\varepsilon)^2}{(1-\varepsilon)} \leq (1+7\varepsilon)\,. \qquad \square$$

As a last ingredient to our approximation algorithm, we design a procedure that deals with the special case that the minimum cycle mean is 0. Since our goal is an algorithm with multiplicative error, we have to be able to compute the solution exactly in that case. This can be done in linear time because the edge-weights are nonnegative.

Proposition 9. *Given a graph with nonnegative integer edge weights, we can find out all vertices x such that $\mu(x) = 0$ in time $O(m)$.*

Proof. Note that in the case of nonnegative edge weights we have $\mu(x) \geq 0$. Furthermore, a cycle can only have mean weight 0 if all edges on this cycle have weight 0. Thus, it will be sufficient to detect cycles in the graph that only contain edges that have weight 0.

We proceed as follows. First, we compute the strongly connected components of G, the original graph. Each strongly connected component G_i (where $1 \leq i \leq k$) is a subgraph of G with a set of vertices V_i and a set of edges E_i. For every $1 \leq i \leq k$ we let $G_i^0 = (E_i^0, V_i)$ denote the subgraph of G_i that only contains edges of weight 0, i.e., $E_i^0 = \{e \in E_i | w(e) = 0\}$. As argued above, G_i contains a zero-mean cycle if and only if G_i^0 contains a cycle. We can check whether G_i^0 contains a cycle by computing the strongly connected components of G_i^0: G_i^0 contains a cycle if and only if it has a strongly connected component of size at least 2 (we can assume w.l.o.g. that there are no self-loops). Let Z be the set of all vertices in strongly connected components of G that contain a zero-mean cycle. The vertices in Z are not the only vertices that can reach a zero-mean cycle. We can identify all vertices that can reach a zero-mean cycle by performing a linear-time graph traversal to identify all vertices that can reach Z.

Since all steps take linear time, the total running time of this algorithm is $O(m)$. $\square$

Finally, we wrap up all arguments to obtain our algorithm for approximating the minimum cycle mean. This algorithms performs $\log t$ approximate min-plus matrix multiplications to compute an approximation of D^t and $\delta_t(x)$. Lemma 8 tells us that $t = n^2 W/\varepsilon$ is just the right number to guarantee that our approximation of $\delta_t(x)$ can be used to obtain an approximation of $\mu(x)$. The value of t is relatively large but the running time of our algorithm depends on t only in a logarithmic way.

Theorem 10. *Given a graph with nonnegative integer edge weights, we can compute an approximation $\hat{\mu}(x)$ of the minimum cycle mean for every vertex x such that $\mu(x) \leq \hat{\mu}(x) \leq (1+\varepsilon)\mu(x)$ for $0 < \varepsilon \leq 1$ in time*

$$O\left(\frac{n^\omega}{\varepsilon}\log^3\left(\frac{nW}{\varepsilon}\right)\log^2\left(\frac{\log\left(\frac{nW}{\varepsilon}\right)}{\varepsilon}\right)\log(n)\right) = \widetilde{O}\left(\frac{n^\omega}{\varepsilon}\log^3\left(\frac{nW}{\varepsilon}\right)\right)\,.$$

Proof. First we find all vertices x with $\mu(x) = 0$. By Proposition 9 this takes time $O(n^2)$ for $m = O(n^2)$. For the remaining vertices x we approximate $\mu(x)$ as follows.

Let $\varepsilon' := \varepsilon/7$. If we execute Algorithm 1 with weight matrix D, error bound ε' and t such that t is the smallest power of two with $t \geq n^2W/\varepsilon'$, we obtain a $(1+\varepsilon')$-approximation $F[x,y]$ of $D^t[x,y]$ for all vertices x and y (Lemma 5). By calculating for every x the minimum entry of $F[x,y]$ over all y we have a $(1+\varepsilon')$-approximation of $\delta_t(x)$ (Lemma 6). By Lemma 8 $\hat{\mu}(x) := \hat{\delta}_t(x)/((1-\varepsilon')t)$ is for this choice of t an approximation of $\mu(x)$ such that $\mu(x) \leq \hat{\mu}(x) \leq (1+7\varepsilon')\mu(x)$. By substituting ε' with $\varepsilon/7$ we get $\mu(x) \leq \hat{\mu}(x) \leq (1+\varepsilon)\mu(x)$ i.e., a $(1+\varepsilon)$-approximation of $\mu(x)$.

By Lemma 5 the running time of Algorithm 1 for $t = 2^{\lceil \log(n^2W/\varepsilon') \rceil} = O(n^2W/\varepsilon)$ is

$$O\left(\frac{n^\omega}{\varepsilon}\log^2\left(\frac{n^2W}{\varepsilon}\right)\log\left(\frac{n^2W^2}{\varepsilon}\right)\log^2\left(\frac{\log\left(\frac{n^2W}{\varepsilon}\right)}{\varepsilon}\right)\log(n)\right).$$

With $\log(n^2W) \leq \log((nW)^2) = O(\log(nW))$ we get that Algorithm 1 runs in time

$$O\left(\frac{n^\omega}{\varepsilon}\log^3\left(\frac{nW}{\varepsilon}\right)\log^2\left(\frac{\log\left(\frac{nW}{\varepsilon}\right)}{\varepsilon}\right)\log(n)\right). \tag{7}$$

□

5 Open problems

We hope that this work draws attention to the problem of approximating the minimum cycle mean. It would be interesting to study whether there is a faster approximation algorithm for the minimum cycle mean problem, maybe at the cost of a worse approximation. The running time of our algorithm immediately improves if faster algorithms for classic matrix multiplication, min-plus matrix multiplication or approximate min-plus multiplication are found. However, a different approach might lead to better results and might shed new light on how well the problem can be approximated. Therefore it would be interesting to remove the dependence on fast matrix multiplication and develop a so-called combinatorial algorithm.

Another obvious extension is to allow negative edge weights in the input graph. Furthermore, we only consider the minimum cycle mean problem, while it might be interesting to actually output a cycle with approximately optimal mean weight.

References

[1] Alfred V. Aho, John E. Hopcroft & Jeffrey D. Ullman (1974): *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

[2] Ravindra K. Ahuja, Thomas L. Magnanti & James B. Orlin (1993): *Network flows: theory, algorithms, and applications*. Prentice Hall.

[3] Noga Alon, Zvi Galil & Oded Margalit (1997): *On the Exponent of the All Pairs Shortest Path Problem*. *Journal of Computer and System Sciences* 54(2), pp. 255–262, doi:10.1006/jcss.1997.1388. Announced at FOCS '91.

[4] Roderick Bloem, Karin Greimel, Thomas A. Henzinger & Barbara Jobstmann (2009): *Synthesizing robust systems*. In: *FMCAD*, pp. 85–92, doi:10.1109/FMCAD.2009.5351139.

[5] Endre Boros, Khaled Elbassioni, Mahmoud Fouz, Vladimir Gurvich, Kazuhisa Makino & Bodo Manthey (2011): *Stochastic Mean Payoff Games: Smoothed Analysis and Approximation Schemes*. In: *ICALP*, pp. 147–158, doi:10.1007/978-3-642-22006-7_13.

[6] Peter Butkovic & Raymond A. Cuninghame-Green (1992): *An $O(n^2)$ algorithm for the maximum cycle mean of an $n \times n$ bivalent matrix*. *Discrete Applied Mathematics* 35(2), pp. 157–162, doi:`10.1016/0166-218X(92)90039-D`.

[7] Timothy M. Chan (2010): *More Algorithms for All-Pairs Shortest Paths in Weighted Graphs*. *SIAM Journal on Computing* 39(5), pp. 2075–2089, doi:`10.1137/08071990X`. Announced at STOC ’07.

[8] Ali Dasdan & Rajesh K. Gupta (1998): *Faster Maximum and Minimum Mean Cycle Algorithms for System-Performance Analysis*. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17(10), pp. 889–899, doi:`10.1109/43.728912`.

[9] Manfred Droste & Ingmar Meinecke (2010): *Describing Average- and Longtime-Behavior by Weighted MSO Logics*. In: *MFCS*, pp. 537–548, doi:`10.1007/978-3-642-15155-2_47`.

[10] Andrzej Ehrenfeucht & Jan Mycielski (1979): *Positional strategies for mean payoff games*. *International Journal of Game Theory* 8(2), pp. 109–113, doi:`10.1007/BF01768705`.

[11] Michael L. Fredman (1976): *New Bounds on the Complexity of the Shortest Path Problem*. *SIAM Journal on Computing* 5(1), pp. 83–89, doi:`10.1137/0205006`.

[12] Andrew V. Goldberg (1995): *Scaling Algorithms for the Shortest Paths Problem*. *SIAM Journal on Computing* 24(3), pp. 494–504, doi:`10.1137/S0097539792231179`.

[13] V.A. Gurvich, A.V. Karzanov & L.G. Khachiyan (1988): *Cyclic games and an algorithm to find minimax cycle means in directed graphs*. *USSR Computational Mathematics and Mathematical Physics* 28(5), pp. 85–91, doi:`10.1016/0041-5553(88)90012-2`.

[14] Thomas Dueholm Hansen & Uri Zwick (2010): *Lower Bounds for Howard’s Algorithm for Finding Minimum Mean-Cost Cycles*. In: *ISAAC*, pp. 415–426, doi:`10.1007/978-3-642-17517-6_37`.

[15] Mark Hartmann & James B. Orlin (1993): *Finding Minimum Cost to Time Ratio Cycles With Small Integral Transit Times*. *Networks* 23(6), pp. 567–574, doi:`10.1002/net.3230230607`.

[16] Ronald A. Howard (1960): *Dynamic Programming and Markov Processes*. MIT Press.

[17] Richard M. Karp (1978): *A characterization of the minimum cycle mean in a digraph*. *Discrete Mathematics* 23(3), pp. 309–311, doi:`10.1016/0012-365X(78)90011-0`.

[18] Richard M. Karp & James B. Orlin (1981): *Parametric shortest path algorithms with an application to cyclic staffing*. *Discrete Applied Mathematics* 3(1), pp. 37–45, doi:`10.1016/0166-218X(81)90026-3`.

[19] Eugène L. Lawler (1976): *Combinatorial optimization: Networks and Matroids*. Dover Publications.

[20] Omid Madani (2002): *Polynomial Value Iteration Algorithms for Deterministic MDPs*. In: *UAI*, pp. 311–318. Available at `http://dl.acm.org/citation.cfm?id=2073913`.

[21] S. Thomas McCormick (1993): *Approximate Binary Search Algorithms for Mean Cuts and Cycles*. *Operations Research Letters* 14(3), pp. 129–132, doi:`10.1016/0167-6377(93)90022-9`.

[22] James B. Orlin & Ravindra K. Ahuja (1992): *New scaling algorithms for the assignment and minimum mean cycle problems*. *Mathematical Programming* 54(1-3), pp. 41–56, doi:`10.1007/BF01586040`.

[23] Aaron Roth, Maria-Florina Balcan, Adam Kalai & Yishay Mansour (2010): *On the Equilibria of Alternating Move Games*. In: *SODA*, pp. 805–816. Available at `http://dl.acm.org/citation.cfm?id=1873667`.

[24] Piotr Sankowski (2005): *Shortest Paths in Matrix Multiplication Time*. In: *ESA*, pp. 770–778, doi:`10.1007/11561071_68`.

[25] Virginia Vassilevska Williams (2012): *Multiplying Matrices Faster Than Coppersmith-Winograd*. In: *STOC*, pp. 887–898, doi:`10.1145/2213977.2214056`.

[26] Virginia Vassilevska Williams & Ryan Williams (2010): *Subcubic Equivalences between Path, Matrix and Triangle Problems*. In: *FOCS*, pp. 645–654, doi:`10.1109/FOCS.2010.67`.

[27] Neal E. Young, Robert Endre Tarjan & James B. Orlin (1991): *Faster Parametric Shortest Path and Minimum-Balance algorithms*. *Networks* 21(2), pp. 205–221, doi:`10.1002/net.3230210206`.

[28] Gideon Yuval (1976): *An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications*. *Information Processing Letters* 4(6), pp. 155–156, doi:10.1016/0020-0190(76)90085-5.

[29] Eitan Zemel (1987): *A Linear Time Randomizing Algorithm for Searching Ranked Functions*. *Algorithmica* 2(1-4), pp. 81–90, doi:10.1007/BF01840350.

[30] Uri Zwick (2002): *All Pairs Shortest Paths using Bridging Sets and Rectangular Matrix Multiplication*. *Journal of the ACM* 49(3), pp. 289–317, doi:10.1145/567112.567114. Announced at FOCS '98.

[31] Uri Zwick & Mike Paterson (1996): *The complexity of mean payoff games on graphs*. *Theoretical Computer Science* 158(1-2), pp. 343–359, doi:10.1016/0304-3975(95)00188-3. Announced at COCOON '95.

Probabilistic data flow analysis: a linear equational approach

Alessandra Di Pierro
University of Verona
Verona, Italy
alessandra.dipierro@univr.it

Herbert Wiklicky
Imperial College London
London, UK
herbert@doc.ic.ac.uk

Speculative optimisation relies on the estimation of the probabilities that certain properties of the control flow are fulfilled. Concrete or estimated branch probabilities can be used for searching and constructing advantageous speculative and bookkeeping transformations. We present a probabilistic extension of the classical equational approach to data-flow analysis that can be used to this purpose. More precisely, we show how the probabilistic information introduced in a control flow graph by branch prediction can be used to extract a system of linear equations from a program and present a method for calculating correct (numerical) solutions.

1 Introduction

In the last two decades probabilistic aspects of software have become a particularly popular subject of research. The reason for this is arguably in *economical* and *resource conscious* questions involving modern computer systems. While program verification and analysis originally focused on qualitative issues, e.g. whether code is correct or if compiler optimisations are valid, the focus is now more often also on the costs of operations.

Speculative optimisation is part of this trend; it plays an important role in the design of modern compiler and run time architectures. A speculative approach has been adopted in various models where cost optimisation claims for a more optimistic interpretation of the results of a program analysis. It is in fact often the case that possible optimisations are discarded because the analysis cannot guarantee their correctness. The alternative to this sometimes overly pessimistic analysis is to speculatively assume in those cases that optimisations are correct and then eventually backtrack and redo the computation if at a later check the assumption turns out to be incorrect.

Speculative optimisation relies on the optimal estimation of the probabilities that certain properties of the control flow are fulfilled. This is different from the classical (pessimistic) thinking where one aims in providing bounds for what can happen during execution [8].

A number of frameworks and tools to analyse systems's probabilistic aspects have been developed, which can be seen as probabilistic versions of classical techniques such as model checking and abstract interpretation. To provide a basis for such analysis various semantical model involving discrete and continuous time and also non-deterministic aspects have been developed (e.g. DTMCs, CTMCs, MDPs, process algebraic approaches etc.). There also exist some powerful tools which implement these methods, e.g. PRISM [14], just to name one.

Our own contribution in this area has been a probabilistic version of the abstract interpretation framework [6], called Probabilistic Abstract Interpretation (PAI) [12, 9]. This analysis framework, in its basic form, is concerned with purely probabilistic, discrete time models. Its purpose is to give optimal estimates of the probability that a certain property holds rather than providing probabilities bounds. As such, we think it is well suited as a base for speculative optimisation.

S	`::=`	`skip`	S	`::=`	$[$`skip`$]^\ell$
	$\mid$	x `:=` $e(x_1,\ldots,x_n)$		$\mid$	$[x$ `:=` $e(x_1,\ldots,x_n)]^\ell$
	$\mid$	x `?=` ρ		$\mid$	$[x$ `?=` $\rho]^\ell$
	$\mid$	S_1 ; S_2		$\mid$	S_1 ; S_2
	$\mid$	`if` b `then` S_1 `else` S_2 `fi`		$\mid$	`if` $[b]^\ell$ `then` S_1 `else` S_2 `fi`
	$\mid$	`while` b `do` S `od`		$\mid$	`while` $[b]^\ell$ `do` S `od`

Table 1: The syntax

The aim of this paper is to provide a framework for a probabilistic analysis of programs in the style of a classical data flow approach [18, 1]. In particular, we are interested in a formal basis for (non-static) branch prediction. The analysis technique we present consists of three phases: (i) abstract branch prediction, (ii) specification of the actual data-flow equations based on the estimates of the branch probabilities, and (iii) finding solutions. We will use vector space structures to specify the properties and analysis of a program. This allows for the construction of solutions via numerical (linear algebraic) methods as opposed to the lattice-theoretic fixed-point construction of the classical analysis.

2 A Probabilistic Language

2.1 Syntax and Operational Semantics

We use as a reference language a simple imperative language whose syntax is given in Table 1. Following the approach in [18] we extend this syntax with unique program labels $\ell \in \mathbf{Lab}$ in order to be able to refer to certain program points during the analysis.

The dummy statement `skip` has no computational effect. For the arithmetic *expressions* $e(x_1,\ldots,x_n)$ on the right hand side (RHS) of the assignment as well as for the tests $b = b(x_1,\ldots,x_n)$ in `if` and `while` statements, we leave the details of the syntax open as they are irrelevant for our treatment. The RHS of a random assignment x `?=` ρ is a distribution ρ over some set of values with the meaning that x is assigned one of the possible constant values c with probability $\rho(c)$.

An operational semantics in the SOS style is given in Table 2.2. This defines a probabilistic transition relation on configurations in $\mathbf{Conf} = \mathbf{Stmt} \times \mathbf{State}$ with $\mathbf{Stmt}$ the set of all statements in our language together with `stop` which indicates termination and $\mathbf{State} = \mathbf{Var} \to \mathbf{Value}$. The details of the semantics of arithmetic and boolean expressions $[\![a]\!] = \mathscr{E}(a)$ and $[\![b]\!] = \mathscr{E}(b)$ respectively are again left open in our treatment here and can be found in [10].

2.2 Computational States

In any concrete computation or execution – even when it is involving probabilistic elements – the computational situation is uniquely defined by a mapping $s : \mathbf{Var} \to \mathbf{Value}$ to which we refer to as a *classical state*. Every variable in $\mathbf{Var}$ has a unique value in $\mathbf{Value}$ possibly including $\perp \in \mathbf{Value}$ to indicate undefinedness. We denote by $\mathbf{State}$ the set of all classical states.

In order to keep the mathematical treatment simple we will assume here that every variable can take values in a finite set $\mathbf{Value}$. These sets can be nevertheless quite large and cover, for example, all finitely representable integers on a given machine.

R0	$\langle \texttt{stop}, s\rangle \Rightarrow_1 \langle \texttt{stop}, s\rangle$	**R4**$_1$	$\dfrac{\langle S_1, s\rangle \Rightarrow_p \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow_p \langle S_1'; S_2, s'\rangle}$
R1	$\langle \texttt{skip}, s\rangle \Rightarrow_1 \langle \texttt{stop}, s\rangle$		
R2	$\langle v \texttt{ := } e, s\rangle \Rightarrow_1 \langle \texttt{stop}, s[v \mapsto \mathscr{E}(e)s]\rangle$	**R4**$_2$	$\dfrac{\langle S_1, s\rangle \Rightarrow_p \langle \texttt{stop}, s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow_p \langle S_2, s'\rangle}$
R3	$\langle v \texttt{ ?= } \rho, s\rangle \Rightarrow_{\rho(r)} \langle \texttt{stop}, s[v \mapsto r]\rangle$		
R5$_1$	$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}, s\rangle \Rightarrow_1 \langle S_1, s\rangle$		if $\mathscr{E}(b)s = \textbf{true}$
R5$_2$	$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}, s\rangle \Rightarrow_1 \langle S_2, s\rangle$		if $\mathscr{E}(b)s = \textbf{false}$
R6$_1$	$\langle \texttt{while } b \texttt{ do } S \texttt{ od}, s\rangle \Rightarrow_1 \langle S; \texttt{ while } b \texttt{ do } S \texttt{ od}, s\rangle$		if $\mathscr{E}(b)s = \textbf{true}$
R6$_2$	$\langle \texttt{while } b \texttt{ do } S \texttt{ od}, s\rangle \Rightarrow_1 \langle \texttt{stop}, s\rangle$		if $\mathscr{E}(b)s = \textbf{false}$

Table 2: The rules of the SOS semantics

For a finite set X we denote by $\mathscr{P}(X)$ the power-set of X and by $\mathscr{V}(X)$ the free vector space over X, i.e. the set of formal linear combinations of elements in X. We represent vectors via their coordinates $(x_1, \ldots, x_n)$ as rows, i.e. elements in $\mathbb{R}^{|X|}$ with $|X|$ denoting the cardinality of X and use post-multiplication with matrices representing linear maps, i.e. $\mathbf{A}(x) = x \cdot \mathbf{A}$. The set $\mathbf{Dist}(X)$ of distributions on X – i.e. $\rho : X \to [0,1]$ and $\sum_i \rho(x_i) = 1$ – clearly correspond to a sub-set of $\mathscr{V}(X)$. We will also use a tuple notation for distributions: $\rho = \{\langle a, \frac{1}{2}\rangle, \langle b, \frac{1}{4}\rangle, \langle c, \frac{1}{4}\rangle\}$ will denote a distribution where a has probability $\rho(a) = \frac{1}{2}$ and b and c both have probability $\frac{1}{4}$. For uniform distributions we will simply specify the underlying set, e.g. $\{a, b, c\}$ instead of $\langle a, \frac{1}{3}\rangle, \langle b, \frac{1}{3}\rangle, \langle c, \frac{1}{3}\rangle\}$.

The tensor product is an essential element of the description of probabilistic states. The tensor product[1] of two vectors $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_m)$ is given by $(x_1y_1, \ldots, x_1y_m, \ldots, x_ny_1, \ldots, x_ny_m)$ an nm dimensional vector. Similarily for matrices. The tensor product of two vector spaces $\mathscr{V} \otimes \mathscr{W}$ can be defined as the formal linear combinations of the tensor products $v_i \otimes w_j$ with v_i and w_j base vectors in $\mathscr{V}$ and $\mathscr{W}$, respectively. For further details we refer e.g to [19, Chap. 14].

Importantly, the isomorphism $\mathscr{V}(X \times Y) = \mathscr{V}(X) \otimes \mathscr{V}(Y)$ allows us to identify set of all distributions on the cartesian product of two sets with the tensor product of the spaces of distributions on X and Y.

We define a *probabilistic state* σ as any probability distribution over classical states, i.e. $\sigma \in \mathbf{Dist}(\mathbf{State})$. This can also be seen as $\sigma \in \mathscr{V}(\mathbf{State}) = \mathscr{V}(\mathbf{Var} \to \mathbf{Value}) = \mathscr{V}(\mathbf{Value}^{|\mathbf{Var}|}) = \mathscr{V}(\mathbf{Value})^{\otimes v}$ the v-vold tensor product of $\mathscr{V}(\mathbf{Value})$ with $v = |\mathbf{Var}|$.

In our setting we represent (semantical) functions and predicates or tests as linear operators on the probabilistic state space, i.e. as matrices. For any function $f : X \mapsto Y$ we define a linear representation $|X| \times |Y|$ matrix by:

$$(\mathbf{F}_f)_{ij} = (\mathbf{F}(f))_{ij} = (\mathbf{F})_{ij} = \begin{cases} 1 & \text{if } f(x_i) = y_j \\ 0 & \text{otherwise.} \end{cases}$$

where we assume some fixed enumeration on both X and Y. For an equivalence relation on X we can also represent the function which maps every element in X to its equivalence class $c : x \mapsto [x]$ in this way. Such a *classification matrix* contains in every row exactly one non-zero entry 1. Classification matrices (modulo reordering of indices) are in a one-to-one correspondence with the equivalence relations on a set X and we will use them to define probabilistic abstractions for our analysis (cf. Section 4.2). A predicate $p : X \to \{\textbf{true}, \textbf{false}\}$ is represented by a diagonal $|X| \times |X|$ matrix:

$$(\mathbf{P}_p)_{ij} = (\mathbf{P}(p))_{ij} = (\mathbf{P})_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } p(x_i) = \textbf{true} \\ 0 & \text{otherwise.} \end{cases}$$

[1]More precisely, the Kronecker product – the coordinate based version of the abstract concept of a tensor product.

2.3 Probabilistic Abstraction

The analysis technique we present in this paper will make use of a particular notion of abstraction of the state space (given as a vector space) which is formalised in terms of Moore-Penrose pseudo-inverse [19].

Definition 1 *Let $\mathscr{C}$ and $\mathscr{D}$ be two finite dimensional vector spaces, and let $\mathbf{A} : \mathscr{C} \to \mathscr{D}$ be a linear map between them. The linear map $\mathbf{A}^\dagger = \mathbf{G} : \mathscr{D} \to \mathscr{C}$ is the* Moore-Penrose pseudo-inverse *of* $\mathbf{A}$ *iff*

$$\mathbf{A} \circ \mathbf{G} = \mathbf{P}_A \text{ and } \mathbf{G} \circ \mathbf{A} = \mathbf{P}_G$$

where $\mathbf{P}_A$ and $\mathbf{P}_G$ denote orthogonal projections onto the ranges of $\mathbf{A}$ *and* $\mathbf{G}$.

An operator or matrix is an *orthogonal projection* if $\mathbf{P}^* = \mathbf{P}^2 = \mathbf{P}$ where $.^*$ denotes the *adjoint* which for real matrices correspond simply to the transpose matrix $\mathbf{P}^* = \mathbf{P}^t$ [19, Ch 10].

For invertible matrices the Moore-Penrose pseudo-inverse is the same as the inverse. A special example is the *forgetful abstraction* $\mathbf{A}_f$ which corresponds to a map $f : X \to \{*\}$ which maps all elements of X onto a single abstract one. It is represented by a $|X| \times 1$ matrix containing only 1, and its Moore-Penrose pseudo-inverse is given by $1 \times |X|$ matrix with all entries $\frac{1}{|X|}$.

The Moore-Penrose pseudo-inverse allows us to construct the closest, in a least square sense (see for example [5, 3]), approximation $\mathbf{F}^\# : \mathscr{D} \to \mathscr{D}$ of a concrete linear operator $\mathbf{F} : \mathscr{C} \to \mathscr{C}$ for a given abstraction $\mathbf{A} : \mathscr{C} \to \mathscr{D}$ as

$$\mathbf{F}^\# = \mathbf{A}^\dagger \cdot \mathbf{F} \cdot \mathbf{A} = \mathbf{G} \cdot \mathbf{F} \cdot \mathbf{A} = \mathbf{A} \circ \mathbf{F} \circ \mathbf{G}.$$

This notion of probabilistic abstraction is central in the Probabilistic Abstract Interpretation (PAI) framework. For further details we refer to e.g. [10]. As we will use this notion later for abstracting branching probabilities, it is important here to point out the guarantees that such abstractions are able to provide. In fact, these are not related to any correctness notion in the classical sense. The theory of the least-square approximation [7, 3] tells us that if $\mathscr{C}$ and $\mathscr{D}$ be two finite dimensional vector spaces, $\mathbf{A} : \mathscr{C} \mapsto \mathscr{D}$ a linear map between them, and $\mathbf{A}^\dagger = \mathbf{G} : \mathscr{D} \mapsto \mathscr{C}$ its Moore-Penrose pseudo-inverse, then the vector $x_0 = y \cdot \mathbf{G}$ is the one minimising the distance between $x \cdot \mathbf{A}$, for any vector x in $\mathscr{C}$, and y, i.e.

$$\inf_{x \in \mathscr{C}} \|x \cdot \mathbf{A} - y\| = \|x_0 \cdot \mathbf{A} - y\|.$$

This guarantees that our probabilistic abstractions correspond to the **closest** approximations in a metric sense of the concrete situations, as they are constructed using the Moore-Penrose pseudo-inverse.

3 Data-Flow Analysis

Data-flow analysis is based on a statically determined flow relation. This is defined in terms of two auxiliary operations, namely $\mathit{init} : \mathbf{Stmt} \to \mathbf{Lab}$ and $\mathit{final} : \mathbf{Stmt} \to \mathscr{P}(\mathbf{Lab})$, defined as follows:

$$\begin{array}{ll}
\mathit{init}([\texttt{skip}]^\ell) = \ell & \mathit{final}([\texttt{skip}]^\ell) = \{\ell\} \\
\mathit{init}([v := e]^\ell) = \ell & \mathit{final}([v := e]^\ell) = \{\ell\} \\
\mathit{init}([v \mathrel{?}= e]^\ell) = \ell & \mathit{final}([v \mathrel{?}= e]^\ell) = \{\ell\} \\
\mathit{init}(S_1;S_2) = \mathit{init}(S_1) & \mathit{final}(S_1;S_2) = \mathit{final}(S_2) \\
\mathit{init}(\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}) = \ell & \mathit{final}(\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}) = \mathit{final}(S_1) \cup \mathit{final}(S_2) \\
\mathit{init}(\texttt{while } [b]^\ell \texttt{ do } S \texttt{ od}) = \ell & \mathit{final}(\texttt{while } [b]^\ell \texttt{ do } S \texttt{ od}) = \{\ell\}.
\end{array}$$

The control flow $\mathscr{F}(S)$ in $S \in \mathbf{Stmt}$ is defined via the function $\mathit{flow} : \mathbf{Stmt} \to \mathscr{P}(\mathbf{Lab} \times \mathbf{Lab})$:

$$\begin{aligned}
\mathit{flow}([\texttt{skip}]^\ell) = \mathit{flow}([v \texttt{ := } e]^\ell) = \mathit{flow}([v \texttt{ ?= } e]^\ell) = \emptyset \\
\mathit{flow}(S_1;S_2) = \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \cup \{(\ell, \mathit{init}(S_2)) \mid \ell \in \mathit{final}(S_1)\} \\
\mathit{flow}(\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}) = \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \cup \{(\ell, \underline{\mathit{init}(S_1)}), (\ell, \mathit{init}(S_2))\} \\
\mathit{flow}(\texttt{while } [b]^\ell \texttt{ do } S \texttt{ od}) = \mathit{flow}(S) \cup \{(\ell, \underline{\mathit{init}(S)})\} \cup \{(\ell', \ell) \mid \ell' \in \mathit{final}(S)\}
\end{aligned}$$

The definition of flow only records that a certain control flow step is possible. For tests b in conditionals and loops we indicate the branch corresponding to the case when the test is successful by underlining it. We identify a statement S with the block $[S]^\ell$ that contains it and with the (unique) label ℓ associated to the block. We will denote by $\mathbf{Block} = \mathbf{Block}(P)$ the set of all the blocks occurring in P, and use indistinctly **Block** and **Lab** to refer to blocks.

3.1 Monotone Framework

The classical data-flow analysis is made up of two components: a "local" part which describes how the information representing the analysis changes when execution passes through a given block/label, and a "global" collection part which describes how information is accumulated when a number of different control flow paths (executions) come together.

This is formalised in a general scheme, called Monotone Framework in [18, Section 2.3], where a data-flow analysis is defined via a number of equations over the lattice L modelling the property to be analysed. For every program label ℓ we have two equations: one describing the generalised 'entry' in terms of the generalised 'exit' of the block in question, and the other describing 'exit' in terms of 'entry' – for forward analysis we have $\circ$=entry and $\bullet$=exit, for a backward analysis the situation is reversed.

$$\begin{aligned}
\mathit{Analysis}_\bullet(\ell) &= f_\ell(\mathit{Analysis}_\circ(\ell)) \\
\mathit{Analysis}_\circ(\ell) &= \begin{cases} \iota, \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\}, \text{otherwise} \end{cases}
\end{aligned}$$

For the typical classical analyses, such as Live Variable LV and Reaching Definition RD, the property lattice L is often the power-set of some underlying set (like **Var** as in the case of the LV analysis). For a may-analysis the collecting operation $\sqcup$ of L is represented by set union $\cup$ and for must-analysis it is the intersection operation $\cap$. The flow relation F can be the forward or backward flow. ι specifies the initial or final analysis information on "extreme" labels in E, where E is $\{\mathit{init}(S_\star)\}$ or $\mathit{final}(S_\star)$, and f_ℓ is the transfer function associated with $B^\ell \in \mathbf{Block}(S)$ [18, Section 2.3].

3.2 Live Variable Analysis

We will illustrate the basic principles of the equational approach to data flow analysis by considering Live Variable analysis (LV) following the presentation in [18, Section 2.1]. The problem is to identify at any program point those variables which are *live*, i.e. which may later be used in an assignment or test.

There are two phases of classical LV analysis: (i) formulation of data-flow equations as set equations (or more generally over a property lattice L), (ii) finding or constructing solutions to these equations, for example, via a fixed-point construction. In the classical analysis we associate to every program point or label ℓ – to be precise the entry and the exit of each label – the information which describes (a super-set of) those variables which are alive at this program point.

Based on the auxiliary functions $\mathit{gen}_{\mathsf{LV}} : \mathbf{Block} \to \mathscr{P}(\mathbf{Var})$ and $\mathit{kill}_{\mathsf{LV}} : \mathbf{Block} \to \mathscr{P}(\mathbf{Var})$ which only depend on the syntax of the local block $[B]^\ell$ and are defined as

$$\begin{array}{rclrcl} kill_{\mathsf{LV}}([x := a]^\ell) & = & \{\mathtt{x}\} & gen_{\mathsf{LV}}([x := a]^\ell) & = & FV(a) \\ kill_{\mathsf{LV}}([x \mathrel{?=} \rho]^\ell) & = & \{\mathtt{x}\} & gen_{\mathsf{LV}}([x \mathrel{?=} \rho]^\ell) & = & \emptyset \\ kill_{\mathsf{LV}}([\mathtt{skip}]^\ell) & = & \emptyset & gen_{\mathsf{LV}}([\mathtt{skip}]^\ell) & = & \emptyset \\ kill_{\mathsf{LV}}([b]^\ell) & = & \emptyset & gen_{\mathsf{LV}}([b]^\ell) & = & FV(b) \end{array}$$

we can define the transfer functions for the *LV* analysis $f_\ell^{LV} : \mathscr{P}(\mathbf{Var}_\star) \to \mathscr{P}(\mathbf{Var}_\star)$ by

$$f_\ell^{LV}(X) = X \setminus kill_{\mathsf{LV}}([B]^\ell) \cup gen_{\mathsf{LV}}([B]^\ell)$$

This allows us to define equations over the property space $L = \mathscr{P}(\mathbf{Var})$, i.e. set equations, which associate to every label entry and exit the analysis information $\mathsf{LV}_{entry} : \mathbf{Lab} \to \mathscr{P}(\mathbf{Var})$ and $\mathsf{LV}_{exit} : \mathbf{Lab} \to \mathscr{P}(\mathbf{Var})$. These set equations are of the general form for a backward may analysis:

$$\begin{array}{rcl} \mathsf{LV}_{entry}(\ell) & = & f_\ell^{LV}(\mathsf{LV}_{exit}(\ell)) \\ \mathsf{LV}_{exit}(\ell) & = & \bigcup_{(\ell,\ell') \in flow} \mathsf{LV}_{entry}(\ell') \end{array}$$

At the beginning of the analysis (i.e. for final labels, as this is a backward analysis) we set $\mathsf{LV}_{exit}(\ell) = \emptyset$.

Example 1 *Consider the following program:*

$$\begin{array}{l} [x \mathrel{?=} \{0,1\}]^1;\ [y \mathrel{?=} \{0,1,2,3\}]^2;\ [x := x + y \bmod 4]^3; \\ \mathtt{if}\ [x > 2]^4\ \mathtt{then}\ [z := x]^5\ \mathtt{else}\ [z := y]^6\ \mathtt{fi} \end{array}$$

Although the program is probabilistic we still can perform a classical analysis by considering non-zero probabilities simply as possibilities. The flow is given by $\{(1,2),(2,3),(3,4),(4,\underline{5}),(4,6)\}$.

With the auxiliary functions $kill_{\mathsf{LV}}$ *and* gen_{LV} *we can now specify the data-flow equations:*

	$gen_{\mathsf{LV}}(\ell)$	$kill_{\mathsf{LV}}(\ell)$
1	$\emptyset$	$\{x\}$
2	$\emptyset$	$\{y\}$
3	$\{x,y\}$	$\{x\}$
4	$\{x\}$	$\emptyset$
5	$\{x\}$	$\{z\}$
6	$\{y\}$	$\{z\}$

$$\begin{array}{rclrcl} \mathsf{LV}_{entry}(1) & = & \mathsf{LV}_{exit}(1) \setminus \{x\} & \mathsf{LV}_{exit}(1) & = & \mathsf{LV}_{entry}(2) \\ \mathsf{LV}_{entry}(2) & = & \mathsf{LV}_{exit}(2) \setminus \{y\} & \mathsf{LV}_{exit}(2) & = & \mathsf{LV}_{entry}(3) \\ \mathsf{LV}_{entry}(3) & = & \mathsf{LV}_{exit}(3) \setminus \{x\} \cup \{x,y\} & \mathsf{LV}_{exit}(3) & = & \mathsf{LV}_{entry}(4) \\ \mathsf{LV}_{entry}(4) & = & \mathsf{LV}_{exit}(4) \cup \{x\} & \mathsf{LV}_{exit}(4) & = & \mathsf{LV}_{entry}(5) \cup \mathsf{LV}_{entry}(6) \\ \mathsf{LV}_{entry}(5) & = & \mathsf{LV}_{exit}(5) \setminus \{z\} \cup \{x\} & \mathsf{LV}_{exit}(5) & = & \emptyset \\ \mathsf{LV}_{entry}(6) & = & \mathsf{LV}_{exit}(6) \setminus \{z\} \cup \{y\} & \mathsf{LV}_{exit}(6) & = & \emptyset \end{array}$$

Then the classical LV analysis of our program gives the solutions:

$$\begin{array}{rclrcl} \mathsf{LV}_{entry}(1) & = & \emptyset & \mathsf{LV}_{exit}(1) & = & \{x\} \\ \mathsf{LV}_{entry}(2) & = & \{x\} & \mathsf{LV}_{exit}(2) & = & \{x,y\} \\ \mathsf{LV}_{entry}(3) & = & \{x,y\} & \mathsf{LV}_{exit}(3) & = & \{x,y\} \\ \mathsf{LV}_{entry}(4) & = & \{x,y\} & \mathsf{LV}_{exit}(4) & = & \{x,y\} \\ \mathsf{LV}_{entry}(5) & = & \{x\} & \mathsf{LV}_{exit}(5) & = & \emptyset \\ \mathsf{LV}_{entry}(6) & = & \{y\} & \mathsf{LV}_{exit}(6) & = & \emptyset. \end{array}$$

4 The Probabilistic Setting

In order to specify a probabilistic data flow analysis using the analogue of the classical equational approach (as presented in the previous sections), we have to define the main ingredients of the analysis in a probabilistic setting namely a vector space as property space (replacing the property lattice L), a linear operator representing the transfer functions f_ℓ, and a method for the information collection (in place of the $\bigsqcup$ operation of the classical monotone framework). Moreover, as we will work with probabilistic states, the second point implies that the control-flow graph will be labelled by some probability information.

As a **property space** we consider distributions $\mathbf{Dist}(L) \subseteq \mathcal{V}(L)$ over a set L, e.g. the corresponding classical property space. For a relational analysis, where the classical property lattice corresponds to $L = L_1 \times L_2$ (cf [11]), the probabilistic property space will be the tensor product $\mathcal{V}(L_1) \otimes \mathcal{V}(L_2)$; this allows us to represent properties via joint probabilities which are able to express the dependency or correlation between states.

We can define probabilistic **transfer functions** by using the linear representation of the classical f_ℓ, i.e. a matrix $\mathbf{F}_\ell = \mathbf{F}_{f_\ell}$ as introduced above in Section 2.2. In general, we will define a probabilistic transfer function by means of an appropriate abstraction of the concrete semantics $[\![[B]^\ell]\!]$ of a given block $[B]^\ell$ according to PAI, i.e. $\mathbf{F}_\ell = \mathbf{A}^\dagger [\![[B]^\ell]\!] \mathbf{A}$ for the relevant abstraction matrix $\mathbf{A}$.

In the classical analysis we treat tests b non-deterministically, to avoid problems with the potential undecidability of predicates. Moreover, we take everything which is possible i.e. the collection of what can happen along the different execution paths, e.g. the two branches of an `if` statement. In the probabilistic setting we **collect information** by means of weighted sums, where the 'weights' are the probabilities associated to each branch. These probabilities come from an estimation of the (concrete or abstract) branch probabilities and are propagated along the control flow graph representing the **flow relation**.

4.1 Control Flow Probabilities

If we execute a program in classical states s which have been chosen randomly according to some probability distribution ρ then this also induces a probability distribution on the possible control flow steps.

Definition 2 *Given a program S_ℓ with $init(S_\ell) = \ell$ and a probability distribution ρ on* **State***, the probability $p_{\ell,\ell'}(\rho)$ that the control is flowing from ℓ to ℓ' is defined as:*

$$p_{\ell,\ell'}(\rho) = \sum_s \left\{ p \cdot \rho(s) \mid \exists s' \text{ s.t. } \langle S_\ell, s\rangle \Rightarrow_p \langle S_{\ell'}, s'\rangle \right\}.$$

In other words, if we provide with a certain probability $\rho(s)$ a concrete execution environment or classical state s for a program S_ℓ, then the control flow probability $p_{\ell,\ell'}(\rho)$ is the probability that we end up with a configuration $\langle S_{\ell'}, \ldots\rangle$ for whatever state in the successor configuration.

Example 2 *Consider the program:* $[x \texttt{ ?= } \{0,1\}]^1$; `if` $[x > 0]^2$ `then` $[\texttt{skip}]^3$ `else` $[x := 0]^4$ `fi`. *We can have two possible states at label 2, namely $s_0 = [x \mapsto 0]$ and $s_1 = [x \mapsto 1]$. After the first statement has been executed in one of two possible ways (with any intial state s):*

$$\left\langle [x \texttt{ ?= } \{0,1\}]^1; \texttt{ if } [x > 0]^2 \texttt{ then } [\texttt{skip}]^3 \texttt{ else } [x := 0]^4 \texttt{ fi}, s \right\rangle \Rightarrow_{\frac{1}{2}}$$
$$\Rightarrow_{\frac{1}{2}} \quad \left\langle \texttt{if } [x > 0]^2 \texttt{ then } [\texttt{skip}]^3 \texttt{ else } [x := 0]^4 \texttt{ fi}, s_0 \right\rangle$$
$$or \quad \left\langle [x \texttt{ ?= } \{0,1\}]^1; \texttt{ if } [x > 0]^2 \texttt{ then } [\texttt{skip}]^3 \texttt{ else } [x := 0]^4 \texttt{ fi}, s \right\rangle \Rightarrow_{\frac{1}{2}}$$
$$\Rightarrow_{\frac{1}{2}} \quad \left\langle \texttt{if } [x > 0]^2 \texttt{ then } [\texttt{skip}]^3 \texttt{ else } [x := 0]^4 \texttt{ fi}, s_1 \right\rangle$$

the distribution over states is obviously $\rho = \{\langle s_0, \frac{1}{2}\rangle \langle s_1, \frac{1}{2}\rangle\}$. *However, in each execution path we have at any moment a definite value for x (the distribution* ρ *describes a property of the set of all executions, not of one execution alone).*

The branch probability in this case (independently of the state s and of any distribution ρ*) is simply* $p_{1,2}(\rho) = 1$ *because, although there are two possible execution steps, the successor configurations are 'coincidently' equipped with the same program* $\texttt{if}\ [x > 0]^2\ \texttt{then}\ [\texttt{skip}]^3\ \texttt{else}\ [x := 0]^4\ \texttt{fi}$.

The successive control steps from label 2 *to* 3 *and* 4*, respectively, both occur with probability* 1 *as in each state* s_0 *and* s_1 *the value of x is a definite one.*

$$\begin{array}{rlcl} & \langle \texttt{if}\ [x > 0]^2\ \texttt{then}\ [\texttt{skip}]^3\ \texttt{else}\ [x := 0]^4\ \texttt{fi}, s_0\rangle & \Rightarrow_1 & \langle [x := 0]^4, s_0\rangle \\ \textit{and} & \langle \texttt{if}\ [x > 0]^2\ \texttt{then}\ [\texttt{skip}]^3\ \texttt{else}\ [x := 0]^4\ \texttt{fi}, s_1\rangle & \Rightarrow_1 & \langle [\texttt{skip}]^3, s_1\rangle \end{array}$$

Thus the branch probabilities with $\rho = \{\langle s_0, \frac{1}{2}\rangle, \langle s_1, \frac{1}{2}\rangle\}$ *are* $p_{2,3}(\rho) = \frac{1}{2}$ *and* $p_{2,4}(\rho) = \frac{1}{2}$. *In general for any* $\rho = \{\langle s_0, p_0\rangle, \langle s_1, p_1\rangle\}$ *we have* $p_{2,3}(\rho) = p_1$ *and* $p_{2,4}(\rho) = p_0$ *despite the fact that the transitions are deterministic. It is the randomness in the probabilistic state that determines in this case the branch probabilities.*

For all blocks in a control flow graph – except for the tests b – there is always only one next statement $S_{\ell'}$ so that the branch probability $p_{\ell,\ell'}(\rho)$ is always 1 for all ρ. For tests b in `if` and `while` statements we have only two different successor statements, one corresponding to the case where $[b]^\ell$ evaluates to **true** and one for **false**. As the corresponding probabilities must sum up to 1 we only need to specify the first case which we denote by $p_\ell(\rho)$.

The probability distributions over states at every execution point are thus critical for the analysis as they determine the branch probabilities for tests, and we need to provide them. The problem is, of course that analysing these probabilities is nearly as expensive as analysing the concrete computation or program executions. It is therefore reasonable to investigate abstract branch probabilities, based on classes of states, or abstract states. It is always possible to lift concrete distributions to ones over (equivalence) classes.

Definition 3 *Given a probability distribution* ρ *on* **State** *and an equivalence relation* $\sim$ *on states then we denote by* $\rho^\# = \rho^\#_\sim$ *the probability distribution on the set of equivalence classes* $\mathbf{State}^\# = \mathbf{State}/\!\sim$ *defined by*

$$\rho^\#([s]_\sim) = \sum_{s' \in [s]_\sim} \rho(s')$$

where $[s]_\sim$ *denotes the equivalence classes of s wrt* $\sim$.

4.2 Estimating Abstract Branch Probabilities

In order to determine concrete or abstract branch probabilities we need to investigate – as we have seen in Example 2 – the interplay between distribution over states and the test $[b]^\ell$ we are interested in. We need for this the linear representation $\mathbf{P}_b$ of the test predicate b as defined in Section 2.2, which for a given distribution over states determines a sub-distribution of those states that lead into one of the two branches by filtering out those states where this happens.

Example 3 *Consider the simple program* $\texttt{if}\ [x >= 1]^1\ \texttt{then}\ [x := x - 1]^2\ \texttt{else}\ [\texttt{skip}]^3\ \texttt{fi}$ *and assume that x has values in* $\{0, 1, 2\}$ *(enumerated in the obvious way). Then the test* $b = (x >= 1)$ *is represented* *by the projection matrix:*

$$\mathbf{P}(x >= 1) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } \mathbf{P}(x >= 1)^{\perp} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{P}(x = 0)$$

For any given concrete probability distribution over states $\rho = \{\langle 0, p_0\rangle, \langle 1, p_1\rangle, \langle 2, p_2\rangle\} = (p_0, p_1, p_2)$ we can easily compute the probabilities to go from label 1 *to label* 2 *as $\rho\mathbf{P}(x >= 1) = (0, p_1, p_2)$ and thus*

$$p_{1,2}(\rho) = \|\rho \cdot \mathbf{P}(x >= 1)\|_1 = p_1 + p_2,$$

where $\|.\|_1$ is the 1-norm of vectors, i.e. $\|(x_i)_i\| = \sum_i |x_i|$, which we use here to aggregate the total probabilities. Similarly, for the else branch, with $\mathbf{P}^{\perp} = \mathbf{I} - \mathbf{P}$:

$$p_{1,3}(\rho) = \|\rho \cdot \mathbf{P}^{\perp}(x >= 1)\|_1 = p_0.$$

In general, the branching behaviour at a test b is described by the projection operator $\mathbf{P}(b)$ and its complement $\mathbf{P}^{\perp}(b) = \mathbf{P}(\neg b)$. For a branching point $[b]^{\ell}$ with $(\ell, \underline{\ell'}), (\ell, \ell'') \in$ *flow*, we denote $\mathbf{P}(b)$ by $\mathbf{P}(\ell, \ell')$ and $\mathbf{P}(\neg b) = \mathbf{P}(b)^{\perp}$ by $\mathbf{P}(\ell, \ell'')$. Each branch probability can be computed for any given input distribution as $p_{\ell,\ell'}(\rho) = \|\rho\mathbf{P}(\ell, \ell')\|_1$ and $p_{\ell,\ell''}(\rho) = \|\rho\mathbf{P}(\ell, \ell'')\|_1$, respectively.

Sometimes it could be useful or practically more appropriate to consider abstract branch probabilities. These can be obtained by means of abstractions on the state space corresponding to classifications c : **State** $\to$ **State**$^{\#}$ that, as explained in Section 2.2, can be lifted to *classification matrices*. Given an equivalence relation $\sim$ on the states and its matrix representation $\mathbf{A}_{\sim}$, we can compute the individual chance of abstract states (i.e. equivalence classes of states) to take the **true** or **false** branch of a test by multiplying the abstract distribution $\rho^{\#}$ by an abstract version $\mathbf{P}(b)^{\#}$ of $\mathbf{P}(b)$ that we can use to select those classes of states satisfying b. In doing so we must guarantee that:

$$\begin{aligned} \rho\mathbf{P}(b)\mathbf{A} &= \rho^{\#}\mathbf{P}^{\#}(b) \\ \rho\mathbf{P}(b)\mathbf{A} &= \rho\mathbf{A}\mathbf{P}^{\#}(b) \\ \mathbf{P}(b)\mathbf{A} &= \mathbf{A}\mathbf{P}^{\#}(b) \end{aligned}$$

In order to give an explicit description of $\mathbf{P}^{\#}$ we only would need to multiply the last equation from the left with $\mathbf{A}^{-1}$. However, $\mathbf{A}$ is in general not a square matrix and thus not invertible. So we use instead the Moore-Penrose pseudo-inverse to have the closest, least-square approximation possible.

$$\begin{aligned} \mathbf{A}^{\dagger}\mathbf{P}(b)\mathbf{A} &= \mathbf{A}^{\dagger}\mathbf{A}\mathbf{P}^{\#}(b) \\ \mathbf{A}^{\dagger}\mathbf{P}(b)\mathbf{A} &= \mathbf{P}^{\#}(b) \end{aligned}$$

The abstract test matrix $\mathbf{P}^{\#}(b)$ contains all the information we need in order to estimate the abstract branch probabilities. Again, we denote by $\mathbf{P}(\ell, \ell')^{\#} = \mathbf{P}^{\#}(b)$ and $\mathbf{P}(\ell, \ell'')^{\#} = \mathbf{P}^{\#}(\neg b) = \mathbf{P}^{\#}(b)^{\perp}$ for a branching point $[b]^{\ell}$ with $(\ell, \underline{\ell'}), (\ell, \ell'') \in$ *flow*.

Branch prediction/predictors in hardware design has long history [16, 20]. It is used at test points $[b]^{\ell}$ to allow pre-fetching of instructions of the expected branch before the test is actually evaluated. If the prediction is wrong the prefetched instructions need to be discarded and the correct ones to be fetched. Ultimately, wrong predictions “just” lead to longer running times, the correctness of the program is not concerned. It can be seen as a form of speculative optimisation. Typical applications or cases where branch prediction is relevant is for nested tests (loops or ifs). Here we get exactly the interplay between different tests and/or abstractions. We illustrate this in the following example.

Example 4 *Consider the following program that counts the prime numbers.*

$$[i := 2]^1;\ \texttt{while}\ [i < 100]^2\ \texttt{do if}\ [prime(i)]^3\ \texttt{then}\ [p := p+1]^4\ \texttt{else}\ [\texttt{skip}]^5\ \texttt{fi};\ [i := i+1]^6\ \texttt{od}$$

Within our framework we can simulate to a certain degree a history dependent branch prediction. If the variable p has been updated in the previous iteration it is highly unlikely it will so again in the next – in fact that only happens in the first two iterations. One can also interpret this as follows: For i even the branch probability $p_{3,4}(\rho_e)$ at label 3 is practically zero for any reasonable distribution, e.g. a uniform distribution ρ_e, on evens. To see this, we need to investigate only the form of

$$\mathbf{P}(prime(i))^\# = \mathbf{A}_e^\dagger \mathbf{P}(prime(i))\mathbf{A}_e,$$

where A_e is the abstraction corresponding to the classification in even and odd.

In order to understand how an abstract property interacts with the branching in the program, as in the previous example we look at $\mathbf{A}^\dagger\mathbf{P}(b)\mathbf{A}$ in order to evaluate how good a branch prediction is for a certain predicate/test b if it is based on a certain abstraction/property $\mathbf{A}$. This is explained in the following example where we consider two properties/abstractions and corresponding tests.

Example 5 *Let us consider two tests for numbers in the range $i = 0, 1, 2, 3, \ldots, n$):*

$$\mathbf{P}_e = (\mathbf{P}(even(n)))_{ii} = \begin{cases} 1 & \text{if } i = 2k \\ 0 & otherwise \end{cases} \qquad \mathbf{P}_p = (\mathbf{P}(prime(n)))_{ii} = \begin{cases} 1 & \text{if } prime(i) \\ 0 & otherwise \end{cases}$$

Likewise we can consider two corresponding abstractions ($j \in \{1 = \mathbf{true}, 2 = \mathbf{false}\}$):

$$(\mathbf{A}_e)_{ij} = \begin{cases} 1 & \text{if } i = 2k+1 \wedge j = 2 \\ 1 & \text{if } i = 2k \wedge j = 1 \\ 0 & otherwise \end{cases} \qquad (\mathbf{A}_p)_{ij} = \begin{cases} 1 & \text{if } prime(i) \wedge j = 2 \\ 1 & \text{if } \neg prime(i) \wedge j = 1 \\ 0 & otherwise \end{cases}$$

Then we can use $\mathbf{P}^\#$ and its orthogonal complement, $(\mathbf{P}^\#)^\perp = \mathbf{I} - \mathbf{P}^\#$ to determine information about the quality of a certain property or its corresponding abstraction via the number of false positives. In fact, this will tell us how precise the abstraction is with respect to tests (such as those controlling a loop or conditional). With rounding the values to 2 significant digits we get, for example the following results for different concrete ranges of the concrete values $0, \ldots, n$.

$$\begin{array}{ccccc}
 & \mathbf{A}_e^\dagger\mathbf{P}_p\mathbf{A}_e & \mathbf{A}_e^\dagger\mathbf{P}_p^\perp\mathbf{A}_e & \mathbf{A}_p^\dagger\mathbf{P}_e\mathbf{A}_p & \mathbf{A}_p^\dagger\mathbf{P}_e^\perp\mathbf{A}_p \\
n = 10 & \begin{pmatrix} 0.20 & 0.00 \\ 0.00 & 0.60 \end{pmatrix} & \begin{pmatrix} 0.80 & 0.00 \\ 0.00 & 0.40 \end{pmatrix} & \begin{pmatrix} 0.25 & 0.00 \\ 0.00 & 0.67 \end{pmatrix} & \begin{pmatrix} 0.75 & 0.00 \\ 0.00 & 0.33 \end{pmatrix} \\
n = 100 & \begin{pmatrix} 0.02 & 0.00 \\ 0.00 & 0.48 \end{pmatrix} & \begin{pmatrix} 0.98 & 0.00 \\ 0.00 & 0.52 \end{pmatrix} & \begin{pmatrix} 0.04 & 0.00 \\ 0.00 & 0.65 \end{pmatrix} & \begin{pmatrix} 0.96 & 0.00 \\ 0.00 & 0.35 \end{pmatrix} \\
n = 1000 & \begin{pmatrix} 0.00 & 0.00 \\ 0.00 & 0.33 \end{pmatrix} & \begin{pmatrix} 1.00 & 0.00 \\ 0.00 & 0.67 \end{pmatrix} & \begin{pmatrix} 0.01 & 0.00 \\ 0.00 & 0.60 \end{pmatrix} & \begin{pmatrix} 0.99 & 0.00 \\ 0.00 & 0.40 \end{pmatrix} \\
n = 10000 & \begin{pmatrix} 0.00 & 0.00 \\ 0.00 & 0.25 \end{pmatrix} & \begin{pmatrix} 1.00 & 0.00 \\ 0.00 & 0.75 \end{pmatrix} & \begin{pmatrix} 0.00 & 0.00 \\ 0.00 & 0.57 \end{pmatrix} & \begin{pmatrix} 1.00 & 0.00 \\ 0.00 & 0.43 \end{pmatrix}
\end{array}$$

Note that the positive and negative versions of these matrices always add up to the identity matrix $\mathbf{I}$. *Also, the entries in the upper left corner of $\mathbf{A}_e^\dagger\mathbf{P}_p\mathbf{A}_e$ give us information about the chances that an even* *number is also a prime number: For small n the percentage is a fifth (indeed* 2 *is a prime and it is one out of* 5 *even numbers under* 10*); the larger n gets the less relevant is this single even prime. With* $\mathbf{A}_p^\dagger \mathbf{P}_e \mathbf{A}_p$ *we get the opposite information: Among the prime numbers* $\{2,3,5,7\}$ *smaller than* 10 *there is one which is even, i.e. 25%; again this effect diminishes for larger n. Finally, the lower right entry in these matrices gives us the percentage that a non-prime number is odd and/or that an odd number is not prime, respectively.*

4.3 Linear Equations Framework

A general framework for our probabilistic data-flow analysis can be defined in analogy with the classical monotone framework by defining the following linear equations:

$$\begin{aligned} \mathit{Analysis}_\bullet(\ell) &= \mathit{Analysis}_\circ(\ell) \cdot \mathbf{F}_\ell \\ \mathit{Analysis}_\circ(\ell) &= \begin{cases} \iota, \text{if } \ell \in E \\ \sum\{\mathit{Analysis}_\bullet(\ell') \cdot \mathbf{P}(\ell',\ell)^\# \mid (\ell',\ell) \in F\}, \text{otherwise} \end{cases} \end{aligned}$$

The first equation is a straight forward generalisation of the classical case, while the second one is defined by means of the linear sums over vectors. A simpler version is obtained by considering static branch prediction:

$$\mathit{Analysis}_\circ(\ell) = \sum\{p_{\ell',\ell} \cdot \mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\}$$

with $p_{\ell',\ell}$ is a numerical value representing a *static* branch probability.

We have as many variables in this systems of equations as there are individual equations. As a result we get unique solutions rather than least fix-points as in the classical setting.

This general scheme must be extended to include a preliminary phase of probability estimation if one wants to improve the quality of the branch prediction. In this case, the abstract state should carry two kinds of information: One, Prob, to provide estimates for probabilities, the other, Analysis, to analyse the actual property in question. The same abstract branch probabilities $\mathbf{P}(\ell',\ell)^\#$ – which we obtain via Prob – can then be used in both cases, but we have different information or properties and different transfer functions for Prob and Analysis.

4.4 Probabilistic Live Variable Analysis

We can use the previously defined probabilistic setting for a data flow analysis, to define a probabilistic version of the Live Variable analysis extending the one in [18] in order to also cover for random assignments and to provide estimates for 'live' probabilities.

The transfer functions, which describe how the program analysis information changes when we pass through a block $[B]^\ell$, is for the classical analysis given via the two auxiliary functions $\mathit{gen}_{\mathsf{LV}}$ and $\mathit{kill}_{\mathsf{LV}}$ (cf. Example 1). Probabilistic versions of these operations can be defined as follows. Consider two properties d for 'dead', and l for 'live' and the space $\mathcal{V}(\{0,1\}) = \mathcal{V}(\{d,l\}) = \mathbb{R}^2$ as the property space corresponding to a single variable. On this space define the operators:

$$\mathbf{L} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{K} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}.$$

The matrix $\mathbf{L}$ changes the "liveliness" of a variable from whatever it is (dead or alive) into alive, while $\mathbf{K}$ does the opposite. The local transfer operators

$$\mathbf{F}_\ell = \mathbf{F}_\ell^{LV} : \mathcal{V}(\{0,1\})^{\otimes|\mathbf{Var}|} \to \mathcal{V}(\{0,1\})^{\otimes|\mathbf{Var}|}$$

for the block $[x := a]^\ell$ can thus be defined as (with $\mathbf{I}$ the identity matrix)

$$\mathbf{F}_\ell = \bigotimes_{x_i \in \mathbf{Var}} \mathbf{X}_i \text{ with } \mathbf{X}_i = \begin{cases} \mathbf{L} & \text{if } x_i \in FV(a) \\ \mathbf{K} & \text{if } x_i = x \wedge x_i \notin FV(a) \\ \mathbf{I} & \text{otherwise.} \end{cases}$$

and similarly for tests $[b]^\ell$

$$\mathbf{F}_\ell = \bigotimes_{x_i \in \mathbf{Var}} \mathbf{X}_i \text{ with } \mathbf{X}_i = \begin{cases} \mathbf{L} & \text{if } x_i \in FV(b) \\ \mathbf{I} & \text{otherwise.} \end{cases}$$

For $[\texttt{skip}]^\ell$ and random assignments $[x \texttt{ ?= } \rho]^\ell$ we simply have $\mathbf{F}_\ell = \bigotimes_{x_i \in \mathbf{Var}} \mathbf{I}$.

In the following example we demonstrate the use of our general framework for probabilistic data-flow analysis by defining a probabilistic *LV* analysis for the program in Example 1.

Example 6 *For the program in Example 1 we present a LV analysis based on concrete branch probabilities. That means that in the first phase of the analysis (which determines the branch probabilities) we will not abstract the values of x and y (and ignore z all together). If the concrete state of each variable is a value in* $\{0,1,2,3\}$*, then the probabilistic state is an element in* $\mathcal{V}(\{0,1,2,3\})^{\otimes 3} = \mathbb{R}^{4^3} = \mathbb{R}^{64}$. *The abstraction we use when we compute the concrete branch probabilities is* $\mathbf{I} \otimes \mathbf{I} \otimes \mathbf{A}_f$*, i.e. z is ignored. This allows us to reduce the dimensions of the probabilistic state space from* 64 *down to just* 16. *The abstract transfer functions for the first* 3 *statements are given in the Appendix.*

We can now compute the probability distribution at label 4 *for any given input distribution. The abstract transfer functions* $\mathbf{F}_5^\#$ *and* $\mathbf{F}_6^\#$ *are the identity as we have restricted ourselves only to the variables x and y.*

We can now set the linear equations for the joint distributions over x and y at the entry and exit to each of the labels:

$$\begin{array}{lcl@{\qquad}lcl}
\mathsf{Prob}_{entry}(1) & = & \rho & \mathsf{Prob}_{exit}(1) & = & \mathsf{Prob}_{entry}(1) \cdot \mathbf{F}_1^\# \\
\mathsf{Prob}_{entry}(2) & = & \mathsf{Prob}_{exit}(1) & \mathsf{Prob}_{exit}(2) & = & \mathsf{Prob}_{entry}(1) \cdot \mathbf{F}_2^\# \\
\mathsf{Prob}_{entry}(3) & = & \mathsf{Prob}_{exit}(2) & \mathsf{Prob}_{exit}(3) & = & \mathsf{Prob}_{entry}(1) \cdot \mathbf{F}_3^\# \\
\mathsf{Prob}_{entry}(4) & = & \mathsf{Prob}_{exit}(3) & \mathsf{Prob}_{exit}(4) & = & \mathsf{Prob}_{entry}(4) \\
\mathsf{Prob}_{entry}(5) & = & \mathsf{Prob}_{exit}(4) \cdot \mathbf{P}_4^\# & \mathsf{Prob}_{exit}(5) & = & \mathsf{Prob}_{entry}(5) \\
\mathsf{Prob}_{entry}(6) & = & \mathsf{Prob}_{exit}(4) \cdot (\mathbf{I} - \mathbf{P}_4^\#) & \mathsf{Prob}_{exit}(6) & = & \mathsf{Prob}_{entry}(6)
\end{array}$$

These equations are easy to solve. In particular we can explicitly determine

$$\begin{array}{lcl}
\mathsf{Prob}_{entry}(5) & = & \rho \cdot \mathbf{F}_1^\# \cdot \mathbf{F}_2^\# \cdot \mathbf{F}_3^\# \cdot \mathbf{P}_4^\# \\
\mathsf{Prob}_{entry}(6) & = & \rho \cdot \mathbf{F}_1^\# \cdot \mathbf{F}_2^\# \cdot \mathbf{F}_3^\# \cdot \mathbf{P}_4^\#,
\end{array}$$

that give us the static branch probabilities $p_{4,5}(\rho) = \|\mathsf{Prob}_{entry}(5)\|_1 = \frac{1}{4}$ *and* $p_{4,6}(\rho) = \|\mathsf{Prob}_{entry}(6)\|_1 = \frac{3}{4}$. *These distributions can explicitly be computed and do not depend on the initial distribution* ρ.

We then perform a probabilistic LV analysis using these probabilities as required. Using the abstract property space and the auxiliary operators we get:

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{LV}_{entry}(1) & = & \mathsf{LV}_{exit}(1)\cdot(\mathbf{K}\otimes\mathbf{I}\otimes\mathbf{I}) & \mathsf{LV}_{exit}(1) & = & \mathsf{LV}_{entry}(2) \\
\mathsf{LV}_{entry}(2) & = & \mathsf{LV}_{exit}(2)\cdot(\mathbf{I}\otimes\mathbf{K}\otimes\mathbf{I}) & \mathsf{LV}_{exit}(2) & = & \mathsf{LV}_{entry}(3) \\
\mathsf{LV}_{entry}(3) & = & \mathsf{LV}_{exit}(3)\cdot(\mathbf{L}\otimes\mathbf{L}\otimes\mathbf{I}) & \mathsf{LV}_{exit}(3) & = & \mathsf{LV}_{entry}(4) \\
\mathsf{LV}_{entry}(4) & = & \mathsf{LV}_{exit}(4)\cdot(\mathbf{L}\otimes\mathbf{I}\otimes\mathbf{I}) & \mathsf{LV}_{exit}(4) & = & p_{4,5}\mathsf{LV}_{entry}(5)+p_{4,6}\mathsf{LV}_{entry}(6) \\
\mathsf{LV}_{entry}(5) & = & \mathsf{LV}_{exit}(5)\cdot(\mathbf{L}\otimes\mathbf{I}\otimes\mathbf{K}) & \mathsf{LV}_{exit}(5) & = & (1,0)\otimes(1,0)\otimes(1,0) \\
\mathsf{LV}_{entry}(6) & = & \mathsf{LV}_{exit}(6)\cdot(\mathbf{I}\otimes\mathbf{L}\otimes\mathbf{K}) & \mathsf{LV}_{exit}(6) & = & (1,0)\otimes(1,0)\otimes(1,0)
\end{array}$$

And thus the solutions for the probabilistic LV analysis are given by:

$$\begin{array}{rcl}
\mathsf{LV}_{entry}(1) & = & (1,0)\otimes(1,0)\otimes(1,0) \\
\mathsf{LV}_{entry}(2) & = & (0,1)\otimes(1,0)\otimes(1,0) \\
\mathsf{LV}_{entry}(3) & = & 0.25\cdot(0,1)\otimes(0,1)\otimes(1,0)+ \\
 & + & 0.75\cdot(0,1)\otimes(0,1)\otimes(1,0) \\
 & = & (0,1)\otimes(0,1)\otimes(1,0) \\
\mathsf{LV}_{entry}(4) & = & 0.25\cdot(0,1)\otimes(1,0)\otimes(1,0)+ \\
 & + & 0.75\cdot(0,1)\otimes(0,1)\otimes(1,0) \\
\mathsf{LV}_{entry}(5) & = & (0,1)\otimes(1,0)\otimes(1,0) \\
\mathsf{LV}_{entry}(6) & = & (1,0)\otimes(0,1)\otimes(1,0)
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{LV}_{exit}(1) & = & (0,1)\otimes(1,0)\otimes(1,0) \\
\mathsf{LV}_{exit}(2) & = & (0,1)\otimes(0,1)\otimes(1,0) \\
\mathsf{LV}_{exit}(3) & = & 0.25\cdot(0,1)\otimes(1,0)\otimes(1,0)+ \\
 & + & 0.75\cdot(0,1)\otimes(0,1)\otimes(1,0) \\
\mathsf{LV}_{exit}(4) & = & 0.25\cdot(0,1)\otimes(1,0)\otimes(1,0)+ \\
 & + & 0.75\cdot(1,0)\otimes(0,1)\otimes(1,0) \\
\mathsf{LV}_{exit}(5) & = & (1,0)\otimes(1,0)\otimes(1,0) \\
\mathsf{LV}_{exit}(6) & = & (1,0)\otimes(1,0)\otimes(1,0)
\end{array}$$

This means that, for example, at the beginning label 4*, i.e. the test* $x > 2$ *there are two situations: It can be with probability* $\frac{1}{4}$ *that only the variable x is alive, or with probability* $\frac{3}{4}$ *both variables x and y are alive. One could say that x for sure is alive and y only with a 75% chance. At the exit of label* 4 *the probabilistic LV analysis tells us that with 25% chance* only *x is alive and with 75% that y is the* only *live variable. To say that x is alive with probability* 0.25 *and y with* 0.75 *probability would be wrong: It is either x or y which is alive and this is reflected in the joint distributions represented as tensors, which we obtain as solution. This illustrates that the probabilistic property space cannot be just* $\mathcal{V}(\{x,y,z\})$ *but that we need indeed* $\mathcal{V}(\{d,l\})^{\otimes 3}$.

5 Conclusions and Related Work

This paper highlights two important aspects of probabilistic program analysis in a data-flow style: (i) the use of tensor products in order to represent the correlation between a number of variables, and (ii) the use of Probabilistic Abstract Interpretation to estimate branch probabilities and to construct probabilistic transfer functions. In particular, we argue that static program analysis does not mean necessarily considering *static branch prediction*. Instead – by extending single numbers $p_{\ell,\ell'}$ as branch probabilities to matrices as abstract branch probabilities $\mathbf{P}(\ell,\ell)^{\#}$ – the PAI framework allows us to express dynamic or conditional aspects.

The framework presented here aims in providing a formal basis for speculative optimisation. Speculative optimisation [15, 2] has been an element of hardware design for some time, in particular to branch prediction [16] or for cache optimisation [17]. More recently, related ideas have also been discussed in the context of speculative multi-threading [4] or probabilistic pointer analysis [9, 13].

The work we have presented in this paper concentrates on the conceptual aspects of probabilistic analysis and not on optimal realisation of, for example, concrete branch predictors. Further work should however include practical implementations of the presented framework in order to compare its performance with the large number of predictors in existence. Another research direction concerns the automatic construction of abstractions so that the induced $\mathbf{P}(\ell,\ell)^{\#}$ are optimal and maximally predictive.

References

[1] A.V. Aho, M.S. Lam, R. Sethi & J.D. Ullman (2007): *Compilers: Principles, Techniques, and Tools*, second edition. Pearson Education.

[2] A. A. Belevantsev, S. S. Gaisaryan & V. P. Ivannikov (2008): *Construction of Speculative Optimization Algorithms*. *Programming and Computer Software* 34(3), pp. 138–153, doi:10.1134/S036176880803002X.

[3] A. Ben-Israel & T.N.E. Greville (2003): *Generalised Inverses*, 2nd edition. Springer Verlag.

[4] A. Bhowmik & M. Franklin (2004): *A General Compiler Framework for Speculative Multithreaded Processors*. *IEEE Transactions on Parallel and Distributed Syststems* 15(8), pp. 713–724, doi:10.1109/TPDS.2004.26.

[5] S.L. Campbell & D. Meyer (1979): *Generalized Inverse of Linear Transformations*. Constable, London.

[6] P. Cousot & R. Cousot (1977): *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. In: *POPL'77*, pp. 238–252, doi:10.1145/512950.512973.

[7] F. Deutsch (2001): *Bet Approximation in Inner Product Spaces*. *CMS Books in Mathematics* 7, Springer Verlag, New York — Berlin, doi:10.1007/978-1-4684-9298-9.

[8] A. Di Pierro, C. Hankin & H. Wiklicky (2007): *Abstract Interpretation for Worst and Average Case Analysis*. In: *Program Analysis and Compilation, Theory and Practice*, *LNCS* 4444, Springer Verlag, pp. 160–174, doi:10.1007/978-3-540-71322-7_8.

[9] A. Di Pierro, C. Hankin & H. Wiklicky (2007): *A Systematic Approach to Probabilistic Pointer Analysis*. In Z. Shao, editor: *Proceedings of APLAS'07*, *LNCS* 4807, Springer Verlag, pp. 335–350, doi:10.1007/978-3-540-76637-7_23.

[10] A. Di Pierro, C. Hankin & H. Wiklicky (2010): *Probabilistic Semantics and Analysis*. In: *Formal Methods for Quantitative Aspects of Programming Languages*, *LNCS* 6155, Springer Verlag, pp. 1–42, doi:10.1007/978-3-642-13678-8_1.

[11] A. Di Pierro, P. Sotin & H. Wiklicky (2008): *Relational Analysis and Precision via Probabilistic Abstract Interpretation*. In C. Baier & A. Aldini, editors: *Proceedings of QAPL'08*, Electronic Notes in Theoretical Computer Science, Elsevier, pp. 23–42, doi:10.1016/j.entcs.2008.11.017.

[12] A. Di Pierro & H. Wiklicky (2000): *Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation*. In: *PPDP'00*, pp. 127–138, doi:10.1145/351268.351284.

[13] M.-Y. Hung, P.-S. Chen, Y-S. Hwang, R. D.-C. Ju & J. K. Lee (2012): *Support of Probabilistic Pointer Analysis in the SSA Form*. *IEEE Transactions on Parallel Distributed Syststems* 23(12), pp. 2366–2379, doi:10.1109/TPDS.2012.73.

[14] M.Z. Kwiatkowska, G. Norman & D. Parker (2004): *PRISM 2.0: A Tool for Probabilistic Model Checking*. In: *International Conference on Quantitative Evaluation of Systems (QEST 2004)*, IEEE Computer Society, pp. 322–323, doi:10.1109/QEST.2004.10016.

[15] J. Lin, T. Chen, W.-C. Hsu, P.-C. Yew, R. D.-C. Ju, T.-F. Ngai & S. Chan (2003): *A compiler framework for speculative analysis and optimizations*. In: *Proceedings Conference on Programming Language Design and Implementation (PLDI)*, pp. 289–299, doi:10.1145/781131.781164.

[16] S. McFarling (1993): *Combining Branch Predictors*. Technical Report WLR TN-36, Digital.

[17] D. Nicolaescu, B. Salamat & A.V. Veidenbaum (2006): *Fast Speculative Address Generation and Way Caching for Reducing L1 Data Cache Energy*. In: *Proceedings of the 24th International Conference on Computer Design (ICCD 2006)*, IEEE, pp. 101–107, doi:10.1109/ICCD.2006.4380801.

[18] F. Nielson, H. Riis Nielson & C. Hankin (1999): *Principles of Program Analysis*. Springer Verlag, Berlin – Heidelberg, doi:10.1007/978-3-662-03811-6.

[19] S. Roman (2005): *Advanced Linear Algebra*, 2nd edition. Springer Verlag.

[20] H. Styles & W. Luk (2004): *Exploiting Program Branch Probabilities in Hardware Compilation*. *IEEE Transaction on Computers* 53(11), pp. 1408–1419, doi:10.1109/TC.2004.96.

Appendix

For completeness, we present here the abstract transfer functions in the probabilistic analysis of Example 6.

$$\mathbf{F}_1^{\#} = \begin{pmatrix}
\frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

$$\mathbf{F}_2^{\#} = \begin{pmatrix}
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}
\end{pmatrix}$$

$$\mathbf{F}_3^{\#} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}$$

$$\mathbf{P}_4^{\#} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Slot Games for Detecting Timing Leaks of Programs

Aleksandar S. Dimovski
Faculty of Information-Communication Tech., FON University, Skopje, 1000, MKD
aleksandar.dimovski@fon.edu.mk

In this paper we describe a method for verifying secure information flow of programs, where apart from direct and indirect flows a secret information can be leaked through covert timing channels. That is, no two computations of a program that differ only on high-security inputs can be distinguished by low-security outputs and timing differences. We attack this problem by using slot-game semantics for a quantitative analysis of programs. We show how slot-games model can be used for performing a precise security analysis of programs, that takes into account both extensional and intensional properties of programs. The practicality of this approach for automated verification is also shown.

1 Introduction

Secure information flow analysis is a technique which performs a static analysis of a program with the goal of proving that it will not leak any sensitive (secret) information improperly. If the program passes the test, then we say that it is secure and can be run safely. There are several ways in which secret information can be leaked to an external observer. The most common are direct and indirect leakages, which are described by the so-called non-interference property [13, 18]. We say that a program satisfies the non-interference property if its high-security (secret) inputs do not affect its low-security (public) outputs, which can be seen by external observers.

However, a program can also leak information through its timing behaviour, where an external observer can measure its total running time. Such timing leaks are difficult to detect and prevent, because they can exploit low-level implementation details. To detect timing leaks, we need to ensure that the total running time of a program do not depend on its high-security inputs.

In this paper we describe a game semantics based approach for performing a precise security analysis. We have already shown in [8] how game semantics can be applied for verifying the non-interference property. Now we use slot-game semantics to check for timing leaks of closed and open programs. We focus here only on detecting covert timing channels, since the non-interference property can be verified similarly as in [8]. Slot-game semantics was developed in [11] for a quantitative analysis of Algol-like programs. It is suitable for verifying the above security properties, since it takes into account both extensional (*what* the program computes) and intensional (*how* the program computes) properties of programs. It represents a kind of denotational semantics induced by the theory of operational improvement of Sands [19]. Improvement is a refinement of the standard theory of operational approximation, where we say that one program is an improvement of another if its execution is more efficient in any program context. We will measure efficiency of a program as the sum of costs associated with basic operations it can perform. It has been shown that slot-game semantics is fully abstract (sound and complete) with respect to operational improvement, so we can use it as a denotational theory of improvement to analyse programming languages.

The advantages of game semantics (denotational) based approach for verifying security are several. We can reason about open programs, i.e. programs with non-locally defined identifiers. Moreover, game semantics is compositional, which enables analysis about program fragments to be combined into an analysis of a larger program. Also the model hides the details of local-state manipulation of a program, which results in small models with maximum level of abstraction where are represented only visible input-output behaviours enriched with costs that measure their efficiency. All other behaviour is abstracted away, which makes this model very suitable for security analysis. Finally, the game model for some language fragments admits finitary representation by using regular languages or CSP processes [10, 6], and has already been applied to automatic program verification. Here we present another application of algorithmic game semantics for automatically verifying security properties of programs.

Related work. The most common approach to ensure security properties of programs is by using security-type systems [14]. Here for every program component are defined security types, which contain information about their types and security levels. Programs that are well-typed under these type systems satisfy certain security properties. Type systems for enforcing non-interference of programs have been proposed by Volpano and Smith in [20], and subsequently they have been extended to detect also covert timing channels in [21, 2]. A drawback of this approach is its imprecision, since many secure programs are not typable and so are rejected. A more precise analysis of programs can be achieved by using semantics-based approaches [15].

2 Syntax and Operational Semantics

We will define a secure information flow analysis for Idealized Algol (IA), a small Algol-like language introduced by Reynolds [16] which has been used as a metalanguage in the denotational semantics community. It is a call-by-name λ-calculus extended with imperative features and locally-scoped variables. In order to be able to perform an automata-theoretic analysis of the language, we consider here its second-order recursion-free fragment (IA$_2$ for short). It contains finitary data types D: $\mathsf{int}_n = \{0, \dots, n-1\}$ and $\mathsf{bool} = \{tt, ff\}$, and first-order function types: $T ::= B \mid B \to T$, where B ranges over base types: expressions ($\mathsf{exp}D$), commands (com), and variables ($\mathsf{var}D$).

Syntax of the language is given by the following grammar:

$$\begin{aligned} M ::= & x \mid v \mid \mathsf{skip} \mid \mathsf{diverge} \mid M \,\mathsf{op}\, M \mid M;M \mid \mathsf{if}\, M \,\mathsf{then}\, M \,\mathsf{else}\, M \mid \mathsf{while}\, M \,\mathsf{do}\, M \\ & \mid M := M \mid !M \mid \mathsf{new}_D\, x := v \,\mathsf{in}\, M \mid \mathsf{mkvar}_D MM \mid \lambda x.M \mid MM \end{aligned}$$

where v ranges over constants of type D.

Typing judgements are of the form $\Gamma \vdash M : T$, where Γ is a type *context* consisting of a finite number of typed free identifiers. Typing rules of the language are standard [1], but the general application rule is broken up into the linear application and the contraction rule [1].

$$\frac{\Gamma \vdash M : B \to T \qquad \Delta \vdash N : B}{\Gamma, \Delta \vdash MN : T} \qquad\qquad \frac{\Gamma, x_1 : T, x_2 : T \vdash M : T'}{\Gamma, x : T \vdash M[x/x_1, x/x_2] : T'}$$

We use these two rules to have control over multiple occurrences of free identifiers in terms during typing.

Any input/output operation in a term is done through global variables, i.e. free identifiers of type $\mathsf{var}D$. So an input is read by de-referencing a global variable, while an output is written by an assignment to a global variable.

[1] $M[N/x]$ denotes the capture-free substitution of N for x in M.

$$
\boxed{\begin{array}{l}
\Gamma \vdash n_1\,\mathsf{op}\,n_2, \mathsf{s} \longrightarrow^{k_{op}} n, \mathsf{s},\ \text{where}\, n = n_1 \mathsf{op} n_2 \\
\Gamma \vdash \mathsf{skip};\ \mathsf{skip}, \mathsf{s} \longrightarrow^{k_{seq}} \mathsf{skip}, \mathsf{s}' \\
\Gamma \vdash \mathsf{if}\, tt\, \mathsf{then}\, M_1\, \mathsf{else} M_2, \mathsf{s} \longrightarrow^{k_{if}} M_1, \mathsf{s} \\
\Gamma \vdash \mathsf{if}\, f\!f\, \mathsf{then}\, M_1\, \mathsf{else} M_2, \mathsf{s} \longrightarrow^{k_{if}} M_2, \mathsf{s} \\
\Gamma \vdash x := v', \mathsf{s} \otimes (x \mapsto v) \longrightarrow^{k_{asg}} \mathsf{skip}, \mathsf{s} \otimes (x \mapsto v') \\
\Gamma \vdash !x, \mathsf{s} \otimes (x \mapsto v) \longrightarrow^{k_{der}} v, \mathsf{s} \otimes (x \mapsto v) \\
\Gamma \vdash (\lambda\, x.M)M', \mathsf{s} \longrightarrow^{k_{app}} M[M'/x], \mathsf{s} \\
\Gamma \vdash \mathsf{new}_D\, x := v\, \mathsf{in}\, \mathsf{skip}, \mathsf{s} \longrightarrow^{k_{new}} \mathsf{skip}, \mathsf{s}
\end{array}}
$$

Table 1: Basic Reduction Rules

The operational semantics is defined in terms of a small-step evaluation relation using a notion of an evaluation context [9]. A small-step evaluation (reduction) relation is of the form:

$$\Gamma \vdash M, \mathsf{s} \longrightarrow M', \mathsf{s}'$$

where Γ is a so-called var-context which contains only identifiers of type var$D$; s, s$'$ are Γ-states which assign data values to the variables in Γ; and M, M' are terms. The set of all Γ-states will be denoted by $St(\Gamma)$.

Evaluation contexts are contexts [2] containing a single hole which is used to identify the next sub-term to be evaluated (reduced). They are defined inductively by the following grammar:

$$E ::= [-] \mid EM \mid E; M \mid \mathsf{skip}; E \mid E\, \mathsf{op} M \mid v\, \mathsf{op} E \mid \mathsf{if}\, E\, \mathsf{then}\, M\, \mathsf{else} M \mid M := E \mid E := v \mid !E$$

The operational semantics is defined in two stages. First, a set of basic reduction rules are defined in Table 1. We assign different (non-negative) costs to each reduction rule, in order to denote how much computational time is needed for a reduction to complete. They are only descriptions of time and we can give them different interpretations describing how much real time they denote. Such an interpretation can be arbitrarily complex. So the semantics is parameterized on the interpretation of costs. Notice that we write $\mathsf{s} \otimes (x \mapsto v)$ to denote a $\{\Gamma, x\}$-state which properly extends s by mapping x to the value v.

We also have reduction rules for iteration, local variables, and mkvar$_D$ construct, which do not incur additional costs.

$$\Gamma \vdash \mathsf{while}\, b\, \mathsf{do} M, \mathsf{s} \longrightarrow \mathsf{if}\, b\, \mathsf{then}\, (M;\ \mathsf{while}\, b\, \mathsf{do} M)\, \mathsf{else}\, \mathsf{skip}, \mathsf{s}$$

$$\frac{\Gamma, y \vdash M[y/x], \mathsf{s} \otimes (y \mapsto v) \longrightarrow M', \mathsf{s}' \otimes (y \mapsto v')}{\Gamma \vdash \mathsf{new}_D\, x := v\, \mathsf{in} M, \mathsf{s} \longrightarrow \mathsf{new}_D\, x := v'\, \mathsf{in} M'[x/y], \mathsf{s}'}$$

$$\Gamma \vdash (\mathsf{mkvar}_D M_1 M_2) := v, \mathsf{s} \longrightarrow M_1 v, \mathsf{s} \qquad \Gamma \vdash !(\mathsf{mkvar}_D M_1 M_2), \mathsf{s} \longrightarrow M_2, \mathsf{s}$$

Next, the in-context reduction rules for arbitrary terms are defined as:

$$\frac{\Gamma \vdash M, \mathsf{s} \longrightarrow^{n} M', \mathsf{s}'}{\Gamma \vdash E[M], \mathsf{s} \longrightarrow^{n} E[M'], \mathsf{s}'}$$

The small-step evaluation relation is deterministic, since arbitrary term can be uniquely partitioned into an evaluation context and a sub-term, which is next to be reduced.

We define the reflexive and transitive closure of the small-step reduction relation as follows:

[2] A context $C[-]$ is a term with (several occurrences of) a hole in it, such that if $\Gamma \vdash M : T$ is a term of the same type as the hole then $C[M]$ is a well-typed closed term of type com, i.e. $\vdash C[M]$: com.

$$\frac{\Gamma \vdash M, \mathsf{s} \longrightarrow^{n} M', \mathsf{s}'}{\Gamma \vdash M, \mathsf{s} \rightsquigarrow^{n} M', \mathsf{s}'} \qquad \frac{\Gamma \vdash M, \mathsf{s} \rightsquigarrow^{n} M', \mathsf{s}' \qquad \Gamma \vdash M', \mathsf{s}' \rightsquigarrow^{n'} M'', \mathsf{s}''}{\Gamma \vdash M, \mathsf{s} \rightsquigarrow^{n+n'} M'', \mathsf{s}''}$$

Now a theory of operational improvement is defined [19]. Let $\Gamma \vdash M : \mathsf{com}$ be a term, where Γ is a var-context. We say that M *terminates in n steps* at state s, written $M, \mathsf{s} \Downarrow^{n}$, if $\Gamma \vdash M, \mathsf{s} \rightsquigarrow^{n} \mathsf{skip}, \mathsf{s}'$ for some state s'. If M is a closed term and $M, \emptyset \Downarrow^{n}$, then we write $M \Downarrow^{n}$. If $M \Downarrow^{n}$ and $n \leq n'$, we write $M \Downarrow^{\leq n'}$. We say that a term $\Gamma \vdash M : T$ may be *improved* by $\Gamma \vdash N : T$, denoted by $\Gamma \vdash M \gtrsim N$, if and only if for all contexts $C[-]$, if $C[M] \Downarrow^{n}$ then $C[N] \Downarrow^{\leq n}$. If two terms improve each other they are considered *improvment-equivalent*, denoted by $\Gamma \vdash M \approx N$.

Let $\Gamma, \Delta \vdash M : T$ be a term where Γ is a *var*-context and Δ is an arbitrary context. Such terms are called *split terms*, and we denote them as $\Gamma \mid \Delta \vdash M : T$. If Δ is empty, then these terms are called *semi-closed*. The semi-closed terms have only some global variables, and the operational semantics is defined only for them. We say that a semi-closed term $h : \mathsf{varD} \mid - \vdash M : \mathsf{com}$ does not have *timing leaks* if the initial value of the high-security variable h does not influence the number of reduction steps of M. More formally, we have:

Definition 1. *A semi-closed term* $h : \mathsf{varD} \mid - \vdash M : \mathsf{com}$ *has no* timing leaks *if*

$$\begin{aligned} \forall s_1, s_2 \in St(\{h\}). \quad & s_1(h) \neq s_2(h) \wedge \\ & h : \mathsf{varD} \vdash M, \mathsf{s}_1 \rightsquigarrow^{n_1} \mathsf{skip}, {\mathsf{s}_1}' \wedge h : \mathsf{varD} \vdash M, \mathsf{s}_2 \rightsquigarrow^{n_2} \mathsf{skip}, {\mathsf{s}_2}' \\ & \Rightarrow n_1 = n_2 \end{aligned} \tag{1}$$

Definition 2. *We say that a* split term $h : \mathsf{varD} \mid \Delta \vdash M : \mathsf{com}$ *does not have timing leaks, where* $\Delta = x_1 : T_1, \ldots, x_k : T_k$*, if for all closed terms* $\vdash N_1 : T_1, \ldots, \vdash N_k : T_k$*, we have that the term* $h : \mathsf{varD} \mid - \vdash M[N_1/x_1, \ldots, N_k/x_k] : \mathsf{com}$ *does not have timing leaks.*

The formula (1) can be replaced by an equivalent formula, where instead of two evaluations of the same term we can consider only one evaluation of the sequential composition of the given term with another its copy [3]. So sequential composition enables us to place these two evaluations one after the other. Let $h : \mathsf{varD} \vdash M : \mathsf{com}$ be a term, we define M' to be α-equivalent to $M[h'/h]$ where all bound variables are suitable renamed. The following can be shown: $h \vdash M, \mathsf{s}_1 \rightsquigarrow^{n} \mathsf{skip}, {\mathsf{s}_1}' \wedge h' \vdash M', \mathsf{s}_2 \rightsquigarrow^{n'} \mathsf{skip}, {\mathsf{s}_2}'$ iff $h, h' \vdash M;\ M', \mathsf{s}_1 \otimes \mathsf{s}_2 \rightsquigarrow^{n+n'} \mathsf{skip};\ \mathsf{skip}, {\mathsf{s}_1}' \otimes {\mathsf{s}_2}'$. In this way, we provide an alternative definition to formula (1) as follows. We say that a semi-closed term $h \mid - \vdash M : T$ has no *timing leaks* if

$$\begin{aligned} \forall s_1 \in St(\{h\}), s_2 \in St(\{h'\}). \quad & s_1(h) \neq s_2(h') \wedge \\ & h, h' \vdash M;\ M', \mathsf{s}_1 \otimes \mathsf{s}_2 \rightsquigarrow^{n_1} \mathsf{skip};\ M', {\mathsf{s}_1}' \otimes \mathsf{s}_2 \rightsquigarrow^{n_2} \mathsf{skip};\ \mathsf{skip}, {\mathsf{s}_1}' \otimes {\mathsf{s}_2}' \\ & \Rightarrow n_1 = n_2 \end{aligned} \tag{2}$$

3 Algorithmic Slot-Game Semantics

We now show how slot-game semantics for IA$_2$ can be represented algorithmically by regular-languages. In this approach, types are interpreted as games, which have two participants: the Player representing the term, and the Opponent representing its context. A game (arena) is defined by means of a set of moves, each being either a question move or an answer move. Each move represents an observable action that a term of a given type can perform. Apart from moves, another kind of action, called *token* (slot), is used to take account of quantitative aspects of terms. It represents a payment that a participant needs to pay in order to use a resource such as time. A computation is interpreted as a play-with-costs, which is given as a sequence of moves and token-actions played by two participants in turns.

We will work here with complete plays-with-costs which represent the observable effects along with incurred costs of a completed computation. Then a term is modelled by a strategy-with-costs, which is a set of complete plays-with-costs. In the regular-language representation of game semantics [10], types (arenas) are expressed as *alphabets of moves*, computations (plays-with-costs) as *words*, and terms (strategies-with-costs) as *regular-languages* over alphabets.

Each type T is interpreted by an alphabet of moves $\mathscr{A}_{[\![T]\!]}$, which can be partitioned into two subsets of *questions* $Q_{[\![T]\!]}$ and *answers* $A_{[\![T]\!]}$. For expressions, we have: $Q_{[\![\mathsf{exp}D]\!]} = \{q\}$ and $A_{[\![\mathsf{exp}D]\!]} = D$, i.e. there are a question move q to ask for the value of the expression and values from D are possible answers. For commands, we have: $Q_{[\![\mathsf{com}]\!]} = \{run\}$ and $A_{[\![\mathsf{com}]\!]} = \{done\}$, i.e. there are a question move *run* to initiate a command and an answer move *done* to signal successful termination of a command. For variables, we have: $Q_{[\![\mathsf{var}D]\!]} = \{read, write(a) \mid a \in D\}$ and $A_{[\![\mathsf{var}D]\!]} = D \cup \{ok\}$, i.e. there are moves for writing to the variable, $write(a)$, acknowledged by the move *ok*, and for reading from the variable, we have a question move *read*, and an answer to it can be any value from D. For function types, we have $\mathscr{A}_{[\![B_1^1 \to \ldots \to B_k^k \to B]\!]} = \sum_{1 \le i \le k} \mathscr{A}^i_{[\![B_i]\!]} + \mathscr{A}_{[\![B]\!]}$, where $+$ means a disjoint union of alphabets. We will use superscript tags to keep record from which type of the disjoint union each move comes from. We denote the token-action by $\circledS$. A sequence of n token-actions $\circledS$ will be written as $ⓝ$.

For any (β-normal) term we define a regular language specified by an *extended regular expression R*. Apart from the standard operations for generating regular expressions, we will use some more specific operations. We define composition of regular expressions R defined over alphabet $\mathscr{A}^1 + \mathscr{B}^2 + \{\circledS\}$ and S over $\mathscr{B}^2 + \mathscr{C}^3 + \{\circledS\}$ as follows:

$$R \,{}_{\mathscr{B}^2}^{\circ}\, S = \{w\big[s/a^2 \cdot b^2\big] \;\mid\; w \in S, a^2 \cdot s \cdot b^2 \in R\}$$

where R is a set of words of the form $a^2 \cdot s \cdot b^2$, such that a^2, $b^2 \in \mathscr{B}^2$ and s contains only letters from $\mathscr{A}^1$ and $\{\circledS\}$. Notice that the composition is defined over $\mathscr{A}^1 + \mathscr{C}^3 + \{\circledS\}$, and all letters of $\mathscr{B}^2$ are hidden. The shuffle operation $R \bowtie S$ generates the set of all possible interleavings from words of R and S, and the restriction operation $R\mid_{\mathscr{A}'}$ (R defined over $\mathscr{A}$ and $\mathscr{A}' \subseteq \mathscr{A}$) removes from words of R all letters from $\mathscr{A}'$.

If w, w' are words, m is a move, and R is a regular expression, define $m \cdot w \frown w' = m \cdot w' \cdot w$, and $R \frown w' = \{w \frown w' \mid w \in R\}$. Given a word with costs w defined over $\mathscr{A} + \{\circledS\}$, we define the underlying word of w as $w^\dagger = w\mid_{\{\circledS\}}$, and the cost of w as $w\mid_{\mathscr{A}} = ⓝ$, which we denote as $\mid w \mid = n$.

The regular expression for $\Gamma \vdash M : T$ is denoted $[\![\Gamma \vdash M : T]\!]$ and is defined over the alphabet $\mathscr{A}_{[\![\Gamma \vdash T]\!]} = \big(\sum_{x:T' \in \Gamma} \mathscr{A}^x_{[\![T']\!]}\big) + \mathscr{A}_{[\![T]\!]} + \{\circledS\}$. Every word in $[\![\Gamma \vdash M : T]\!]$ corresponds to a complete play-with-costs in the strategy-with-costs for $\Gamma \vdash M : T$.

Free identifiers $x \in \Gamma$ are interpreted by the copy-cat regular expressions, which contain all possible computations that terms of that type can have. Thus they provide the most general closure of an open term.

$$[\![\Gamma, x : B_1^{x,1} \to \ldots B_k^{x,k} \to B^x \vdash x : B_1^1 \to \ldots B_k^k \to B]\!] = \\ \sum_{q \in Q_{[\![B]\!]}} q \cdot q^x \cdot \big(\sum_{1 \le i \le k} \big(\sum_{q_1 \in Q_{[\![B_i]\!]}} q_1^{x,i} \cdot q_1^i \cdot \sum_{a_1 \in A_{[\![B_i]\!]}} a_1^i \cdot a_1^{x,i} \big)\big)^* \cdot \sum_{a \in A_{[\![B]\!]}} a^x \cdot a$$

When a first-order non-local function is called, it may evaluate any of its arguments, zero or more times, and then it can return any value from its result type as an answer. For example, the term $[\![\Gamma, x : \mathsf{exp}D^x \vdash x : \mathsf{exp}D]\!]$ is modelled by the regular expression: $q \cdot q^x \cdot \sum_{n \in D} n^x \cdot n$.

The linear application is defined as:

$$[\![\Gamma, \Delta \vdash M N : T]\!] = [\![\Delta \vdash N : B^1]\!] \,{}_{\mathscr{A}^1_{[\![B]\!]}}^{\circ}\, [\![\Gamma \vdash M : B^1 \to T]\!]$$

Since we work with terms in β-normal form, function application can occur only when the function term is a free identifier. In this case, the interpretation is the same as above except that we add the cost k_{app} corresponding to function application. Notice that k_{app} denotes certain number of $\textcircled{\$}$ units that are needed for a function application to take place. The contraction $[\![\Gamma, x : T^x \vdash M[x/x_1, x/x_2] : T']\!]$ is obtained from $[\![\Gamma, x_1 : T^{x_1}, x_2 : T^{x_2} \vdash M : T']\!]$, such that the moves associated with x_1 and x_2 are de-tagged so that they represent actions associated with x.

To represent local variables, we first need to define a (storage) 'cell' regular expression cell_v which imposes the good variable behaviour on the local variable. So cell_v responds to each *write*(n) with *ok*, and plays the most recently written value in response to *read*, or if no value has been written yet then answers the *read* with the initial value v. Then we have:

$$\begin{aligned}
&\mathsf{cell}_v = (read \cdot v)^* \cdot \big(\sum_{n \in D} write(n) \cdot ok \cdot (read \cdot n)^*\big)^* \\
&[\![\Gamma, x : \mathsf{var}D \vdash M]\!] \circ \mathsf{cell}_v^x = \big([\![\Gamma, x : \mathsf{var}D \vdash M]\!] \cap (\mathsf{cell}_v^x \bowtie (\mathscr{A}_{[\![\Gamma \vdash B]\!]} + \textcircled{\$})^*)\big) \mid_{\mathscr{A}^x_{[\![\mathsf{var}D]\!]}} \\
&[\![\Gamma \vdash \mathsf{new}_D\, x := v \,\mathsf{in}\, M]\!] = [\![\Gamma, x : \mathsf{var}D \vdash M]\!] \circ \mathsf{cell}_v^x \frown k_{var}
\end{aligned}$$

Note that all actions associated with x are hidden away in the model of new, since x is a local variable and so not visible outside of the term.

Language constants and constructs are interpreted as follows:

$$\begin{aligned}
&[\![\mathsf{v} : \mathsf{exp}D]\!] = \{q \cdot v\} \quad [\![\mathsf{skip} : \mathsf{com}]\!] = \{run \cdot done\} \quad [\![\mathsf{diverge} : \mathsf{com}]\!] = \emptyset \\
&[\![\mathsf{op} : \mathsf{exp}D^1 \times \mathsf{exp}D^2 \to \mathsf{exp}D']\!] = q \cdot k_{op} \cdot q^1 \cdot \textstyle\sum_{m \in D} m^1 \cdot q^2 \cdot \sum_{n \in D} n^2 \cdot (m\, op\, n) \\
&[\![; : \mathsf{com}^1 \to \mathsf{com}^2 \to \mathsf{com}]\!] = run \cdot run^1 \cdot done^1 \cdot k_{seq} \cdot run^2 \cdot done^2 \cdot done \\
&[\![\mathsf{if} : \mathsf{expbool}^1 \to \mathsf{com}^2 \to \mathsf{com}^3 \to \mathsf{com}]\!] = run \cdot k_{if} \cdot q^1 \cdot tt^1 \cdot run^2 \cdot done^2 \cdot done\, + \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad run \cdot k_{if} \cdot q^1 \cdot ff^1 \cdot run^3 \cdot done^3 \cdot done \\
&[\![\mathsf{while} : \mathsf{expbool}^1 \to \mathsf{com}^2 \to \mathsf{com}]\!] = run \cdot (k_{if} \cdot q^1 \cdot tt^1 \cdot run^2 \cdot done^2)^* \cdot k_{if} \cdot q^1 \cdot ff^1 \cdot done \\
&[\![:= : \mathsf{var}D^1 \to \mathsf{exp}D^2 \to \mathsf{com}]\!] = \textstyle\sum_{n \in D} run \cdot k_{asg} \cdot q^2 \cdot n^2 \cdot write(n)^1 \cdot ok^1 \cdot done \\
&[\![! : \mathsf{var}D^1 \to \mathsf{exp}D]\!] = \textstyle\sum_{n \in D} q \cdot k_{der} \cdot read^1 \cdot n^1 \cdot n
\end{aligned}$$

Although it is not important at what position in a word costs are placed, for simplicity we decide to attach them just after the initial move. The only exception is the rule for sequential composition (;), where the cost is placed between two arguments. The reason will be explained later on.

We now show how slot-games model relates to the operational semantics. First, we need to show how to represent the state explicitly in the model. A Γ-state s is interpreted as follows:

$$[\![\mathsf{s} : \mathsf{var}D_1^{x_1} \times \ldots \times \mathsf{var}D_k^{x_k}]\!] = \mathsf{cell}^{x_1}_{\mathsf{s}(x_1)} \bowtie \ldots \bowtie \mathsf{cell}^{x_k}_{\mathsf{s}(x_k)}$$

The regular expression $[\![s]\!]$ is defined over the alphabet $\mathscr{A}^{x_1}_{[\![\mathsf{var}D_1]\!]} + \ldots + \mathscr{A}^{x_k}_{[\![\mathsf{var}D_k]\!]}$, and words in $[\![s]\!]$ are such that projections onto x_i-component are the same as those of suitable initialized $\mathsf{cell}_{\mathsf{s}(x_i)}$ strategies. Note that $[\![s]\!]$ is a regular expression without costs. The interpretation of $\Gamma \vdash M : \mathsf{com}$ at state s is:

$$[\![\Gamma \vdash M]\!] \circ [\![\mathsf{s}]\!] = \big([\![\Gamma \vdash M]\!] \cap ([\![\mathsf{s}]\!] \bowtie (\mathscr{A}_{[\![\mathsf{com}]\!]} + \textcircled{\$})^*)\big) \mid_{\mathscr{A}_{[\![\Gamma]\!]}}$$

which is defined over the alphabet $\mathscr{A}_{[\![\mathsf{com}]\!]} + \{\textcircled{\$}\}$. The interpretation $[\![\Gamma \vdash M]\!] \circ [\![s]\!]$ can be studied more closely by considering words in which moves from $\mathscr{A}_{[\![\Gamma]\!]}$ are not hidden. Such words are called *interaction sequences*. For any interaction sequence $run \cdot t \cdot done \bowtie \textcircled{n}$ from $[\![\Gamma \vdash M]\!] \circ [\![\mathsf{s}]\!]$, where t is an even-length word over $\mathscr{A}_{[\![\Gamma]\!]}$, we say that it leaves the state s' if the last write moves in each x_i-component are such that x_i is set to the value $\mathsf{s}'(x_i)$. For example, let $\mathsf{s} = (x \mapsto 1, y \mapsto 2)$, then the following interaction: $run \cdot write(5)^y \cdot ok^y \cdot read^x \cdot 1^x \cdot done$ leaves the state $s' = (x \mapsto 1, y \mapsto 5)$. Any two-move word of the form: $run^{x_i} \cdot n^{x_i}$ or $write(n)^{x_i} \cdot ok^{x_i}$ will be referred to as *atomic state operation* of $\mathscr{A}_{[\![\Gamma]\!]}$. The following results are proved in [11] for the full ICA (IA plus parallel composition and semaphores), but they also hold for the restricted fragment of it.

Proposition 1. *If $\Gamma \vdash M : \{\mathsf{com}, \mathsf{expD}\}$ and $\Gamma \vdash M, s \longrightarrow^n M', s'$, then for each interaction sequence $i \cdot t$ from $[\![\Gamma \vdash M']\!] \circ [\![s']\!]$ (i is an initial move) there exists an interaction $i \cdot t_a \cdot t \frown ⓝ \in [\![\Gamma \vdash M]\!] \circ [\![s]\!]$ such that t_a is an empty word or an atomic state operation of $\mathscr{A}_{[\![\Gamma]\!]}$ which leaves the state s'.*

Proposition 2. *If $\Gamma \vdash M, \mathrm{s} \rightsquigarrow^n M', s'$ then $[\![\Gamma \vdash M']\!] \circ [\![s']\!] \bowtie ⓝ \subseteq [\![\Gamma \vdash M]\!] \circ [\![s]\!]$.*

Theorem 1 (Consistency)**.** *If $M, \mathrm{s} \Downarrow^n$ then $\exists w \in [\![\Gamma \vdash M]\!] \circ [\![s]\!]$ such that $\mid w \mid = n$ and $w^\dagger = run \cdot done$.*

Theorem 2 (Computational Adequacy)**.** *If $\exists w \in [\![\Gamma \vdash M]\!] \circ [\![s]\!]$ such that $\mid w \mid = n$ and $w^\dagger = run \cdot done$, then $M, \mathrm{s} \Downarrow^n$.*

We say that a regular expression R is improved by S, denoted as $R \gtrsim S$, if $\forall w \in R, \exists t \in S$, such that $w^\dagger = t^\dagger$ and $\mid w \mid \geq \mid t \mid$.

Theorem 3 (Full Abstraction)**.** $\Gamma \vdash M \gtrsim N$ *iff* $[\![\Gamma \vdash M]\!] \gtrsim [\![\Gamma \vdash N]\!]$.

This shows that the two theories of improvement based on operational and game semantics are identical.

4 Detecting Timing Leaks

In this section slot-game semantics is used to detect whether a term with a secret global variable h can leak information about the initial value of h through its timing behaviour.

For this purpose, we define a special command $\mathsf{skip}^{\#}$ which similarly as skip does nothing, but its slot-game semantics is: $[\![\mathsf{skip}^{\#}]\!] = \{run \cdot \# \cdot done\}$, where # is a new special action, called *delimiter*. Since we verify security of a term by running two copies of the same term one after the other, we will use the command $\mathsf{skip}^{\#}$ to specify the boundary between these two copies. In this way, we will be able to calculate running times of the two terms separately.

Theorem 4. *Let $h : \mathsf{varD} \mid - \vdash M : \mathsf{com}$ be a semi-closed term, and* [3]

$$R = [\![k : \mathsf{exp}D \vdash \mathsf{new}_D\, h{:=}k \,\mathsf{in}\, M;\ \mathsf{skip}^{\#};\ \mathsf{new}_D\, h'{:=}k \,\mathsf{in}\, M' : \mathsf{com}]\!] \tag{3}$$

Any word of R is of the form $w = w_1 \cdot \# \cdot w_2$ such that $\mid w_1 \mid = \mid w_2 \mid$ iff M has no timing leaks, i.e. the fact (2) holds.

Proof. Suppose that any word $w \in R$ is of the form $w = w_1 \cdot \# \cdot w_2$ such that $\mid w_1 \mid = \mid w_2 \mid$. Let us analyse the regular expression R defined in (3). We have:

$$\begin{aligned} R = \{ & run \cdot k_{var} \cdot q^k \cdot v^k \cdot w_1 \cdot k_{seq} \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot v'^k \cdot w_2 \cdot done \mid \\ & run \cdot w_1 \cdot done \in [\![h \vdash M]\!] \circ \mathsf{cell}_v^h, run \cdot w_2 \cdot done \in [\![h' \vdash M']\!] \circ \mathsf{cell}_{v'}^{h'} \} \end{aligned}$$

for arbitrary values $v, v' \in D$. In order to ensure that one k_{seq} unit of cost occurs before and after the delimiter action, k_{seq} is played between two arguments of the sequential composition as was described in Section 3. Given that $run \cdot w_1 \cdot done \in [\![h \vdash M]\!] \circ \mathsf{cell}_v^h$ and $run \cdot w_2 \cdot done \in [\![h' \vdash M']\!] \circ \mathsf{cell}_{v'}^{h'}$ for any

[3]The free identifier k in (3) is used to initialize the variables h and h' to arbitrary values from D.

$v, v' \in D$, by Computational Adequacy we have that $M, (h \mapsto v) \Downarrow^{|w_1|}$ and $M', (h' \mapsto v') \Downarrow^{|w_2|}$. Since $| w_1 |=| w_2 |$, it follows that the fact (2) holds.

Let us consider the opposite direction. Suppose that the fact (2) holds. The term in (3) is α-equivalent to $k \vdash \mathsf{new}_D\, h := k\, \mathsf{in}\, \mathsf{new}_D\, h' := k\, \mathsf{in}\, M;\ \mathsf{skip}^{\#};\ M'$. Consider $[\![h, h' \vdash M;\ \mathsf{skip}^{\#};\ M']\!] \circ [\![(h \mapsto v) \otimes (h' \mapsto v')]\!]$, where $v, v' \in D$. By Consistency, we have that $\exists w_1 \in [\![h, h' \vdash M]\!] \circ [\![(h \mapsto v) \otimes (h' \mapsto v')]\!]$ such that $| w_1 |= n$ and w_1 leaves the state $(h \mapsto v_1) \otimes (h' \mapsto v')$, and $\exists w_2 \in [\![h, h' \vdash M']\!] \circ [\![(h \mapsto v_1) \otimes (h' \mapsto v')]\!]$ such that $| w_2 |= n$ and w_2 leaves the state $(h \mapsto v_1) \otimes (h' \mapsto v_1')$. Any word $w \in R$ is obtained from w_1 and w_2 as above ($| w_1 |=| w_2 |$), and so satisfies the requirements of the theorem. $\square$

We can detect timing leaks from a semi-closed term by verifying that all words in the model in (3) are in the required form. To do this, we restrict our attention only to the costs of words in R.

Example 1. Consider the term:

$$h : \mathsf{var\,int}_2 \vdash \mathsf{if}\,(!h > 0)\,\mathsf{then}\, h := !h + 1;\ \mathsf{else\,skip} : \mathsf{com}$$

The slot-game semantics of this term extended as in (3) is:

$$run \cdot k_{var} \cdot q^k \cdot \big(0^k \cdot k_{seq} \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot (0^k \cdot done + 1^k \cdot k_{der} \cdot k_+ \cdot done) \\ + 1^k \cdot k_{seq} \cdot k_{der} \cdot k_+ \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot (0^k \cdot done + 1^k \cdot k_{der} \cdot k_+ \cdot done)\big)$$

This model includes all possible observable interactions of the term with its environment, which contains only the identifier k, along with the costs measuring its running time. Note that the first value for k read from the environment is used to initialize h, while the second value for k is used to initialize h'.

By inspecting we can see that the model contains the word:

$$run \cdot k_{var} \cdot q^k \cdot 0^k \cdot k_{seq} \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot 1^k \cdot k_{der} \cdot k_+ \cdot done$$

which is not of the required form. This word (play) corresponds to two computations of the given term where initial values of h are 0 and 1 respectively, such that the cost of the second computation has additional $k_{der} + k_+$ units more than the first one. $\square$

We now show how to detect timing leaks of a split (open) term $h : \mathsf{varD} \mid \Delta \vdash M : \mathsf{com}$, where $\Delta = x_1 : T_1, \ldots, x_k : T_k$. To do this, we need to check timing efficiency of the following model:

$$[\![h, h' : \mathsf{varD} \vdash M[N_1/x_1, \ldots, N_k/x_k];\ \mathsf{skip}^{\#};\ M'[N_1/x_1, \ldots, N_k/x_k]]\!] \qquad (4)$$

at state $(h \mapsto v, h' \mapsto v')$, for any closed terms $\vdash N_1 : T_1, \ldots, \vdash N_k : T_k$, and for any values $v, v' \in D$. As we have shown slot-game semantics respects theory of operational improvement, so we will need to examine whether all its complete plays-with-costs s are of the form $s_1 \cdot \# \cdot s_2$ where $| s_1 |=| s_2 |$. However, the model in (4) can not be represented as a regular language, so it can not be used directly for detecting timing leaks.

Let us consider more closely the slot-game model in (4). Terms M and M' are run in the same context Δ, which means that each occurrence of a free identifier x_i from Δ behaves uniformly in both M and M'. So any complete play-with-costs of the model in (4) will be a concatenation of complete plays-with-costs from models for M and M' with additional constraints that behaviours of free identifiers from Δ are the same in M and M'. If these additional constraints are removed from the above model, then we generate a model which is an over-approximation of it and where free identifiers from Δ can behave freely in M and M'. Thus we obtain:

$$[\![h, h' : \mathsf{varD} \vdash M[N_1/x_1, \ldots, N_k/x_k];\ \mathsf{skip}^{\#};\ M'[N_1/x_1, \ldots, N_k/x_k]]\!] \subseteq \\ [\![h, h' : \mathsf{varD} \vdash M;\ \mathsf{skip}^{\#};\ M'[N_1/x_1, \ldots, N_k/x_k]]\!]$$

If $\vdash N_1 : T_1, \ldots, \vdash N_k : T_k$ are arbitrary closed terms, then they are interpreted by identity (copy-cat) strategies corresponding to their types, and so we have:

$$[\![h,h' : \mathsf{varD} \vdash M; \mathsf{skip}^{\#}; M'[N_1/x_1,\ldots,N_k/x_k]]\!] = [\![h,h' : \mathsf{varD}, \Delta \vdash M; \mathsf{skip}^{\#}; M']\!]$$

This model is a regular language and we can use it to detect timing leaks.

Theorem 5. *Let $h : \mathsf{varD} \mid \Delta \vdash M : \mathsf{com}$ be a split (open) term, where $\Delta = x_1 : T_1, \ldots, x_k : T_k$, and*

$$S = [\![k : \mathsf{exp}D, \Delta \vdash \mathsf{new}_D\, h := k\, \mathsf{in}\, M;\ \mathsf{skip}^{\#};\ \mathsf{new}_D\, h' := k\, \mathsf{in}\, M' : \mathsf{com}]\!] \tag{5}$$

If any word of S is of the form $w = w_1 \cdot \# \cdot w_2$ such that $\mid w_1 \mid = \mid w_2 \mid$, Then $h : \mathsf{varD} \mid \Delta \vdash M$ has no timing leaks.

Note that the opposite direction in the above result does not hold. That is, if there exists a word from S which is not of the required form then it does not follow that M has timing leaks, since the found word (play) may be spurious introduced due to over-approximation in the model in (5), and so it may be not present in the model in (4).

Example 2. Consider the term:

$$h : \mathsf{varint}_2, f : \mathsf{expint}_2{}^{f,1} \to \mathsf{com}^f \vdash f(!h) : \mathsf{com}$$

where f is a non-local call-by-name function.

The slot-game model for this term is as follows:

$$run \cdot k_{app} \cdot run^f \cdot (q^{f,1} \cdot k_{der} \cdot read^h \cdot (0^h \cdot 0^{f,1} + 1^h \cdot 1^{f,1}))^* \cdot done^f \cdot done$$

Once f is called, it may evaluate its argument, zero or more times, and then it terminates successfully. Notice that moves tagged with f represent the actions of calling and returning from the function f, while moves tagged with $f,1$ indicate actions of the first argument of f.

If we generate the slot-game model of this term extended as in (5), we obtain a word which is not in the required form:

$$run \cdot k_{var} \cdot q^k \cdot 0^k \cdot k_{app} \cdot run^f \cdot q^{f,1} \cdot k_{der} \cdot 0^{f,1} \cdot done^f \cdot k_{seq} \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot 1^k \cdot k_{app} \cdot run^f \cdot done^f \cdot done$$

This word corresponds to two computations of the term, where the first one calls f which evaluates its argument once, and the second calls f which does not evaluate its argument at all. The first computation will have the cost of k_{der} units more that the second one. However, this is a spurious counter-example, since f does not behave uniformly in the two computations, i.e. it calls its argument in the first but not in the second computation. □

To handle this problem, we can generate an under-approximation of the model given in (4) which can be represented as a regular language. Let $h : \mathsf{varD} \mid \Delta \vdash M$ be a term derived without using the contraction rule for any identifier from Δ. Consider the following model:

$$\begin{aligned}[\![h,h' : \mathsf{varD} \mid \Delta \vdash M;\ \mathsf{skip}^{\#};\ M']\!]^m = {} & [\![h,h' : \mathsf{varD} \mid \Delta \vdash M;\ \mathsf{skip}^{\#};\ M']\!] \cap \\ & (\mathsf{delta}^{x_1}_{T_1,m} \bowtie \ldots \bowtie \mathsf{delta}^{x_k}_{T_k,m} \bowtie (\mathscr{A}_{[\![h,h':\mathsf{varD}\vdash\mathsf{com}]\!]} + \textcircled{\$})^*)\end{aligned} \tag{6}$$

where $m \geq 0$ denotes the number of times that free identifiers of function types may evaluate its arguments at most. The regular expressions $\mathsf{delta}_{T,m}$ are used to repeat zero or once an arbitrary behaviour for terms of type T, and are defined as follows.

$$\mathsf{delta}_{\mathsf{exp}D,0} = q \cdot \textstyle\sum_{n \in D} n \cdot (\varepsilon + q \cdot n) \qquad \mathsf{delta}_{\mathsf{com},0} = run \cdot done \cdot (\varepsilon + run \cdot done)$$
$$\mathsf{delta}_{\mathsf{var}D,0} = (read \cdot \textstyle\sum_{n \in D} n \cdot (\varepsilon + read \cdot n))\ +\ (\textstyle\sum_{n \in D} write(n) \cdot ok \cdot (\varepsilon + write(n) \cdot ok))$$

If T is a first-order function type, then $\mathsf{delta}_{T,m}$ will be a regular language only when the number of times its arguments can be evaluated is limited. For example, we have that:

$$\mathsf{delta}_{\mathsf{com}^1 \to \mathsf{com},m} = run \cdot \sum_{r=0}^{m} (run^1 \cdot done^1)^r \cdot done \cdot (\varepsilon + run \cdot (run^1 \cdot done^1)^r \cdot done)$$

If T is a function type with k arguments, then we have to remember not only how many times arguments are evaluated in the first call, but also the exact order in which arguments are evaluated.

Notice that we allow an arbitrary behavior of type T to be repeated zero or once in $\mathsf{delta}_{T,m}$, since it is possible that depending on the current value of h an occurrence of a free identifier from Δ to be run in M but not in M', or vice versa. For example, consider the term:

$$h : \mathsf{var}\,\mathsf{int}_2 \mid x, y : \mathsf{exp}\,\mathsf{int}_2 \vdash \mathsf{new}_{int_2}\, z := 0\, \mathsf{in}\, \mathsf{if}\, (!h > 0)\, \mathsf{then}\, z := x\, \mathsf{else}\, z := y + 1$$

This term has timing leaks, and the corresponding counter-example contains only one interaction with x occurred in a computation, and one interaction with y occurred in the other computation. This counter-example will be included in the model in (6), only if $\mathsf{delta}_{T,m}$ is defined as above.

Let $h : \mathsf{varD} \mid \Delta \vdash M$ be an arbitrary term where identifiers from Δ may occur more than once in M. Let $h : \mathsf{varD} \mid \Delta_1 \vdash M_1$ be derived without using the contraction for Δ_1, such that $h : \mathsf{varD} \mid \Delta \vdash M$ is obtained from it by applying one or more times the contraction rule for identifiers from Δ. Then $[\![h, h' : \mathsf{varD} \mid \Delta \vdash M;\ \mathsf{skip}^{\#};\ M']\!]^m$ is obtained by first computing $[\![h, h' : \mathsf{varD} \mid \Delta_1 \vdash M_1;\ \mathsf{skip}^{\#};\ M_1']\!]^m$ as defined in (6), and then by suitable tagging all moves associated with several occurrences of the same identifier from Δ as described in the interpretation of contraction. We have that:

$$[\![h, h' : \mathsf{varD}, \Delta \vdash M;\ \mathsf{skip}^{\#};\ M']\!]^m \subseteq [\![h, h' : \mathsf{varD} \vdash M[N_1/x_1, \ldots, N_k/x_k];\mathsf{skip}^{\#}; M'[N_1/x_1, \ldots, N_k/x_k]]\!]$$

for any $m \geq 0$ and arbitrary closed terms $\vdash N_1 : T_1, \ldots, \vdash N_k : T_k$.

In the case that Δ contains only identifiers of base types B which do not occur in any while-subterm of M, then in the above formula the subset relation becomes the equality for $m = 0$. If a free identifier occurs in a while-subterm of M, then it can be called arbitrary many times in M, and so we cannot reproduce its behaviour in M'.

Theorem 6. *Let $h : \mathsf{varD} \mid \Delta \vdash M$ be a split (open) term, where $\Delta = x_1 : T_1, \ldots, x_k : T_k$, and*

$$T = \quad [\![k : \mathsf{exp}D, \Delta \vdash \mathsf{new}_D\, h := k\, \mathsf{in}\, M;\ \mathsf{skip}^{\#};\ \mathsf{new}_D\, h' := k\, \mathsf{in}\, M' : \mathsf{com}]\!]^m \qquad (7)$$

(i) Let Δ contains only identifiers of base types B, which do not occur in any while*-subterm of M. Any word of T (where $m = 0$) is of the form $w_1 \cdot \# \cdot w_2$ such that $\mid w_1 \mid = \mid w_2 \mid$ iff M has no timing leaks.*

(ii) Let Δ be an arbitrary context. If there exists a word $w = w_1 \cdot \# \cdot w_2 \in T$ such that $\mid w_1 \mid \neq \mid w_2 \mid$, Then M does have timing leaks.

Note that if a counter-example witnessing a timing leakage is found, then it provides a specific context Δ, i.e. a concrete definition of identifiers from Δ, for which the given open term have timing leaks.

5 Detecting Timing-Aware Non-interference

The slot-game semantics model contains enough information to check the non-interference property of terms along with timing leaks. The method for verifying the non-interference property is analogous to the one described in [8], where we use the standard game semantics model. As slot-game semantics can be considered as the standard game semantics augmented with the information about quantitative assessment of time usage, we can use it as underlying model for detection of both non-interference property and timing leaks, which we call *timing-aware non-interference*.

In what follows, we show how to verify timing-aware non-interference property for closed terms. In the case of open terms, the method can be extended straightforwardly by following the same ideas for handling open terms described in Section 4.

Let $l : \mathsf{varD}, h : \mathsf{varD}' \vdash M : \mathsf{com}$ be a term where l and h represent low- and high-security global variables respectively. We define $\Gamma_1 = l : \mathsf{var}D, h : \mathsf{var}D'$, $\Gamma_1' = l' : \mathsf{var}D, h' : \mathsf{var}D'$, and M' is α-equivalent to $M[l'/l, h'/h]$ where all bound variables are suitable renamed. We say that $\Gamma_1 \mid - \vdash M : \mathsf{com}$ satisfies *timing-aware non-interference* if

$$\begin{aligned}\forall s_1 \in St(\Gamma_1), s_2 \in St(\Gamma_1'). \quad & s_1(l) = s_2(l') \wedge s_1(h) \neq s_2(h') \wedge \\ & \Gamma_1 \vdash M;\, M', \mathsf{s}_1 \otimes \mathsf{s}_2 \leadsto^{n_1} \mathsf{skip};\, M', \mathsf{s}_1{}' \otimes \mathsf{s}_2 \leadsto^{n_2} \mathsf{skip};\, \mathsf{skip}, \mathsf{s}_1{}' \otimes \mathsf{s}_2{}' \\ & \Rightarrow s_1'(l) = s_2'(l') \;\wedge\; n_1 = n_2\end{aligned}$$

Suppose that abort is a special free identifier of type com^{abort} in Γ. We say that a term $\Gamma \vdash M$ is *safe* iff $\Gamma \vdash M[\mathsf{skip}/\mathsf{abort}] \sqsubseteq\!\!\!\sim M[\mathsf{diverge}/\mathsf{abort}]$ [4]; otherwise we say that a term is *unsafe*. It has been shown in [5] that a term $\Gamma \vdash M$ is safe iff $[\![\Gamma \vdash M]\!]$ does not contain any play with moves from $\mathscr{A}^{abort}_{[\![\mathsf{com}]\!]}$, which we call unsafe plays. For example, $[\![\mathsf{abort} : \mathsf{com}^{\mathsf{abort}} \vdash \mathsf{skip}\,;\ \mathsf{abort} : \mathsf{com}]\!] = run \cdot run^{abort} \cdot done^{abort} \cdot done$, so this term is unsafe.

By using Theorem 4 from Section 4 and the corresponding result for closed terms from [8], it is easy to show the following result.

$$\begin{aligned}L = [\![k : \mathsf{exp}D, k' : \mathsf{exp}D', \mathsf{abort} : \mathsf{com} \vdash\ & \mathsf{new}_D\, l \mathop{:=} k \,\mathsf{in}\, \mathsf{new}_{D'}\, h \mathop{:=} k' \,\mathsf{in} \\ & \mathsf{new}_D\, l' \mathop{:=} !l \,\mathsf{in}\, \mathsf{new}_{D'}\, h' \mathop{:=} k' \,\mathsf{in} \\ & \mathsf{skip}^{\#};\, M;\, \mathsf{skip}^{\#};\, M';\, \mathsf{skip}^{\#};\, \mathsf{if}\,(!l \neq !l')\,\mathsf{then}\,\mathsf{abort} : \mathsf{com}]\!]\end{aligned} \tag{8}$$

The regular expression L contains no unsafe word (plays) and all its words are of the form $w = w_1 \cdot \# \cdot w_2 \cdot \# \cdot w_3 \cdot \# \cdot w_4$ such that $\mid w_2 \mid = \mid w_3 \mid$ iff M satisfies the timing-aware non-interference property.

Notice that the free identifier k in (8) is used to initialize the variables l and l' to any value from D which is the same for both l and l', while k' is used to initialize h and h' to any values from D'. The last if command is used to check values of l and l' in the final state after evaluating the term in (8). If their values are different, then abort is run.

6 Application

We can also represent slot-game semantics model of IA_2 by using the CSP process algebra. This can be done by extending the CSP representation of standard game semantics given in [6], by attaching the costs corresponding to each translation rule. In the same way, we have adapted the verification tool in [6] to automatically convert an IA_2 term into a CSP process [17] that represents its slot-game semantics. The CSP process outputted by our tool is defined by a script in machine readable CSP which can be analyzed by the FDR tool. It represents a model checker for the CSP process algebra, and in this way a range of properties of terms can be verified by calls to it.

[4] $\sqsubseteq\!\!\!\sim$ denotes observational approximation of terms (see [1])

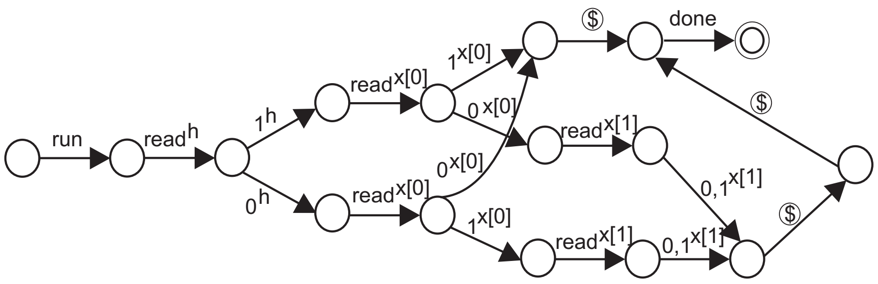


Figure 1: Slot-game semantics for the linear search with k=2

In the input syntax of terms, we use simple type annotations to indicate what finite sets of integers will be used to model free identifiers and local variables of type integer. An operation between values of types int_{n_1} and int_{n_2} produces a value of type $\mathsf{int}_{max\{n_1,n_2\}}$. The operation is performed modulo $max\{n_1,n_2\}$.

In order to use this tool to check for timing leaks in terms, we need to encode the required property as a CSP process (i.e. regular-language). This can be done only if we know the cost of the worst plays (paths) in the model of a given term. We can calculate the worst-case cost of a term by generating its model, and then by counting the number of tokens in its plays. The property we want to check will be: $\sum_{i=0}^{n} ⓘ \cdot \# \cdot ⓘ$, where n denotes the worst-case cost of a term.

To demonstrate practicality of this approach for automated verification, we consider the following implementation of the linear-search algorithm.

```
$h : \mathsf{varint}_2, x[k] : \mathsf{varint}_2 \vdash$
    $\mathsf{new}_{int_2}\, a[k] := 0\, \mathsf{in}$
    $\mathsf{new}_{int_{k+1}}\, i := 0\, \mathsf{in}$
    $\mathsf{while}\,(i < k)\, \mathsf{do}\, \{a[i] := !x[i];\;\; i := !i + 1;\; \}$
    $\mathsf{new}_{int_2}\, y := !h\, \mathsf{in}$
    $\mathsf{new}_{bool}\, present := ff\, \mathsf{in}$
    $\mathsf{while}\,(i < k \,\&\&\, \neg present)\, \mathsf{do}\, \{$
        $\mathsf{if}\,(compare(!a[i], !y))\, \mathsf{then}\, present := tt;$
        $i := !i + 1;$
    $\} : \mathsf{com}$
```

The meta variable $k > 0$ represents the array size. The term copies the input array x into a local array a, and the input value of h into a local variable y. The linear-search algorithm is then used to find whether the value stored in y is in the local array. At the moment when the value is found in the array, the term terminates successfully. Note that arrays are introduced in the model as syntactic sugar by using existing term formers. So an array $x[k]$ is represented as a set of k distinct variables $x[0], \ldots, x[k-1]$ (see [6, 10] for details).

Suppose that we are only interested in measuring the efficiency of the term relative to the number of *compare* operations. It is defined as follows $compare : \mathsf{expint}_2 \to \mathsf{expint}_2 \to \mathsf{expbool}$, and its semantics compares for equality the values of two arguments with cost Ⓢ:

$$[\![compare : \mathsf{expint}_2^1 \to \mathsf{expint}_2^2 \to \mathsf{expbool}]\!] = q \cdot Ⓢ \cdot q^1 \cdot (\sum_{m \neq n} m^1 \cdot q^2 \cdot n^2 \cdot ff) + (\sum_{m=n} m^1 \cdot q^2 \cdot n^2 \cdot tt)$$

where $m, n \in \{0, 1\}$. We assume that the costs of all other operations are relatively negligible (e.g. $k_{var} = k_{der} = \ldots = 0$).

We show the model for this term with $k = 2$ in Fig. 1. The worst-case cost of this term is equal to the array's size k, which occurs when the search fails or the value of h is compared with all elements of the array. We can perform a security analysis for this term by considering the model extended as in (7), where $m = 0$. We obtain that this term has timing leaks, with a counter-example corresponding to two computations, such that initial values of h are different, and the search succeeds in the one after only one iteration of while and fails in the other. For example, this will happen when all values in the array x are 0's, and the value of h is 0 in the first computation and 1 in the second one.

We can also automatically analyse in an analogous way terms where the array size k is much larger. Also the set of data that can be stored into the global variable h and array x can be larger than $\{0,1\}$. In these cases we will obtain models with much bigger number of states, but they still can be automatically analysed by calls to the FDR tool.

7 Conclusion

In this paper we have described how game semantics can be used for verifying security properties of open sequential programs, such as timing leaks and non-interference. This approach can be extended to terms with infinite data types, such as integers, by using some of the existing methods and tools based on game semantics for verifying such terms. Counter-example guided abstraction refinement procedure (ARP) [5] and symbolic representation of game semantics model [7] are two methods which can be used for this aim. The technical apparatus introduced here applies not only to time as a resource but to any other observable resource, such as power or heating of the processor. They can all be modeled in the framework of slot games and checked for information leaks.

We have focussed here on analysing the IA language, but we can easily extend this approach to any other language for which game semantics exists. Since fully abstract game semantics was also defined for probabilistic [4], concurrent [12], and programs with exceptions [1], it will be interesting to extend this approach to such programs.

References

[1] Abramsky, S., and McCusker, G: Game Semantics. In Proceedings of *the 1997 Marktoberdorf Summer School: Computational Logic* , (1998), 1–56. Springer.

[2] Agat, J: Transforming out Timing Leaks. In: Wegman, M.N., Reps, T.W. (eds.) POPL 2000. ACM, pp. 40–53. ACM, New York (2000), doi:10.1145/325694.325702.

[3] Barthe, G., D'Argenio, P.R., Rezk, T: Secure information flow by self-composition. In: IEEE CSFW 2004. pp. 100–114. IEEE Computer Society Press, (2004), doi:10.1109/CSFW.2004.17.

[4] V. Danos and R. Harmer. Probabilistic Game Semantics. In Proceedings of LICS 2000. 204–213. IEEE Computer Society Press, Los Alamitos (2000), doi:10.1109/LICS.2000.855770.

[5] Dimovski, A., Ghica, D. R., Lazić, R. Data-Abstraction Refinement: A Game Semantic Approach. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS vol. 3672, pp. 102–117. Springer, Heidelberg (2005), doi:10.1007/11547662_9.

[6] Dimovski, A., Lazić, R: Compositional Software Verification Based on Game Semantics and Process Algebras. In *Int. Journal on STTT* **9(1)**, pp. 37–51, (2007), doi:10.1007/s10009-006-0005-y.

[7] Dimovski, A: Symbolic Representation of Algorithmic Game Semantics. In: Faella, M., Murano, A. (eds.) GandALF 2012. EPTCS vol. 96, pp. 99–112. Open Publishing Association, (2012), doi:10.4204/EPTCS.96.8.

[8] Dimovski, A: Ensuring Secure Non-interference of Programs by Game Semantics. Submitted for publication.

[9] Cartwright, R., Curien, P. L., and Felleisen, M: Fully abstract semantics for observably sequential languages. In *Information and Computation* **111(2)**, pp. 297–401, (1994), doi:10.1006/inco.1994.1047.

[10] Ghica, D. R., McCusker, G: The Regular-Language Semantics of Second-order Idealized Algol. Theoretical Computer Science **309** (1–3), pp. 469–502, (2003), doi:10.1016/S0304-3975(03)00315-3.

[11] Ghica, D. R. Slot Games: a quantitative model of computation. In Palsberg, J., Abadi, M. (eds.) POPL 2005. ACM, pp. 85–97. ACM Press, New York (1998), doi:10.1145/1040305.1040313.

[12] Ghica, D. R., Murawski, A: Compositional Model Extraction for Higher-Order Concurrent Programs. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS vol. 3920, pp. 303–317. Springer, Heidelberg (2006), doi:10.1007/11691372_20.

[13] Goguen, J., Meseguer, J: Security polices and security models. In: IEEE Symp. on Security and Privacy 1982. pp. 11–20. IEEE Computer Society Press, (1982).

[14] Heintze, N., Riecke, J.G: The SLam calculus: programming with secrecy and integrity. In: MacQueen, D.B., Cardelli, L. (eds.) POPL 1998. ACM, pp. 365–377. ACM, New York (1998), doi:10.1145/268946.268976.

[15] Joshi, R., and Leino, K.R.M: A semantic approach to secure information flow. In *Science of Computer Programming* **37**, pp. 113–138, (2000), doi:10.1016/S0167-6423(99)00024-6.

[16] Reynolds, J. C: The essence of Algol. In: O'Hearn, P.W, and Tennent, R.D. (eds), *Algol-like languages*. (Birkhaüser, 1997).

[17] Roscoe, W. A: *Theory and Practice of Concurrency*. Prentice-Hall, 1998.

[18] Sabelfeld, A., and Myers, A.C: Language-based information-flow security. In IEEE *Journal on Selected Areas in Communications* **21(1)**, (2003), 5–19, doi:10.1109/JSAC.2002.806121.

[19] Sands, D: *Improvement Theory and its Applications*. Cambridge University Press, 1998.

[20] Volpano, D., Smith, G., and Irvine, C: A sound type system for secure flow analysis. In *Journal of Computer Security* **4(2/3)**, (1996), 167–188, doi:10.3233/JCS-1996-42-304.

[21] Volpano, D., Smith, G: Eliminating covert flows with minimum typings. In: IEEE *Computer Security Foundations Workshop (CSFW)*, 1997, 156–169. IEEE Computer Society Press, (1997), doi:10.1109/CSFW.1997.596807.

Social Network Games with Obligatory Product Selection

Krzysztof R. Apt
Centre for Mathematics and Computer Science (CWI),
ILLC, University of Amsterdam, The Netherlands
k.r.apt@cwi.nl

Sunil Simon
Centre for Mathematics and Computer Science (CWI)
s.e.simon@cwi.nl

Recently, we introduced in [1] a model for product adoption in social networks with multiple products, where the agents, influenced by their neighbours, can adopt one out of several alternatives (products). To analyze these networks we introduce social network games in which product adoption is obligatory.

We show that when the underlying graph is a simple cycle, there is a polynomial time algorithm allowing us to determine whether the game has a Nash equilibrium. In contrast, in the arbitrary case this problem is NP-complete. We also show that the problem of determining whether the game is weakly acyclic is co-NP hard.

Using these games we analyze various types of paradoxes that can arise in the considered networks. One of them corresponds to the well-known Braess paradox in congestion games. In particular, we show that social networks exist with the property that by adding an additional product to a specific node, the choices of the nodes will unavoidably evolve in such a way that everybody is strictly worse off.

1 Introduction

Social networks became a huge interdisciplinary research area with important links to sociology, economics, epidemiology, computer science, and mathematics. A flurry of numerous articles, notably the influential [11], and books, e.g., [7, 3], helped to delineate better this area. It deals with many diverse topics such as epidemics, spread of certain patterns of social behaviour, effects of advertising, and emergence of 'bubbles' in financial markets.

Recently, we introduced in [1] *social networks with multiple products*, in which the agents (players), influenced by their neighbours, can adopt one out of several alternatives (products). To study the situation when the product adoption is obligatory we introduce here social network games in which product adoption is obligatory. An example of a studied situation is when a group of people chooses an obligatory 'product', for instance, an operating system or a mobile phone provider, by taking into account the choice of their friends. The resulting games exhibit the following ***join the crowd*** property:

> the payoff of each player weakly increases when more players choose his strategy.

that we define more precisely in Subsection 2.3.

The considered games are a modification of the strategic games that we recently introduced in [14] and more fully in [15], in which the product adoption was optional. The insistence on product selection leads to a different analysis and different results than the ones reported there. In particular, Nash equilibria need not exist already in the case when the underlying graph is a simple cycle. We show that one can determine in polynomial time whether for such social networks a Nash equilibrium exists. We prove that for arbitrary networks, determining whether a Nash equilibrium exists is NP-complete. We also show that for arbitrary networks and for networks whose underlying graph has no source nodes, determining whether the game is weakly acyclic is co-NP hard.

The considered social networks allow us to analyze various paradoxes that were identified in the literature. One example is the *paradox of choice* first formulated in [13]. It has been summarised in [6, page 38] as follows:

> The more options one has, the more possibilities for experiencing conflict arise, and the more difficult it becomes to compare the options. There is a point where more options, products, and choices hurt both seller and consumer.

The point is that consumers choices depend on their friends' and acquaintances' preferences.

Another example is a 'bubble' in a financial market, where a decision of a trader to switch to some new financial product triggers a sequence of transactions, as a result of which all traders involved become worse off.

Such paradoxes are similar to the renowned Braess paradox which states that in some road networks the travel time can actually increase when new roads are added, see, e.g., [12, pages 464-465] and a 'dual' version of Braess paradox that concerns the removal of road segments, studied in [4, 5]. Both paradoxes were studied by means of congestion games. However, in contrast to congestion games, Nash equilibria do not need to exist in the games we consider here. Consequently, one needs to rely on different arguments. Moreover, there are now two new types of paradoxes that correspond to the situations when an addition, respectively, removal, of a product can lead to a game with no Nash equilibrium.

For each of these four cases we present a social network that exhibits the corresponding paradox. These paradoxes were identified first in [2] in the case when the adoption of a product was not obligatory. In contrast to the case here considered the existence of a strongest paradox within the framework of [2] remains an open problem.

2 Preliminaries

2.1 Strategic games

A ***strategic game*** for $n > 1$ players, written as $(S_1,\dots,S_n,p_1,\dots,p_n)$, consists of a non-empty set S_i of ***strategies*** and a ***payoff function*** $p_i : S_1 \times \cdots \times S_n \to \mathbb{R}$, for each player i.

Fix a strategic game $G := (S_1,\dots,S_n,p_1,\dots,p_n)$. We denote $S_1 \times \cdots \times S_n$ by S, call each element $s \in S$ a ***joint strategy***, denote the ith element of s by s_i, and abbreviate the sequence $(s_j)_{j \neq i}$ to s_{-i}. Occasionally we write (s_i, s_{-i}) instead of s.

We call a strategy s_i of player i a ***best response*** to a joint strategy s_{-i} of his opponents if $\forall s'_i \in S_i$ $p_i(s_i,s_{-i}) \geq p_i(s'_i,s_{-i})$. We call a joint strategy s a ***Nash equilibrium*** if each s_i is a best response to s_{-i}. Further, we call a strategy s'_i of player i a ***better response*** given a joint strategy s if $p_i(s'_i,s_{-i}) > p_i(s_i,s_{-i})$.

By a ***profitable deviation*** we mean a pair (s,s') of joint strategies such that $s' = (s'_i,s_{-i})$ for some s'_i and $p_i(s') > p_i(s)$. Following [10], an ***improvement path*** is a maximal sequence of profitable deviations. Clearly, if an improvement path is finite, then its last element is a Nash equilibrium. A game is called ***weakly acyclic*** (see [16, 9]) if for every joint strategy there exists a finite improvement path that starts at it. In other words, in weakly acyclic games a Nash equilibrium can be reached from every initial joint strategy by a sequence of unilateral deviations. Given two joint strategies s and s' we write

- $s > s'$ if for all i, $p_i(s) > p_i(s')$.

When $s > s'$ holds we say that s' is ***strictly worse*** than s.

2.2 Social networks

We are interested in strategic games defined over a specific type of social networks introduced in [1] that we recall first.

Let $V = \{1,\dots,n\}$ be a finite set of ***agents*** and $G = (V,E,w)$ a weighted directed graph with $w_{ij} \in [0,1]$ being the weight of the edge (i,j). Given a node i of G, we denote by $N(i)$ the set of nodes from which there is an incoming edge to i. We call each $j \in N(i)$ a ***neighbour*** of i in G. We assume that for each node i such that $N(i) \neq \emptyset$, $\sum_{j\in N(i)} w_{ji} \leq 1$. An agent $i \in V$ is said to be a ***source node*** in G if $N(i) = \emptyset$. Given a (to be defined) network $\mathscr{S}$ we denote by *source*$(\mathscr{S})$ the set of source nodes in the underlying graph G.

By a ***social network*** (from now on, just ***network***) we mean a tuple $\mathscr{S} = (G,\mathscr{P},P,\theta)$, where

- G is a weighted directed graph,
- $\mathscr{P}$ is a finite set of alternatives or ***products***,
- P is function that assigns to each agent i a non-empty set of products $P(i)$ from which it can make a choice,
- θ is a ***threshold function*** that for each $i \in V$ and $t \in P(i)$ yields a value $\theta(i,t) \in (0,1]$.

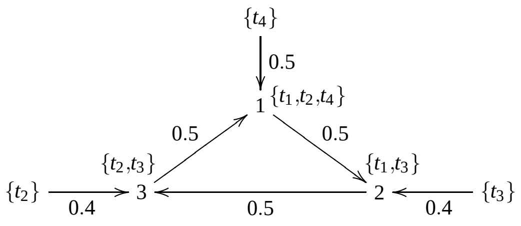


Figure 1: A social network

Example 1. Figure 1 shows an example of a network. Let the threshold be 0.3 for all nodes. The set of products $\mathscr{P}$ is $\{t_1,t_2,t_3,t_4\}$, the product set of each agent is marked next to the node denoting it and the weights are labels on the edges. Each source node is represented by the unique product in its product set. □

Given two social networks $\mathscr{S}$ and $\mathscr{S}'$ we say that $\mathscr{S}'$ is an ***expansion*** of $\mathscr{S}$ if it results from adding a product to the product set of a node in $\mathscr{S}$. We say then also that $\mathscr{S}$ is a ***contraction*** of $\mathscr{S}'$.

2.3 Social network games

Next, introduce the strategic games over the social networks. They form a modification of the games studied in [14, 15] in that we do not admit a strategy representing the fact that a player abstains from choosing a product.

Fix a network $\mathscr{S} = (G,\mathscr{P},P,\theta)$. With each network $\mathscr{S}$ we associate a strategic game $\mathscr{G}(\mathscr{S})$. The idea is that the agents simultaneously choose a product. Subsequently each node assesses his choice by comparing it with the choices made by his neighbours. Formally, we define the game as follows:

- the players are the agents (i.e., the nodes),
- the set of strategies for player i is $S_i := P(i)$,

- For $i \in V$, $t \in P(i)$ and a joint strategy s, let $\mathscr{N}_i^t(s) := \{j \in N(i) \mid s_j = t\}$, i.e., $\mathscr{N}_i^t(s)$ is the set of neighbours of i who adopted in s the product t.

  The payoff function is defined as follows, where c_0 is some given in advance positive constant:

  – for $i \in source(\mathscr{S})$,
    $p_i(s) := c_0$,
  – for $i \notin source(\mathscr{S})$,
    $p_i(s) := \sum_{j \in \mathscr{N}_i^t(s)} w_{ji} - \theta(i,t)$, where $s_i = t$ and $t \in P(i)$.

In the first case we assume that the payoff function for the source nodes is constant only for simplicity. The second case of the payoff definition is motivated by the following considerations. When agent i is not a source node, his 'satisfaction' from a joint strategy depends positively from the accumulated weight (read: 'influence') of his neighbours who made the same choice as him, and negatively from his threshold level (read: 'resistance') to adopt this product. The assumption that $\theta(i,t) > 0$ reflects the view that there is always some resistance to adopt a product.

We call these games ***social network games with obligatory product selection***, in short, ***social network games***.

Example 2. Consider the network given in Example 1 and the joint strategy s where each source node chooses the unique product in its product set and nodes 1, 2 and 3 choose t_2, t_3 and t_2 respectively. The payoffs are then given as follows:

- for the source nodes, the payoff is the fixed constant c_0,
- $p_1(s) = 0.5 - 0.3 = 0.2$,
- $p_2(s) = 0.4 - 0.3 = 0.1$,
- $p_3(s) = 0.4 - 0.3 = 0.1$.

Let s' be the joint strategy in which player 3 chooses t_3 and the remaining players make the same choice as given in s. Then (s,s') is a profitable deviation since $p_3(s') > p_3(s)$. In what follows, we represent each profitable deviation by a node and a strategy it switches to, e.g., $3 : t_3$. Starting at s, the sequence of profitable deviations $3 : t_3, 1 : t_4$ is an improvement path which results in the joint strategy in which nodes 1, 2 and 3 choose t_4, t_3 and t_3 respectively and, as before, each source node chooses the unique product in its product set. □

By definition, the payoff of each player depends only on the strategies chosen by his neighbours, so the social network games are related to graphical games of [8]. However, the underlying dependence structure of a social network game is a directed graph. Further, note that these games satisfy the ***join the crowd*** property that we define as follows:

> Each payoff function p_i depends only on the strategy chosen by player i and the set of players who also chose his strategy. Moreover, the dependence on this set is monotonic.

The last qualification is exactly opposite to the definition of congestion games with player-specific payoff functions of [9] in which the dependence on the above set is antimonotonic. That is, when more players choose the strategy of player i, then his payoff weakly decreases.

3 Nash equilibria

The first natural question we address is whether the social network games have a Nash equilibrium.

3.1 Simple cycles

In contrast to the case of games studied in [14] the answer is negative already for the case when the underlying graph is a simple cycle.

Example 3. Consider the network given in Figure 2, where the product set of each agent is marked next to the node denoting it and the weights are all equal and put as labels on the edges.

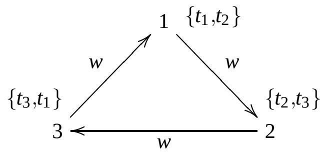


Figure 2: A simple cycle no Nash equilibrium

Let the thresholds be defined as follows: $\theta(1,t_1) = \theta(2,t_2) = \theta(3,t_3) = r_1$ and $\theta(1,t_2) = \theta(2,t_3) = \theta(3,t_1) = r_2$ where $r_1 > r_2$. We also assume that $w > r_1 - r_2$. Hence for all s_2 and s_3

$$p_1(t_1,s_2,t_1) > p_1(t_2,s_2,s_3) > p_1(t_1,s_2,t_3)$$

and similarly for the payoff functions p_2 and p_3. So it is more profitable for player i to adopt strategy t_i provided its neighbour also adopts t_i.

It is easy to check that the game associated with this network has no Nash equilibrium. Indeed, here is the list of all the joint strategies, where we underline the strategy that is not a best response to the choice of other players: $(t_1,\underline{t_2},t_1)$, $(t_1,\underline{t_2},t_3)$, $(t_1,t_3,\underline{t_1})$, $(\underline{t_1},t_3,t_3)$, $(\underline{t_2},t_2,t_1)$, $(t_2,t_2,\underline{t_3})$, $(\underline{t_2},t_3,t_1)$, $(t_2,\underline{t_3},t_3)$. □

This example can be easily generalized to the case of an arbitrary simple cycle. Below, $i \oplus 1$ and $i \ominus 1$ stand for addition and subtraction defined cyclically over the set $\{1,\ldots,n\}$. So $n \oplus 1 = 1$ and $1 \ominus 1 = n$. Indeed, consider a social network with n nodes that form a simple cycle and assume that each player i has strategies t_i and $t_{i\oplus 1}$. Choose for each player i the weights $w_{i\ominus 1\, i}$ and the threshold function $\theta(i,t)$ so that

$$w_{i\ominus 1\, i} - \theta(i,t_i) > -\theta(i,t_{i\oplus 1}) > -\theta(i,t_i),$$

so that (we put on first two positions, respectively, the strategies of players $i \ominus 1$ and i, while the last argument is a joint strategy of the remaining $n-2$ players)

$$p_i(t_i,t_i,s) > p_i(t',t_{i\oplus 1},s') > p_i(t_{i\ominus 1},t_i,s''),$$

where t', s, s' and s'' are arbitrary. It is easy to check then that the resulting social network game has no Nash equilibrium.

A natural question is what is the complexity of determining whether a Nash equilibrium exists. First we consider this question for the special case when the underlying graph is a simple cycle.

Theorem 4. *Consider a network $\mathscr{S}$ whose underlying graph is a simple cycle. It takes $O(n \cdot |\mathscr{P}|^4)$ time to decide whether the game $\mathscr{G}(\mathscr{S})$ has a Nash equilibrium.*

Proof. Suppose $\mathscr{S} = (G, \mathscr{P}, P, \theta)$. When the underlying graph of $\mathscr{S}$ is a simple cycle, the concept of a best response of player $i \oplus 1$ to a strategy of player i is well-defined. Let

$$R_i := \{(t_i,t_{i\oplus 1}) \mid t_i \in P(i), t_{i\oplus 1} \in P(i\oplus 1), t_{i\oplus 1} \text{ is a best response to } t_i\},$$

$$I := \{(t,t) \mid t \in \mathscr{P}\},$$

and let $\circ$ stand for the composition of binary relations.

The question whether $\mathscr{G}(\mathscr{S})$ has a Nash equilibrium is then equivalent to the problem whether there exists a sequence $a_1,...,a_n$ such that $(a_1,a_2) \in R_1,...,(a_{n-1},a_n) \in R_{n-1},(a_n,a_1) \in R_n$. In other words, is $(R_1 \circ \cdots \circ R_n) \cap I$ non-empty?

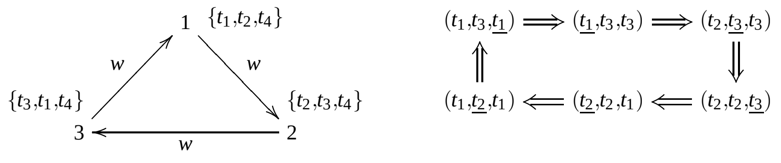
To answer this question we first construct successively $n-1$ compositions $R_1 \circ R_2$, $(R_1 \circ R_2) \circ R_3, \ldots,$ $(\ldots(R_1 \circ R_2) \cdots \circ R_{n-1}) \circ R_n$.

Each composition construction can be carried out in $|\mathscr{P}|^4$ steps. Indeed, given two relations $A,B \subseteq \mathscr{P} \times \mathscr{P}$, to compute their composition $A \circ B$ requires for each pair $(a,b) \in A$ to find all pairs $(c,d) \in B$ such that $b = c$. Finally, to check whether the intersection of $R_1 \circ \cdots \circ R_n$ with I is non-empty requires at most $|\mathscr{P}|$ steps.

So to answer the original question takes $O(n \cdot |\mathscr{P}|^4)$ time. □

Note that this proof applies to any strategic game in which there is a reordering of players $\pi(1),\ldots,\pi(n)$ such that the payoff of player $\pi(i)$ depends only on his strategy and the strategy chosen by player $\pi(i \ominus i)$.

It is worthwhile to note that for the case of simple cycles, the existence of Nash equilibrium in the associated social network game does not imply that the game is weakly acyclic.


1
$\{t_1,t_2,t_4\}$
w
w
$\{t_3,t_1,t_4\}$
$\{t_2,t_3,t_4\}$
3
2
w
$(t_1,t_3,\underline{t_1}) \Longrightarrow (\underline{t_1},t_3,t_3) \Longrightarrow (t_2,\underline{t_3},t_3)$
$(t_1,\underline{t_2},t_1) \Longleftarrow (\underline{t_2},t_2,t_1) \Longleftarrow (t_2,t_2,\underline{t_3})$

(a) (b)

Figure 3: A simple cycle and an infinite improvement path

Example 5. Consider the network in Figure 3(a) which is a modification of the network in Figure 2. We add a new product t_4 to the product set of all the nodes i with $\theta(i,t_4) > r_1$. We also assume that $w - \theta(i,t_4) > -r_2$. Then the joint strategy (t_4,t_4,t_4) is a Nash equilibrium. However, Figure 3(b) shows the unique improvement path starting in (t_1,t_3,t_1) which is infinite. For each joint strategy in the figure, we underline the strategy that is not a best response. This shows that the game is not weakly acyclic. □

In Section 4 we shall study the complexity of checking whether a social network game is weakly acyclic.

3.2 Arbitrary social networks

In this section we establish two results which show that deciding whether a social network has a Nash equilibrium is computationally hard.

Theorem 6. *Deciding whether for a social network $\mathscr{S}$ the game $\mathscr{G}(\mathscr{S})$ has a Nash equilibrium is NP-complete.*

To prove the result we first construct another example of a social network game with no Nash equilibrium and then use it to determine the complexity of the existence of Nash equilibria.

Example 7. Consider the network given in Figure 4, where the product set of each agent is marked next to the node denoting it and the weights are labels on the edges. Nodes with a unique product in the product set is simply represented by the product.

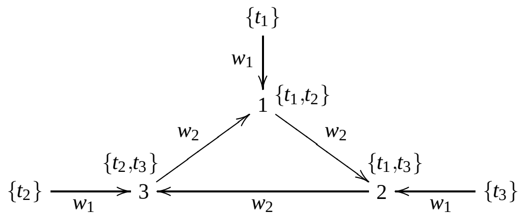


Figure 4: A network with no Nash equilibrium

We assume that each threshold is a constant θ, where $\theta < w_1 < w_2$. So it is more profitable to a player residing on a triangle to adopt the product adopted by his neighbour residing on a triangle than by the other neighbour.

The game associated with this network has no Nash equilibrium. It suffices to analyze the joint strategies involving nodes 1, 2 and 3 since the other nodes have exactly one product in their product sets. Here we provide a listing of all such joint strategies, where we underline the strategy that is not a best response to the choice of other players: $(\underline{t_1},t_1,t_2)$, $(t_1,t_1,\underline{t_3})$, $(t_1,t_3,\underline{t_2})$, $(t_1,\underline{t_3},t_3)$, $(t_2,\underline{t_1},t_2)$, $(t_2,\underline{t_1},t_3)$, $(t_2,t_3,\underline{t_2})$, $(\underline{t_2},t_3,t_3)$. In contrast, what will be of relevance in a moment, if we replace $\{t_1\}$ by $\{t_1'\}$, then the corresponding game has a Nash equilibrium, namely the joint strategy corresponding to the triple (t_2,t_3,t_3). □

Proof of Theorem 6: As in [1], to show NP-hardness, we use a reduction from the NP-complete PARTITION problem, which is: given n positive rational numbers $(a_1,\ldots,a_n)$, is there a set S such that $\sum_{i\in S} a_i = \sum_{i\notin S} a_i$? Consider an instance I of PARTITION. Without loss of generality, suppose we have normalised the numbers so that $\sum_{i=1}^{n} a_i = 1$. Then the problem instance sounds: is there a set S such that $\sum_{i\in S} a_i = \sum_{i\notin S} a_i = \frac{1}{2}$?

To construct the appropriate network we employ the networks given in Figure 4 and in Figure 5, where for each node $i \in \{1,\ldots,n\}$ we set $w_{ia} = w_{ib} = a_i$, and assume that the thresholds of the nodes a and b are constant and equal $\frac{1}{2}$.

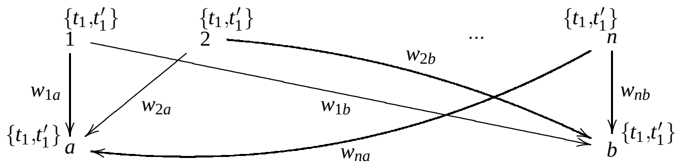


Figure 5: A network related to the PARTITION problem

To finalize the construction we use two copies of the network given in Figure 4, one unchanged and the other in which the product t_1 is replaced everywhere by t_1', and construct the desired network $\mathscr{S}$ by identifying with the node marked by $\{t_1\}$ in the network from Figure 4, the node a of the network from Figure 5 and with the node marked by $\{t_1'\}$ in the modified version of the network from Figure 4 the node b.

Suppose that a solution to the considered instance of the PARTITION problem exists, i.e., for some set $S \subseteq \{1,\ldots,n\}$ we have $\sum_{i\in S} a_i = \sum_{i\notin S} a_i = \frac{1}{2}$. Consider the game $\mathscr{G}(\mathscr{S})$ and the joint strategy s formed by the following strategies:

- t_1 assigned to each node $i \in S$ in the network from Figure 5,
- t_1' assigned to each node $i \in \{1,\ldots,n\} \setminus S$ in the network from Figure 5,

- t'_1 assigned to the nodes a and t_1 to the node b,
- t_2 assigned to node 1 and t_3 assigned to the nodes 2, 3 in both versions of the networks from Figure 4,
- t_2 and t_3 assigned respectively to the nodes marked by $\{t_2\}$ and $\{t_3\}$.

We claim that s is a Nash equilibrium. Consider first the player (i.e., node) a. The accumulated weight of its neighbours who chose strategy t'_1 is $\frac{1}{2}$. Therefore, the payoff for a in the joint strategy s is 0. The accumulated weight of its neighbours who chose strategy t_1 is $\frac{1}{2}$, as well. Therefore t'_1 is indeed a best response for player a as both strategies yield the same payoff. For the same reason, t_1 is a best response for player b. The analysis for the other nodes is straightforward.

Conversely, suppose that a strategy profile s is a Nash equilibrium in $\mathcal{G}(\mathcal{S})$. From Example 7 it follows that $s_a = t'_1$ and $s_b = t_1$. This implies that t'_1 is a best response of node a to s_{-a} and therefore $\sum_{i \in \{1,\dots,n\} | s_i = t'_1} w_{ia} \geq \sum_{i \in \{1,\dots,n\} | s_i = t_1} w_{ia}$. By a similar reasoning, for node b we have $\sum_{i \in \{1,\dots,n\} | s_i = t_1} w_{ib} \geq \sum_{i \in \{1,\dots,n\} | s_i = t'_1} w_{ib}$. Since $\sum_{i=1}^{n} a_i = 1$ and for $i \in \{1,\dots,n\}$, $w_{ia} = w_{ib} = a_i$, and $s_i \in \{t_1, t'_1\}$ we have for $S := \{i \in \{1,\dots,n\} \mid s_i = t_1\}$, $\sum_{i \in S} a_i = \sum_{i \notin S} a_i$. In other words, there exists a solution to the considered instance of the partition problem. □

Theorem 8. *For a network $\mathcal{S}$ whose underlying graph has no source nodes, deciding whether the game $\mathcal{G}(\mathcal{S})$ has a Nash equilibrium is NP-complete.*

Proof. The proof extends the proof of the above theorem. Given an instance of the PARTITION problem we use the following modification of the network. We 'twin' each node $i \in \{1,\dots,n\}$ in Figure 5 with a new node i' with the product set $\{t_1, t'_1\}$, by adding the edges (i,i') and (i',i). We also 'twin' nodes marked $\{t_2\}$ and $\{t_3\}$ in Figure 4 with new nodes with the product set $\{t_2\}$ and $\{t_3\}$ respectively. Additionally, we choose the weights on the new edges $w_{ii'}$, $w_{i'i}$ and the corresponding thresholds so that when i and i' adopt a common product, their payoff is positive. Then the underlying graph of the resulting network does not have any source nodes and the above proof remains valid for this new network. □

4 Weakly acyclic games

In this section we study the complexity of checking whether a social network game is weakly acyclic. We establish two results that are analogous to the ones established in [15] for the case of social networks in which the nodes may decide not to choose any product. The proofs are based on similar arguments though the details are different.

Theorem 9. *For an arbitrary network $\mathcal{S}$, deciding whether the game $\mathcal{G}(\mathcal{S})$ is weakly acyclic is co-NP hard.*

Proof. We again use an instance of the PARTITION problem in the form of n positive rational numbers $(a_1,\dots,a_n)$ such that $\sum_{i=1}^{n} a_i = 1$. Consider the network given in Figure 6. For each node $i \in \{1,\dots,n\}$ we set $P(i) = \{t_1, t_2\}$. The product set for the other nodes are marked in the figure. As before, we set $w_{ia} = w_{ib} = a_i$.

Since for all $i \in \{1,\dots,n\}$, a_i is rational, it has the form $a_i = \frac{l_i}{r_i}$. Let $\tau = \frac{1}{2 \cdot r_1 \cdot \ldots \cdot r_n}$. The following property holds.

Property 1. *Given an instance $(a_1,\dots,a_n)$ of the PARTITION problem and τ defined as above, for all $S \subseteq \{1,\dots,n\}$*

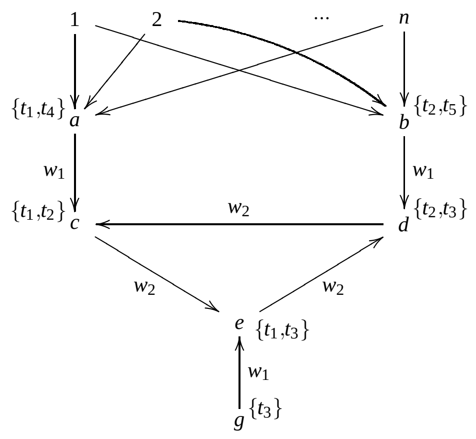


Figure 6: A network related to weakly acyclic games

(i) *if* $\sum_{i \in S} a_i < \frac{1}{2}$, *then* $\sum_{i \in S} a_i \leq \frac{1}{2} - \tau$,

(ii) *if* $\sum_{i \in S} a_i > \frac{1}{2}$, *then* $\sum_{i \in S} a_i \geq \frac{1}{2} + \tau$.

Proof. By definition, each a_i and $\frac{1}{2}$ is a multiple of τ. Thus $\sum_{i \in S} a_i = x \cdot \tau$ and $\frac{1}{2} = y \cdot \tau$ where x and y are integers.
(i) If $x \cdot \tau < y \cdot \tau$, then $x \cdot \tau \leq (y-1) \cdot \tau$. Therefore $\sum_{i \in S} a_i \leq \frac{1}{2} - \tau$.
The proof of *(ii)* is analogous. □

Note that given $(a_1, \ldots, a_n)$, τ can be defined in polynomial time. Let the thresholds be defined as follows: $\theta(a,t_1) = \theta(b,t_2) = \frac{1}{2}$ and $0 < \theta(a,t_4) = \theta(b,t_5) < \tau$. The threshold for nodes c,d and e is a constant θ_1 such that $\theta_1 < w_1 < w_2$. Thus, like in the network in Figure 4, it is more profitable to a player residing on a triangle to adopt the product adopted by his neighbour residing on a triangle than by the other neighbour.

Suppose that a solution to the considered instance of the PARTITION problem exists. That is, for some set $S \subseteq \{1, \ldots, n\}$ we have $\sum_{i \in S} a_i = \sum_{i \notin S} a_i = \frac{1}{2}$. In the game $\mathcal{G}(\mathcal{S})$, take the joint strategy s formed by the following strategies:

- t_1 assigned to each node $i \in S$ and the nodes a and c,
- t_2 assigned to each node $i \in \{1, \ldots, n\} \setminus S$ and the nodes b and d,
- t_3 assigned to the nodes e and g.

Any improvement path that starts in this joint strategy will not change the strategies assigned to the nodes a,b and g. So if such an improvement path terminates, it produces a Nash equilibrium in the game associated with the network given in Figure 4 of Example 7. But we argued that this game does not have a Nash equilibrium. Consequently, there is no finite improvement path in the game $\mathcal{G}(\mathcal{S})$ that starts in the above joint strategy and therefore $\mathcal{G}(\mathcal{S})$ is not weakly acyclic.

Now suppose that the considered instance of the PARTITION problem does not have a solution. Then we show that the game $\mathcal{G}(\mathcal{S})$ is weakly acyclic. To this end, we order the nodes of $\mathcal{S}$ as follows (note the positions of the nodes c,d and e): $1,2,\ldots,n,g,a,b,c,e,d$. Given a joint strategy, consider an improvement path in which at each step the first node in the above list that did not select a best response switches to a best response. After at most n steps the nodes $1,2,\ldots,n$ all selected a product t_1 or t_2. Let s be the resulting joint strategy.

First suppose that $\sum_{i \in \{1,...,n\} | s_i = t_1} w_{ia} > \frac{1}{2}$. This implies that $\sum_{i \in \{1,...,n\} | s_i = t_2} w_{ib} < \frac{1}{2}$. By Property 1, $\sum_{i \in \{1,...,n\} | s_i = t_2} w_{ib} \leq \frac{1}{2} - \tau$. The payoff of the node b depends only on the choices made by the source nodes $1, 2, \ldots, n$, so we have $p_b(t_2, s_{-b}) \leq -\tau$. Since $\theta(b, t_5) < \tau$, we also have $p_b(t_5, s_{-b}) > -\tau$ and therefore t_5 is a best response for node b. Let s^b be the resulting strategy in which node b selects t_5. Consider the prefix of ξ starting at s^b (call it ξ^b). We argue that in ξ^b, t_2 is never a better response for node d. Suppose that $s^b_d = t_3$. We have the following two cases:

- $s^b_e = t_3$: then $p_d(s^b) = w_2 - \theta_1$ and so t_3 is the best response for node d.
- $s^b_e = t_1$: then $p_d(s^b) = -\theta_1$ and if node d switches to t_2 then $p_d(t_2, s^b_{-b}) = -\theta_1$ (since $s^b_b = t_5$). Thus t_2 is not a better response.

Using the above observation, we conclude that there exists a suffix of ξ^b (call it ξ^d) such that node d never chooses t_2. This means that in ξ^d the unique best response for node c is t_1 and for node e is t_1. This shows that ξ^d is finite and hence ξ is finite, as well.

The case when $\sum_{i \in \{1,...,n\} | s_i = t_2} w_{ib} > \frac{1}{2}$ is analogous with all improvement paths terminating in a joint strategy where node a chooses t_4 and node c chooses t_2. □

Theorem 10. *For a network $\mathcal{S}$ whose underlying graph has no source nodes, deciding whether the game $\mathcal{G}(\mathcal{S})$ is weakly acyclic is co-NP hard.*

Proof. The proof extends the proof of the above theorem. Given an instance of the PARTITION problem we use the following modification of the network given in Figure 6. We 'twin' each node $i \in \{1, \ldots, n\}$ with a new node i', also with the product set $\{t_1, t_2\}$, by adding the edges (i, i') and (i', i). We also 'twin' the node g with a new node g', also with the product set $\{t_3\}$, by adding the edges (g, g') and (g', g). Additionally, we choose the weights $w_{ii'}$ and $w_{i'i}$ and the corresponding thresholds so that when i and i' adopt a common product, their payoff is positive.

Suppose that a solution to the considered instance of the PARTITION problem exists. Then we extend the joint strategy considered in the proof of Theorem 9 by additionally assigning t_1 to each node i' such that $i \in S$, t_2 to each node i' such that $i \in \{1, \ldots, n\} \setminus S$ and t_3 to the node g'. Then, as before, there is no finite improvement path starting in this joint strategy, so $\mathcal{G}(\mathcal{S})$ is not weakly acyclic.

Suppose now that no solution to the considered instance of the PARTITION problem exists. Take the following order of the nodes of $\mathcal{S}$:

$$1, 1', 2, 2', \ldots, n, n', g, g', a, b, c, e, d,$$

and as in the previous proof, given a joint strategy, we consider an improvement path ξ in which at each step the first node in the above list that did not select a best response switches to a best response.

Note that each node from the list $1, 1', 2, 2', \ldots, n, n', g, g'$ is scheduled at most once. So there exists a suffix of ξ in which only the nodes a, b, c, e, d are scheduled. Using now the argument given in the proof of Theorem 9 we conclude that there exists a suffix of ξ that is finite. This proves that $\mathcal{G}(\mathcal{S})$ is weakly acyclic. □

5 Paradoxes

In [2] we identified various paradoxes in social networks with multiple products and studied them using the social network games introduced in [14]. Here we carry out an analogous analysis for the case when the product selection is obligatory. This qualification, just like in the case of social network games, substantially changes the analysis. We focus on the main four paradoxes that we successively introduce and analyze.

5.1 Vulnerable networks

The first one is the following. We say that a social network $\mathcal{S}$ is ***vulnerable*** if for some Nash equilibrium s in $\mathcal{G}(\mathcal{S})$, an expansion $\mathcal{S}'$ of $\mathcal{S}$ exists such that each improvement path in $\mathcal{G}(\mathcal{S}')$ leads from s to a joint strategy s' which is a Nash equilibrium both in $\mathcal{G}(\mathcal{S}')$ and $\mathcal{G}(\mathcal{S})$ such that $s > s'$. So the newly added product triggers a sequence of changes that unavoidably move the players from one Nash equilibrium to another one that is strictly worse for everybody.

The following example shows that vulnerable networks exist. Here and elsewhere the relevant expansion is depicted by means of a product and the dotted arrow pointing to the relevant node.

Example 11. Consider the directed graph given in Figure 7, in which the product set of each node is marked next to it.

$$\begin{tikzcd}
\{\underline{t_1},t_3,t_4\}\ 1 \arrow[r, bend left] \arrow[d] & 2\ \{\underline{t_1},t_4\} \arrow[l, bend left] & t_2 \arrow[l, dotted] \\
\{\underline{t_2},t_3\}\ 3 \arrow[r, bend left] & 4\ \{\underline{t_2},t_3\} \arrow[l, bend left] \arrow[u]
\end{tikzcd}$$

Figure 7: A directed graph

We complete it to the desired social network below. Let '_' stand for an arbitrary strategy of the relevant player. We stipulate that

$$\begin{aligned}
&p_2(\_,t_2,\_,t_2) > p_2(t_1,t_1,\_,\_),\\
&p_1(t_3,t_2,\_,\_) > p_1(t_1,t_2,\_,\_) > p_1(t_4,t_2,\_,\_),\\
&p_3(t_3,\_,t_3,\_) > p_3(\_,\_,t_2,t_2),\\
&p_4(\_,\_,t_3,t_3) > p_4(\_,\_,t_3,t_2),\\
&p_2(\_,t_4,\_,\_) > p_2(\_,t_2,\_,t_3),\\
&p_1(t_4,t_4,\_,\_) > p_1(t_3,\_,\_,\_) > p_1(t_1,t_4,\_,\_),
\end{aligned}$$

so that $2:t_2, 1:t_3, 3:t_3, 4:t_3, 2:t_4, 1:t_4$ is a unique improvement path that starts in (t_1,t_1,t_2,t_2) and ends in (t_4,t_4,t_3,t_3).

Additionally we stipulate that

$$\begin{aligned}
&p_1(t_1,t_1,\_,\_) > p_1(t_4,t_4,\_,\_),\\
&p_2(t_1,t_1,\_,\_) > p_2(t_4,t_4,\_,\_),\\
&p_3(\_,\_,t_2,t_2) > p_3(\_,\_,t_3,t_3),\\
&p_4(\_,\_,t_2,t_2) > p_4(\_,\_,t_3,t_3),
\end{aligned}$$

so that $(t_1,t_1,t_2,t_2) >_s (t_4,t_4,t_3,t_3)$.

These requirements entail constraints on the weights and thresholds that are for instance realized by

$$w_{12} = 0,\ w_{21} = 0.2,\ w_{42} = 0.3,\ w_{13} = 0.2,\ w_{34} = 0.2,\ w_{43} = 0,$$

and

$$\begin{aligned}
&\theta(1,t_1) = 0.2,\ \theta(1,t_3) = 0.1,\ \theta(1,t_4) = 0.3,\ \theta(2,t_1) = 0.1,\ \theta(2,t_2) = 0.3,\\
&\theta(2,t_4) = 0.2,\ \theta(3,t_2) = 0.1,\ \theta(3,t_3) = 0.2,\ \theta(4,t_2) = 0.1,\ \theta(4,t_3) = 0.2.
\end{aligned}$$ □

It is useful to note that in the setup of [2], in which for each player the 'abstain' strategy is allowed, it remains an open problem whether vulnerable networks (called there because of various other alternatives $\forall s$-vulnerable networks) exist.

5.2 Fragile networks

Next, we consider the following notion. We say that a social network $\mathscr{S}$ is ***fragile*** if $\mathscr{G}(\mathscr{S})$ has a Nash equilibrium while for some expansion $\mathscr{S}'$ of $\mathscr{S}$, $\mathscr{G}(\mathscr{S}')$ does not. The following example shows that fragile networks exist.

Example 12. Consider the network $\mathscr{S}$ given in Figure 8, where the product set of each node is marked next to it.

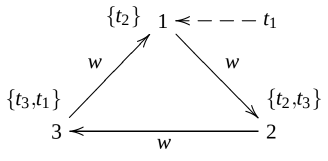


Figure 8: A fragile network

Let the thresholds be defined as follows: $\theta(2,t_2) = \theta(3,t_3) = r_1$ and $\theta(1,t_2) = \theta(2,t_3) = \theta(3,t_1) = r_2$ where $r_1 > r_2$. We also assume that $w > r_1 - r_2$.

Consider the joint strategy s, in which nodes 1, 2 and 3 choose t_2, t_2 and t_1 respectively. It can be verified that s is a Nash equilibrium in $\mathscr{G}(\mathscr{S})$. Now consider the expansion $\mathscr{S}'$ of $\mathscr{S}$ in which product t_1 is added to the product set of node 1 and let $\theta(1,t_1) = r_1$. Then $\mathscr{S}'$ is the network in Example 3 which, as we saw, does not have a Nash equilibrium. □

5.3 Inefficient networks

We say that a social network $\mathscr{S}$ is ***inefficient*** if for some Nash equilibrium s in $\mathscr{G}(\mathscr{S})$, a contraction $\mathscr{S}'$ of $\mathscr{S}$ exists such that each improvement path in $\mathscr{G}(\mathscr{S}')$ starting in s leads to a joint strategy s' which is a Nash equilibrium both in $\mathscr{G}(\mathscr{S}')$ and $\mathscr{G}(\mathscr{S})$ such that $s' > s$. We note here that if the contraction was created by removing a product from the product set of node i, we impose that any improvement path in $\mathscr{G}(\mathscr{S}')$, given a starting joint strategy from $\mathscr{G}(\mathscr{S})$, begins by having node i making a choice (we allow any choice from his remaining set of products as an improvement move). Otherwise the initial payoff of node i in $\mathscr{G}(\mathscr{S}')$ is not well-defined.

Example 13. We exhibit in Figure 9 an example of an inefficient network. The weight of each edge is assumed to be w, and we also have the same product-independent threshold, θ, for all nodes, with $w > \theta$.

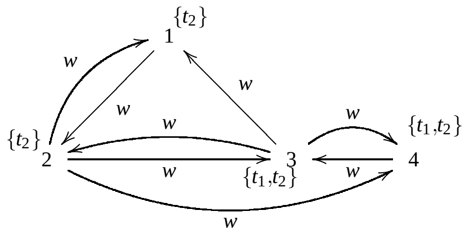


Figure 9: An example of an inefficient network

Consider as the initial Nash equilibrium the joint strategy $s = (t_2, t_2, t_1, t_1)$. It is easy to check that this is indeed a Nash equilibrium, with the payoff equal to $w - \theta$ for all nodes. Suppose now that we remove product t_1 from the product set of node 3. We claim that the unique improvement path then leads to the Nash equilibrium in which all nodes adopt t_2.

To see this, note that node 3 moves first in any improvement path and it has a unique choice, t_2. Then node 4 moves and necessarily switches to t_2. This yields a Nash equilibrium in which each node selected t_2 with the payoff of $2w - \theta$, which is strictly better than the payoff in s. $\square$

5.4 Unsafe networks

Finally, we analyze the following notion. We call a social network $\mathscr{S}$ ***unsafe*** if $\mathscr{G}(\mathscr{S})$ has a Nash equilibrium, while for some contraction $\mathscr{S}'$ of $\mathscr{S}$, $\mathscr{G}(\mathscr{S}')$ does not. The following example shows that unsafe networks exist.

Example 14. Let $\mathscr{S}_1$ be the modification of the network $\mathscr{S}$ given in Figure 2 where node 1 has the product set $\{t_1, t_2, t_4\}$, where $\theta(1, t_4) < r_2$. Then the joint strategy (t_4, t_3, t_3) is a Nash equilibrium in $\mathscr{G}(\mathscr{S}_1)$. Now consider the contraction $\mathscr{S}_1'$ of $\mathscr{S}_1$ where product t_4 is removed from node 1. Then $\mathscr{S}_1'$ is the network $\mathscr{S}$, which as we saw in Example 3 has no Nash equilibrium. $\square$

6 Conclusions

In this paper we studied dynamic aspects of social networks with multiple products using the basic concepts of game theory. We used the model of social networks, originally introduced in [1] that we subsequently studied using game theory in [14], [15] and [2].

However, in contrast to these three references the product adoption in this paper is obligatory. This led to some differences. For example, in contrast to the case of [14], a Nash equilibrium does not need to exist when the underlying graph is a simple cycle. Further, in contrast to the setup of [2], we were able to construct a social network that exhibits the strongest form of the paradox of choice. On the other hand, some complexity results, namely the ones concerning weakly acyclic games, remain the same as in [14], though the proofs had to be appropriately modified.

References

[1] K. R. Apt & E. Markakis (2011): *Diffusion in Social Networks with Competing Products*. In: *Proc. 4th International Symposium on Algorithmic Game Theory (SAGT11), Lecture Notes in Computer Science* 6982, Springer, pp. 212–223, doi:10.1007/978-3-642-24829-0_20.

[2] K. R. Apt, E. Markakis & S. Simon (2013): *Paradoxes in Social Networks with Multiple Products*. Manuscript, CWI, Amsterdam, The Netherlands. Computing Research Repository (CoRR), http://arxiv.org/abs/1301.7592.

[3] David Easley & Jon Kleinberg (2010): *Networks, Crowds, and Markets*. Cambridge University Press.

[4] D. Fotakis, A. C. Kaporis, T. Lianeas & P. G. Spirakis (2012): *On the Hardness of Network Design for Bottleneck Routing Games*. In: *SAGT*, pp. 156–167, doi:10.1007/978-3-642-33996-7_14.

[5] D. Fotakis, A. C. Kaporis & P. G. Spirakis (2012): *Efficient methods for selfish network design*. *Theor. Comput. Sci.* 448, pp. 9–20, doi:10.1016/j.tcs.2012.04.033.

[6] G. Gigerenzer (2008): *Gut Feelings: The Intelligence of the Unconscious*. Penguin. Reprint edition.

[7] M.O. Jackson (2008): *Social and Economic Networks*. Princeton University Press, Princeton.

[8] M. Kearns, M. Littman & S. Singh (2001): *Graphical models for game theory*. In: *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI '01)*, Morgan Kaufmann, pp. 253–260.

[9] I. Milchtaich (1996): *Congestion Games with Player-Specific Payoff Functions*. *Games and Economic Behaviour* 13, pp. 111–124, doi:10.1006/game.1996.0027.

[10] D. Monderer & L. S. Shapley (1996): *Potential Games*. *Games and Economic Behaviour* 14, pp. 124–143, doi:`10.1006/game.1996.0044`.

[11] S. Morris (2000): *Contagion*. *The Review of Economic Studies* 67(1), pp. 57–78, doi:`10.1111/1467-937X.00121`.

[12] N. Nisan, T. Roughgarden, É. Tardos & V. J. Vazirani, editors (2007): *Algorithmic Game Theory*. Cambridge University Press.

[13] B. Schwartz (2005): *Paradox of Choice: Why More Is Less*. Harper Perennial.

[14] S. Simon & K. R. Apt (2012): *Choosing Products in Social Networks*. In: *Proc. 8th International Workshop on Internet and Network Economics (WINE), Lecture Notes in Computer Science* 7695, Springer, pp. 100–113, doi:`10.1007/978-3-642-35311-6_8`.

[15] S. Simon & K. R. Apt (2013): *Social Network Games*. *Journal of Logic and Computation*, doi:`10.1093/logcom/ext012`. To appear.

[16] H. Peyton Young (1993): *The evolution of conventions*. *Econometrica* 61(1), pp. 57–84, doi:`10.2307/2951778`.

Alternating-time temporal logic with finite-memory strategies

Steen Vester
DTU Compute
Technical University of Denmark

stve@dtu.dk

Model-checking the alternating-time temporal logics ATL and ATL^* with incomplete information is undecidable for perfect recall semantics. However, when restricting to memoryless strategies the model-checking problem becomes decidable. In this paper we consider two other types of semantics based on finite-memory strategies. One where the memory size allowed is bounded and one where the memory size is unbounded (but must be finite). This is motivated by the high complexity of model-checking with perfect recall semantics and the severe limitations of memoryless strategies. We show that both types of semantics introduced are different from perfect recall and memoryless semantics and next focus on the decidability and complexity of model-checking in both complete and incomplete information games for ATL/ATL^*. In particular, we show that the complexity of model-checking with bounded-memory semantics is Δ_2^p-complete for ATL and $PSPACE$-complete for ATL^* in incomplete information games just as in the memoryless case. We also present a proof that ATL and ATL^* model-checking is undecidable for $n \geq 3$ players with finite-memory semantics in incomplete information games.

1 Introduction

The alternating-time temporal logics ATL and ATL^* have been studied with perfect recall semantics and memoryless semantics in both complete and incomplete information concurrent game structures [2, 3, 12]. The model-checking problems for these logics have applications in verification and synthesis of computing systems in which different entities interact. The complexity of model-checking with perfect recall semantics, where players are allowed to use an infinite amount of memory, is very high in some cases and even undecidable in the case of ATL [3, 8] with incomplete information. On the other hand, model-checking with memoryless semantics, where players are not allowed to use any memory about the history of a game, is decidable and has a much lower complexity [12]. The drawback is that there are many games where winning strategies exist for some coalition, but where no memoryless winning strategies exist. In this paper, we focus on the tradeoff between complexity and strategic ability with respect to the memory available to the players. Instead of considering the extreme cases of memoryless strategies and infinite memory strategies we look at finite-memory strategies as an intermediate case of the two. The motivation is the possibility to solve more games than with memoryless strategies, but without the cost that comes with infinite memory.

We introduce two new types of semantics called bounded-memory semantics and finite-memory semantics respectively. For bounded-memory semantics there is a bound on the amount of memory available to the players, whereas for finite-memory semantics players can use any finite amount of memory. We will study the expressiveness of these new types of semantics compared to memoryless and perfect recall semantics in ATL and ATL^* with both complete and incomplete information. Afterwards we focus on the complexity and decidability of the model-checking problem for the different cases.

Our approach have similarities with the work done in [12], [5] and [1]. It is a natural extension of the framework used in [12] where memoryless semantics and perfect recall semantics are considered. In [5] ATL/ATL^* with bounded-memory semantics and strategy context is introduced for complete information games, where bounded-memory strategies are defined essentially in the same way as here. However, their use of strategy context makes the problems and algorithms considered different from ours. In [1] a version with bounded-recall is considered where agents can only remember the last m states of the play. This contrasts our approach where the players can decide what to store in the memory about the past.

2 Concurrent game structures

A concurrent game is played on a finite graph by a finite number of players, where the players interact by moving a token between different states along the edges of the graph. The game is played an infinite number of rounds where each round is played by letting every player independently and concurrently choose an action. The combination of actions chosen by the players along with the current state uniquely determines the successor state of the game. More formally,

Definition 1. *A concurrent game structure (CGS) with n players*

$$\mathcal{G} = (\mathrm{States}, \mathrm{Agt}, \mathrm{Act}, \mathrm{Mov}, \mathrm{Tab})$$

consists of

- States - *A finite non-empty set of states*
- $\mathrm{Agt} = \{1, ..., n\}$ - *A finite non-empty set of players*
- Act - *A finite non-empty set of actions*
- $\mathrm{Mov} : \mathrm{States} \times \mathrm{Agt} \to 2^{\mathrm{Act}} \setminus \{\emptyset\}$ - *A function specifying the legal actions at a given state of a given player*
- $\mathrm{Tab} : \mathrm{States} \times \mathrm{Act}^n \to \mathrm{States}$ - *A transition function defined for each* $(a_1, ..., a_n) \in \mathrm{Act}^n$ *and state* s *such that* $a_j \in \mathrm{Mov}(s, j)$ *for* $1 \leq j \leq n$

Unless otherwise noted, we implicitly assume from now on that the players in a game are named $1, ..., n$ where $n = |\mathrm{Agt}|$. Note that every player must have at least one legal action in each state. The transition function Tab is defined for each state and all legal tuples of actions in that state. We also refer to such legal tuples of actions as moves. To add meaning to concurrent game structures we introduce the concept of a concurrent game model which consists of a concurrent game structure as well as a labeling of the states in the structure with propositions from some fixed, finite set Prop of proposition symbols.

Definition 2. *A concurrent game model (CGM) is a pair* $(\mathcal{G}, \pi)$ *where* $\mathcal{G}$ *is a concurrent game structure and* $\pi : \mathrm{States} \to \mathcal{P}(\mathrm{Prop})$ *is a labeling function.*

An example of a CGM can be seen in Figure 1, where the states are drawn as nodes. Transitions are drawn as edges between nodes such that there is an edge from s to s' labeled with the move $(a_1, ..., a_n)$ if $\mathrm{Tab}(s, (a_1, ..., a_n)) = s'$. The states are labelled with propositions from the set $\mathrm{Prop} = \{p, q\}$ in the figure.

We define an incomplete information concurrent game structure as a CGS where each player j has an equivalence relation $\sim_j$ on the set of states. The intuitive meaning is that $s \sim_j s'$ if player j cannot distinguish between the states s and s'.

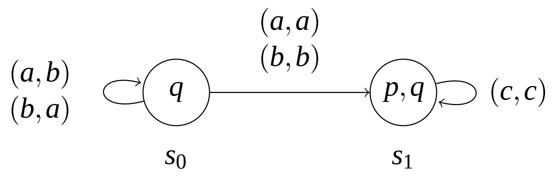


Figure 1: CGM $\mathscr{M}$

Definition 3. *A concurrent game structure with incomplete information (iCGS) with n players is a tuple*

$$\mathscr{G} = (\text{States}, \text{Agt}, \text{Act}, \text{Mov}, \text{Tab}, (\sim_j)_{1\leq j\leq n})$$

where

- $(\text{States}, \text{Agt}, \text{Act}, \text{Mov}, \text{Tab})$ *is a CGS*
- $\sim_j \subseteq \text{States} \times \text{States}$ *is an equivalence relation for all* $1 \leq j \leq n$
- *If* $s \sim_j s'$ *then* $\text{Mov}(s, j) = \text{Mov}(s', j)$ *for all* $s, s' \in \text{States}$ *and all* $j \in \text{Agt}$

Note that we require the set of actions available to a player in two indistinguishable states to be the same. We extend the notion to concurrent game models with incomplete information in the natural way.

Definition 4. *A concurrent game model with incomplete information (iCGM) is a pair* $(\mathscr{G}, \pi)$ *where* $\mathscr{G}$ *is an iCGS and* $\pi : \text{States} \to 2^{\text{Prop}}$ *is a labeling function.*

For each player j, the relation $\sim_j$ induces a set $[\cdot]_j$ of equivalence classes of states. We denote by $[s]_j$ the class that state s belongs to for player j. These classes are refered to as the observation sets of player j. Since the set of legal actions of player j is required to be the same in states from the same observation set, we can define $\text{Mov}([s]_j, j) = \text{Mov}(s, j)$ for all states s. Note that the concepts of iCGS and iCGM generalize CGS and CGM respectively, since they are the special cases where $\sim_j$ is the identity relation for all players j.

3 Outcomes, histories and strategies

Let $\mathscr{G} = (\text{States}, \text{Agt}, \text{Act}, \text{Mov}, \text{Tab})$ be a CGS with n players. An outcome (or play) of a concurrent game is an infinite sequence of states in the game structure that corresponds to an infinite sequence of legal moves. Formally, the set of outcomes $\text{Out}_{\mathscr{G}}(s)$ of $\mathscr{G}$ from $s \in \text{States}$ is defined as

$$\text{Out}_{\mathscr{G}}(s) = \{\rho_0\rho_1... \in \text{States}^\omega \mid \rho_0 = s \wedge \forall j \geq 0. \exists m \in \text{Act}^n. \text{Tab}(\rho_j, m) = \rho_{j+1}\}$$

$\text{Out}_{\mathscr{G}} = \bigcup_{s\in\text{States}} \text{Out}_{\mathscr{G}}(s)$ is the set of all outcomes of $\mathscr{G}$. A history of a concurrent game is a non-empty, finite prefix of an outcome. The set $\text{Hist}_{\mathscr{G}}(s)$ of histories of $\mathscr{G}$ from $s \in \text{States}$ is defined as

$$\text{Hist}_{\mathscr{G}}(s) = \{\rho_0\rho_1...\rho_k \in \text{States}^+ \mid \rho_0 = s \wedge \exists \rho' \in \text{Out}_{\mathscr{G}}(s). \forall 0 \leq j \leq k. \rho_j = \rho'_j\}$$

$\text{Hist}_{\mathscr{G}} = \bigcup_{s\in\text{States}} \text{Hist}_{\mathscr{G}}(s)$ is the set of all histories of $\mathscr{G}$. For a (finite or infinite) sequence ρ of states we write ρ_0 for the first state, ρ_j for the $(j+1)$th state. $\rho_{\leq j}$ is the prefix $\rho_0\rho_1...\rho_j$ of ρ and $\rho_{\geq j}$ is the suffix $\rho_j\rho_{j+1}...$ of ρ. When $\rho = \rho_0...\rho_k$ is a finite sequence we denote the length of ρ by $|\rho| = k$ and write $\text{last}(\rho) = \rho_k$.

For a given CGS $\mathscr{G} = (\text{States}, \text{Agt}, \text{Act}, \text{Mov}, \text{Tab})$ we define a strategy for player j as a mapping $\sigma_j : \text{Hist}_{\mathscr{G}} \to \text{Act}$ such that for all $h \in \text{Hist}_{\mathscr{G}}$ we have $\sigma_j(h) \in \text{Mov}(\text{last}(h), j)$. Thus, a strategy for player j maps any given history to an action that is legal for player j in the final state of the history. We will also refer to these strategies as perfect recall strategies or infinite-memory strategies, since a player using such a strategy can use the entire history of a play up to the decision point to choose his next action. A memoryless (positional, no recall) strategy for player j is a strategy σ_j such that for all $h, h' \in \text{Hist}_{\mathscr{G}}$ with $\text{last}(h) = \text{last}(h')$ we have $\sigma_j(h) = \sigma_j(h')$. It is called a memoryless strategy since the player is only using the last state of the history to decide on his action. We denote by Strat_j^R the set of perfect recall strategies for player j and by Strat_j^r the set of memoryless strategies for player j. We write $\text{Out}_{\mathscr{G}}(s, \sigma)$ for a strategy $\sigma = (\sigma_a)_{a \in \text{Agt}}$ for coalition A and a state s to denote the set of possible outcomes from state s when players in coalition A play according to σ.

Next, we define finite-memory strategies in which a player is only allowed to store a finite amount of memory of the history of the game. He can then combine his memory with the current state of the game to choose an action. To model a strategy with finite memory we use a deterministic finite-state transducer (DFST). A DFST is a 6-tuple $(M, m_0, \Sigma, \Gamma, T, G)$ where M is a finite, non-empty set of states, m_0 is the initial state, Σ is the input alphabet, Γ is the output alphabet, $T : M \times \Sigma \to M$ is the transition function and $G : M \times \Sigma \to \Gamma$ is the output function. The set of states of the DFST are the possible values of the internal memory of the strategy. We will also call these memory states. The initial state corresponds to the initial memory value. The input symbols are the states of the game and the set of output symbols is the set of actions of the game. In each round of the game the DFST reads a state of the game. Then it updates its memory based on the current memory value and the input state and performs an action based on the current memory value and the input state. More formally, we say that a strategy σ_j for player j is a finite-memory strategy if there exists a DFST $A = (M, m_0, \text{States}, \text{Act}, T, G)$ such that for all $h \in \text{Hist}_{\mathscr{G}}$ we have

$$\sigma_j(h) = G(\mathscr{T}(m_0, h_{\leq |h|-1}), \text{last}(h))$$

where $\mathscr{T}$ is defined recursively by $\mathscr{T}(m, \rho) = T(m, \rho_0)$ for any memory state m and any history ρ with $|\rho| = 0$ and $\mathscr{T}(m, \rho) = T(\mathscr{T}(m, \rho_{\leq |\rho|-1}), \text{last}(\rho))$ for any memory state m and any history ρ with $|\rho| \geq 1$. Intuitively $\mathscr{T}$ is the function that repeatedly applies the transition function T on a sequence of inputs to calculate the memory state after a given history. We call $\mathscr{T}$ the repeated transition function. We say that σ_j is a k-memory strategy if the number of states of the DFST is k. We also say that the strategy σ_j is represented by the DFST A. We denote the set of finite-memory strategies for player j by Strat_j^F and the set of k-memory strategies for player j by $\text{Strat}_j^{F_k}$. Thus, $\text{Strat}_j^F = \bigcup_{k \geq 1} \text{Strat}_j^{F_k}$. In addition, we have that the memoryless strategies are exactly the finite-memory strategies with one memory state, i.e. $\text{Strat}_j^{F_1} = \text{Strat}_j^r$.

Next, we generalize the notions of strategies to incomplete information games by defining them on observation histories rather than on histories, since players observe sequences of observation sets during the play rather than sequences of states. We define the set $\text{Hist}_{\mathscr{G}}^j$ of observation histories for player j in iCGS $\mathscr{G}$ as

$$\text{Hist}_{\mathscr{G}}^j = \{[s_0]_j [s_1]_j ... [s_k]_j \mid s_0 s_1 ... s_k \in \text{Hist}_{\mathscr{G}}\}$$

For each player, a given history induces a particular observation history which is observed by the player. Then, strategies are defined as mappings from observation histories to actions, memoryless strategies are strategies where the same action is chosen for any observation history ending with the same observation set and finite-memory strategies are represented by DFSTs where the input symbols are observation sets rather than states of the game. Note that the definitions coincide for complete information games.

4 ATL/ATL^* with finite-memory and bounded-memory semantics

The alternating-time temporal logics ATL and ATL^* generalize the computation tree logics CTL and CTL^* with the strategic operator $\langle\langle A\rangle\rangle\varphi$ which expresses that coalition A has a strategy to ensure the property φ. For a fixed, finite set Agt of agents and finite set Prop of proposition symbols the ATL^* formulas are constructed from the following grammar

$$\varphi ::= p \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \mathbf{X}\varphi_1 \mid \varphi_1\mathbf{U}\varphi_2 \mid \langle\langle A\rangle\rangle\varphi_1$$

where $p \in$ Prop, φ_1, φ_2 are ATL^* formulas and $A \subseteq$ Agt is a coalition of agents. The connectives $\wedge$, $\rightarrow$, $\Leftrightarrow$, **G** and **F** are defined in the usual way. The universal path quantifier **A** of computation tree logic can be defined as $\langle\langle\emptyset\rangle\rangle$. ATL is the subset of ATL^* defined by the following grammar

$$\varphi ::= p \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \langle\langle A\rangle\rangle\mathbf{X}\varphi_1 \mid \langle\langle A\rangle\rangle\mathbf{G}\varphi_1 \mid \langle\langle A\rangle\rangle(\varphi_1\mathbf{U}\varphi_2)$$

where $p \in$ Prop, φ_1, φ_2 are ATL formulas and $A \subseteq$ Agt is a coalition of agents.

We distinguish between state formulas and path formulas, which are evaluated on states and paths of a game respectively. The state formulas are defined as follows

- p is a state formula if $p \in$ Prop
- If φ_1 and φ_2 are state formulas, then $\neg\varphi_1$ and $\varphi_1 \vee \varphi_2$ are state formulas
- If φ_1 is an ATL^* formula and $A \subseteq$ Agt, then $\langle\langle A\rangle\rangle\varphi_1$ is a state formula

All ATL^* formulas are path formulas. Note that all ATL formulas are state formulas.

In [12] ATL and ATL^* are defined with different semantics based on (1) whether the game is with complete or incomplete information (2) whether perfect recall strategies or only memoryless strategies are allowed. Here i and I are used to denote incomplete and complete information respectively. r and R are used to denote memoryless and perfect recall strategies respectively. We extend this framework by considering finite-memory semantics where only finite-memory strategies are allowed and denote this by F. In addition we extend it with an infinite hierarchy of bounded-memory semantics, where F_k for $k \geq 1$ denotes that only k-memory strategies are allowed. We denote the satisfaction relations $\models_{XY}$ where $X \in \{i, I\}$ and $Y \in \{r, F_1, F_2, ..., F, R\}$. We will also write ATL_{XY} and ATL^*_{XY} to denote the logics obtained with the different types of semantics.

The semantics of formulas in alternating-time temporal logic is given with respect to a fixed CGM $\mathcal{M} = (\mathcal{G}, \pi)$ where the players that appear in the formulas must appear in $\mathcal{G}$ and the propositions present in the formulas are in Prop. For state formulas we define for all CGMs $\mathcal{M} = (\mathcal{G}, \pi)$, all states s, all propositions $p \in$ Prop, all state formulas φ_1 and φ_2, all path formulas φ_3, all coalitions $A \in$ Agt and all $Y \in \{r, F_1, F_2, ..., F, R\}$

$\mathcal{M}, s \models_{IY} p$	if $p \in \pi(s)$
$\mathcal{M}, s \models_{IY} \neg\varphi_1$	if $\mathcal{M}, s \not\models_{IY} \varphi_1$
$\mathcal{M}, s \models_{IY} \varphi_1 \vee \varphi_2$	if $\mathcal{M}, s \models_{IY} \varphi_1$ or $\mathcal{M}, s \models_{IY} \varphi_2$
$\mathcal{M}, s \models_{IY} \langle\langle A\rangle\rangle \varphi_3$	if there exist strategies $(\sigma_A)_{a\in A} \in \prod_{a\in A} \mathrm{Strat}_a^Y$ such that $\forall \rho \in \mathrm{Out}_{\mathcal{G}}(s, \sigma_A).\mathcal{M}, \rho \models_{IY} \varphi_3$

For path formulas we define for all CGMs $\mathcal{M} = (\mathcal{G}, \pi)$, all paths ρ, all propositions $p \in \mathrm{Prop}$, all state formulas φ_1, all path formulas φ_2 and φ_3, all coalitions $A \in \mathrm{Agt}$ and all $Y \in \{r, F_1, F_2, ..., F, R\}$

$\mathcal{M}, \rho \models_{IY} \varphi_1$	if $\mathcal{M}, \rho_0 \models_{IY} \varphi_1$
$\mathcal{M}, \rho \models_{IY} \neg\varphi_2$	if $\mathcal{M}, \rho \not\models_{IY} \varphi_2$
$\mathcal{M}, \rho \models_{IY} \varphi_2 \vee \varphi_3$	if $\mathcal{M}, \rho \models_{IY} \varphi_2$ or $\mathcal{M}, \rho \models_{IY} \varphi_3$
$\mathcal{M}, \rho \models_{IY} \mathbf{X}\varphi_2$	if $\mathcal{M}, \rho_{\geq 1} \models_{IY} \varphi_2$
$\mathcal{M}, \rho \models_{IY} \varphi_2 \mathbf{U} \varphi_3$	if $\exists k.\mathcal{M}, \rho_{\geq k} \models_{IY} \varphi_3$ and $\forall j < k.\mathcal{M}, \rho_{\geq j} \models_{IY} \varphi_2$

For iCGMs the semantics are defined similarly, but for $\langle\langle A\rangle\rangle\varphi$ to be true in state s the coalition A must have a strategy to make sure φ is satisfied in all plays starting in states that are indistinguishable from s to one of the players in A. Now, for state formulas we define for all iCGMs $\mathcal{M} = (\mathcal{G}, \pi)$, all states s, all propositions $p \in \mathrm{Prop}$, all state formulas φ_1 and φ_2, all path formulas φ_3, all coalitions $A \in \mathrm{Agt}$ and all $Y \in \{r, F_1, F_2, ..., F, R\}$

$\mathcal{M}, s \models_{iY} p$	if $p \in \pi(s)$
$\mathcal{M}, s \models_{iY} \neg\varphi_1$	if $\mathcal{M}, s \not\models_{iY} \varphi_1$
$\mathcal{M}, s \models_{iY} \varphi_1 \vee \varphi_2$	if $\mathcal{M}, s \models_{iY} \varphi_1$ or $\mathcal{M}, s \models_{iY} \varphi_2$
$\mathcal{M}, s \models_{iY} \langle\langle A\rangle\rangle \varphi_3$	if there exist strategies $(\sigma_A)_{a\in A} \in \prod_{a\in A} \mathrm{Strat}_a^Y$ such that for every $a \in A$, every $s' \sim_a s$ and every $\rho \in \mathrm{Out}_{\mathcal{G}}(s', \sigma_A)$ we have $\mathcal{M}, \rho \models_{iY} \varphi_3$

For path formulas we define for all iCGMs $\mathcal{M} = (\mathcal{G}, \pi)$, all paths ρ, all propositions $p \in \mathrm{Prop}$, all state formulas φ_1, all path formulas φ_2 and φ_3, all coalitions $A \in \mathrm{Agt}$ and all $Y \in \{r, F_1, F_2, ..., F, R\}$

$\mathcal{M}, \rho \models_{iY} \varphi_1$	if $\mathcal{M}, \rho_0 \models_{iY} \varphi_1$
$\mathcal{M}, \rho \models_{iY} \neg\varphi_2$	if $\mathcal{M}, \rho \not\models_{iY} \varphi_2$
$\mathcal{M}, \rho \models_{iY} \varphi_2 \vee \varphi_3$	if $\mathcal{M}, \rho \models_{iY} \varphi_2$ or $\mathcal{M}, \rho \models_{iY} \varphi_3$
$\mathcal{M}, \rho \models_{iY} \mathbf{X}\varphi_2$	if $\mathcal{M}, \rho_{\geq 1} \models_{iY} \varphi_2$
$\mathcal{M}, \rho \models_{iY} \varphi_2 \mathbf{U} \varphi_3$	if $\exists k.\mathcal{M}, \rho_{\geq k} \models_{iY} \varphi_3$ and $\forall j < k.\mathcal{M}, \rho_{\geq j} \models_{iY} \varphi_2$

We will occasionally write $\models_{XY}^{L}$ to emphasize that the semantics is for the logic L, but omit it when the logic is clear from the context as above.

5 Expressiveness

With the new types of semantics introduced we are interested in when the new types of semantics are different and when they are equivalent. For instance, in [12] it was noted that $\models_{Ir}$ and $\models_{IR}$ are equivalent for ATL, but not ATL^*. We do a similar comparison for the different kinds of semantics in order to understand the capabilities of different amounts of memory in different games. In addition, since there is equivalence in some cases this gives us fewer different cases to solve when considering the model-checking problem. We start by looking only at formulas of the form $\langle\langle A\rangle\rangle\varphi$ where $A \subseteq \mathrm{Agt}$ and φ is an

LTL formula. Denote the fragments of ATL and ATL^* restricted to this kind of formulas by ATL_0 and ATL_0^* respectively. A nice property of these fragments is the following proposition, which tells us that to have equivalence of semantics for two types of memory for either ATL or ATL^* it is sufficient to consider the fragments ATL_0 and ATL_0^* respectively.

Proposition 5. *For $X \in \{i,I\}$ and $Y,Z \in \{r,F_1,F_2,...,F,R\}$ we have*

1. $\models_{XY}^{ATL} \; = \; \models_{XZ}^{ATL}$ if and only if $\models_{XY}^{ATL_0} \; = \; \models_{XZ}^{ATL_0}$
2. $\models_{XY}^{ATL^*} \; = \; \models_{XZ}^{ATL^*}$ if and only if $\models_{XY}^{ATL_0^*} \; = \; \models_{XZ}^{ATL_0^*}$

Proof. We treat both cases simultaneously and let $L \in \{ATL, ATL^*\}$. $(\Rightarrow)$ The first direction is trivial, since the set of L_0 formulas is included in the set of L formulas. $(\Leftarrow)$ For the second direction suppose $\models_{XY}^{L_0} \; = \; \models_{XZ}^{L_0}$. Let $\mathscr{M} = (\mathscr{G}, \pi)$ be an (i)CGM over the set Prop of proposition symbols. Let φ be an arbitrary formula from L that contains k strategy quantifiers. Let $\varphi = \varphi_0$ and $\pi = \pi_0$. We transform φ_0 and π_0 in k rounds, in each round $1 \leq j \leq k$ the innermost subformula φ' of φ_{j-1} with a strategy quantifier as main connective is replaced by a new proposion $p_j \notin$ Prop to obtain φ_j. The labeling function is extended such that for all states s we have

$$\pi_j(s) = \begin{cases} \pi_{j-1}(s) \cup \{p_j\} & \text{if } (\mathscr{G}, \pi_{j-1}), s \models_{XY}^{L_0} \varphi' \\ \pi_{j-1}(s) & \text{otherwise} \end{cases}$$

Note that because of our initial assumption we have $(\mathscr{G}, \pi_{j-1}), s \models_{XY} \varphi'$ if and only if $(\mathscr{G}, \pi_{j-1}), s \models_{XZ} \varphi'$ since φ' is an L_0 formula. Therefore, for each j and all paths ρ we also have

$(\mathscr{G}, \pi_{j-1}), \rho \models_{XY} \varphi_{j-1}$ if and only if $(\mathscr{G}, \pi_j), \rho \models_{XY} \varphi_j$ and

$(\mathscr{G}, \pi_{j-1}), \rho \models_{XZ} \varphi_{j-1}$ if and only if $(\mathscr{G}, \pi_j), \rho \models_{XZ} \varphi_j$

In particular, φ_k is an LTL formula and therefore for all ρ we have $(\mathscr{G}, \pi_k), \rho \models_{XY} \varphi_k$ if and only if $(\mathscr{G}, \pi_k), \rho \models_{XZ} \varphi_k$. Together with the above we get for all ρ that

$(\mathscr{G}, \pi_0), \rho \models_{XY} \varphi_0$ iff $(\mathscr{G}, \pi_1), \rho \models_{XY} \varphi_1$ iff ... iff $(\mathscr{G}, \pi_k), \rho \models_{XY} \varphi_k$ iff

$(\mathscr{G}, \pi_k), \rho \models_{XZ} \varphi_k$ iff ... iff $(\mathscr{G}, \pi_1), \rho \models_{XZ} \varphi_1$ iff $(\mathscr{G}, \pi_0), \rho \models_{XZ} \varphi_0$

Thus, $\models_{XY}^{L} \; = \; \models_{XZ}^{L}$ since φ and $\mathscr{M}$ was chosen arbitrarily.

□

The relations between different types of semantics presented in Figure 2 provide insights about the need of memory for winning strategies in games with various amounts of information and types of LTL objectives that can be specified in ATL_0/ATL_0^*. In addition, according to Proposition 5 the cases of equivalence in Figure 2 are exactly the cases of equivalence for the full ATL/ATL^*. We will use the rest of this section to prove the results of this table.

Logic	Expressiveness									
ATL_0 w. complete info	$\models_{Ir}^{ATL_0}$	$=$	$\models_{IF_2}^{ATL_0}$	$=$	$\models_{IF_3}^{ATL_0}$	$=$	$\ldots$	$=$	$\models_{IF}^{ATL_0}$	$=$ $\models_{IR}^{ATL_0}$
ATL_0 w. incomplete info	$\models_{ir}^{ATL_0}$	$\subset$	$\models_{iF_2}^{ATL_0}$	$\subset$	$\models_{iF_3}^{ATL_0}$	$\subset$	$\ldots$	$\subset$	$\models_{iF}^{ATL_0}$	$\subset$ $\models_{iR}^{ATL_0}$
ATL_0^* w. complete info	$\models_{Ir}^{ATL_0^*}$	$\subset$	$\models_{IF_2}^{ATL_0^*}$	$\subset$	$\models_{IF_3}^{ATL_0^*}$	$\subset$	$\ldots$	$\subset$	$\models_{IF}^{ATL_0^*}$	$=$ $\models_{IR}^{ATL_0^*}$
ATL_0^* w. incomplete info	$\models_{ir}^{ATL_0^*}$	$\subset$	$\models_{iF_2}^{ATL_0^*}$	$\subset$	$\models_{iF_3}^{ATL_0^*}$	$\subset$	$\ldots$	$\subset$	$\models_{iF}^{ATL_0^*}$	$\subset$ $\models_{iR}^{ATL_0^*}$

Figure 2: Relations between the different types of semantics

5.1 Complete information games

For complete information games, the question of whether a (memoryless/finite-memory/perfect recall) winning strategy exists for a coalition A can be reduced to the question of whether a (memoryless/finite-memory/perfect recall) winning strategy exists for player 1 in a two-player turn-based game. The idea is to let player 1 control coalition A and let player 2 control coalition $\mathrm{Agt} \setminus A$ and give player 2 information about the action of player 1 before he has to choose in each round of the game in order to make it turn-based. Since ATL_0 can only be used to express reachability ($\langle\langle A \rangle\rangle \varphi_1 \mathbf{U} \varphi_2$), safety ($\langle\langle A \rangle\rangle \mathbf{G} \varphi_1$) and 1-step reachability ($\langle\langle A \rangle\rangle \mathbf{X} \varphi_1$) objectives where no memory is needed for winning strategies [9], it follows that all types of semantics considered are equal in ATL with complete information as noted in [12]. Since ATL_0^* can only be used to express LTL objectives, it follows that $\models_{IF}^{ATL^*} = \models_{IR}^{ATL^*}$ since only finite memory is needed for winning strategies in such games [11].

5.2 The bounded-memory hierarchy

The bounded-memory hierarchy is increasing for ATL_0/ATL_0^* because when a coalition has a k-memory winning strategy, then it also has a $k+1$-memory winning strategy which can be obtained by adding a disconnected memory-state to the DFST representing the strategy. For ATL_0^* with complete information the hierarchy is strict. This can be seen since the family $\varphi_k = \langle\langle \{1\} \rangle\rangle \mathbf{X}^k p$ of formulas for $k \geq 1$ has the property that $\mathcal{M}, s_0 \models_{IF_k} \varphi_k$ and $\mathcal{M}, s_0 \not\models_{IF_{k-1}} \varphi_k$ for $k \geq 2$ for the one-player CGM $\mathcal{M}$ illustrated in Figure 3. Here player 1 wins if he chooses w (wait) the first $k-1$ rounds and then chooses g (go) in the kth round.

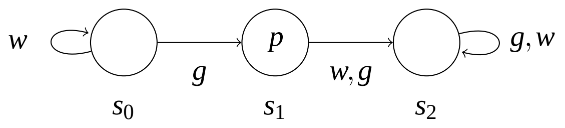


Figure 3: CGM $\mathcal{M}$

The reason that the property $\mathbf{X}^k p$ cannot be forced by player 1 using a $(k-1)$-memory strategy is that the DFST representing the strategy would have to output the action w in the first $k-1$ rounds followed by an output of the action g when reading the same input s_0 in every round. This is not possible, because after $k-1$ rounds there must have been at least one repeated memory-state and from such a repeated state, the DFST would keep repeating its behavior. Therefore, it will either output w forever or output

g before the kth round, making it unable to enforce $\mathbf{X}^k p$. For ATL_0/ATL_0^* with incomplete information, we can show the same result for the formula $\psi = \langle\!\langle\{1\}\rangle\!\rangle\mathbf{F}p$ for the family $\mathscr{M}_k$ of iCGMs illustrated in Figure 4 where $k \geq 1$. In this game all states except s_0 are in the same observation set for player 1. Here we have $\mathscr{M}_k, s_0 \models_{iF_k} \psi$ and $\mathscr{M}_k, s_0 \not\models_{iF_{k-1}} \psi$.

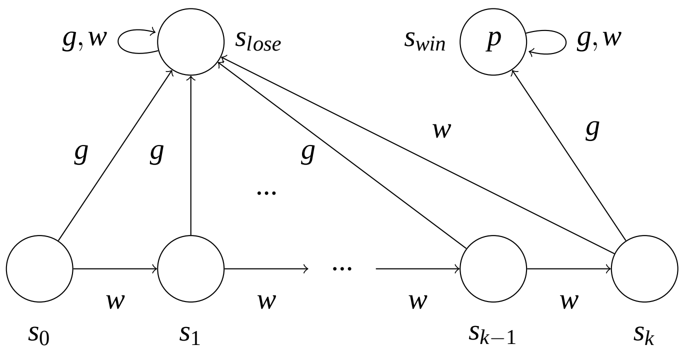


Figure 4: iCGM $\mathscr{M}_k$

Player 1 wins exactly if he chooses w for the first k rounds and then g, which is not possible for a $(k-1)$-memory strategy when it receives the same input symbol in every round after the initial round as in the previous example.

The reason why the bounded-memory hierarchies are not increasing for ATL/ATL^* in general is the possibility of using negation of strategically quantified formulas. For instance, given an ATL_0 formula φ, an iCGM $\mathscr{M}$ and a state s such that $\mathscr{M}, s \models_{iF_k} \varphi$ and $\mathscr{M}, s \not\models_{iF_{k-1}} \varphi$ for some k, then for the ATL formula $\neg\varphi$ we have $\mathscr{M}, s \not\models_{iF_k} \neg\varphi$ and $\mathscr{M}, s \models_{iF_{k-1}} \neg\varphi$.

5.3 Infinite memory is needed

Finally, infinite memory is actually needed in some cases for ATL_0/ATL_0^* with incomplete information. This is shown in a slightly different framework in [4] where an example of a game is given with initial state s_0 such that $\mathscr{M}, s_0 \models_{iR} \langle\!\langle\{1,2\}\rangle\!\rangle\mathbf{G}\neg p$ and $\mathscr{M}, s_0 \not\models_{iF} \langle\!\langle\{1,2\}\rangle\!\rangle\mathbf{G}\neg p$ for a proposition p. We will not repeat the example here, but in the undecidability proof in Section 6.3 an example of such a game is given. This means that $\models_{iF}^{L} \neq \models_{iR}^{L}$ for $L \in \{ATL_0, ATL_0^*\}$. We have $\models_{iF}^{L} \subseteq \models_{iR}^{L}$ since all finite-memory strategies are perfect recall strategies and therefore $\models_{iF}^{L} \subset \models_{iR}^{L}$ which concludes the last result of Figure 2.

6 Model-checking

In this section we look at the decidability and complexity of model-checking ATL/ATL^* with the new semantics introduced and compare with the results for memoryless and perfect recall semantics. We adopt the same way of measuring input size as in [2, 3, 12, 10] where the input is measured as the size of the game structure and the size of the formula to be checked. In the case of bounded-memory semantics, we also include in the input size the size of the memory-bound k encoded in unary. Our results can be seen in Figure 5 along with known results for memoryless and perfect recall semantics.

As can be seen in the figure, we obtain the same complexity for bounded-memory semantics as for memoryless semantics in all the cases which is positive, since we can solve many more games while

	ATL	ATL^*
$\models_{Ir}$	$PTIME$ [3]	$PSPACE$ [12]
$\models_{IF_k}$	$PTIME$	$PSPACE$
$\models_{IF}$	$PTIME$	$2EXPTIME$
$\models_{IR}$	$PTIME$ [3]	$2EXPTIME$ [3]

	ATL	ATL^*
$\models_{ir}$	Δ_2^p [12, 10]	$PSPACE$ [12]
$\models_{iF_k}$	Δ_2^p	$PSPACE$
$\models_{iF}$	Undecidable	Undecidable
$\models_{iR}$	Undecidable [3, 8]	Undecidable [3, 8]

Figure 5: Model-checking complexity for ATL, ATL^*. All complexity results are completeness results.

staying in the same complexity class. We also obtain the same complexity for finite-memory semantics as perfect recall semantics, including undecidability for incomplete information games, which is disappointing. We will use the rest of the section to prove these results. In many cases this is done by using known results and techniques and modifying them slightly as well as using the results from Section 5.

6.1 Using expressiveness results

In section 5 it was shown that $\models_{Ir}^{ATL} = \models_{IF_2}^{ATL} = \models_{IF_3}^{ATL} = ... = \models_{IF}^{ATL}$ which means that the model-checking problem is the same for these cases. Since $\models_{Ir}^{ATL}$ is known to be $PTIME$-complete [3] the result is the same for finite-memory semantics and bounded-memory semantics. It was also shown that $\models_{IF}^{ATL^*} = \models_{IR}^{ATL^*}$. Since model-checking ATL_{IR}^* is $2EXPTIME$-complete [3] so is model-checking ATL_{IF}^* since it is the same problem.

6.2 Bounded-memory semantics

For model-checking $ATL_{iF_k}, ATL_{IF_k}^*$ and $ATL_{iF_k}^*$ we employ some of the same ideas as in [12] for memoryless semantics, but extend them to deal with bounded-memory strategies. We first consider model-checking ATL_0^* formulas with iF_k semantics. Model-checking an ATL_0^* formula $\langle\langle A\rangle\rangle\varphi$ in an iCGM $\mathscr{M} = (\mathscr{G}, \pi)$ with $\mathscr{G} = (\text{States}, \text{Agt}, \text{Act}, \text{Mov}, \text{Tab}, (\sim_j)_{1\leq j\leq n})$ and initial state s_0 can be done using non-determinism as follows. First, assume without loss of generality that $A = \{1, ..., r\}$ with $r \leq n$. Use non-determinism to guess a k-memory strategy $\sigma = (\sigma_j)_{1\leq j\leq r}$ for each of the players in A represented by DFSTs $\mathscr{A}_j = (M_j, m_{j0}, []_j, \text{Act}, T_j, G_j)$ for $j \in A$. Check that this strategy enforces φ by creating a labelled and initialized transition system $T(s_0', \sigma) = (Q, R, L, q_0)$ for all $s_0' \sim_j s_0$ for some $1 \leq j \leq r$ in which the set of paths corresponds to the σ-outcomes from s_0' in $\mathscr{G}$. The set Q of states, the transition relation $R \subseteq Q \times Q$, the labeling function $L : Q \in 2^{\text{Prop}}$ and the initial state q_0 are constructed as follows.

- $Q = \text{States} \times \prod_{j=1}^{r} M_j$
- $([s, (m_1, ..., m_r)], [s', (m_1', ..., m_r')]) \in R$ if and only if there exists $a_{r+1}, ..., a_n \in \text{Act}$ so
  - $\text{Tab}(s, (G_1(m_1, [s]_1), ..., G_r(m_r, [s]_r), a_{r+1}, ..., a_n)) = s'$ and
  - $T_j(m_j, [s]_j) = m_j'$ for $1 \leq j \leq r$
- $L(s, (m_1, ..., m_r)) = \pi(s)$ for all $(s, (m_1, ..., m_r)) \in Q$

- $q_0 = (s'_0, (m_{10}, ..., m_{r0}))$

Intuitively, each state in the transition system corresponds to a state of the game as well as possible combinations of memory values for players in A. It can then be shown that $\rho = \rho_0\rho_1...$ is a σ-outcome in $\mathscr{G}$ from $\rho_0 = s'_0$ if and only if there exists $(m_{1j}, ..., m_{rj}) \in \prod_{j=1}^{r} M_r$ for $j \geq 0$ such that $\rho' = (\rho_0, (m_{10}, ..., m_{r0}))(\rho_1, (m_{11}, ..., m_{r1}))...$ is a path in $T(s'_0, \sigma)$. This means that σ is a witness that $\mathscr{M}, s_0 \models_{iF_k} \langle\!\langle A \rangle\!\rangle \varphi$ if and only if $T(s'_0, \sigma), q_0 \models_{CTL^*} \mathbf{A}\varphi$ for all $s'_0 \sim_j s_0$ for some $1 \leq j \leq r$. Note that the size of the transition systems are polynomial in the size of the input because $|Q| = k^r$, the number n of agents is fixed and $r \leq n$. In addition, the transition systems $T(s_1, \sigma)$ and $T(s_2, \sigma)$ are equal for any $s_1, s_2 \in$ States except for the initial state of the transition systems. Thus, we can use the same transition system to do the check for the different initial states. We can perform this check of a strategy σ in *PSPACE* since CTL^* model-checking can be done in *PSPACE* [7]. Moreover, when $\langle\!\langle A \rangle\!\rangle \varphi$ is an ATL_0 formula, the check can be done in $PTIME$ since CTL model-checking can be done in $PTIME$ [6]. Thus, we can do model-checking of ATL_0 and ATL_0^* with iF_k semantics in *NP* and *PSPACE* respectively.

We extend the above algorithm to full ATL and ATL^* by evaluating the strategically quantified subformulas in a bottom up fashion, starting with the innermost formula and moving outwards resembling the technique typically used in CTL^* model-checking [7]. In both cases we need to make a linear amount of calls to the ATL_0/ATL_0^* algorithm in the size of the formula to be checked. This gives us a $\Delta_2^p = P^{NP}$ algorithm and a *PSPACE* algorithm in ATL and ATL^* respectively. Since ATL^* with IF_k semantics is a special case, the *PSPACE* algorithm also works here. The *PSPACE*-hardness for $ATL^*_{iF_k}$ and $ATL^*_{IF_k}$ follows from *PSPACE*-hardness of ATL^*_{Ir} [12] since this is a special case of the two. In the same way Δ_2^p-hardness of ATL_{iF_k} follows from Δ_2^p hardness of ATL_{ir} [10].

6.3 Undecidability of finite-memory semantics

In [8] it was proven that model-checking ATL and ATL^* with *iR* semantics is undecidable, even for as simple a formula as $\langle\!\langle A \rangle\!\rangle \mathbf{G} \neg p$ for $n \geq 3$ players. We provide a proof sketch for the same result for *iF* semantics inspired by a technique from [4] which also illustrates that infinite memory is needed in some games. The idea is to reduce the problem of whether a deterministic Turing machine with a semi-infinite tape that never writes the blank symbol repeats some configuration twice when started with an empty input tape, with the convention that the Turing machine will keep looping in a halting configuration forever if a halting state is reached. This problem is undecidable since the halting problem can be reduced to it. From a given Turing machine $T = (Q, q_0, \Sigma, \delta, B, F)$ of this type where Q is the set of states, q_0 is the initial state, Σ is the tape alphabet, $\delta : Q \times (\Sigma \cup \{B\}) \to Q \times \Sigma \times \{L, R\}$ is the transition function, B is the blank symbol and F is the set of accepting states, we generate a three-player concurrent game model $\mathscr{M}_T = (\mathscr{G}_T, \pi_T)$ with a state s_0 such that $\mathscr{M}_T, s_0 \models_{iF} \langle\!\langle \{1,2\} \rangle\!\rangle \mathbf{G} \neg p$ if and only if T repeats some configuration twice.

Consider the three-player game $\mathscr{M}_T$ in Figure 6. To make the figure more simple, we only write the actions of player 1 and 2 along edges and let player 3 choose a successor state, given the choices of player 1 and 2. If player 1 and 2 choose an action tuple that is not present on an edge from the current state of the game, the play goes to a sink state where p is true. In all other states p is false. Both player 1 and 2 have three observation sets, which are denoted 0, $\cdot$ and I (though, they are not equal for the two players). In the figure we write $x \mid y$ in a state if the state is in observation set x for player 1 and y for player 2. The play starts in s_0 which is the only state in observation set 0 for both player 1 and 2. The rules of the game are such that player 3 can choose when to let player 1 receive observation I. He can also choose to either let player 2 receive observation I at the same time as player 1 or let him receive it in the immediately following state of the game. Both player 1 and 2 can observe I at most once during the game. It can be seen from the game graph that both player 1 and 2 must play action a until they receive observation I in order not to lose. We design the game so they must play the vth configuration of the Turing machine T when receiving observation I after v rounds in a winning strategy for all $v \geq 1$. To do this we let the tape alphabet and the set of control states of T be legal actions for player 1 and 2. By playing a configuration, we mean playing the contents of the non-blank part of the tape of T one symbol at a time from left to right and playing the control state immediately before the content of the cell that the tape head points to.

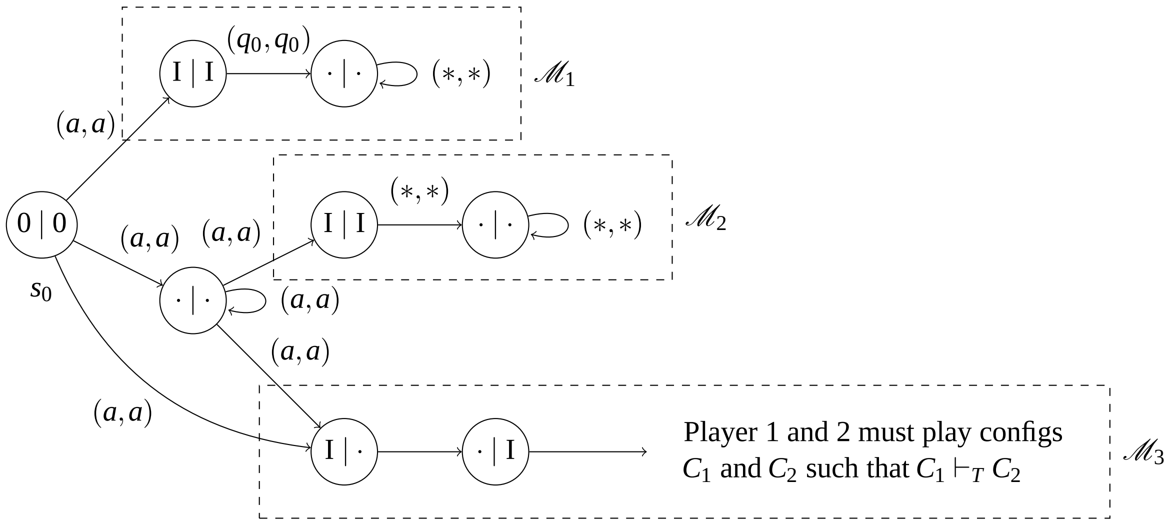


Figure 6: iCGM $\mathscr{M}_T$

We design $\mathscr{M}_T$ with three modules $\mathscr{M}_1, \mathscr{M}_2$ and $\mathscr{M}_3$ as shown in Figure 6. They are designed with the following properties

- $\mathscr{M}_1$ is designed such that when player 1 and 2 both observe I after the first round, then in a winning strategy they must both play the initial configuration (i.e. q_0) in order to maintain $\neg p$. If they don't, then player 3 has a counter-strategy that takes the play to $\mathscr{M}_1$.
- $\mathscr{M}_2$ is designed such that when player 1 and 2 both observe I at the same time, then in a winning strategy they must both play the same sequence of symbols after observing I ($*$ stands for any action and $(*,*)$ means any action pair where the two actions are equal). If there is a number $r > 1$ so they don't comply with this when observing I after round r, then player 3 has a counter-strategy that takes the play to $\mathscr{M}_2$ after round r.
- $\mathscr{M}_3$ is designed such that if player 1 observes I in the round before player 2 does, then in a winning strategy they must player configurations C_1 and C_2 respectively such that $C_1 \vdash_T C_2$ where $\vdash_T$ is the successor relation for configurations of T. Due to space limitations, the specific design of this module is omitted here.

Now, suppose T has a repeated configuration. Then player 1 and 2 have a winning strategy σ that consists in both players playing the jth configuration of the run of T when observing I after the jth round. This strategy is winning because no matter if player 3 chooses to go to module $\mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3$ or none of them, then $\neg p$ will always hold given how they are designed when player 1 and 2 play according to σ. Next, the sequence of configurations in the run of T is of the form $\pi \cdot \tau^{\omega}$ where π and τ are finite sequences of configurations since T has a repeated configuration. Then, player 1 and 2 only need finite memory to play according to σ since they only need to remember a finite number of configurations and how far on the periodic path $\pi \cdot \tau^{\omega}$ the play is. Thus, they have a finite-memory winning strategy.

Suppose on the other hand that T does not have a repeated configuration and assume for contradiction that player 1 and 2 have a k-memory winning strategy σ for some k. Since player 1 and 2 cannot see whether the play is in $\mathscr{M}_1, \mathscr{M}_2$ or $\mathscr{M}_3$ player 1 must, when playing according to σ, play the first configuration D_1 of the run of T when observing I after the first round. Otherwise, player 3 has a counter-strategy taking the play to $\mathscr{M}_1$ after the first round. Then, player 2 must play the second configuration D_2 of the run of T when observing I after the second round. Otherwise, player 3 has a counter-strategy taking the play to $\mathscr{M}_3$ after the first round since player 1 must play D_1 when observing I after the first round and player 2 must play a successor configuration of what player 1 plays. Next, when using σ, player 1 must play D_2 when observing I after the second round. Otherwise, player 3 has a counter-strategy that takes the play to $\mathscr{M}_2$ after the second round since player 2 plays D_2 when observing I after the second round. Repeating this argument, it can be seen that σ must consist of player 1 and 2 playing the jth configuration of the run of T when observing I after the jth round for all $j \geq 1$. However, this is not possible for a k-memory strategy when the run of T does not have a repeated configuration. This is because the current memory value of the DFST representing the strategy at the point when I is observed determines which sequence of symbols the strategy will play (since it will receive the same input symbol for the rest of the game). Thus, it is not capable of playing more than k different configurations. And since for any k a winning strategy must be able to play more than k different configurations there is a contradiction and a finite-memory winning strategy therefore cannot exist.

In conclusion $\mathscr{M}_T, s_0 \models_{iF} \langle\langle\{1,2\}\rangle\rangle \mathbf{G} \neg p$ if and only if T repeats some configuration twice, which means that the model-checking problem is undecidable for ATL and ATL^* with *iF* semantics. This game also illustrates that infinite memory is needed in some games, since player 1 and 2 can win the game with perfect recall strategies when T does not have a repeated configuration. This is simply done by playing the sequence of configurations of the run of T.

7 Concluding Remarks

We have motivated the extension of the alternating-time temporal logics ATL/ATL^* with bounded-memory and finite-memory semantics and have explored the expressiveness for both complete and incomplete information games. Both finite-memory semantics and the infinite hierarchy of bounded-memory semantics were shown to be different from memoryless and perfect recall semantics. We have also obtained complexity and decidability results for the model-checking problems that emerged from the newly introduced semantics. In particular, the model-checking results for bounded-memory semantics were positive with as low a complexity as for memoryless semantics for ATL/ATL^* and complete/incomplete information games. Unfortunately model-checking with finite-memory semantics was shown to be as hard as with perfect recall semantics in the cases considered, even though it was shown to be a different problem.

References

[1] Thomas Ågotnes & Dirk Walther (2009): *A Logic of Strategic Ability Under Bounded Memory*. *Journal of Logic, Language and Information* 18(1), pp. 55–77, doi:`10.1007/s10849-008-9075-4`.

[2] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (1997): *Alternating-time Temporal Logic*. In: *FOCS*, pp. 100–109, doi:`10.1109/SFCS.1997.646098`.

[3] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-time temporal logic*. *J. ACM* 49(5), pp. 672–713, doi:`10.1145/585265.585270`.

[4] Dietmar Berwanger & Lukasz Kaiser (2010): *Information Tracking in Games on Graphs*. *Journal of Logic, Language and Information* 19(4), pp. 395–412, doi:`10.1007/s10849-009-9115-8`.

[5] Thomas Brihaye, Arnaud Da Costa Lopes, François Laroussinie & Nicolas Markey (2009): *ATL with Strategy Contexts and Bounded Memory*. In: *LFCS*, pp. 92–106, doi:`10.1007/978-3-540-92687-0_7`.

[6] Edmund M. Clarke & E. Allen Emerson (1981): *Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic*. In: *Logic of Programs*, pp. 52–71, doi:`10.1007/BFb0025774`.

[7] Edmund M. Clarke, E. Allen Emerson & A. Prasad Sistla (1986): *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications*. *ACM Trans. Program. Lang. Syst.* 8(2), pp. 244–263. Available at `http://doi.acm.org/10.1145/5397.5399`.

[8] Catalin Dima & Ferucio Laurentiu Tiplea (2011): *Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable*. *CoRR* abs/1102.4225. Available at `http://arxiv.org/abs/1102.4225`.

[9] E. Allen Emerson & Charanjit S. Jutla (1991): *Tree Automata, Mu-Calculus and Determinacy (Extended Abstract)*. In: *FOCS*, pp. 368–377, doi:`10.1109/SFCS.1991.185392`.

[10] Wojciech Jamroga & Jürgen Dix (2008): *Model Checking Abilities of Agents: A Closer Look*. *Theory Comput. Syst.* 42(3), pp. 366–410, doi:`10.1007/s00224-007-9080-z`.

[11] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of a Reactive Module*. In: *POPL*, pp. 179–190, doi:`10.1145/75277.75293`.

[12] Pierre-Yves Schobbens (2004): *Alternating-time logic with imperfect recall*. *Electr. Notes Theor. Comput. Sci.* 85(2), pp. 82–93, doi:`10.1016/S1571-0661(05)82604-0`.

Satisfiability of ATL with strategy contexts

François Laroussinie
LIAFA – Univ. Paris Diderot & CNRS
francoisl@liafa.univ-paris-diderot.fr

Nicolas Markey
LSV – ENS Cachan & CNRS
markey@lsv.ens-cachan.fr

Various extensions of the temporal logic ATL have recently been introduced to express rich properties of multi-agent systems. Among these, ATL_{sc} extends ATL with *strategy contexts*, while Strategy Logic has *first-order quantification* over strategies. There is a price to pay for the rich expressiveness of these logics: model-checking is non-elementary, and satisfiability is undecidable.

We prove in this paper that satisfiability is decidable in several special cases. The most important one is when restricting to *turn-based* games. We prove that decidability also holds for concurrent games if the number of moves available to the agents is bounded. Finally, we prove that restricting strategy quantification to memoryless strategies brings back undecidability.

1 Introduction

Temporal logics are a convenient tool to reason about computerised systems, in particular in the setting of verification [Pnu77, CE82, QS82]. When systems are interactive, the models usually involve several agents (or players), and relevant properties to be checked often question the existence of *strategies* for these agents to achieve their goals. To handle these, *alternating-time temporal logic* was introduced, and its algorithmic properties were studied: model checking is PTIME-complete [AHK02], while satisfiability was settled EXPTIME-complete [WLWW06].

While model checking is tractable, ATL still suffers from a lack of expressiveness. Over the last five years, several extensions or variants of ATL have been developed, among which ATL *with strategy contexts* [BDLM09] and *Strategy Logic* [CHP07, MMV10]. The model-checking problem for these logics has been proved non-elementary [DLM10, DLM12], while satisfiability is undecidable, both when looking for finite-state or infinite-state models [MMV10, TW12]. Several fragments of these logics have been defined and studied, with the aim of preserving a rich expressiveness and at the same time lowering the complexity of the decision problems [WHY11, MMPV12, HSW13].

In this paper we prove that satisfiability is decidable (though with non-elementary complexity) for the full logic ATL_{sc} (and SL) in two important cases: first, when satisfiability is restricted to turn-based games (this solves a problem left open in [MMV10] for SL), and second, when the number of moves available to the players is bounded. We also consider a third variation, where quantification is restricted to *memoryless* strategies; in that setting, the satisfiability problem is proven undecidable, even for turn-based games.

Our results heavily rely on a tight connection between ATL_{sc} and QCTL [DLM12], the extension of CTL with quantification over atomic propositions. For instance, the QCTL formula $\exists p.\ \varphi$ states that it is possible to label the unwinding of the model under consideration with proposition p in such a way that φ holds. This labeling with additional proposition allows us to mark the strategies of the agents and the model-checking problem for ATL_{sc} can then be reduced to the model-checking problem for QCTL.

This work was partly supported by ERC Starting grant EQualIS (308087) and by European project Cassting (FP7-ICT-601148).

However, in this transformation, the resulting QCTL formula depends both on the ATL_{sc} formula to be checked and on the game where the formula is being checked. This way, the procedure does not extend to satisfiability, which is actually undecidable. We prove here that this difficulty can be overcome when considering turn-based games, or when the number of available moves is fixed. The satisfiability problem for ATL_{sc} is then reduced to the satisfiability problem for QCTL, which we proved decidable (with non-elementary complexity) in [LM13]. When restricting to memoryless strategies, a similar reduction to QCTL exists, but in a setting where the quantified atomic propositions directly label the model, instead of its unwinding. The satisfiability problem for QCTL under that semantics is undecidable [Fre01, LM13], and we adapt the proof of that result to show that satisfiability of ATL^0_{sc} (in which quantification is restricted to memoryless strategies) is also undecidable.

2 Definitions

2.1 ATL with strategy contexts

In this section, we define the framework of concurrent game structures, and define the logic ATL with strategy contexts. We fix once and for all a set AP of atomic propositions.

Definition 1. *A* Kripke structure *$\mathscr{S}$ is a* 3*-tuple $\langle Q,R,\ell\rangle$ where Q is a countable set of states, $R \subseteq Q^2$ is a total relation (*i.e.*, for all $q \in Q$, there is $q' \in Q$ s.t. $(q,q') \in R$) and $\ell \colon Q \to 2^{\mathsf{AP}}$ is a labelling function.*

A path in a Kripke structure $\mathscr{S}$ is a mapping $\rho \colon \mathbb{N} \to Q$ such that $(\rho(i),\rho(i+1)) \in R$ for all i. We write $\mathsf{first}(\rho) = \rho(0)$. Given a path ρ and an integer i, the i-th suffix of ρ, is the path $\rho_{\geq i} \colon n \mapsto \rho(i+n)$; the i-th prefix of ρ, denoted $\rho_{\leq i}$, is the finite sequence made of the $i+1$ first state of ρ. We write $\mathsf{Exec}^{\mathsf{f}}(q)$ for the set of finite prefixes of paths (or *histories*) with first state q. We write $\mathsf{last}(\pi)$ for the last state of a history π. Given a history $\rho_{\leq i}$ and a path π such that $\mathsf{last}(\rho_{\leq i}) = \mathsf{first}(\pi)$, the concatenation $\lambda = \rho_{\leq i} \cdot \pi$ is defined by $\lambda(j) = \rho(j)$ when $j \leq i$ and $\lambda(j) = \pi(j-i)$ when $j > i$.

Definition 2 ([AHK02])**.** *A* Concurrent Game Structure *(*CGS*) $\mathscr{C}$ is a* 7*-tuple $\langle Q,R,\ell,\mathsf{Agt},\mathscr{M},\mathsf{Mov},\mathsf{Edge}\rangle$ where: $\langle Q,R,\ell\rangle$ is a (possibly infinite-state) Kripke structure, $\mathsf{Agt} = \{a_1,\ldots,a_p\}$ is a finite set of* agents*, $\mathscr{M}$ is a non-empty set of moves, $\mathsf{Mov} \colon Q \times \mathsf{Agt} \to \mathscr{P}(\mathscr{M}) \smallsetminus \{\varnothing\}$ defines the set of available moves of each agent in each state, and $\mathsf{Edge} \colon Q \times \mathscr{M}^{\mathsf{Agt}} \to R$ is a transition table associating, with each state q and each set of moves of the agents, the resulting transition departing from q.*

The size of a CGS $\mathscr{C}$ is $|Q| + |\mathsf{Edge}|$. For a state $q \in Q$, we write $\mathsf{Next}(q)$ for the set of all states reachable by the possible moves from q, and $\mathsf{Next}(q,a_j,m_j)$, with $m_j \in \mathsf{Mov}(q,a_j)$, for the restriction of $\mathsf{Next}(q)$ to possible transitions from q when player a_j plays move m_j. We extend Mov and Next to coalitions (*i.e.*, sets of agents) in the natural way. We say that a CGS is *turn-based* when each state q is controlled by a given agent, called the owner of q (and denoted $\mathsf{Own}(q)$). In other terms, for every $q \in Q$, for any two move vectors m and m' in which $\mathsf{Own}(q)$ plays the same move, it holds $\mathsf{Edge}(q,m) = \mathsf{Edge}(q,m')$ (which can be achieved by letting the sets $\mathsf{Mov}(q,a)$ be singletons for every $a \neq \mathsf{Own}(q)$).

A *strategy* for some player $a_i \in \mathsf{Agt}$ in a CGS $\mathscr{C}$ is a function f_i that maps any history to a possible move for a_i, *i.e.*, satisfying $f_i(\pi) \in \mathsf{Mov}(\mathsf{last}(\pi),a_i)$. A strategy f_i is *memoryless* if $f_i(\pi) = f_i(\pi')$ whenever $\mathsf{last}(\pi) = \mathsf{last}(\pi')$. A strategy for a coalition A is a mapping assigning a strategy to each agent in A. The set of strategies for A is denoted $\mathsf{Strat}(A)$. The *domain* $\mathsf{dom}(F_A)$ of $F_A \in \mathsf{Strat}(A)$ is A. Given a coalition B, the strategy $(F_A)_{|B}$ (resp. $(F_A)_{\smallsetminus B}$) denotes the restriction of F_A to the coalition $A \cap B$ (resp. $A \smallsetminus B$). Given two strategies $F \in \mathsf{Strat}(A)$ and $F' \in \mathsf{Strat}(B)$, we define $F \circ F' \in \mathsf{Strat}(A \bigcup B)$ as $(F \circ F')_{|a_j}(\rho) = F_{|a_j}(\rho)$ (resp. $F'_{|a_j}(\rho)$) if $a_j \in A$ (resp. $a_j \in B \smallsetminus A$).

Let ρ be a history. A strategy $F_A = (f_j)_{a_j \in A}$ for some coalition A induces a set of paths from ρ, called the *outcomes* of F_A after ρ, and denoted $\mathsf{Out}(\rho, F_A)$: an infinite path $\pi = \rho \cdot q_1 q_2 \ldots$ is in $\mathsf{Out}(\rho, F_A)$ if, and only if, writing $q_0 = \mathsf{last}(\rho)$, for all $i \geq 0$ there is a set of moves $(m_k^i)_{a_k \in \mathsf{Agt}}$ such that $m_k^i \in \mathsf{Mov}(q_i, a_k)$ for all $a_k \in \mathsf{Agt}$, $m_k^i = f_k(\pi_{|\rho|+i})$ if $a_k \in A$, and q_{i+1} is the unique element of $\mathsf{Next}(q_i, \mathsf{Agt}, (m_k^i)_{a_k \in \mathsf{Agt}})$. Also, given a history ρ and a strategy $F_A = (f_j)_{a_j \in A}$, the strategy F_A^ρ is the sequence of strategies $(f_j^\rho)_{a_j \in A}$ such that $f_j^\rho(\pi) = f_j(\rho \cdot \pi)$, assuming $\mathsf{last}(\rho) = \mathsf{first}(\pi)$.

We now introduce the extension of ATL with strategy contexts [BDLM09, DLM10]:

Definition 3. *Given a set of atomic propositions* AP *and a set of agents* Agt*, the syntax of* ATL^*_{sc} *is defined as follows (where p ranges over AP and A over* 2^{Agt}*):*

$$\mathsf{ATL}^*_{sc} \ni \varphi_{\mathsf{state}}, \psi_{\mathsf{state}} ::= p \mid \neg\,\varphi_{\mathsf{state}} \mid \varphi_{\mathsf{state}} \vee \psi_{\mathsf{state}} \mid \cdot\rangle A \langle\cdot\, \varphi_{\mathsf{state}} \mid \langle\cdot A \cdot\rangle\, \varphi_{\mathsf{path}}$$
$$\varphi_{\mathsf{path}}, \psi_{\mathsf{path}} ::= \varphi_{\mathsf{state}} \mid \neg\,\varphi_{\mathsf{path}} \mid \varphi_{\mathsf{path}} \vee \psi_{\mathsf{path}} \mid \mathbf{X}\,\varphi_{\mathsf{path}} \mid \varphi_{\mathsf{path}}\,\mathbf{U}\,\psi_{\mathsf{path}}.$$

That a (state or path) formula φ is satisfied at a position i of a path ρ of a CGS $\mathscr{C}$ under a strategy context $F \in \mathsf{Strat}(B)$ (for some coalition B), denoted $\mathscr{C}, \rho, i \models_F \varphi$, is defined as follows (omitting atomic propositions and Boolean operators):

$$\begin{array}{rll}
\mathscr{C}, \rho, i \models_F \cdot\rangle A \langle\cdot\, \varphi_{\mathsf{state}} & \text{iff} & \mathscr{C}, \rho, i \models_{F_{\setminus A}} \varphi_{\mathsf{state}} \\
\mathscr{C}, \rho, i \models_F \langle\cdot A \cdot\rangle\, \varphi_{\mathsf{path}} & \text{iff} & \exists F_A \in \mathsf{Strat}(A).\ \forall \rho' \in \mathsf{Out}(\rho_{\leq i}, F_A \circ F).\ \mathscr{C}, \rho', i \models_{F_A \circ F} \varphi_{\mathsf{path}} \\
\mathscr{C}, \rho, i \models_F \mathbf{X}\,\varphi_{\mathsf{path}} & \text{iff} & \mathscr{C}, \rho, i+1 \models_F \varphi_{\mathsf{path}} \\
\mathscr{C}, \rho, i \models_F \varphi_{\mathsf{path}}\,\mathbf{U}\,\psi_{\mathsf{path}} & \text{iff} & \exists j \geq 0.\ \mathscr{C}, \rho, i+j \models_F \psi_{\mathsf{path}} \text{ and } \forall 0 \leq k < j.\ \mathscr{C}, \rho, i+k \models_F \varphi_{\mathsf{path}}
\end{array}$$

Notice how the (existential) strategy quantifier contains an implicit universal quantification over the set of outcomes of the selected strategies. Also notice that state formulas do not really depend on the selected path: indeed one can easily show that

$$\mathscr{C}, \rho, i \models_F \varphi_{\mathsf{state}} \quad \text{iff} \quad \mathscr{C}, \rho', j \models_{F'} \varphi_{\mathsf{state}}$$

where we assume $\rho(i) = \rho'(j)$ and where F and F' verifies: $F(\rho_{\leq i} \cdot \rho'') = F'(\rho'_{\leq j} \cdot \rho'')$ for any finite ρ'' starting in $\rho(i)$. In particular this is the case when the $\rho_{\leq i} = \rho'_{\leq j}$ and $F = F'$.

In the sequel we equivalently write $\mathscr{C}, \pi(0) \models_F \varphi_{\mathsf{state}}$ in place of $\mathscr{C}, \pi, 0 \models_F \varphi_{\mathsf{state}}$ when dealing with state formulas.

For convenience, in the following we allow the construct $\langle\cdot A \cdot\rangle\, \varphi_{\mathsf{state}}$, defining it as a shorthand for $\langle\cdot A \cdot\rangle \perp \mathbf{U}\, \varphi_{\mathsf{state}}$. We also use the classical modalities $\mathbf{F}$ and $\mathbf{G}$, which can be defined using $\mathbf{U}$. Also, $[\cdot A \cdot]\, \varphi_{\mathsf{path}} = \neg \langle\cdot A \cdot\rangle \neg \varphi_{\mathsf{path}}$ expresses that any A-strategy has at least one outcome where φ_{path} holds.

The fragment ATL_{sc} of ATL^*_{sc} is defined as usual, by restricting the set of path formulas to

$$\varphi_{\mathsf{path}}, \psi_{\mathsf{path}} ::= \neg\,\varphi_{\mathsf{path}} \mid \mathbf{X}\,\varphi_{\mathsf{state}} \mid \varphi_{\mathsf{state}}\,\mathbf{U}\,\psi_{\mathsf{state}}.$$

It was proved in [BDLM09] that ATL_{sc} is actually as expressive as ATL^*_{sc}. Moreover, for any given set of players, any ATL_{sc} formula can be written without using negation in path formulas, replacing for instance $\langle\cdot A \cdot\rangle\, \mathbf{G}\,\varphi$ with $\langle\cdot A \cdot\rangle \neg \langle\cdot \mathsf{Agt} \setminus (A \cup B) \cdot\rangle\, \mathbf{F} \neg \varphi$, where B is the domain of the context in which the formula is being evaluated. While this is not a generic equivalence (it depends on the context and on the set of agents), it provides a way of removing negation from any given ATL_{sc} formula.

2.2 Quantified CTL

In this section, we introduce QCTL, and define its *tree semantics*.

Definition 4. *Let Σ be a finite alphabet, and S be a (possibly infinite) set of directions. A* Σ-labelled S-tree *is a pair $\mathscr{T} = \langle T, l\rangle$, where $T \subseteq S^*$ is a non-empty set of finite words on S s.t. for any non-empty word $n = m \cdot s$ in T with $m \in S^*$ and $s \in S$, the word m is also in T; and $l\colon T \to \Sigma$ is a labelling function.*

The *unwinding* (or *execution tree*) of a Kripke structure $\mathscr{S} = \langle Q, R, \ell\rangle$ from a state $q \in Q$ is the 2^{AP}-labelled Q-tree $\mathscr{T}_{\mathscr{S}}(q) = \langle \mathsf{Exec}^{\mathsf{f}}(q), \ell_{\mathscr{T}}\rangle$ with $\ell_{\mathscr{T}}(q_0 \cdots q_i) = \ell(q_i)$. Note that $\mathscr{T}_{\mathscr{S}}(q) = \langle \mathsf{Exec}^{\mathsf{f}}(q), \ell_{\mathscr{T}}\rangle$ can be seen as an (infinite-state) Kripke structure where the set of states is $\mathsf{Exec}^{\mathsf{f}}(q)$, labelled according to $\ell_{\mathscr{T}}$, and with transitions $(m, m\cdot s)$ for all $m \in \mathsf{Exec}^{\mathsf{f}}(q)$ and $s \in Q$ s.t. $m \cdot s \in \mathsf{Exec}^{\mathsf{f}}(q)$.

Definition 5. *For $P \subseteq \mathsf{AP}$, two 2^{AP}-labelled trees $\mathscr{T} = \langle T, \ell\rangle$ and $\mathscr{T}' = \langle T', \ell'\rangle$ are* P-equivalent *(denoted by $\mathscr{T} \equiv_P \mathscr{T}'$) whenever $T = T'$, and $\ell(n) \cap P = \ell'(n) \cap P$ for any $n \in T$.*

In other terms, $\mathscr{T} \equiv_P \mathscr{T}'$ if $\mathscr{T}'$ can be obtained from $\mathscr{T}$ by modifying the labelling function of $\mathscr{T}$ for propositions not in P. We now define the syntax and semantics of QCTL^*:

Definition 6. *The syntax of* QCTL^* *is defined by the following grammar:*

$$\mathsf{QCTL}^* \ni \varphi_{\mathsf{state}}, \psi_{\mathsf{state}} ::= p \mid \neg\,\varphi_{\mathsf{state}} \mid \varphi_{\mathsf{state}} \vee \psi_{\mathsf{state}} \mid \mathbf{E}\varphi_{\mathsf{path}} \mid \mathbf{A}\varphi_{\mathsf{path}} \mid \exists p.\ \varphi_{\mathsf{state}}$$
$$\varphi_{\mathsf{path}}, \psi_{\mathsf{path}} ::= \varphi_{\mathsf{state}} \mid \neg\,\varphi_{\mathsf{path}} \mid \varphi_{\mathsf{path}} \vee \psi_{\mathsf{path}} \mid \mathbf{X}\,\varphi_{\mathsf{path}} \mid \varphi_{\mathsf{path}}\,\mathbf{U}\,\psi_{\mathsf{path}}.$$

QCTL^* is interpreted here over Kripke structures through their unwindings[1]: given a Kripke structure $\mathscr{S}$, a state q and a formula $\varphi \in \mathsf{QCTL}^*$, that φ holds at q in $\mathscr{S}$, denoted with $\mathscr{S}, q \models_t \varphi$, is defined by the truth value of $\mathscr{T}_{\mathscr{S}}(q) \models \varphi$ that uses the standard inductive semantics of CTL^* over trees extended with the following case:

$$\mathscr{T} \models \exists p.\varphi_{\mathsf{state}} \quad \text{iff} \quad \exists \mathscr{T}' \equiv_{\mathsf{AP}\setminus\{p\}} \mathscr{T} \text{ s.t. } \mathscr{T}' \models \varphi_{\mathsf{state}}.$$

Universal quantification over atomic propositions, denoted with the construct $\forall p.\ \varphi$, is obtained by dualising this definition. We refer to [LM13] for a detailed study of QCTL^* and QCTL. Here we just recall the following important properties of these logics. First note that QCTL is actually as expressive as QCTL^* (with an effective translation) [Fre01, DLM12]. Secondly model checking and satisfiability are decidable but non elementary. More precisely given a QCTL formula φ and a (finite) set of degrees $\mathscr{D} \subseteq \mathbb{N}$, one can build a tree automaton $\mathscr{A}_{\varphi,\mathscr{D}}$ recognizing the $\mathscr{D}$-trees satisfying φ. This provides a decision procedure for model checking as the Kripke structure $\mathscr{S}$ fixes the set $\mathscr{D}$, and it remains to check whether the unwinding of $\mathscr{S}$ is accepted by $\mathscr{A}_{\varphi,\mathscr{D}}$. For satisfiability the decision procedure is obtained by building a formula φ_2 from φ such that φ_2 is satisfied by some $\{1,2\}$-tree iff φ is satisfied by some finitely-branching tree. Finally it remains to notice that a QCTL formula is satisfiable iff it is satisfiable in a finitely-branching tree (as QCTL is as expressive as MSO) to get the decision procedure for QCTL satisfiability. By consequence we also have that a QCTL formula is satisfiable iff it is satisfied by a regular tree (corresponding to the unwinding of some finite Kripke structure).

3 From ATL_{sc} to QCTL

The main results of this paper concern the satisfiability problem for ATL_{sc}: given a formula in ATL_{sc}, does there exists a CGS $\mathscr{C}$ and a state q such that $\mathscr{C}, q \models_\varnothing \varphi$ (with empty initial context)? Before we

[1]Note that several semantics are possible for QCTL^* and the one we use here is usually called the *tree semantics*.

present these results in the next sections, we briefly explain how we reduce the model-checking problem for ATL_{sc} (which consists in deciding whether a given state q of a given CGS $\mathscr{C}$ satisfies a given ATL_{sc} formula φ) to the model-checking problem for QCTL. This reduction will serve as a basis for proving our main result.

3.1 Model checking

Let $\mathscr{C} = \langle Q, R, \ell, \mathsf{Agt}, \mathscr{M}, \mathsf{Mov}, \mathsf{Edge}\rangle$ be a finite-state CGS, with a finite set of moves $\mathscr{M} = \{m_1, \ldots, m_k\}$. We consider the following sets of fresh atomic propositions: $\mathsf{P}_Q = \{\mathsf{p}_q \mid q \in Q\}$, $\mathsf{P}^j_{\mathscr{M}} = \{\mathsf{m}^j_1, \ldots, \mathsf{m}^j_k\}$ for every $a_j \in \mathsf{Agt}$, and write $\mathsf{P}_{\mathscr{M}} = \bigcup_{a_j \in \mathsf{Agt}} \mathsf{P}^j_{\mathscr{M}}$. Let $\mathscr{S}_{\mathscr{C}}$ be the Kripke structure $\langle Q, R, \ell_+\rangle$ where for any state q, we have: $\ell_+(q) = \ell(q) \cup \{\mathsf{p}_q\}$. A strategy for an agent a_j can be seen as a function $f_j \colon \mathsf{Exec}^{\mathsf{f}}(q) \to \mathsf{P}^j_{\mathscr{M}}$ labeling the execution tree of $\mathscr{S}_{\mathscr{C}}$ with propositions in $\mathsf{P}^j_{\mathscr{M}}$.

Let $F \in \mathsf{Strat}(C)$ be a strategy context and $\Phi \in \mathsf{ATL}_{sc}$. We reduce the question whether $\mathscr{C}, q \models_F \Phi$ to a model-checking instance for QCTL^* over $\mathscr{S}_{\mathscr{C}}$. For this, we define a QCTL^* formula $\overline{\Phi}^C$ inductively: for non-temporal formulas,

$$\overline{\cdot\rangle A\langle\cdot\, \varphi}^C = \overline{\varphi}^{C\setminus A} \qquad \overline{\varphi \wedge \psi}^C = \overline{\varphi}^C \wedge \overline{\psi}^C \qquad \overline{\neg\psi}^C = \neg\,\overline{\varphi}^C \qquad \overline{p}^C = p$$

For a formula of the form $\langle\cdot A\cdot\rangle\, \mathbf{X}\,\varphi$ with $A = \{a_{j_1}, \ldots, a_{j_l}\}$, we let:

$$\overline{\langle\cdot A\cdot\rangle\, \mathbf{X}\,\varphi}^C = \exists \mathsf{m}^{j_1}_1 \ldots \mathsf{m}^{j_1}_k \ldots \mathsf{m}^{j_l}_1 \ldots \mathsf{m}^{j_l}_k .\ \bigwedge_{a_j \in A} \mathbf{AG}\left(\Phi_{\mathsf{strat}}(a_j)\right) \wedge \mathbf{A}\left(\Phi^{[C\cup A]}_{\mathsf{out}} \Rightarrow \mathbf{X}\,\overline{\varphi}^{C\cup A}\right)$$

where:

$$\Phi_{\mathsf{strat}}(a_j) = \bigvee_{q \in Q} \Big(\mathsf{p}_q \wedge \bigvee_{m_i \in \mathsf{Mov}(q,a_j)} (\mathsf{m}^j_i \wedge \bigwedge_{l \neq i} \neg\, \mathsf{m}^j_l)\Big)$$

$$\Phi^{[A]}_{\mathsf{out}} = \mathbf{G}\left[\bigwedge_{\substack{q\in Q\\ m \in \mathsf{Mov}(q,A)}} \Big((\mathsf{p}_q \wedge m) \Rightarrow \mathbf{X}\big(\bigvee_{q' \in \mathsf{Next}(q,A,m)} \mathsf{p}_{q'}\big)\Big)\right]$$

where m is a move $(m^j)_{a_j \in A} \in \mathsf{Mov}(q,A)$ for A and P_m is the propositional formula $\bigwedge_{a_j \in A} m^j$ characterizing m. Formula $\Phi_{\mathsf{strat}}(a_j)$ ensures that the labelling of propositions m^j_i describes a feasible strategy for a_j. Formula $\Phi^{[A]}_{\mathsf{out}}$ characterizes the outcomes of the strategy for A that is described by the atomic propositions in the model. Note that $\Phi^{[A]}_{\mathsf{out}}$ is based on the transition table Edge of $\mathscr{C}$ (via $\mathsf{Next}(q,A,m)$). For a formula of the form $\langle\cdot A\cdot\rangle\,(\varphi\,\mathbf{U}\,\psi)$ with $A = \{a_{j_1}, \ldots, a_{j_l}\}$, we let:

$$\overline{\langle\cdot A\cdot\rangle\,(\varphi\,\mathbf{U}\,\psi)}^C = \exists \mathsf{m}^{j_1}_1 \ldots \mathsf{m}^{j_1}_k \ldots \mathsf{m}^{j_l}_1 \ldots \mathsf{m}^{j_l}_k .\ \bigwedge_{a_j \in A} \mathbf{AG}\left(\Phi_{\mathsf{strat}}(a_j)\right) \wedge \mathbf{A}\left(\Phi^{[C\cup A]}_{\mathsf{out}} \Rightarrow (\overline{\varphi}^{C\cup A}\,\mathbf{U}\,\overline{\psi}^{C\cup A})\right)$$

Then:

Theorem 7. [DLM12] *Let q be a state in a CGS $\mathscr{C}$. Let Φ be an ATL_{sc} formula and F be a strategy context for some coalition C. Let $\mathscr{T}'$ be the execution tree $\mathscr{T}_{\mathscr{S}_{\mathscr{C}}}(q)$ with a labelling function ℓ' s.t. for every $\pi \in \mathsf{Exec}^{\mathsf{f}}(q)$ of length i and any $a_j \in C$, $\ell'(\pi) \cap \mathsf{P}^j_{\mathscr{M}} = \mathsf{m}^j_i$ if, and only if, $F(\pi)_{|a_j} = m_i$. Then $\mathscr{C}, q \models_F \Phi$ if, and only if, $\mathscr{T}', q \models_t \overline{\Phi}^C$.*

Combined with the (non-elementary) decision procedure for QCTL^* model checking, we get a model-checking algorithm for model checking ATL_{sc}. Notice that our reduction above is into QCTL^*, but as explained before every QCTL^* formula can be translated into QCTL. Finally note that model checking is non elementary (k-EXPTIME-hard for any k) both for QCTL and ATL_{sc} [DLM12].

3.2 Satisfiability

We now turn to satisfiability. The reduction to QCTL we just developed for model checking does not extend to satisfiability, because the QCTL formula we built depends both on the formula and on the structure. Actually, satisfiability is undecidable for ATL_{sc}, both for infinite CGS and when restricting to finite CGS [TW12]. It is worth noticing that both problems are relevant, as ATL_{sc} does not have the finite-model property (nor does it have the finite-branching property). This can be derived from the fact that the modal logic $S5^n$ does not have the finite-model property [Kur02], and from the elegant reduction of satisfiability of $S5^n$ to satisfiability of ATL_{sc} given in [TW12] [2].

In what follows, we prove decidability of satisfiability in two different settings: first in the setting of turn-based games, and then in the setting of a bounded number of actions allowed to the players. A consequence of our decidability proofs is that in both cases (based on automata constructions), ATL_{sc} does have the finite-model property (thanks to Rabin's regularity theorem). We also consider the setting where quantification is restricted to memoryless strategies, but prove that then satisfiability is undecidable (even on turn-based games and with a fixed number of actions).

Before we proceed to the algorithms for satisfiability, we prove a generic result [3] about the number of agents needed in a CGS to satisfy a formula involving a given set of agents. This result has already been proved for ATL (*e.g.* in [WLWW06]). Given a formula $\Phi \in \mathsf{ATL}_{sc}$, we use $\mathsf{Agt}(\Phi)$ to denote the set of agents involved in the strategy quantifiers in Φ.

Proposition 8. *An* ATL_{sc} *formula* Φ *is satisfiable iff, it is satisfiable in a CGS with* $|\mathsf{Agt}(\Phi)|+1$ *agents.*

Proof. Assume Φ is satisfied in a CGS $\mathscr{C} = \langle Q, R, \ell, \mathsf{Agt}, \mathscr{M}, \mathsf{Mov}, \mathsf{Edge}\rangle$. If $|\mathsf{Agt}| \leq \mathsf{Agt}(\Phi)$, one can easily add extra players in $\mathscr{C}$ in such a way that they play no role in the behavior of the game structure. Otherwise, if $|\mathsf{Agt}| > \mathsf{Agt}(\Phi)+1$, we can replace the agents in Agt that do not belong to $\mathsf{Agt}(\Phi)$ by a unique agent mimicking the action of the removed players. For example, a coalition $A = \{a_1, \ldots, a_k\}$ can be replaced by a player a whose moves are k-tuples in $\mathscr{M}^k$. $\square$

4 Turn-based case

Let Φ be an ATL_{sc} formula, and assume $\mathsf{Agt}(\Phi)$ is the set $\{a_1, \ldots, a_n\}$. Following Prop. 8, let Agt be the set of agents $\mathsf{Agt}(\Phi) \cup \{a_0\}$, where a_0 is an additional player. In the following, we use an atomic propositions $(\mathsf{turn}_j)_{a_j \in \mathsf{Agt}}$ to specify the owner of the states. A strategy for an agent a_j can be encoded by an atomic proposition mov_j: indeed it is sufficient to mark one *successor* of every a_j-state (notice that this is a crucial difference with CGS). The outcomes of such a strategy are the runs in which every a_j-state is followed by a state labelled with mov_j; this is the main idea of the reduction below.

Given a coalition C (which we intend to represent the agents that have a strategy in the current context), we define a QCTL* formula $\widehat{\Phi}^C$ inductively:

- for non-temporal formulas we let:

$$\widehat{\cdot\rangle A\langle\cdot\,\varphi}^C = \widehat{\varphi}^{C\setminus A} \qquad \widehat{\varphi\wedge\psi}^C = \widehat{\varphi}^C \wedge \widehat{\psi}^C \qquad \widehat{\neg\psi}^C = \neg\,\widehat{\varphi}^C \qquad \widehat{P}^C = P$$

- for path formulas, we define:

$$\widehat{\mathbf{X}\,\varphi}^C = \mathbf{X}\,\widehat{\varphi}^C \qquad \widehat{\varphi\,\mathbf{U}\,\psi}^C = \widehat{\varphi}^C\,\mathbf{U}\,\widehat{\psi}^C$$

[2] Indeed the finite-branching property for ATL_{sc} would imply the finite-model property for $S5^n$.
[3] Note that it still holds true when restricting to turn-based games.

- for formulas of the form $\langle\cdot A\cdot\rangle\, \varphi$ with $A = \{a_{j_1}, \ldots, a_{j_l}\}$, we let:

$$\widehat{\langle\cdot A\cdot\rangle\, \varphi}^{C} = \exists \mathsf{mov}_{j_1} \ldots \mathsf{mov}_{j_l}. \left[\mathbf{AG} \bigwedge_{a_j \in A} (\mathsf{turn}_j \Rightarrow \mathbf{EX}_1\, \mathsf{mov}_j) \wedge \mathbf{A}\Big[\mathbf{G}\Big(\bigwedge_{a_j \in A\cup C} (\mathsf{turn}_j \Rightarrow \mathbf{X}\, \mathsf{mov}_j)\Big) \Rightarrow \widehat{\varphi}^{C\cup A}\Big]\right]$$

  where $\mathbf{EX}_1\, \alpha$ is a shorthand for $\mathbf{EX}\, \alpha \wedge \forall p. \Big(\mathbf{EX}\, (\alpha \wedge p) \Rightarrow \mathbf{AX}\, (\alpha \Rightarrow p)\Big)$, specifying the existence of a unique successor satisfying α.

Now we have the following proposition, whose proof is done by structural induction over the formula:

Proposition 9. *Let $\Phi \in \mathsf{ATL}_{sc}$, and $\mathsf{Agt} = \mathsf{Agt}(\Phi) \cup \{a_0\}$ as above. Let $\mathscr{C}$ be a turn-based CGS, q be a state of $\mathscr{C}$, and F be a strategy context. Let $\mathscr{T}_{\mathscr{C}}(q) = \langle T, \ell\rangle$ be the execution tree of the underlying Kripke structure of $\mathscr{C}$ (including a labelling with propositions $(\mathsf{turn}_j)_{a_j \in \mathsf{Agt}}$). Let ℓ_F be the labelling extending ℓ such that for every node ρ of T belonging to some $a_j \in \mathsf{dom}(F)$ (i.e., such that $\mathsf{last}(\rho) \in Own(a_j)$), its successor $\rho \cdot q$ according to F (i.e., such that $F_j(\rho) = q$) is labelled with mov_j. Then we have:*

$$\mathscr{C}, q \models_F \Phi \quad \textit{iff} \quad \langle T, \ell_F\rangle \models \widehat{\Phi}^{\mathsf{dom}(F)}$$

Proof. The proof is by structural induction over Φ. The cases of atomic propositions and Boolean operators are straightforward.

- $\Phi = \langle\cdot A\cdot\rangle\, (\varphi\, \mathbf{U}\, \psi)$: assume $\mathscr{C}, q \models_F \Phi$. Then there exists $F_A \in \mathsf{Strat}(A)$ s.t. for any $\rho \in \mathsf{Out}(q, F_A \circ F)$, there exists $i \geq 0$ s.t. $\mathscr{C}, \rho(i) \models_{(F_A \circ F)^{\rho_{\leq i}}} \psi$ and $\forall 0 \leq j < i$, we have $\mathscr{C}, \rho(j) \models_{(F_A \circ F)^{\rho_{\leq j}}} \varphi$. Let $\ell_{F_A \circ F}$ be the extension of ℓ labelling T with propositions $(\mathsf{mov}_j)_{a_j \in \mathsf{Agt}}$ according to the strategy context $F_A \circ F$. By induction hypothesis, the following two statements hold true:
  - $\langle T, \ell_{F_A \circ F}\rangle_{\rho_{\leq i}} \models \widehat{\psi}^{\mathsf{dom}(F) \cup A}$, and
  - $\langle T, \ell_{F_A \circ F}\rangle_{\rho_{\leq j}} \models \widehat{\varphi}^{\mathsf{dom}(F) \cup A}$ for any $0 \leq j < i$.

  (where $\langle U, l\rangle_\pi$ is the subtree of $\langle U, l\rangle$ rooted at node $\pi \in U$). As this is true for every ρ in the outcomes induced by $F_A \circ F$, it holds for every path in the execution tree satisfying the constraint over the labelling of $(\mathsf{turn}_j)_{a_j \in \mathsf{Agt}}$ and $(\mathsf{mov}_j)_{a_j \in \mathsf{Agt}}$. It follows that

$$\langle T, \ell_{F_A \circ F}\rangle \models \mathbf{A}\Big[\mathbf{G}\Big(\bigwedge_{a_j \in A\cup C} (\mathsf{turn}_j \Rightarrow \mathbf{X}\, \mathsf{mov}_j)\Big) \Rightarrow \widehat{\varphi}^{\mathsf{dom}(F) \cup A}\Big]$$

  Moreover we also know that $\mathbf{AG} \bigwedge_{a_j \in A} (\mathsf{turn}_j \Rightarrow \mathbf{EX}_1 \mathsf{mov}_j)$ holds true in $\langle T, \ell_{F_A \circ F}\rangle$ since the labelling $\ell_{F_A \circ F}$ includes the strategy F_A. Hence $\langle T, \ell_F\rangle \models \widehat{\Phi}^{\mathsf{dom}(F)}$, with the labelling for $(\mathsf{mov}_j)_{a_j \in A}$ being obtained from F_A.

  Now assume $\langle T, \ell_F\rangle \models \widehat{\Phi}^{\mathsf{dom}(F)}$. Write $A = \{a_{j_1}, \ldots, a j_l\}$. Then we have:

$$\langle T, \ell_F\rangle \models \exists \mathsf{mov}_{j_1} \ldots \mathsf{mov}_{j_l}. \Big[\mathbf{AG} \bigwedge_{a_j \in A} (\mathsf{turn}_j \Rightarrow \mathbf{EX}_1 \mathsf{mov}_j) \wedge \mathbf{A}\Big[\mathbf{G}\Big(\bigwedge_{a_j \in A\cup C} (\mathsf{turn}_j \Rightarrow \mathbf{X}\, \mathsf{mov}_j)\Big) \Rightarrow (\widehat{\varphi}^{\mathsf{dom}(F) \cup A}\, \mathbf{U}\, \widehat{\psi}^{\mathsf{dom}(F) \cup A})\Big]\Big]$$

The first part of the formula, namely $\mathbf{AG} \bigwedge_{a_j \in A}(\mathsf{turn}_j \Rightarrow \mathbf{EX}_1 \mathsf{mov}_j)$, ensures that the labeling with $(\mathsf{mov}_j)_{a_j\in A}$ defines a strategy for the coalition A. The second part states that every run belonging to the outcomes of $F_A \circ F$ (remember that ℓ_F already contains the strategy context F) satisfies $(\widehat{\varphi}^{\mathsf{dom}(F)\cup A}\,\mathbf{U}\,\widehat{\psi}^{\mathsf{dom}(F)\cup A})$. Finally it remains to use the induction hypothesis over states along the execution to deduce $\mathscr{C}, q \models_F \langle\cdot A\cdot\rangle\,(\varphi\,\mathbf{U}\,\psi)$.

- $\Phi = \cdot\rangle A\langle\cdot\,\psi$: assume $\mathscr{C}, q \models_F \Phi$. Then $\mathscr{C}, q \models_{F_{\mathsf{dom}(F)\setminus A}} \psi$. Applying the induction hypothesis, we get $\langle T, \ell_{F_{\mathsf{dom}(F)\setminus A}}\rangle \models \widehat{\psi}^{\mathsf{dom}(F)\setminus A}$. And it follows that $\langle T, \ell_F\rangle \models \widehat{\psi}^{\mathsf{dom}(F)\setminus A}$ because the labeling of strategies for coalition A in F is not used for evaluating $\widehat{\psi}^{\mathsf{dom}(F)\setminus A}$. Conversely, assume $\langle T, \ell_F\rangle \models \widehat{\psi}^{\mathsf{dom}(F)\setminus A}$. Then we have $\langle T, \ell_{F_{\mathsf{dom}(F)\setminus A}}\rangle \models \widehat{\psi}^{\mathsf{dom}(F)\setminus A}$ (again the labeling of A strategies in F is not used for evaluating the formula). Applying induction hypothesis, we get $\mathscr{C}, q \models_{F_{\mathsf{dom}(F)\setminus A}} \psi$ and then $\mathscr{C}, q \models_F \Phi$.
- $\Phi = \langle\cdot A\cdot\rangle\,\mathbf{X}\,\varphi$ and $\Phi = \cdot\rangle A\langle\cdot\,\mathbf{X}\,\varphi$: the proofs are similar to the previous ones. □

Finally, let Φ_{tb} be the following formula, used to make the game turn-based:

$$\Phi_{tb} = \mathbf{AG}\Big[\bigvee_{a_j\in\mathsf{Agt}}\Big(\mathsf{turn}_j \wedge \bigwedge_{a_l\neq a_j} \neg\mathsf{turn}_l\Big)\Big]$$

and let $\widetilde{\Phi}$ be the formula $\Phi_{tb} \wedge \widehat{\Phi}^{\varnothing}$. Then we have:

Theorem 10. *Let Φ be an ATL_{sc} formula and $\widetilde{\Phi}$ be the QCTL^* formula defined as above. Φ is satisfiable in a turn-based CGS if, and only if, $\widetilde{\Phi}$ is satisfiable (in the tree semantics).*

Proof. If Φ is satisfiable in a turn-based structure, then there exists such a structure $\mathscr{C}$ with $|\mathsf{Agt}(\Phi)|+1$ agents. Assume $\mathscr{C}, q \models \Phi$. Now consider the execution tree $\mathscr{T}_{\mathscr{C}}(q)$ with the additional labelling to mark states with the correct propositions $(\mathsf{turn}_j)_{a_j\in\mathsf{Agt}}$, indicating the owner of each state. From Proposition 9, we have $\mathscr{T}_{\mathscr{C}}(q) \models \widehat{\Phi}^{\varnothing}$. Thus clearly $\mathscr{T}_{\mathscr{C}}(q) \models \widetilde{\Phi}$.

Conversely assume $\mathscr{T} \models \widetilde{\Phi}$. As explained in Section 2, we can assume that $\mathscr{T}$ is regular. Thus $\mathscr{T} \models \Phi_{tb} \wedge \widehat{\Phi}^{\varnothing}$: the first part of the formula ensures that every state of the underlying Kripke structure can be assigned to a unique agent, hence defining a turn-based CGS. The second part ensures that Φ holds for the corresponding game, thanks to Proposition 9. □

The above translation from ATL_{sc} into QCTL^* transforms a formula with k strategy quantifiers into a formula with at most $k+1$ nested blocks of quantifiers; satisfiability of a QCTL^* formula with $k+1$ blocks of quantifiers is in $(\mathsf{k}+3)$-$\mathsf{EXPTIME}$ [LM13]. Hence the algorithm has non-elementary complexity. We now prove that this high complexity cannot be avoided:

Theorem 11. *Satisfiability of ATL_{sc} formulas in turn-based CGS is non-elementary (i.e., it is* k-$\mathsf{EXPTIME}$*-hard, for all k).*

Proof (sketch). Model checking ATL_{sc} over turn-based games is non-elementary [DLM12], and it can easily be encoded as a satisfiability problem. Let $\mathscr{C} = \langle Q, R, \ell, \mathsf{Agt}, \mathscr{M}, \mathsf{Mov}, \mathsf{Edge}\rangle$ be a turn-based CGS, and Φ be an ATL_{sc} formula. Let P_q be a fresh atomic proposition for every $q \in Q$. Now we define an ATL_{sc} formula $\Psi_{\mathscr{C}}$ to describe the game $\mathscr{C}$ as follows:

$$\Psi_{\mathscr{C}} = \mathbf{AG}\Big(\bigvee_{q\in Q}(\mathsf{P}_q \wedge \bigwedge_{q'\neq q}\neg\mathsf{P}_{q'} \wedge \bigwedge_{P\in\ell(q)} P \wedge \bigwedge_{P'\notin\ell(q)} \neg P')\Big) \wedge$$
$$\mathbf{AG}\Big[\bigvee_{q\in Q}\Big(\mathsf{P}_q \Rightarrow (\bigwedge_{q\to q'} \langle\!\langle\mathsf{Own}(q)\rangle\!\rangle\,\mathbf{X}\,\mathsf{P}_{q'} \wedge \bigwedge_{q'.\ q\not\to q'} \neg\,\langle\!\langle\mathsf{Own}(q)\rangle\!\rangle\,\mathbf{X}\,\mathsf{P}_{q'})\Big)\Big]$$

where $q \to q'$ denotes the existence of a transition from q to q' in $\mathscr{C}$. Any turn-based CGS satisfying $\Psi_{\mathscr{C}}$ corresponds to some unfolding of $\mathscr{C}$, and then has the same execution tree. Finally we clearly have that $\mathscr{C}, q \models \Phi$ if, and only if, $\Psi_{\mathscr{C}} \wedge \mathsf{P}_q \wedge \Phi$ is satisfiable in a turn-based structure. $\square$

5 Bounded action alphabet

We consider here another setting where the reduction to QCTL^* can be used to solve the satisfiability of ATL_{sc}: we assume that each player has a bounded number of available actions. Formally, it corresponds to the following satisfiability problem:

Problem: $(\mathsf{Agt}, \mathscr{M})$**-satisfiability**
Input: a finite set of moves $\mathscr{M}$, a set of agents Agt, and an ATL_{sc} formula Φ involving the agents in Agt;
Question: does there exist a CGS $\mathscr{C} = \langle Q, R, \ell, \mathsf{Agt}, \mathscr{M}, \mathsf{Mov}, \mathsf{Edge}\rangle$ and a state $q \in Q$ such that $\mathscr{C}, q \models \Phi$.

Assume $\mathscr{M} = \{1, \ldots, \alpha\}$ and $\mathsf{Agt} = \{a_1, \ldots, a_n\}$. With this restriction, we know that we are looking for a CGS whose execution tree has nodes with degrees in the set $\mathscr{D} = \{1, 2, \ldots, \alpha^n\}$. We consider such $\mathscr{D}$-trees where the transition table is encoded as follows: for every agent a_i and move m in $\mathscr{M}$, we use the atomic proposition mov_i^m to specify that agent a_i has played move m in the *previous* node. Any execution tree of such a CGS satisfies formula

$$\Phi_{\mathsf{Edge}} = \mathbf{AG}\Big[\Big(\bigwedge_{\bar{m}\in\mathscr{M}^n} \mathbf{EX}_1 \mathsf{mov}^{\bar{m}}\Big) \wedge \mathbf{AX}\Big(\bigvee_{\bar{m}\in\mathscr{M}^n} \mathsf{mov}^{\bar{m}}\Big)\Big]$$

where $\mathsf{mov}^{\bar{m}}$ stands for $\bigwedge_{a_j\in\mathsf{Agt}} \mathsf{mov}_j^{\bar{m}_j}$. Notice that the second part of the formula is needed because of the way we handle the *implicit* universal quantification associated with the strategy quantifiers of ATL_{sc}.

Given a coalition C, we define a QCTL^* formula $\widehat{\Phi}^C$ inductively as follows:

- for non-temporal formulas we let

$$\widehat{\cdot\rangle A\langle\cdot\, \varphi}^C = \widehat{\varphi}^{C\smallsetminus A} \qquad \widehat{\varphi\wedge\psi}^C = \widehat{\varphi}^C \wedge \widehat{\psi}^C \qquad \widehat{\neg\psi}^C = \neg\,\widehat{\varphi}^C \qquad \widehat{P}^C = P$$

- for temporal modalities, we define

$$\widehat{\mathbf{X}\varphi}^C = \mathbf{X}\,\widehat{\varphi}^C \qquad \widehat{\varphi\,\mathbf{U}\,\psi}^C = \widehat{\varphi}^C\,\mathbf{U}\,\widehat{\psi}^C.$$

- finally, for formulas of the form $\langle\cdot A\cdot\rangle\,\varphi$ with $A = \{a_{j_1}, \ldots, a_{j_l}\}$, we let:

$$\widehat{\langle\cdot A\cdot\rangle\,\varphi}^C = \exists \mathsf{choose}_{j_1}^1 \ldots \mathsf{choose}_{j_1}^\alpha \ldots \mathsf{choose}_{j_l}^1 \ldots \mathsf{choose}_{j_l}^\alpha. \\ \Big[\mathbf{AG}\Big(\bigwedge_{a_j\in A}\ \bigvee_{m=1\ldots\alpha} (\mathsf{choose}_j^m \wedge \bigwedge_{n\neq m} \neg\,\mathsf{choose}_j^n)\Big) \wedge \\ \mathbf{A}\Big[\mathbf{G}\Big(\bigwedge_{a_j\in A\cup C}\ \bigwedge_{m=1\ldots\alpha} (\mathsf{choose}_j^m \Rightarrow \mathbf{X}\,\mathsf{mov}_j^m)\Big) \Rightarrow \widehat{\varphi}^{C\cup A}\Big]\Big].$$

The first part of this formula requires that the atomic propositions choose_j^m describe a strategy, while the second part expresses that every execution following the labelled strategies (including those for C) satisfies the path formula $\widehat{\varphi}^{C\cup A}$.

Now, letting $\widehat{\Phi}$ be the formula $\Phi_{\mathsf{Edge}} \wedge \widehat{\Phi}^{\varnothing}$, we have the following theorem (similar to Theorem 10):

Theorem 12. *Let Φ be an ATL_{sc} formula, $\mathsf{Agt} = \{a_1, \ldots, a_n\}$ be a finite set of agents, $\mathscr{M} = \{1, \ldots, \alpha\}$ be a finite set of moves, and $\widehat{\Phi}$ be the formula defined above. Then Φ is $(\mathsf{Agt}, \mathscr{M})$-satisfiable in a CGS if, and only if, the QCTL^* formula $\widehat{\Phi}$ is satisfiable (in the tree semantics).*

We end up with a non-elementary algorithm (in $(\mathsf{k}+2)$-$\mathsf{EXPTIME}$ for a formula involving k strategy quantifiers) for solving satisfiability of an ATL_{sc} formula for a bounded number of moves, both for a fixed or for an unspecified set of agents (we can infer the set of agents using Prop. 8). Since ATL_{sc} model checking is non-elementary even for a fixed number of moves (the crucial point is the alternation of strategy quantifiers), we deduce:

Corollary 13. *$(\mathsf{Agt}, \mathscr{M})$-satisfiability for ATL_{sc} formulas is non-elementary (i.e., k-$\mathsf{EXPTIME}$-hard, for all k).*

6 Memoryless strategies

Memoryless strategies are strategies that only depend on the present state (as opposed to general strategies, whose values can depend on the whole history). Restricting strategy quantifiers to memoryless strategies in the logic makes model checking much easier: in a finite game, there are only finitely many memoryless strategies to test, and applying a memoryless strategy just amounts to removing some transitions in the graph. Still, quantification over memoryless strategies is not possible in plain ATL_{sc}, and this additional expressive power turns out to make satifiability undecidable, even when restricting to turn-based games. One should notice that the undecidability proof of [TW12] for satisfiability in concurrent games uses one-step games (*i.e.*, they only involve one $\mathbf{X}$ modality), and hence also holds for memoryless strategies.

Theorem 14. *Satisfiability of ATL^0_{sc} (with memoryless-strategy quantification) is undecidable, even when restricting to turn-based games.*

Proof. We prove the result for infinite-state turn-based games, by adapting the corresponding proof for QCTL under the structure semantics [Fre01], which consists in encoding the problem of tiling a quadrant. The result for finite-state turn-based games can be obtained using similar (but more involved) ideas, by encoding the problem of tiling all finite grids (see [LM13] for the corresponding proof for QCTL).

We consider a finite set T of tiles, and two binary relations H and V indicating which tile(s) may appear on the right and above (respectively) a given tile. Our proof consists in writing a formula that is satisfiable only on a grid-shaped (turn-based) game structure representing a tiling of the quadrant (*i.e.*, of $\mathbb{N} \times \mathbb{N}$). The reduction involves two players: Player 1 controls square states (which are labelled with $\square$), while Player 2 controls circle states (labelled with $\bigcirc$). Each state of the grid is intended to represent one cell of the quadrant to be tiled. For technical reasons, the reduction is not that simple, and our game structure will have three kinds of states (see Fig. 1):

- the "main" states (controlled by Player 2), which form the grid. Each state in this main part has a *right* neighbour and a *top* neighbour, which we assume we can identify: more precisely, we make use of two atomic propositions v_1 and v_2 which alternate along the horizontal lines of the grid. The *right* successor of a v_1-state is labelled with v_2, while its *top* successor is labelled with v_1;
- the "tile" states, labelled with one item of T (seen as atomic propositions). Each tile state only has outgoing transition(s) to a tile state labelled with the same tile;

- the "choice" states, which appear between "main" states and "tile" states: there is one choice state associated with each main state, and each choice state has a transition to each tile state. Choice states are controlled by Player 1.

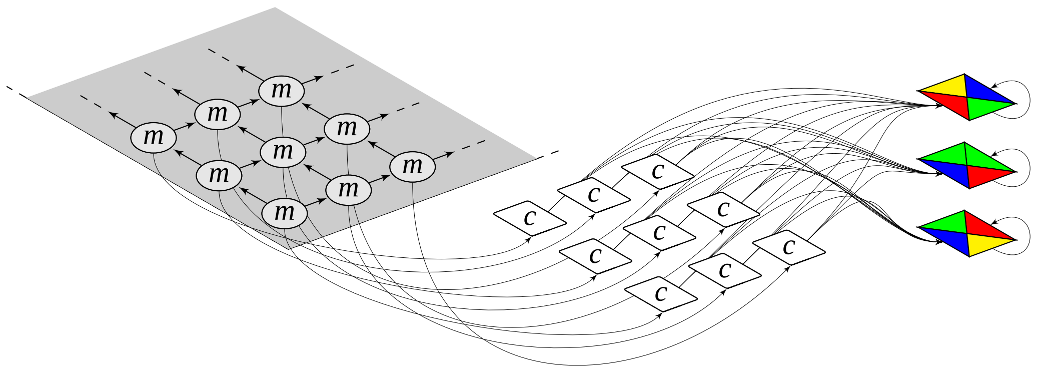


Fig. 1: The turn-based game encoding the tiling problem

Assuming that we have such a structure, a tiling of the grid corresponds to a *memoryless* strategy of Player 1 (who only plays in the "choice" states). Once such a memoryless strategy for Player 1 has been selected, that it corresponds to a valid tiling can be expressed easily: for instance, in any cell of the grid (assumed to be labelled with v_1), there must exist a pair of tiles $(t_1,t_2) \in H$ such that $v_1 \wedge \langle\cdot 2\cdot\rangle_0 \mathbf{X}\mathbf{X} t_1 \wedge \langle\cdot 2\cdot\rangle_0 \mathbf{X}(v_2 \wedge \mathbf{X}\mathbf{X} t_2)$. This would be written as follows:

$$\langle\cdot 1\cdot\rangle_0 \mathbf{G} \left[\begin{array}{cl} & v_1 \Rightarrow \displaystyle\bigvee_{(t_1,t_2)\in H} \langle\cdot 2\cdot\rangle_0 \mathbf{X}\mathbf{X} t_1 \wedge \langle\cdot 2\cdot\rangle_0 \mathbf{X}(v_2 \wedge \mathbf{X}\mathbf{X} t_2) \\ \wedge & \\ & v_2 \Rightarrow \displaystyle\bigvee_{(t_1,t_2)\in H} \langle\cdot 2\cdot\rangle_0 \mathbf{X}\mathbf{X} t_1 \wedge \langle\cdot 2\cdot\rangle_0 \mathbf{X}(v_1 \wedge \mathbf{X}\mathbf{X} t_2) \end{array} \right].$$

The same can be imposed for vertical constraints, and for imposing a fairness constraint on the base line (under the same memoryless strategy for Player 1).

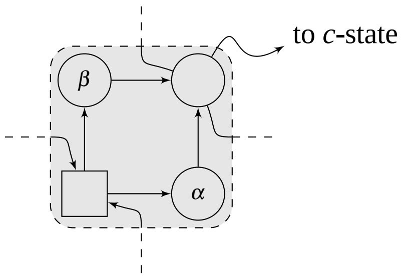


Fig. 2: The cell gadget

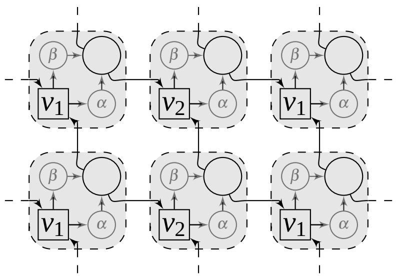


Fig. 3: Several cells forming (part of) a grid

It remains to build a formula characterising an infinite grid. This requires a slight departure from the above description of the grid: each main state will in fact be a gadget composed of four states, as depicted on Fig. 2. The first state of each gadget will give the opportunity to Player 1 to *color* the state with either α or β. This will be used to enforce "confluence" of several transitions to the same state (which we need to express that the two successors of any cell of the grid share a common successor).

We now start writing our formula, which we present as a conjunction of several subformulas. We require that the main states be labelled with m, the choice states be labelled with c, and the tile states be labelled with the names of the tiles. We let $\mathsf{AP}' = \{m,c\} \cup T$ and $\mathsf{AP} = \mathsf{AP}' \cup \{v_1, v_2, \alpha, \beta, \square, \bigcirc\}$. The first part of the formula reads a follows (where universal path quantification can be encoded, as long as the context is empty, using $\langle\cdot\varnothing\cdot\rangle_0$):

$$\mathbf{AG}\left[\bigvee_{p\in\mathsf{AP}'} p \wedge \bigwedge_{p'\in\mathsf{AP}'\setminus\{p\}} \neg p'\right] \wedge \mathbf{A}(m\,\mathbf{W}\,c) \wedge \mathbf{AG}\left[c \Rightarrow \left(\square \wedge \bigwedge_{t\in T} \langle\cdot 1\cdot\rangle_0\, \mathbf{X}\, t \wedge \mathbf{AX}\left(\bigvee_{t\in T} \mathbf{AG}\, t\right)\right)\right] \wedge$$
$$\mathbf{AG}\left[(\square \Leftrightarrow \neg\bigcirc) \wedge \left(\square \Rightarrow \bigwedge_{p\in\mathsf{AP}} (\mathbf{EX}\, p \Leftrightarrow \langle\cdot 1\cdot\rangle_0\, \mathbf{X}\, p)\right) \wedge \left(\bigcirc \Rightarrow \bigwedge_{p\in\mathsf{AP}} (\mathbf{EX}\, p \Leftrightarrow \langle\cdot 2\cdot\rangle_0\, \mathbf{X}\, p)\right)\right] \quad (1)$$

This formula enforces that each state is labelled with exactly one proposition from AP'. It also enforces that any path will wander through the main part until it possibly goes to a choice state (this is expressed as $\mathbf{A}(m\,\mathbf{W}\,c)$, where $m\,\mathbf{W}\,c$ means $\mathbf{G}\,m \vee m\,\mathbf{U}\,c$, and can be expressed a negated-until formula). Finally, the second part of the formula enforces the witnessing structures to be turn-based.

Now we have to impose that the m-part has the shape of a grid: intuitively, each cell has three successors: one "to the right" and one "to the top" in the main part of the grid, and one c-state which we will use for associating a tile with this cell. For technical reasons, the situation is not that simple, and each cell is actually represented by the gadget depicted on Fig. 2. Each state of the gadget is labelled with m. We constrain the form of the cells as follows:

$$\mathbf{AG}\left[m \Rightarrow ((\square \wedge \neg\alpha \wedge \neg\beta) \vee (\bigcirc \wedge \neg(\alpha \wedge \beta)))\right] \wedge \mathbf{AG}\left[\big((m\wedge\square) \Rightarrow (v_1 \Leftrightarrow \neg v_2)\big) \wedge \big((v_1 \vee v_2) \Rightarrow (m \wedge \square)\big)\right] \wedge$$
$$\mathbf{AG}\left[(m\wedge\square) \Rightarrow \left[\mathbf{AX}\left(m \wedge \bigcirc \wedge (\alpha\vee\beta) \wedge \mathbf{AX}\,(m \wedge \bigcirc \wedge \neg\alpha \wedge \neg\beta\right)\right) \wedge \langle\cdot 1\cdot\rangle_0\, \mathbf{X}\,\alpha \wedge \langle\cdot 1\cdot\rangle_0\, \mathbf{X}\,\beta\right]\right] \quad (2)$$

This says that there are four types of states in each cell, and specifies the possible transitions within such cells. We now express constraints on the transitions leaving a cell:

$$\mathbf{AG}\left[(\mathbf{EX}\,c \vee \mathbf{EX}\,v_1 \vee \mathbf{EX}\,v_2) \Rightarrow (m \wedge \bigcirc \wedge \neg\alpha \wedge \neg\beta)\right] \wedge$$
$$\mathbf{AG}\left[(m \wedge \bigcirc \wedge \neg\alpha \wedge \neg\beta) \Rightarrow (\mathbf{EX}\,c \wedge \mathbf{EX}\,v_1 \wedge \mathbf{EX}\,v_2 \wedge \mathbf{AX}\,(c \vee v_1 \vee v_2)\right] \quad (3)$$

It remains to enforce that the successor of the α and β states are the same. This is obtained by the following formula:

$$\mathbf{AG}\left[(m\wedge\square) \Rightarrow [\cdot 2\cdot]_0 \left(\langle\cdot\varnothing\cdot\rangle_0\, \mathbf{X}^3(c \vee v_1) \vee \langle\cdot\varnothing\cdot\rangle_0\, \mathbf{X}^3(c \vee v_2)\right)\right] \quad (4)$$

Indeed, assume that some cell has two different "final" states; then there would exist a strategy for Player 2 (consisting in playing differently in those two final states) that would violate Formula (4). Hence each cell as a single final state.

We now impose that each cell in the main part has exactly two m-successors, and these two m-successors have an m-successor in common. For the former property, Formula (3) already imposes that each cell has at least two m-successors (one labelled with v_1 and one with v_2). We enforce that there cannot be more that two:

$$\mathbf{AG}\left[(m\wedge\square) \Rightarrow [\cdot 1\cdot]_0 \left[\left(\langle\cdot 2\cdot\rangle_0\, \mathbf{X}^3(v_1 \wedge \mathbf{X}\,\alpha) \wedge \langle\cdot 2\cdot\rangle_0\, \mathbf{X}^3(v_2 \wedge \mathbf{X}\,\alpha)\right) \Rightarrow [\cdot 2\cdot]_0\, \langle\cdot\varnothing\cdot\rangle_0\, \mathbf{X}^3\mathbf{X}\,\alpha\right]\right]. \quad (5)$$

Notice that $[\cdot 2 \cdot]_0 \, \langle \cdot \varnothing \cdot \rangle_0 \, \varphi$ means that φ has to hold along any outcome of any *memoryless* strategy of Player 2. Assume that a cell has three (or more) successor cells. Then at least one is labelled with v_1 and at least one is labelled with v_2. There is a strategy for Player 1 to color one v_1-successor cell and one v_2-successor cell with α, and a third successor cell with β, thus violating Formula (5) (as Player 2 has a strategy to reach a successor cell colored with β)

For the latter property (the two successors have a common successor), we add the following formula (as well as its v_2-counterpart):

$$[\cdot 1 \cdot]_0 \, \langle \cdot \varnothing \cdot \rangle_0 \, \mathbf{G} \left[(m \wedge \Box \wedge v_1) \Rightarrow \Big([\langle \cdot 2 \cdot \rangle_0 \, \mathbf{X}^3 (v_1 \wedge [\cdot 2 \cdot]_0 \, \mathbf{X}^3 \mathbf{X} \alpha)] \Rightarrow [\langle \cdot 2 \cdot \rangle_0 \, \mathbf{X}^3 (\neg v_1 \wedge \mathbf{X}^3 (\neg v_1 \wedge \mathbf{X} \alpha))] \Big) \right] \quad (6)$$

In this formula, the initial (universal) quantification over strategies of Player 1 fixes a color for each cell. The formula claims that whatever this choice, if we are in some v_1-cell and can move to another v_1-cell whose two successors have color α, then also we can move to a v_2-cell having one α successor (which we require to be a v_2-cell). As this must hold for any coloring, both successors of the original v_1-cell share a common successor. Notice that this does not prevent the grid to be collapsed: this would just indicate that there is a *regular* infinite tiling.

We conclude by requiring that the initial state be in a square state of a cell in the main part. □

7 Results for Strategy Logic

In this section, we extend the previous results to Strategy Logic (SL). This logic has been initially introduced in [CHP07] for two-player turn-based games. It has then been extended to n-players concurrent games in [MMV10]. As explained in the introduction, satisfiability has been shown undecidable when considering infinite structures [MMV10], and the proof in [TW12] for finite satisfiability of ATL$_{sc}$ straightforwardly extends to SL. Here we show that satisfiability is decidable when considering turn-based games and when fixing a finite alphabet, and that it remains undecidable when only considering memoryless strategies.

Strategy Logic in a nutshell. We start by briefly recalling the main ingredients of SL. The syntax is given by the following grammar:

$$\varphi, \psi ::= p \mid \varphi \wedge \psi \mid \neg \varphi \mid \mathbf{X} \varphi \mid \varphi \, \mathbf{U} \, \psi \mid \langle\!\langle x \rangle\!\rangle \, \varphi \mid (a, x) \varphi$$

where $a \in$ Agt is an agent and x is a (strategy) variable (we use Var to denote the set of these variables). Formula $\langle\!\langle x \rangle\!\rangle \, \varphi$ expresses the existence of a strategy, which is stored in variable x, under which formula φ holds. In φ, the *agent binding* operator (a, x) can be used to bind agent a to follow strategy x. An assignment χ is a partial function from Agt $\cup$ Var to Strat. SL formulas are interpreted over pairs (χ, q) where q is a state of some CGS and χ is an assignment such that every free strategy variable/agent[4] occurring in the formula belongs to $\mathsf{dom}(\chi)$. Note that we have Agt $\subseteq \mathsf{dom}(\chi)$ when temporal modalities $\mathbf{X}$ and $\mathbf{U}$ are interpreted: this implies that the set of outcomes is restricted to a unique execution generated by all the strategies assigned to players in Agt, and the temporal modalities are therefore interpreted over this execution. Here we just give the semantics of the main two constructs (see [MMV10] for a complete definition of SL):

$$\begin{aligned} \mathscr{C}, \chi, q \models \langle\!\langle x \rangle\!\rangle \, \varphi \quad &\text{iff} \quad \exists F \in \mathsf{Strat} \text{ s.t. } \mathscr{C}, \chi[x \mapsto F], q \models \varphi \\ \mathscr{C}, \chi, q \models (a, x) \varphi \quad &\text{iff} \quad \mathscr{C}, \chi[a \mapsto \chi(x)], q \models \varphi \end{aligned}$$

[4] We use the standard notion of freedom for the strategy variables with the hypothesis that $\langle\!\langle x \rangle\!\rangle$ binds x, and for the agents with the hypothesis that (a, x) binds a and that every agent in Agt is free in temporal subformula (*i.e.*, with $\mathbf{U}$ or $\mathbf{X}$ as root).

In the following we assume w.l.o.g. that every quantifier $\langle\!\langle x\rangle\!\rangle$ introduces a fresh strategy variable x: this allows us to permanently use variable x to denote the selected strategy for a.

Turn-based case. The approach we used for ATL_{sc} can be adapted for SL. Given an SL formula Φ and a mapping $V\colon \mathsf{Agt} \to \mathrm{Var}$, we define a QCTL^* formula $\widehat{\Phi}^V$ inductively as follows (Boolean cases omitted):

$$\widehat{\langle\!\langle x\rangle\!\rangle\,\varphi}^V = \exists \mathsf{mov}_x. \Big[\mathbf{AG}\Big(\mathbf{EX}_1 \mathsf{mov}_x\Big) \wedge \widehat{\varphi}^V\Big] \qquad\qquad \widehat{(a,x)\varphi}^V = \widehat{\varphi}^{V[a\to x]}$$

Note that in this case we require that *every* reachable state has a (unique) successor labeled with mov_x: indeed when one quantifies over a strategy x, the agent(s) who will use this strategy are not known yet. However, in the turn-based case, a given strategy should be dedicated to a single agent: there is no natural way to share a strategy for two different agents (or the other way around, any two strategies for two different agents can be seen as a single strategy), as they are not playing in the same states. When the strategy x is assigned to some agent a, only the choices made in the a-states are considered.

The temporal modalities are treated as follows:

$$\widehat{\varphi\,\mathbf{U}\,\psi}^V = \mathbf{A}\Big[\mathbf{G}\Big(\bigwedge_{a_j\in\mathsf{Agt}} (\mathsf{turn}_j \Rightarrow \mathbf{X}\,\mathsf{mov}_{V(a_j)})\Big) \Rightarrow \widehat{\varphi}^V\,\mathbf{U}\,\widehat{\psi}^V\Big]$$
$$\widehat{\mathbf{X}\varphi}^V = \mathbf{A}\Big[\mathbf{G}\Big(\bigwedge_{a_j\in\mathsf{Agt}} (\mathsf{turn}_j \Rightarrow \mathbf{X}\,\mathsf{mov}_{V(a_j)})\Big) \Rightarrow \mathbf{X}\,\widehat{\varphi}^V\Big]$$

Now let $\widetilde{\Phi}$ be the formula $\Phi_{tb} \wedge \widehat{\Phi}^{V_\varnothing}$. Then we have the following theorem:

Theorem 15. *Let Φ be an SL formula and $\widetilde{\Phi}$ be the QCTL^* formula defined as above. Then Φ is satisfiable in a turn-based CGS if, and only if, $\widetilde{\Phi}$ is satisfiable (in the tree semantics).*

Bounded action alphabet Let $\mathscr{M}$ be $\{1,\ldots,\alpha\}$. The reduction carried out for ATL_{sc} can also be adapted for SL in this case. Given an SL formula Φ and a partial function $V\colon \mathsf{Agt} \to \mathrm{Var}$, we define the QCTL^* formula $\widehat{\widehat{\Phi}}^V$ inductively as follows:

$$\widehat{\widehat{\langle\!\langle x\rangle\!\rangle\,\varphi}}^V = \exists \mathsf{choose}^1_x \ldots \exists \mathsf{choose}^\alpha_x. \mathbf{AG}\Big(\bigvee_{1\le m\le\alpha} \mathsf{choose}^m_x \wedge \bigwedge_{n\neq m} \neg\,\mathsf{choose}^n_x\Big) \wedge \widehat{\widehat{\varphi}}^V \qquad \widehat{\widehat{(a,x)\varphi}}^V = \widehat{\widehat{\varphi}}^{V[a\mapsto x]}$$

The temporal modalities are handled as follows:

$$\widehat{\widehat{\varphi\,\mathbf{U}\,\psi}}^V = \mathbf{A}\Big[\Big(\mathbf{G}\bigwedge_{a_j\in\mathsf{Agt}}\ \bigwedge_{1\le m\le\alpha}\Big(\mathsf{choose}^m_{V(a_j)} \Rightarrow \mathbf{X}\,\mathsf{mov}^m_j\Big)\Big) \Rightarrow \Big(\widehat{\widehat{\varphi}}^V\,\mathbf{U}\,\widehat{\widehat{\psi}}^V\Big)\Big]$$
$$\widehat{\widehat{\mathbf{X}\varphi}}^C = \mathbf{A}\Big[\Big(\mathbf{G}\bigwedge_{a_j\in\mathsf{Agt}}\ \bigwedge_{1\le m\le\alpha}\Big(\mathsf{choose}^m_{V(a_j)} \Rightarrow \mathbf{X}\,\mathsf{mov}^m_j\Big)\Big) \Rightarrow \Big(\mathbf{X}\,\widehat{\widehat{\varphi}}^V\Big)\Big]$$

Remember that in this case, mov^m_j labels the possible successors of a state where agent a_j plays m.

Finally, let $\widehat{\Phi}$ be the formula $\Phi_{\mathrm{move}} \wedge \widehat{\widehat{\Phi}}^V_\varnothing$. We have:

Theorem 16. *Let Φ be an SL formula based on the set $\mathsf{Agt} = \{a_1,\ldots,a_n\}$, let $\mathscr{M} = \{1,\ldots,\alpha\}$ be a finite set of moves, and $\widehat{\Phi}$ be the QCTL^* formula defined as above. Then Φ is $(\mathsf{Agt},\mathscr{M})$-satisfiable if, and only if, $\widehat{\Phi}$ is satisfiable (in the tree semantics).*

7.1 Memoryless strategies

We now extend the undecidability result of ATL^0_{sc} to SL with memoryless-strategy quantification. Notice that there is an important difference between ATL^0_{sc} and SL^0 (the logic obtained from SL by quantifying only on memoryless strategies): the ATL_{sc}-quantifier $\langle\cdot A\cdot\rangle_0$ still has an implicit quantification over *all* the strategies of the other players (unless their strategy is fixed by the context), while in SL^0 all strategies must be explicitly quantified. Hence SL^0 and ATL^0_{sc} have uncomparable expressiveness. Still:

Theorem 17. SL^0 *satisfiability is undecidable, even when restricting to turn-based game structures.*

Proof (sketch). The proof uses a similar reduction as for the proof for ATL^0_{sc}. The difference is that the implicitly-quantified strategies in ATL^0_{sc} are now explicitly quantified, hence memoryless. However, most of the properties that our formulas impose are "local" conditions (involving at most four nested "next" modalities) imposed in all the reachable states. Such properties can be enforced even when considering only the ultimately periodic paths that are outcomes of memoryless strategies. The only subformula not of this shape is formula $\mathbf{A} m \mathbf{W} c$, but imposing this property along the outcomes of memoryless strategies is sufficient to have the formula hold true along any path. $\square$

8 Conclusion

While satisfiability for ATL_{sc} and SL is undecidable, we proved in this paper that it becomes decidable when restricting the search to turn-based games. We also considered the case where strategy quantification in those logics is restricted to memoryless strategies: while this makes model checking easier, it makes satisfiability undecidable, even for turn-based structures. These results have been obtained by following the tight and natural link between those temporal logics for games and the logic QCTL, which extends CTL with quantification over atomic propositions. This witnesses the power and usefulness of QCTL, which we will keep on studying to derive more results about temporal logics for games.

Acknowledgement. We thank the anonymous reviewers for their numerous suggestions, which helped us improve the presentation of the paper.

References

[AHK02] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

[BDLM09] Th. Brihaye, A. Da Costa, F. Laroussinie, and N. Markey. ATL with strategy contexts and bounded memory. In *Proceedings of the International Symposium Logical Foundations of Computer Science (LFCS'09)*, LNCS 5407, p. 92–106. Springer, 2009. doi:10.1007/978-3-540-92687-0_7.

[CE82] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the 3rd Workshop on Logics of Programs (LOP'81)*, LNCS 131, p. 52–71. Springer, 1982. doi:10.1007/BFb0025774.

[CHP07] K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*, LNCS 4703, p. 59–73. Springer, 2007. doi:10.1007/978-3-540-74407-8_5.

[DLM10] A. Da Costa, F. Laroussinie, and N. Markey. ATL with strategy contexts: Expressiveness and model checking. In *Proceedings of the 30th Conferentce on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, LIPIcs 8, p. 120–132. Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.120.

[DLM12] A. Da Costa, F. Laroussinie, and N. Markey. Quantified CTL: Expressiveness and model checking. In *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, LNCS 7454, p. 177–192. Springer, 2012. doi:10.1007/978-3-642-32940-1_14.

[Fre01] T. French. Decidability of quantified propositional branching time logics. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AJCAI'01)*, LNCS 2256, p. 165–176. Springer, 2001. doi:10.1007/3-540-45656-2_15.

[HSW13] C.-H. Huang, S. Schewe, and F. Wang. Model-checking iterated games. In *Proceedings of the 19th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'13)*, LNCS 7795, p. 154–168. Springer, 2013. doi:10.1007/978-3-642-36742-7_11.

[Kur02] A. Kurucz. $S5 \times S5 \times S5$ lacks the finite model property. In *Proceedings of the 3rd Workshop on Advances in Modal Logic (AIML'00)*, p. 321–327. World Scientific, 2002.

[LM13] F. Laroussinie and N. Markey. Quantified CTL: expressiveness and complexity. Research Report LSV-13-07, Lab. Spécification & Vérification, ENS Cachan, France, 2013.

[MMPV12] F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. What makes ATL* decidable? a decidable fragment of strategy logic. In *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, LNCS 7454, p. 193–208. Springer, 2012. doi:10.1007/978-3-642-32940-1_15.

[MMV10] F. Mogavero, A. Murano, and M. Y. Vardi. Reasoning about strategies. In *Proceedings of the 30th Conferentce on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, LIPIcs 8, p. 133–144. Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.133.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, p. 46–57. IEEE Comp. Soc. Press, 1977. doi:10.1109/SFCS.1977.32.

[QS82] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on Programming (SOP'82)*, LNCS 137, p. 337–351. Springer, 1982. doi:10.1007/3-540-11494-7_22.

[TW12] N. Troquard and D. Walther. On satisfiability in atl with strategy contexts. In *Proceedings of the 13th European Conference in Logics in Artificial Intelligence (JELIA'12)*, LNCS 7519, p. 398–410. Springer, 2012. doi:10.1007/978-3-642-33353-8_31.

[WHY11] F. Wang, C.-H. Huang, and F. Yu. A temporal logic for the interaction of strategies. In *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR'11)*, LNCS 6901, p. 466–481. Springer, 2011. doi:10.1007/978-3-642-23217-6_31.

[WLWW06] D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL satisfiability is indeed EXPTIME-complete. *Journal of Logic and Computation*, 16(6):765–787, 2006. doi:10.1093/logcom/exl009.

Modularity and Openness in Modeling Multi-Agent Systems

Wojciech Jamroga
Computer Science and Communication, and
Interdisciplinary Centre on Security, Reliability and Trust
University of Luxembourg
wojtek.jamroga@uni.lu

Artur Męski
Institute of Computer Science
Polish Academy of Sciences, Warsaw, Poland,
and FMCS, University of Łódź, Poland
meski@ipipan.waw.pl

Maciej Szreter
Institute of Computer Science
Polish Academy of Sciences, Warsaw, Poland
mszreter@ipipan.waw.pl

We revisit the formalism of modular interpreted systems (MIS) which encourages modular and open modeling of synchronous multi-agent systems. The original formulation of MIS did not live entirely up to its promise. In this paper, we propose how to improve modularity and openness of MIS by changing the structure of interference functions. These relatively small changes allow for surprisingly high flexibility when modeling actual multi-agent systems. We demonstrate this on two well-known examples, namely the trains, tunnel and controller, and the dining cryptographers.

Perhaps more importantly, we propose how the notions of multi-agency and openness, crucial for multi-agent systems, can be precisely defined based on their MIS representations.

1 Introduction

The paradigm of multi-agent systems (MAS) focuses on systems consisting of autonomous entities acting in a common environment. Regardless of whether we deem the entities to be intelligent or not, proactive or reactive, etc., there are two design-level properties that a multi-agent system should satisfy. First, it should be *modular* in the sense that it is inhabited by *loosely coupled* components. That is, interaction between agents is crucial for the system, but it should be relatively scarce compared to the intensity of local computation within agents (otherwise the system is in fact a single-agent system in disguise). Secondly, it should be *open* in the sense that an agent should be able join or leave the system without changing the design of the other components.

Models and representations of MAS can be roughly divided into two classes. On one hand, there are models of various agent logics, most notably modal logics of knowledge, action, time, and strategic ability [7, 8, 2]. These models are well suited for theoretical analysis of general properties of agent systems. However, they are too abstract in the sense that: (a) they are based on abstract notions of global state and global transition so the structure of a model does not reflect the structure of a MAS at all, and (b) they come with neither explicit nor implicit methodology for design and analysis of actual agent systems. At the other extreme there are practical-purpose high-level representation languages like Promela [11], Estelle [6], and Reactive Modules (RM) [1]. They are application-oriented, and usually include too many features to be convenient for theoretical analysis. The middle ground consists of formalisms that originate from abstract logical models but try to encapsulate a particular modeling methodology. For instance, interpreted systems [8] support local design of the state space; however, transitions are still global, i.e., they are defined between global rather than local states. Synchronous automata networks [10] and ISPL specifications [17, 19] push the idea further: they are based on local states and semi-local transitions, i.e., the outcome of a transition is local, but its domain global. This makes agents hard to separate from one another in a model, which hampers its modularity. On the other hand, concurrent programs [16] and asynchronous automata networks [10] are fairly modular but they support only systems whose execution can be appropriately modeled by interleaving of local actions and/or events.

Modular Interpreted Systems (MIS) are a class of models proposed in [13] to achieve separation of the interference between agents from the local processing within agents. The main idea behind MIS was to encapsulate the way agents' actions interfere by so called *interaction tokens* from a given alphabet $\mathscr{I}n$, together with functions out_i, in_i that define the *interface* of agent i. That is, out_i specifies how i's actions influence the evolution of the other agents, whereas in_i specifies how what rest of the world influences the local transition of i. Modular interpreted systems received relatively little attention, though some work was done on studying computational properties of the related verification problem [12], facilitating verification by abstraction [14], and using MIS to analyze homogeneous multi-agent systems [4]. This possibly stems from the fact that, in their original incarnation, MIS are not as modular and open as one would expect. More precisely, the types of functions used to define interference fix the number of agents in the MIS. Moreover, the assumption that all the functions used in a model are deterministic limit the practical applicability, as modeling of many natural scenarios becomes cumbersome.

In this paper, we try to revive MIS as an interesting formalism for modeling multi-agent systems. We propose how to improve modularity and openness of the original class by changing the structure of interference functions out, in. The idea is to use multisets of interference tokens instead of k-tuples. This way, we do not need to "hardwire" information about other modules inside a module. Additionally, we assume that the "manifestation" function *out* can be nondeterministic. These relatively small changes allow for surprisingly high flexibility when modeling MAS. We demonstrate that on two well-known benchmark examples: trains, tunnel and controller, and the dining cryptographers.

Perhaps more importantly, we propose how two important features of multi-agent systems can be formally defined, based on MIS representations. First, we show how to decide if a system is designed in a proper multi-agent way by looking at the relation between the complexity of its interference layer to the complexity of its global unfolding. Moreover, we define the degree of openness of a MIS as the complexity of the minimal transformation that the model must undergo in order to add a new agent to the system, or remove an existing one. We apply the definitions to our benchmark models, and show that different variants of cryptographers grossly differ in the amount of openness that they offer.

The paper has the following structure. In Section 2, the new variant of MIS is defined, along with its execution semantics. Section 3 presents MIS representations for two benchmarks: Tunnel, Trains and Controller (TTC) and Dining Cryptographers (DC). A graphical notation is provided to make the examples easier to read. In Sections 4 and 5, we propose formal definitions of multi-agency and openness, respectively, and apply them to several variants of the benchmarks. Section 6 concludes the paper.

1.1 Related Work

The modeling structures discussed in this paper share many similarities with existing modeling frameworks, in particular with Reactive Modules [1]. Still, MIS and RMs have different perspectives: Reactive Modules is an application-oriented language, while the focus of modular interpreted systems is more theoretic. This results in a higher abstraction level of MIS which are based on abstract states and interaction tokens. MIS aim at separating internal activities of modules and interactions between modules, what is not (explicitly) featured in RM.

Modularity in models and model checking has been the focus of many papers. Most notably, Hierarchical State Machines of Alur et al. [3, 20] and the approach of hierarchical module checking by

Murano et al. [18] feature both "horizontal" and "vertical" modularity, i.e., a system can be constructed by means of parallel composition as well as nesting of modules. Similarly, dynamic modifications and "true openness" of models has been advocated in [9]. In that paper, Dynamic Reactive Modules (DRM) were proposed, which allow for dynamic reconfiguration of the structure of the system (including adding and removing modules). Our approach differs from the ones cited above in two ways. On one hand, we focus on an abstract formulation of the *separation of concerns* between modules (and agents), rather than providing concrete mechanisms that implement the separation. On the other, we define indicators that show *how good the resulting models is*. That is, our measures of agentivity and openness are meant to assess the model "from the outside". In particular, the focus of the DRM is on providing a mechanism for adding and removing agents in the RM representation. We implement these operations on the meta-level, as a basis of the mathematical measure of openness. Our work could in principle be applied to DRMs and other formalisms, but it would require defining the appropriate multi-agent mechanisms which are already present in Interpreted Systems.

2 Modular Interpreted Systems Revisited

Modular interpreted systems were proposed in [13] to encourage modular and open design of synchronous agent systems. Below, we present an update on the formalism. The new version of MIS differs from the original one [13] as follows. First, a single agent can be now modeled by more than one module to allow for compact design of agents' local state spaces and transition functions. Secondly, the type of function in_i is now independent from the structure and cardinality of the set of agents, thus removing the main obstacle to modularity and openness of representation in the previous version. Thirdly, the interaction functions in_i, out_i are nondeterministic in order to enable nondeterministic choice and randomization (needed, e.g., to obtain fair scheduling or secure exchange of information). Fourthly, we separate agents from their names. This way, agents that are not present in the "current" MIS can be referenced in order to facilitate possible future expansion of the MIS.

2.1 New Definition of MIS

Let a bag (multiset) over set X be any function $X \to \mathbb{N}$. The set of all bags over X will be denoted by $\mathscr{B}(X)$, and the union of bags by $\uplus$.

Definition 1 (Modular interpreted system) *We define a modular interpreted system (MIS) as a tuple*

$$S = (Agtnames, Act, \mathscr{I}n, \mathbb{A}\mathrm{gt}),$$

where Agtnames is a finite set of agent names*, Act is a finite set of* action names*, $\mathscr{I}n$ is a finite* interaction alphabet*, and* $\mathbb{A}\mathrm{gt} = \{a_1, \dots, a_k\}$ *is a finite set of* agents *(whose structure is defined in the following paragraph). A set of* directed tokens*, used to specify the recipients of interactions, is defined as $\mathscr{T}ok = \mathscr{I}n \times (Agtnames \cup \{\varepsilon\})$, where ε denotes that the interaction needs to be broadcasted to all the agents in the system.*

Each agent $a_j = (id, \{m_1, \dots, m_n\})$ consists of a unique name id $\in$ Agtnames (also denoted with name(a_j)), and one or more modules *$m_j = (St_j, Init_j, d_j, out_j, in_j, o_j, \Pi_j, \pi_j)$, where:*

- *St_j is a set of* local states*,*
- *$Init_j \subseteq St_j$ is the set of initial states,*
- *$d_j : St_j \to \mathscr{P}(Act)$ defines local availability of actions; for convenience of the notation, we additionally define the set of* situated actions *as $D_j = \{(q_j, \alpha) \mid q_j \in St_j, \alpha \in d_j(q_j)\}$,*

- *out_j, in_j are* interaction *(or* interference*)* functions*:*
  - *$out_j : D_j \to \mathscr{P}(\mathscr{P}(\mathscr{T}ok))$ refers to the set of influences (chosen nondeterministically) that a given situated action (of module m_j) may possibly have on the recipients of the embedded interaction symbols, and*
  - *$in_j : St_j \times \mathscr{B}(\mathscr{I}n) \to \mathscr{P}(\mathscr{I}n)$ translates external manifestations from the other modules into the (nondeterministically chosen) "impression" that they make on module m_j depending on the local state of m_j; we assume $in_j(\cdot) \neq \emptyset$;*
- *$o_j : D_j \times \mathscr{I}n \to \mathscr{P}(St_j)$ is a* local transition function *(possibly nondeterministic),*
- *Π_j is a set of* local propositions *of module m_j (we require that Π_j and Π_m are disjoint when $j \neq m$),*
- *$\pi_j : \Pi_j \to \mathscr{P}(St_j)$ is a* valuation *of these propositions.*

Additionally, we define the cardinality of *S (denoted $card(S)$) as the number of agents in S.*

Typically, each agent in a MIS consists of exactly one module, and we will use the terms interchangeably. Also, we will omit $Init_j$ from the description of a module whenever $Init_j = St_j$.

Note that function in_j is in general infinite. For practical purposes, finite representation of in_j is needed. We use decision lists similarly to [15, 19]. Thus, in_i will be described as an ordered list of pairs of the form $condition \mapsto value$. The first pair on the list with a matching condition decides on the value of the function. The conditions are boolean combinations of membership and cardinality tests, and are defined over the variable s for the conditions defined on states, and over H for the conditions on multisets of received interferences. We require that the last condition on the list is $\top$, so that the function is total. Several examples of MIS's are presented in Sections 3 and 5.

2.2 Execution Semantics for MIS

Definition 2 (Explicit models) *A* nondeterministic concurrent epistemic game structure (NCEGS) *is a tuple $C = (\mathscr{A}, \mathscr{S}t, \mathscr{S}t_0, PV, \mathscr{V}, \mathscr{A}ct, \mathfrak{d}, t, \sim_1, \ldots, \sim_k)$, where: $\mathscr{A} = \{1, \ldots, k\}$ is a nonempty set of agents, $\mathscr{S}t$ is a nonempty set of states, $\mathscr{S}t_0 \subseteq \mathscr{S}t$ is the set of initial states, PV is a set of atomic propositions, $\mathscr{V} : PV \to \mathscr{P}(\mathscr{S}t)$ is a valuation function, $\mathfrak{d} : \mathscr{A} \times \mathscr{S}t \to \mathscr{P}(\mathscr{A}ct)$ assigns nonempty sets of actions available at each state, and t is a (nondeterministic) transition function that assigns a nonempty set $Q = t(q, \alpha_1, \ldots, \alpha_k)$ of outcome states to a state q, and a tuple of actions $(\alpha_1, \ldots, \alpha_k)$ that can be executed in q.*

We define the semantics of MIS through an unfolding to NCEGS.

Definition 3 (Unfolding of MIS) *Unfolding of the modular interpreted system S from Definition 1 to a nondeterministic concurrent epistemic game structure $NCEGS(S) = (\mathscr{A}', \mathscr{S}t', \mathscr{S}t_0', PV', \mathscr{V}', \mathscr{A}ct', \mathfrak{d}', t')$ is defined as follows:*

- *$\mathscr{A}' = \{1, \ldots, k\}$, and $\mathscr{A}ct' = Act$,*
- *$\mathscr{S}t' = \prod_{i=1}^{k} St_i$,*
- *$\mathscr{S}t_0' = \{(q_1, \ldots, q_k) \mid (\forall i \in \{1, \ldots, k\})\ q_i \in Init_i\}$,*
- *$PV' = \bigcup_{i=1}^{k} \Pi_i$, and $\mathscr{V}'(p) = \pi_i(p)$ when $p \in \Pi_i$,*
- *$\mathfrak{d}'(i, q) = d_i(q_i)$ for global state $q = (q_1, \ldots, q_k)$, and $i \in \mathscr{A}'$,*
- *The transition function t' is constructed as follows. Let $q = (q_1, \ldots, q_k)$ be a state, and $\alpha = (\alpha_1, \ldots, \alpha_k)$ be a joint action. We define an auxiliary function $\mathfrak{oi}_i(q_i, \alpha_i)$ of all the possible interferences of agent i, for q_i, and α_i: $\gamma' \in \mathfrak{oi}_i(q_i, \alpha_i)$ iff there exist $T_1, \ldots, T_k$ such that $T_j \in out_j((q_j, \alpha_j))$, and $\gamma' \in in_i(q_i, \mathscr{I}_1 \uplus \ldots \uplus \mathscr{I}_k)$, where $\mathscr{I}_j = \{\gamma_j \mid (\exists r \in \{name(a_j), \varepsilon\})\ (\gamma_j, r) \in T_j\}$ for all $j \in \mathscr{A}'$. Then $(q_1', \ldots, q_k') \in t(q, \alpha_1, \ldots, \alpha_k)$ iff $q_i' \in o_i((q_i, \alpha_i), \gamma)$, where $\gamma \in \mathfrak{oi}_i(q_i, \alpha_i)$;*

- *$q \sim_i q'$ iff q and q' agree on the local states of all the modules in agent a_i.*

Definition 3 immediately provides some important logics (such as CTL, LTL, ATL, epistemic logic, and their combinations) with semantics over modular interpreted systems. By the same token, the model checking and satisfiability problems for those logics are well defined in MIS.

3 Modeling with MIS

We argue that the revised definition of MIS achieves a high level of separation between components in a model. The interaction between an agent and the rest of the world is encapsulated in the agent's interference functions out_i, in_i. Of course, the design of the agent must take into account the tokens that can be sent from modules with which the agent is supposed to interact. For instance, the out, in functions of two communicating agents must be prepared to receive communication tokens from the other party. However, the interference functions can be oblivious to the modules with which the agent does not interact. In this section, we demonstrate the advantages on two benchmark scenarios: Trains, Tunnel, and Controller (TTC), and Dining Cryptographers (DC).

3.1 Tunnel, Trains, and Controller (TTC)

TTC is a variant of classical mutual exclusion, and models n trains moving over cyclic tracks sharing a single tunnel. Because only one train can be in the tunnel at a time, trains need to get a permission from the controller before entering the tunnel. We model the scenario by MIS $TTC_n = (Agt, Act, In)$, where:

- $Agtnames = \{tr_1, \ldots, tr_n, ctrl\}$,
- $Act = \{nop, approach, request, enter, leave\}$,
- $\mathscr{I}n = \{\mathsf{idle}, \mathsf{appr}, \mathsf{try}_1, \ldots, \mathsf{try}_n, \mathsf{retry}, \mathsf{granted}, \mathsf{left}, \mathsf{enter}, \mathsf{aw\_reqs}, \mathsf{grant}, \mathsf{grant}_1, \ldots, \mathsf{grant}_n,$ $\mathsf{no\_reqs}, \mathsf{infd}, \mathsf{ack\_release}, \mathsf{aw\_leave}\}$.
- $\mathbb{A}\mathrm{gt} = \{\mathbf{tr}_1, \ldots, \mathbf{tr}_n, \mathbf{ctrl}\}$,

The system includes n trains $\mathbf{tr}_i = \big(tr_i, \{(St_i, Init_i, d_i, out_i, in_i, o_i, \Pi_i, \pi_i)\}\big)$ for $i \in \{0, \ldots, n\}$ such that:

$St_i = \{out, tun\_needed, granted, in\}$, and $Init_i = \{out\}$. d_i is defined as:

- $out \mapsto \{nop, approach\}$,
- $tun\_needed \mapsto \{request\}$,
- $granted \mapsto \{enter\}$,
- $in \mapsto \{nop, leave\}$

o_i is defined as:

- $((out, nop), \mathsf{idle}) \mapsto \{out\}$
- $((out, approach), \mathsf{appr}) \mapsto \{tun\_needed\}$,
- $((tun\_needed, request), \mathsf{retry}) \mapsto \{tun\_needed\}$,
- $((tun\_needed, request), \mathsf{granted}) \mapsto \{granted\}$,
- $((granted, enter), \mathsf{enter}) \mapsto \{in\}$,
- $((in, nop), \mathsf{idle}) \mapsto \{in\}$,
- $((in, leave), \mathsf{leave}) \mapsto \{out\}$

$\Pi_i = \{in\_tunnel\}$

out_i is defined as:

- $(out, nop) \mapsto \{\{(\mathsf{idle}, tr_i)\}\}$,
- $(out, approach) \mapsto \{\{(\mathsf{appr}, tr_i)\}\}$,
- $(tun\_needed, request) \mapsto \{\{(\mathsf{try}_i, ctrl)\}\}$,
- $(granted, enter) \mapsto \{\{(\mathsf{enter}, tr_i)\}\}$,
- $(in, nop) \mapsto \{\{(\mathsf{idle}, tr_i)\}\}$,
- $(in, leave) \mapsto \{\{(\mathsf{left}, ctrl), (\mathsf{left}, tr_i)\}\}$

in_i is defined as:

- $s = out \wedge \mathsf{appr} \in H \mapsto \{\mathsf{appr}\}$,
- $s = tun\_needed \wedge \mathsf{grant} \in H \mapsto \{\mathsf{granted}\}$,
- $s = tun\_needed \mapsto \{\mathsf{retry}\}$,
- $s = granted \wedge \mathsf{enter} \in H \mapsto \{\mathsf{granted}\}$,
- $s = in \wedge \mathsf{left} \in H \mapsto \{\mathsf{left}\}$,
- $\top \mapsto \{\mathsf{idle}\}$

$\pi_i = \{in \mapsto in\_tunnel\}$

Moreover, the agent **ctrl** $= \big(ctrl, \{(St_c, Init_c, d_c, out_c, in_c, o_c, \Pi_c, \pi_c)\}\big)$ modeling the controller is defined as follows:

$St_c = \{tun\_free, infd, tr_1granted, \dots, tr_ngranted\}$, and $Init_c = \{tun\_free\}$.

d_c is defined as:

- $tun\_free \mapsto \{accepting\}$,
- $infd \mapsto \{waiting\}$,
- $tr_1granted \mapsto \{inform\}$,
- $\dots$
- $tr_ngranted \mapsto \{inform\}$

o_c is defined as:

- $((tun\_free, accepting), \mathsf{no\_reqs}) \mapsto \{tun\_free\}$,
- $((infd, waiting), \mathsf{aw\_leave}) \mapsto \{infd\}$,
- $((infd, waiting), \mathsf{ack\_release}) \mapsto \{tun\_free\}$,
- $((tun\_free, accepting), \mathsf{grant_1}) \mapsto \{tr_1granted\}$,
- $\dots$
- $((tun\_free, accepting), \mathsf{grant_n}) \mapsto \{tr_ngranted\}$,
- $((tr_1granted, inform), \mathsf{infd}) \mapsto \{infd\}$,
- $\dots$
- $((tr_ngranted, inform), \mathsf{infd}) \mapsto \{infd\}$

out_c is defined as:

- $(tun\_free, accepting) \mapsto \{\{(\mathsf{aw\_reqs}, \varepsilon)\}\}$,
- $(infd, waiting) \mapsto \{\{(\mathsf{aw\_leave}, ctrl)\}\}$,
- $(tr_1granted, inform) \mapsto \{\{(\mathsf{grant}, tr_1)\}\}$,
- $\dots$
- $(tr_ngranted, inform) \mapsto \{\{(\mathsf{grant}, tr_n)\}\}$,

in_c is defined as:

- $s = tun\_free \wedge \mathsf{try}_1 \in H \mapsto \{\mathsf{grant_1}\}$,
- $\dots$
- $s = tun\_free \wedge \mathsf{try}_n \in H \mapsto \{\mathsf{grant_n}\}$,
- $s = tun\_free \mapsto \{\mathsf{no\_reqs}\}$,
- $s = tr_1granted \vee \dots \vee s = tr_ngranted \mapsto \{\mathsf{infd}\}$,
- $s = infd \wedge \mathsf{left} \in H \mapsto \{\mathsf{ack\_release}\}$,
- $s = infd \mapsto \{\mathsf{aw\_leave}\}$,
- $\top \mapsto \{\mathsf{idle}\}$

$\Pi_c = \{tunnel\_busy\}$
$\pi_c = \{infd \mapsto tunnel\_busy\}$

The model is illustrated in Figure 1 using the notation introduced in Section 3.2. The protocol focuses on the procedure of gaining a permission to access the tunnel. Before requesting the permission, a train approaches the tunnel, and its state changes to *tun_needed*. In this state it requests the permission from the controller. When the controller grants the permission to one of the nondeterministically chosen trains ($tr_igranted$) it informs the train that got access to the tunnel about this fact, and moves to the state $infd$. The train enters the tunnel in the next step of the protocol, and changes its state to *in*, whereas the remaining trains may continue requesting the access (they remain in *tun_needed*). When the train leaves the tunnel, it changes its state to tun_free.

3.2 Graphical Representation

As the definitions of MIS tend to be verbose, we introduce a simple graphical notation, based on networks of communicating automata. Let us explain it, based on Figure 1, which is a graphical representation of the tunnel, trains, and controller model from Section 3.1:

- Modules defining different **agents** and belonging to the same agent are separated by solid and dashed lines, respectively,
- Circles correspond to **local states**. An arrow with loose end pointing into a circle denotes an initial state,
- Boxes define **local actions** associated with a state,
- For a local action, dashed lines going out of it define **emitted influences**, specified with the receiver and the influence at the left and right side of an harpoon arrow pointed left, respectively. When no receiver is specified, the influence is broadcasted,

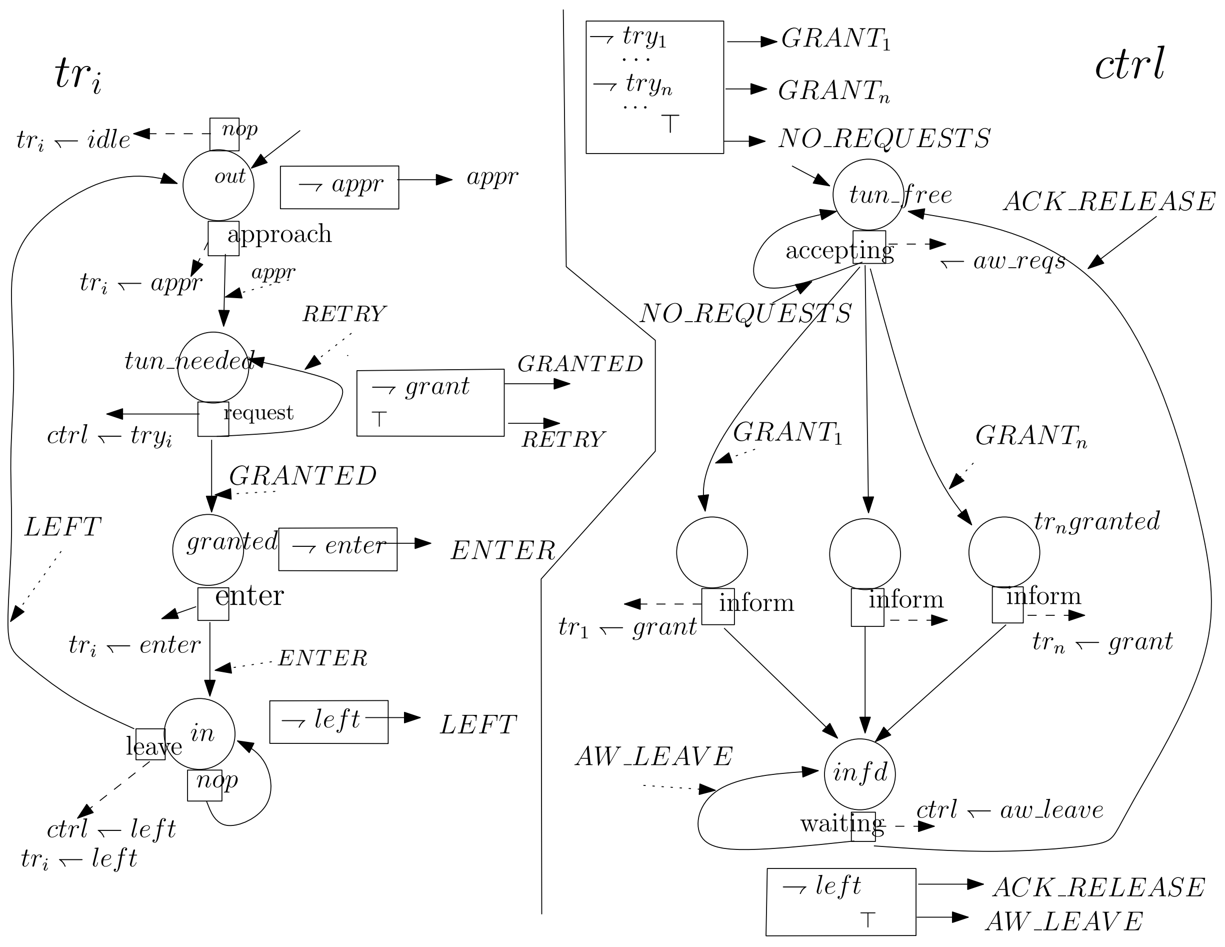


Figure 1: Tunnel, trains and controller (TTC) in the graphical representation

- Solid lines with arrows, connecting an action with a local state, correspond to a **local transition function**,
- For a local state, guarded commands (possibly in a box) define the translation of **external manifestations** received by an agent into **local impressions**. A harpoon arrow pointed right corresponds to a sender at the left side and the message at the right side, and if the sender is not specified it means receiving from anyone. For a transition, dotted arrows pointing at it correspond to application of those impressions.

The number of interactions x received by an agent is denoted with $n(x)$. The notation $*$ labeling a transition means that it is executed when none of the remaining transitions are enabled. For example, it could be used instead of directly specifying the generation and application of *aw_leave* manifestation in the controller.

Some parts can be skipped or abstracted away if it does not lead to confusion. For example, if no influence is emitted and only one transition is associated with an action, this action needs not be directly specified. In Figure 1, the self-loop from the *in* state is not accompanied by the associated local impression nor the impression. Similarly, a single influence addressed to the very module that issued it can be omitted. For example, we do not show the manifestation *idle* in the graph. Valuations of

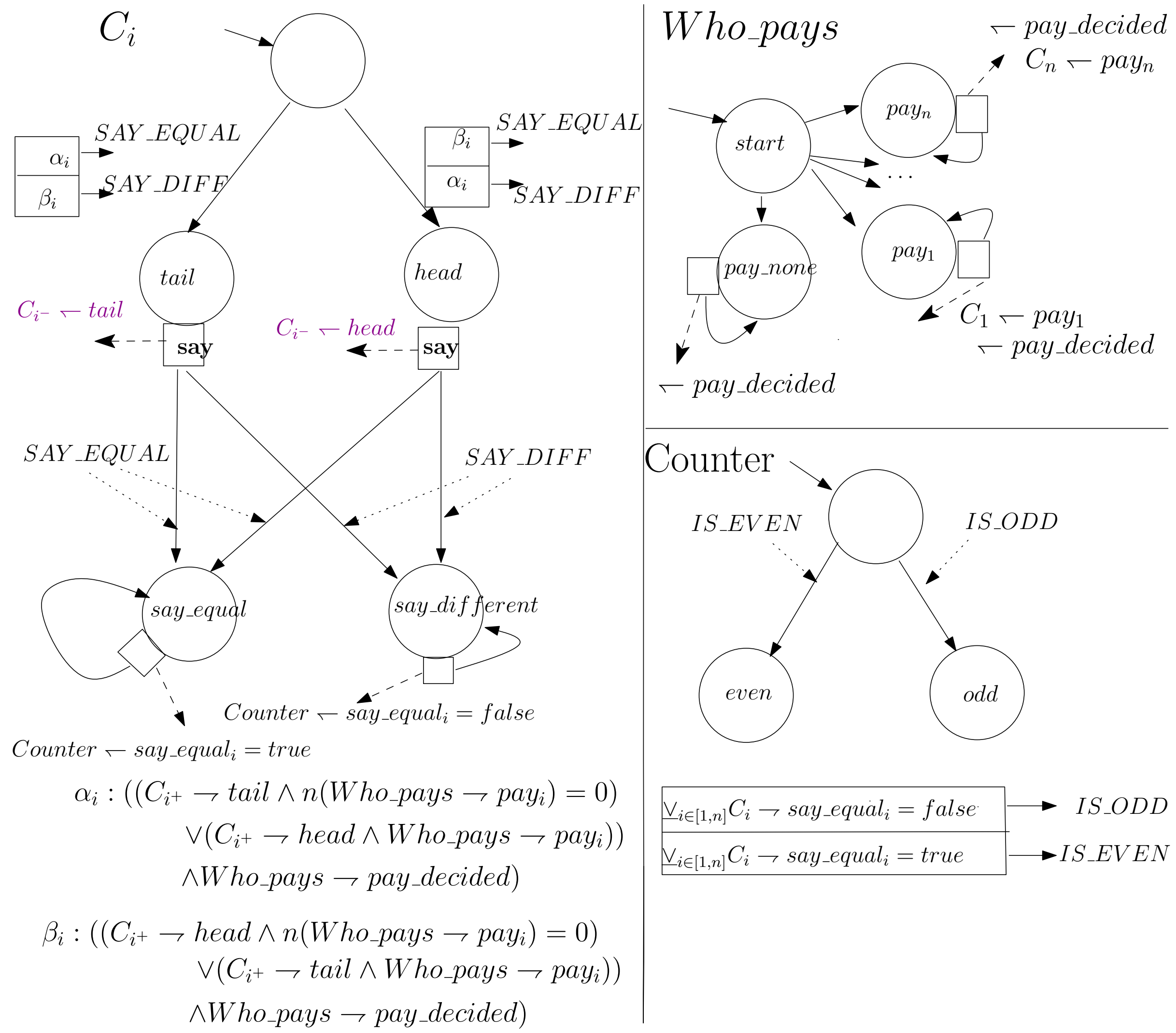

C_i
SAY_EQUAL
SAY_DIFF
α_i
β_i
tail
head
$C_{i^-} \leftharpoondown tail$
$C_{i^-} \leftharpoondown head$
say
say
SAY_EQUAL
SAY_DIFF
say_equal
say_different
$Counter \leftharpoondown say\_equal_i = false$
$Counter \leftharpoondown say\_equal_i = true$
$\alpha_i : ((C_{i^+} \rightharpoondown tail \wedge n(Who\_pays \rightharpoondown pay_i) = 0) \vee (C_{i^+} \rightharpoondown head \wedge Who\_pays \rightharpoondown pay_i)) \wedge Who\_pays \rightharpoondown pay\_decided)$
$\beta_i : ((C_{i^+} \rightharpoondown head \wedge n(Who\_pays \rightharpoondown pay_i) = 0) \vee (C_{i^+} \rightharpoondown tail \wedge Who\_pays \rightharpoondown pay_i)) \wedge Who\_pays \rightharpoondown pay\_decided)$
Who_pays
$\leftharpoondown pay\_decided$
$C_n \leftharpoondown pay_n$
start
pay_n
...
pay_1
pay_none
$C_1 \leftharpoondown pay_1$
$\leftharpoondown pay\_decided$
$\leftharpoondown pay\_decided$
Counter
IS_EVEN
IS_ODD
even
odd
$\bigvee_{i\in[1,n]} C_i \rightharpoondown say\_equal_i = false$ → IS_ODD
$\bigvee_{i\in[1,n]} C_i \rightharpoondown say\_equal_i = true$ → IS_EVEN

Figure 2: MIS for dining cryptographers (DC1)

propositions can be depicted in a similar way as for networks of automata. We omit them in our examples throughout, as they do not play a role in this paper.

3.3 Dining Cryptographers: Standard Version (DC1)

Dining Cryptographers is a well-known benchmark proposed by Chaum [5]. *n* cryptographers are having dinner, and the bill is to be paid anonymously, either by one of them or by their employer. In order to learn which option is the case without disclosing which cryptographer is paying (if any), they run a two-stage protocol. First, every cryptographer is paired with precisely one other participant (they sit around the table), thus forming a cycle. Every pair shares a one-bit secret, say by tossing a coin behind a menu. In the second stage, each cryptographer publicly announces whether he sees an odd or an even number of coin heads, saying the opposite if being the payer.

In the simplest case (DC1) the number of cryptographers is fixed, and each cryptographer is directly bound with its neighbours. Cryptographers announce their utterances by broadcasting them. A modular interpreted system modeling this setting is presented in Figure 2. For n cryptographers, the ith cryptographer is modeled by agent C_i ($0 \leq i \leq n$). We introduce notation i^+ and i^- to refer to the right and the left neighbour of cryptographer i, respectively. The system includes also two additional agents. Who_pays initializes the system by determining who is the payer, and communicating it to the cryptographers. According to the protocol definition, either one of the cryptographers is chosen, or none of the participants pays. Agent $Counter$ counts the utterances of the cryptographers, computes the XOR operation (denoted by $\veebar$ and assuming that utterances *different* and *equal* correspond to true and false values, respectively, thus the result is true iff the number of *different* utterances is odd), and determines the outcome of the protocol. Figure 2 shows the modular interpreted system for DC1.

4 How to Measure Multi-Agency

In this section, we present our preliminary attempt at defining what it means for a design to be multi-agent. Intuitively, separate agents should have only limited coordination and/or communication capabilities. Otherwise, the whole system can be seen as a single agent in disguise. The idea is to measure the complexity of interference between different agents, and relate it to the complexity of the system. The former factor will be captured by the number of directed interaction tokens that a given agent can generate; the latter by the number of global transitions that can occur. We say that the agent is well designed if its interference complexity is reasonably smaller than overall complexity of the system.

Definition 4 (Interaction complexity) *The* interaction complexity of agent *i in modular interpreted system M, denoted $IC(i)$, is defined as follows. Let $\#out_i(q_i)$ be the the maximal number of directed tokens generated by function out_i to modules of other agents in state q_i. Furthermore, let $\#in_i(q_i)$ be the maximal number of tokens admitted by function in_i from modules of other agents in state q_i. Now, $IC(i) = \sum_{q_i \in St_i}(\#out_i(q_i) + \#in_i(q_i))$.*

The interaction complexity of *M is defined as $IC(M) = \sum_{i \in \mathbb{A}\mathrm{gt}} IC(i)$.*

Definition 5 (Global complexity) *The* global complexity of MIS *M, denoted $GC(M)$, is the number of transitions in the NCEGS unfolding of M.*

How can we express that $IC(M)$ is "reasonably smaller" than $GC(M)$? Such a requirement is relatively easy to specify for *classes* of models, parameterized with values of some parameter (for instance, the number of identical trains in the tunnel-controller scenario).

Definition 6 ($\mathscr{C}$-sparse interaction, multi-agent design) *Let $\mathscr{M}$ be a class of MIS and $\mathscr{C}$ a class of complexity functions $f : \mathbb{N} \to \mathbb{R}_+ \cup \{0\}$. We say that $\mathscr{M}$ is characterized by $\mathscr{C}$-*sparse interaction *iff there is a function $f \in \mathscr{C}$ such that $IC(M) \leq f(GC(M))$ for every $M \in \mathscr{M}$.*

Furthermore, we say that $\mathscr{M}$ has multi-agent design *iff $\mathscr{M}$ has LOGTIME-sparse interaction, and $card(M) \geq 2$ for every $M \in \mathscr{M}$.*

Proposition 1 *Classes TTC and DC1 have multi-agent design.*

The proof is straightforward. It is easy to see that the other variants of Dining Cryptographers, discussed in Section 5, also have multi-agent design.

5 How Open is an Open System?

The idea of open systems is important for several communities: not only MAS, but also verification, software engineering, etc. It is becoming even more important now, with modern technologies enabling dynamic networks of devices, users and services whose nodes can be created and removed according to current needs. Traditionally, the term *open system* is understood as a process coupled with the environment, which is rather disappointing given the highly distributed nature of MAS nowadays. One would rather like "openness" to mean that components (agents in our case) can freely join and leave the system without the need to redesign the rest of it.

Perfectly open systems are seldom in practice; usually, adding/removing components requires some transformation of the remaining part (for instance, if a server is to send personalized information to an arbitrary number of clients then it must add the name of each new client to the appropriate distribution lists). So, it is rather the degree of openness that should be captured. We try to answer the question *How open is the system?* (or, to be more precise, its model) in the next subsection.

5.1 A Measure of Openness

We base the measure on the following intuition: openness of a system is *simplicity of adding and removing agents to and from the model*. That is, we consider two natural transformations of models: expansion (adding agents) and reduction (removing agents). We note that the simplicity of a transformation is best measured by its algorithmic complexity, i.e., the number of steps needed to complete the transformation. A perfectly open system requires no transformation at all (0 steps) to accommodate new components, whereas at the other extreme we have systems that require redesigning of the model from scratch whenever a new agent arrives.

Note that the openness of a model depends on *which* agents want to join or leave. For instance, the system with trains and controllers should be able to easily accommodate additional trains, but not necessarily additional controllers. Likewise, departure of a train should be straightforward, but not necessarily that of the controller. No less importantly, the context matters. We are usually not interested in an arbitrary expansion or reduction (which are obviously trivial). We want to add or remove agents while keeping the "essence" of the system's behavior intact. The following definitions formalize the idea.

Definition 7 (Expansion and reduction of a MIS) *Let* $M = (Agtnames, Act, \mathscr{I}n, \mathbb{A}\mathrm{gt})$ *be a MIS, and* $\boldsymbol{a}$ *an agent (in the sense of Definition 1). By* $agt(\boldsymbol{a})$ *(resp.* $act(\boldsymbol{a})$*,* $in(\boldsymbol{a})$*) we denote the set of agent names (resp. action symbols, interaction symbols) occurring in* $\boldsymbol{a}$*. Moreover,* $ns(\boldsymbol{a}, M)$ *will denote the set of* $\boldsymbol{a}$*'s namesakes in* M.[1] *Note that* $ns(\boldsymbol{a}, M)$ *can contain at most* 1 *agent.*

The expansion of M by $\boldsymbol{a}$ *is defined as the modular interpreted system* $M \oplus \boldsymbol{a} = (Agtnames', Act', \mathscr{I}n', \mathbb{A}\mathrm{gt}')$ *where:* $Agtnames' = Agtnames \cup agt(\boldsymbol{a})$*,* $Act' = Act \cup act(\boldsymbol{a})$*,* $\mathscr{I}n' = \mathscr{I}n \cup in(\boldsymbol{a})$*, and* $\mathbb{A}\mathrm{gt}' = \mathbb{A}\mathrm{gt} \setminus ns(\boldsymbol{a}, M) \cup \{\boldsymbol{a}\}$*. The* reduction of M by $\boldsymbol{a}$ *is defined as* $M \ominus \boldsymbol{a} = (Agtnames, Act, \mathscr{I}n, \mathbb{A}\mathrm{gt}')$ *where* $\mathbb{A}\mathrm{gt}' = \mathbb{A}\mathrm{gt} \setminus \{\boldsymbol{a}\}$.

Thus, expansion corresponds to "dumb" pasting an agent into a MIS, and reduction corresponds to simple removal of the agent. The operations are well defined in the following sense.

Proposition 2 *Expansion/reduction of a MIS is always a MIS.*[2]

It is easy to see that removing an agent and pasting it in again does not change the MIS. The reverse sequence of operations does change the MIS. However, both structures have the same unfoldings:

[1] That is, agents in M that have the same id as $\boldsymbol{a}$.

[2] The proofs of results in Section 5 are straightforward from the construction of MIS, and we leave them to the reader.

Proposition 3 *Let $\boldsymbol{a}$ be an agent in M. Then, $(M \ominus \boldsymbol{a}) \oplus \boldsymbol{a} = M$. Moreover, let $\boldsymbol{a}$ be an agent with no namesake in M. Then, $NCEGS((M \oplus \boldsymbol{a}) \ominus \boldsymbol{a}) = NCEGS(M)$.*

Now we can make our first attempt at a measure of openness.

Definition 8 (Degree of openness) *Let θ be a property of models,*[3] *M a modular interpreted system, and $\boldsymbol{a}$ an agent. The* degree of openness *of M wrt expansion by $\boldsymbol{a}$ under constraint θ is defined as the minimal number of steps that transform $M \oplus \boldsymbol{a}$ into a MIS M' such that $card(M') = card(M \oplus \boldsymbol{a})$ and M' satisfies θ.*

Likewise, the degree of openness of modular interpreted system M wrt reduction by agent $\boldsymbol{a}$ under constraint θ is the minimal number of steps that transform $M \ominus \boldsymbol{a}$ into an M' such that $card(M') = card(M \ominus \boldsymbol{a})$ and M' satisfies θ.

The constraint θ can for example refer to liveness of the system or some of its components, fairness in access to some resources, and/or safety of critical sections. Note that the cardinality check is essential in the definition – otherwise, a possible transformation would be to simply delete the newly added agent from $M \oplus \boldsymbol{a}$ (respectively, to restore **a** in $M \ominus \boldsymbol{a}$).

Definition 9 (Openness of a class of models) *Let $\mathscr{M}$ be a class of MIS, $\boldsymbol{a}$ an agent, and θ a property of models. Moreover, let $\mathscr{C}$ be a class of complexity functions $f : \mathbb{N} \to \mathbb{R}_+ \cup \{0\}$. $\mathscr{M}$ is $\mathscr{C}$-*open *wrt expansion (resp. reduction) by $\boldsymbol{a}$ under constraint θ iff there is a complexity function $f \in \mathscr{C}$ such that for every $M \in \mathscr{M}$ the degree of openness of M wrt expansion (resp. reduction) by $\boldsymbol{a}$ under θ is no greater than $f(|M|)$.*

The most cumbersome part of the above definitions is the constraint θ. How can one capture the "essence" of acceptable expansions and reductions? Note that, semantically, θ can be seen as a subclass of models. We postulate that in most scenarios the class that defines acceptable expansions/reductions is the very class whose openness we want to measure. This leads to the following refinement of the previous definitions.

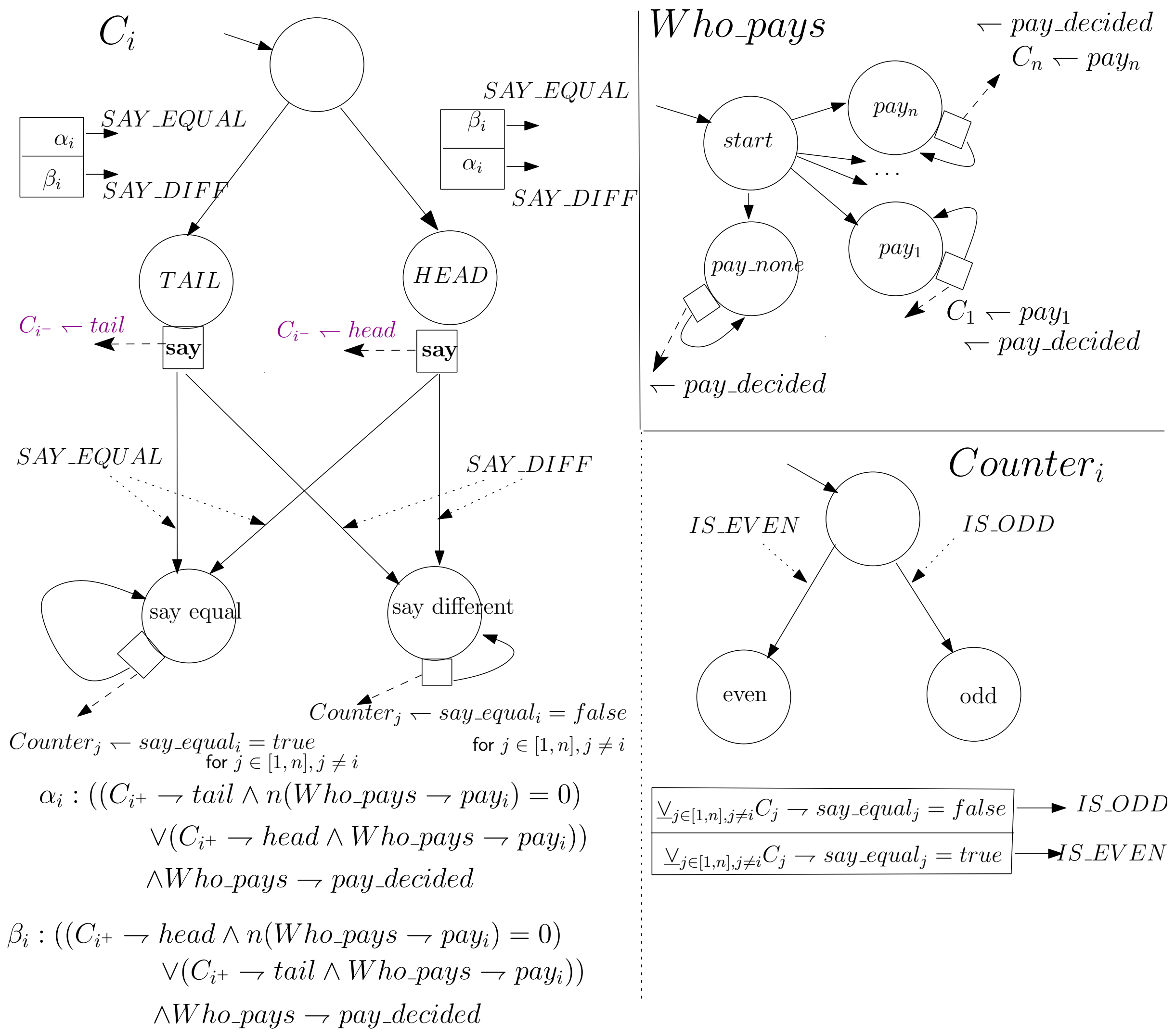
Definition 10 (Openness in a class) *The degree of openness of M wrt expansion (resp. reduction) by $\boldsymbol{a}$ in class $\mathscr{M}$ is the minimal number of steps that transform $M \oplus \boldsymbol{a}$ (resp. $M \ominus \boldsymbol{a}$) into a MIS $M' \in \mathscr{M}$ such that $card(M') = card(M \oplus \boldsymbol{a})$.*

Moreover, $\mathscr{M}$ is $\mathscr{C}$-open wrt expansion (resp. reduction) by $\boldsymbol{a}$ iff there is a complexity function $f \in \mathscr{C}$ such that for every $M \in \mathscr{M}$ the degree of openness of M wrt expansion (resp. reduction) by $\boldsymbol{a}$ in $\mathscr{M}$ is no greater than $f(|M|)$.

We explain the measure in greater detail in the remainder of Section 5. It is important to note that (in contrast to the measure of multi-agentivity proposed in Section 4) our measure of openness is not specific to MIS, and can be applied to other modeling frameworks.

Remark 4 *Alternatively, we could define the openness of M wrt $\boldsymbol{a}$ and θ by the* Kolmogorov complexity *of an appropriate expansion/reduction, i.e., by the size of the shortest algorithm that transforms M in an appropriate way. We chose time complexity instead, for two reasons. First, Kolmogorov complexity often obscures the level of difficulty of a process (e.g., a two-line algorithm with an infinite **while** loop can implement infinitely many changes, which gives the same complexity as changing the names of two communication channels for a controller). Secondly, computing Kolmogorov complexity can be cumbersome as it is Turing-equivalent to answering the halting problem.*

[3] We do not restrict the language in which θ is specified. It can be propositional logic, first-order temporal logic, or even the general language of mathematics. The only requirement is that, for every MIS M, the truth of θ in M is well defined.


C_i
α_i
β_i
SAY_EQUAL
SAY_DIFF
$TAIL$
$HEAD$
$C_{i^-} \leftharpoondown tail$
$C_{i^-} \leftharpoondown head$
say
say
say equal
say different
$Counter_j \leftharpoondown say\_equal_i = true$ for $j \in [1,n], j \neq i$
$Counter_j \leftharpoondown say\_equal_i = false$ for $j \in [1,n], j \neq i$
Who_pays
$\leftharpoondown pay\_decided$
$C_n \leftharpoondown pay_n$
start
pay_n
pay_1
pay_none
$C_1 \leftharpoondown pay_1$
$Counter_i$
IS_EVEN
IS_ODD
even
odd
$\veebar_{j\in[1,n],j\neq i} C_j \rightharpoondown say\_equal_j = false$
$\veebar_{j\in[1,n],j\neq i} C_j \rightharpoondown say\_equal_j = true$

$$\alpha_i : ((C_{i^+} \rightharpoondown tail \wedge n(Who\_pays \rightharpoondown pay_i) = 0) \vee (C_{i^+} \rightharpoondown head \wedge Who\_pays \rightharpoondown pay_i)) \wedge Who\_pays \rightharpoondown pay\_decided$$

$$\beta_i : ((C_{i^+} \rightharpoondown head \wedge n(Who\_pays \rightharpoondown pay_i) = 0) \vee (C_{i^+} \rightharpoondown tail \wedge Who\_pays \rightharpoondown pay_i)) \wedge Who\_pays \rightharpoondown pay\_decided$$

Figure 3: Dining cryptographers version DC2: direct channels instead of broadcast

We observe, however, that a Kolmogorov-style measure of openness can be a good alternative for infinite models, especially ones that require infinitely many steps to accommodate changes in the configuration of components.

5.2 How to Open Up Cryptographers

In Section 3.3 we modeled the standard version of the Dining Cryptographers protocol as a modular interpreted system (class DC1). In this section, we will determine the openness of DC1, plus two other classes of MIS modeling other versions of the protocol. To comply with classical rules of composition, we begin with the least open variant.

5.2.1 DC-Net, Direct Channels, No Broadcasting (DC2)

Let us assume that no broadcast channel is available, or it is too faulty (or insecure) to be of use in multi-party computation. In such case, every pair of cryptographers must use a direct secured channel for communicating the final utterance. The result of the computation is calculated independently by every cryptographer. We denote this class of models by DC2, and construct it as follows. Each cryptographer i is modeled by agent C_i, similar to the cryptographer agents in DC1. Instead of a single global counter of utterances, there is one counter per every cryptographer ($Counter_i$). The final utterance is sent by direct point-to-point channels to the counters of all other participants. The resulting MIS is shown in Figure 3.

Adding a new cryptographer C_i to DC1_n requires the following changes. First, modifying links among the new neighbours of C_i yields 10 changes. Secondly, every agent C_j in DC1_n must be modified in order to establish a communication channel with C_i. This requires $2 \cdot 5$ changes per cryptographer, thus $10n$ changes are needed. Thirdly, for the agent Who_pays, we add the state $pay_{i'}$ with corresponding transitions: a single non-deterministic transition from $start$ to $pay_{i'}$ (17 steps: 2 for d_i + 4 for out_i + 8 for in_i + 3 for o_i), and the loop sending payment information (19 steps: 4 for d_i + 4 for out_i + 4 for in_i + 3 for o_i). Finally, $Counter$ needs to be updated to take into account the new participant. A XOR argument is added with receiving a manifestation, yielding $2 \cdot 4 = 8$ changes. Thus, the overall openness complexity for DC2_n is $10n + 54$.

Proposition 5 *Class DC2 is $O(n)$-open wrt expansion by a cryptographer.*

5.2.2 Dining Cryptographers: Standard Version with Broadcast (DC1)

Let us now go back to the standard version of the protocol, presented in Section 3.3 Adding a new cryptographer C_i requires the following changes. First, modifying links for the new neighbors of C_i requires 10 changes. Secondly, changes in Who_pays and $Counter$ are the same as for DC2_n, yielding 44 steps. Thus, 54 changes are needed to accommodate the new cryptographer, regardless of the number of agents already present in the system.

Proposition 6 *Class DC1 is $O(1)$-open wrt expansion by a cryptographer.*

5.2.3 Fully Open System, Cryptographers without Identifiers (DC0)

In our most radical variant, cryptographers are not arbitrarily assigned as neighbors. Instead, they establish their neighborhood relation on their own before starting the protocol. Every cryptographer is modeled by two modules C_i and Pay_i, and there are two additional agents $Oracle$ and $Counter$, cf. Figure 4. The system proceeds as follows:

Setting up the payer. Every cryptographer sends the oracle his declaration whether he is going to pay or not (chosen nondeterministically). This is performed by module Pay_i. If $Oracle$ receives at most one statement $want\_pay$, it confirms to all cryptographers. If more than one statements $want\_pay$ is sent, the round is repeated until the payment issue becomes resolved.

Establishing the neighbourhood relation and tossing coins. Each cryptographer either nondeterministically tosses a coin and announces the outcome, or listens to such announcements from the other agents. If there is exactly one cryptographer announcing and one listening, they become paired. They register the value of the announcement, and proceed further. A cryptographer who started with announcing will now listen, and vice versa. This takes several rounds, and completes when every cryptographer has been paired with two neighbors (one to whom he listened, and one to whom he announced).

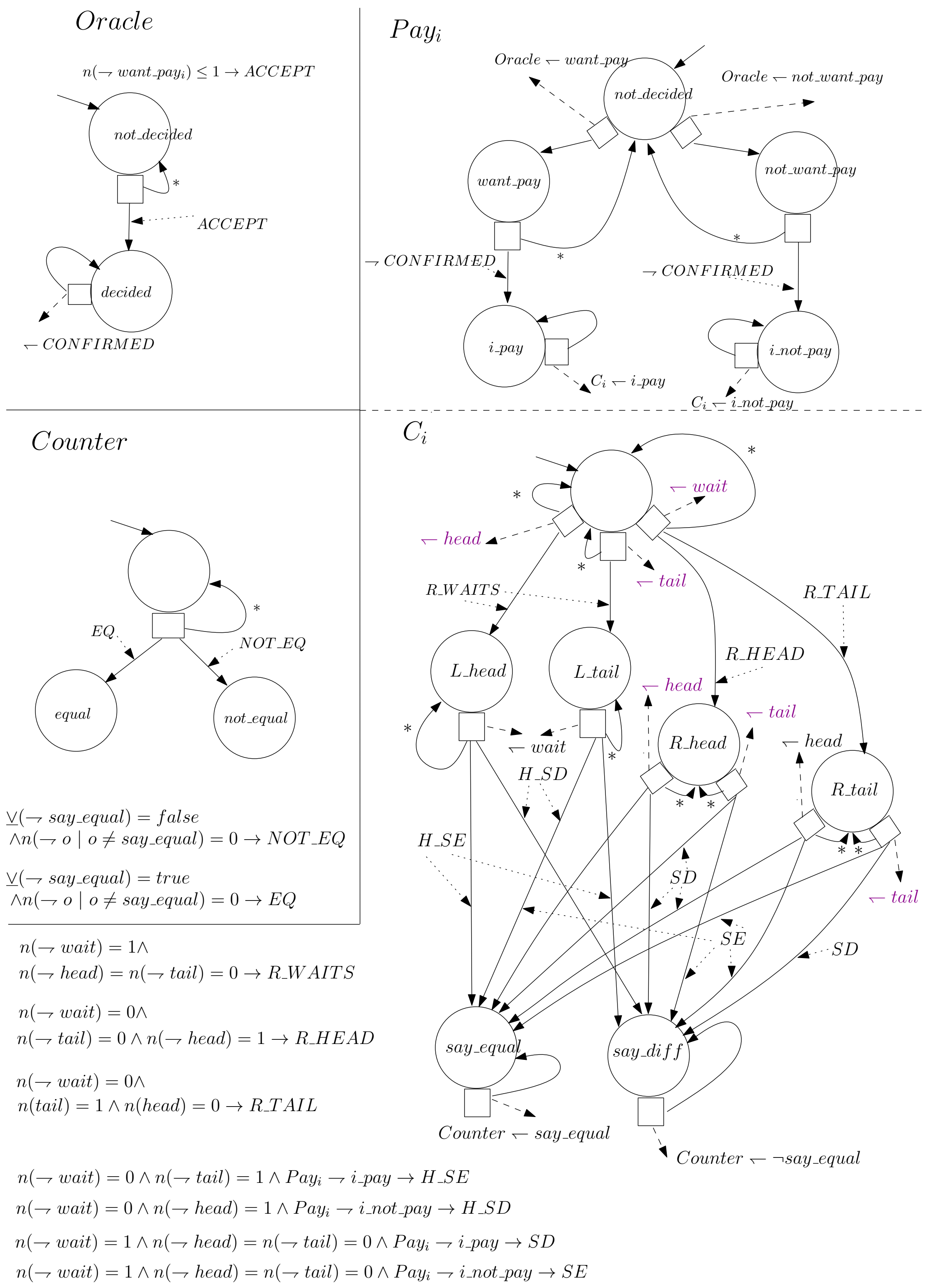

Oracle
$n(\leftharpoondown want\_pay_i) \leq 1 \rightarrow ACCEPT$
not_decided
*
ACCEPT
decided
$\leftharpoonup CONFIRMED$
Pay_i
$Oracle \leftharpoonup want\_pay$
not_decided
$Oracle \leftharpoonup not\_want\_pay$
want_pay
not_want_pay
*
*
$\leftharpoondown CONFIRMED$
$\leftharpoondown CONFIRMED$
i_pay
i_not_pay
$C_i \leftharpoonup i\_pay$
$C_i \leftharpoonup i\_not\_pay$
Counter
*
EQ
NOT_EQ
equal
not_equal
$\underline{\vee}(\leftharpoondown say\_equal) = false$
$\wedge n(\leftharpoondown o \mid o \neq say\_equal) = 0 \rightarrow NOT\_EQ$
$\underline{\vee}(\leftharpoondown say\_equal) = true$
$\wedge n(\leftharpoondown o \mid o \neq say\_equal) = 0 \rightarrow EQ$
$n(\leftharpoondown wait) = 1 \wedge$
$n(\leftharpoondown head) = n(\leftharpoondown tail) = 0 \rightarrow R\_WAITS$
$n(\leftharpoondown wait) = 0 \wedge$
$n(\leftharpoondown tail) = 0 \wedge n(\leftharpoondown head) = 1 \rightarrow R\_HEAD$
$n(\leftharpoondown wait) = 0 \wedge$
$n(tail) = 1 \wedge n(head) = 0 \rightarrow R\_TAIL$
$n(\leftharpoondown wait) = 0 \wedge n(\leftharpoondown tail) = 1 \wedge Pay_i \leftharpoondown i\_pay \rightarrow H\_SE$
$n(\leftharpoondown wait) = 0 \wedge n(\leftharpoondown head) = 1 \wedge Pay_i \leftharpoondown i\_not\_pay \rightarrow H\_SD$
$n(\leftharpoondown wait) = 1 \wedge n(\leftharpoondown head) = n(\leftharpoondown tail) = 0 \wedge Pay_i \leftharpoondown i\_pay \rightarrow SD$
$n(\leftharpoondown wait) = 1 \wedge n(\leftharpoondown head) = n(\leftharpoondown tail) = 0 \wedge Pay_i \leftharpoondown i\_not\_pay \rightarrow SE$
C_i
*
*
$\leftharpoonup wait$
$\leftharpoonup head$
*
$\leftharpoonup tail$
R_WAITS
R_TAIL
R_HEAD
L_head
L_tail
$\leftharpoonup head$
$\leftharpoonup tail$
$\leftharpoonup wait$
$\leftharpoonup head$
R_head
R_tail
*
*
H_SD
* *
* *
H_SE
SD
$\leftharpoonup tail$
SE
SD
say_equal
say_diff
$Counter \leftharpoonup say\_equal$
$Counter \leftharpoonup \neg say\_equal$

Figure 4: Cryptographers without identifiers (DC0)

Computation. A broadcast channel is used for sending around the utterances (*say_equal* or $\neg$*say_equal*). *Counter* counts the utterances and computes their XOR on the spot, in the way described before.

DC0 is fully open, as adding a new cryptographer requires no adaptation of $\mathrm{DC0}_n$.

Proposition 7 *Class DC0 is $O(0)$-open wrt expansion by a cryptographer.*

By comparing their classes of openness, it is clear that DC1 is significantly more open wrt expansion than DC2 (constant vs. linear openness). On the other hand, it seems that the gap between DC1 and DC0 is rather slight ($O(1)$ vs. $O(0)$). Is that really the case? We believe that the difference between $O(1)$-openness and $O(0)$-openness is larger than one is used to in complexity of algorithms. First, constant openness means that, when expanding the MIS by a *set* of new agents, the required transformation can be linear in the size of the set. More importantly, non-zero openness signifies the need to come up with a *correct* procedure of expansion. In contrast, zero openness means zero hassle: the new agents can join the system as they come. There is no need for "maintenance" of the system so that it stays compliant with its (usually implicit) specification.

6 Conclusions

In this paper, we propose a new version of modular interpreted systems. The aim is to let modeling and analysis of multi-agent systems benefit from true separation of interference between agents and the "internals" of their processes that go on in a system. Thanks to that, one can strive for a more modular and open design. Even more importantly, one can use the MIS representation of a system to assess its agentivity and openness through application of simple mathematical measures.

We emphasize that it was *not* our aim to create yet another agent programming language or representations that will be used as input to cutting-edge model checkers. Instead, we propose a class of models which enables to expose the internal structure of a multi-agent system, and to define the concepts of openness and multi-agentivity in a precise mathematical sense. While our definition of multi-agentivity is specific to MIS, the measure of openness is in fact generic, and can be applied to models defined in other formalisms (such as Reactive Modules). We plan to look closer at the degree of openness provided by different representation frameworks in the future.

We would also like to stress that the focus of this paper regarding the measures of agentivity and openness is on formalizing the concepts and showing how they work on benchmarks. An formal study of the measures and their properties is a matter of future work.

Acknowledgements. The authors thank Andrzej Tarlecki for his suggestion to improve modularity of MIS by using multisets, and Thomas Agotnes for discussions. Wojciech Jamroga acknowledges the support of the FNR (National Research Fund) Luxembourg under project GALOT – INTER/DFG/12/06. Artur Męski acknowledges the support of the European Union, European Social Fund. Project PO KL "Information technologies: Research and their interdisciplinary applications" (UDA-POKL.04.01.01-00-051/10-00).

References

[1] R. Alur & T. A. Henzinger (1999): *Reactive Modules*. *Formal Methods in System Design* 15(1), pp. 7–48, doi:10.1023/A:1008739929481.

[2] R. Alur, T. A. Henzinger & O. Kupferman (2002): *Alternating-Time Temporal Logic*. Journal of the ACM 49, pp. 672–713, doi:10.1145/585265.585270.

[3] R. Alur, S. Kannan & M. Yannakakis (1999): *Communicating Hierarchical State Machines*. In: Proceedings of ICALP, pp. 169–178, doi:10.1007/3-540-48523-6_14.

[4] J. Calta (2012): *Synthesis of Strategies for Multi-Agent Systems*. Ph.D. thesis, Humboldt University Berlin.

[5] D. Chaum (1988): *The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability*. Journal of Cryptology 1(1), pp. 65–75, doi:10.1007/BF00206326.

[6] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna & A. Zbrzezny (2003): *Verics: A Tool for Verifying Timed Automata and Estelle Specifications*. In: Proceedings of the of the 9th Int. Conf. on Tools and Algorithms for Construction and Analysis of Systems (TACAS'03), LNCS 2619, Springer, pp. 278–283, doi:10.1007/3-540-36577-X_20.

[7] E. A. Emerson (1990): *Temporal and Modal Logic*. In J. van Leeuwen, editor: Handbook of Theoretical Computer Science, B, Elsevier Science Publishers, pp. 995–1072.

[8] R. Fagin, J. Y. Halpern, Y. Moses & M. Y. Vardi (1995): *Reasoning about Knowledge*. MIT Press.

[9] J. Fisher, T. A. Henzinger, D. Nickovic, N. Piterman, A. V. Singh & M. Y. Vardi (2011): *Dynamic Reactive Modules*. In: Proceedings of CONCUR, pp. 404–418, doi:10.1007/978-3-642-23217-6_27.

[10] F. Gecseg (1986): *Products of Automata*. EATCS Monographs on Theor. Comput. Sci., Springer, doi:10.1007/978-3-642-61611-2.

[11] G. J. Holzmannn (1997): *The Model Checker SPIN*. IEEE Transactions on Software Engineering 23(5), pp. 279–295, doi:10.1109/32.588521.

[12] W. Jamroga & T. Ågotnes (2006): *Modular Interpreted Systems: A Preliminary Report*. Technical Report IfI-06-15, Clausthal University of Technology.

[13] W. Jamroga & T. Ågotnes (2007): *Modular Interpreted Systems*. In: Proceedings of AAMAS'07, pp. 892–899, doi:10.1145/1329125.1329286.

[14] M. Köster & P. Lohmann (2011): *Abstraction for model checking modular interpreted systems over ATL*. In: Proceedings of AAMAS, pp. 1129–1130.

[15] F. Laroussinie, N. Markey & G. Oreiby (2008): *On the Expressiveness and Complexity of ATL*. Logical Methods in Computer Science 4, p. 7, doi:10.2168/LMCS-4(2:7)2008.

[16] O. Lichtenstein & A. Pnueli (1985): *Checking that finite state concurrent programs satisfy their linear specification*. In: POPL '85: Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM, New York, NY, USA, pp. 97–107, doi:10.1145/318593.318622.

[17] A. Lomuscio & F. Raimondi (2006): *MCMAS : A Model Checker for Multi-agent Systems*. In: Proceedings of TACAS, Lecture Notes in Computer Science 4314, pp. 450–454, doi:10.1007/11691372_31.

[18] A. Murano, M. Napoli & M. Parente (2008): *Program Complexity in Hierarchical Module Checking*. In: Proceedings of LPAR, pp. 318–332, doi:10.1007/978-3-540-89439-1_23.

[19] F. Raimondi (2006): *Model Checking Multi-Agent Systems*. Ph.D. thesis, University College London.

[20] S. La Torre, M. Napoli, M. Parente & G. Parlato (2008): *Verification of scope-dependent hierarchical state machines*. Information and Computation 206(9-10), pp. 1161–1177, doi:10.1016/j.ic.2008.03.017.

Model checking coalitional games in shortage resource scenarios[*]

Della Monica, Dario
ICE-TCS, School of Computer Science
Reykjavik University, Iceland
dariodm@ru.is

Napoli, Margherita
Dipartimento di Informatica
University of Salerno, Italy
napoli@dia.unisa.it

Parente, Mimmo
Dipartimento di Informatica
University of Salerno, Italy
parente@unisa.it

Verification of multi-agents systems (MAS) has been recently studied taking into account the need of expressing resource bounds. Several logics for specifying properties of MAS have been presented in quite a variety of scenarios with bounded resources. In this paper, we study a different formalism, called *Priced Resource-Bounded Alternating-time Temporal Logic* (PRB-ATL), whose main novelty consists in moving the notion of resources from a syntactic level (part of the formula) to a semantic one (part of the model). This allows us to track the evolution of the resource availability along the computations and provides us with a formalisms capable to model a number of real-world scenarios. Two relevant aspects are the notion of global availability of the resources on the market, that are shared by the agents, and the notion of price of resources, depending on their availability. In a previous work of ours, an initial step towards this new formalism was introduced, along with an EXPTIME algorithm for the model checking problem. In this paper we better analyze the features of the proposed formalism, also in comparison with previous approaches. The main technical contribution is the proof of the EXPTIME-hardness of the the model checking problem for PRB-ATL, based on a reduction from the acceptance problem for *Linearly-Bounded Alternating Turing Machines*. In particular, since the problem has multiple parameters, we show two *fixed-parameter* reductions.

1 Introduction

Verification of multi-agents systems (MAS) is a topic under investigation by several research groups in computer science in the last ten years ([8]). Most of the research is based on logical formalisms, maybe the most famous being the *Alternating-time Temporal Logics* (ATL) [3] and the *Coalition Logic* (CL) [15, 16], both oriented towards the description of collective behaviors and used as specification languages for open systems. These scenarios are hence naturally modeled as games. In [10] it has been shown that CL can be embedded into ATL. Recently, these two logics have been used for the verification of multi-agent systems (MAS), enhanced with resource constraints [1, 2, 5, 6, 9]. The intuitive idea is that agent actions consume and/or produce resources, thus the choice of a given action of an agent is subject to the availability of the resources. In [1], Alechina et al. introduce the logic *Resource-Bounded Coalition Logic* (RBCL), whose language extends the one of CL with explicit representation of resource bounds. In [2], the same authors propose an analogous extension for ATL, called *Resource-Bounded Alternating-time Temporal Logics* (RB-ATL), and give a model checking procedure that runs in time $O(|\varphi|^{2\cdot r+1} \times |G|)$, where $|\varphi|$ is the length of the formula φ to be checked, $|G|$ is the size of the model

[*]The work of Dario Della Monica has been partially supported by the project *Processes and Modal Logics* (project nr. 100048021) of the Icelandic Research Fund and the project *Decidability and Expressiveness for Interval Temporal Logics* (project nr. 130802-051) of the Icelandic Research Fund in partnership with the European Commission Framework 7 Programme (People) under "Marie Curie Actions". The work of Margherita Napoli has been partially supported by the Italian PRIN 2010 project *Logical Methods of Information Management*. The work of Margherita Napoli and Mimmo Parente has been partially supported by the Italian FARB projects 2010-2012.

G, and *r* is the number of resources. However, the problem of determining a lower bound to the model checking problem is left open. In [6], Bulling and Farwer introduce two *Resource-Bounded Agent Logics*, called RAL and RAL*. The former represents a generalization of Alechina et al.'s RB-ATL, the latter is an analogous extension of ATL* (analogous extensions for, respectively, CTL and CTL* were presented by the same authors in [5]). The authors study several syntactic and semantic variants of RAL and RAL* with respect to the (un)decidability of the model checking problem. In particular, while previous approaches only conceive actions *consuming* resources, they introduce the notion of actions *producing* resources. It turned out that such a new notion makes the model checking problem undecidable. Formulae of the formalisms proposed in [1, 2, 5, 6] allow one to assign an endowment of resources to the agents by means of the so-called *team operators* (borrowed from ATL). The problem is then to determine whether the agents in the *proponent* team have a strategy for the game to carry out the assigned goals with that bounded amount of resources, whatever the agents in the *opponent* team do.

In this paper we study a different formalism, called *Priced Resource-Bounded Alternating-time Temporal Logic* (PRB-ATL), introduced in [9], but in a much less mature version. The key features of this new approach toward the formalization of such complex systems can be summarized as follows.

- *Boundedness of the resources.* This is a crucial point in our formalization. In order to model boundedness of the resources, a notion of *global availability* of resources on the market (or in nature), which evolves depending on both proponent and opponent behaviors, is introduced. Such a global availability is a semantic component (it is part of the structure where the logic is interpreted) and its evolution is tracked during the executions of the system. Agents' moves are affected by the current global availability (e.g., agents cannot consume an unbounded amount of resources).

- *Resources are shared.* Resources are global, that is, they are shared by all the agents. Thus, the agents either consume or produce resources out of a shared pool of bounded capability, and acquisition (resp., release) of a resource by an agent (independently if the agent belongs to the proponent or opponent team) implies that the resource will be available in smaller (resp., greater) quantity. In this way, we can model several scenarios where shared resources are acquired at a cost that depends on that resource current availability (for example in concurrent systems where there is a competition on resources).

- *Money as a meta-resource.* In addition to public shared resources, our setting also allows one to model *private* resources, that is, resources that are possessed by agents (public resources are present in the market and will be acquired by the agents in case they need). The idea is to provide the agents with the unique private resource, *money*, that can be used to acquire (public) resources needed to perform the tasks. In this sense, money represent several resource combinations and can be considered as a meta-resource. Unlike the other resources, it is a syntactic component (money endowment is part of the formula), and is the only (meta-)resource which is private for an agent.
  At this stage, our formalization only features the possibility of assigning to agents one private resource. Nevertheless, in principle, it is possible to extend the idea to admit a vector of private resources. Furthermore, one could think of including the same resource in both the pool of public resources and in the pool of private ones. For instance, in a car race one of the players (the cars) possesses some gasoline in the tank (private resource) but he needs to acquire more gasoline at the gas station (public resource) to complete the race.

- *Resource production.* Production of resources is allowed in a quantity that is not greater than a fixed amount. Thus, we extend the model still preserving the decidability of the model checking problem. Observe that the constraint we impose still allows us to describe many interesting real-world scenarios, such as acquiring memory by a program, or leasing a car during a travel, or, in general, any release of resources previously acquired. A similar setting has been already observed also in [6].

- *Opponent power.* First observe that we use the standard terminology which separates the role of the agents in a proponent team and those in the opponent team. This distinction is not within the game structure, but it is due to the formula under consideration. Agents of the opponent team are subject to resource availability in choosing the action to perform, in the same way as the agent of the proponent team, thus the opponent team cannot interfere with a proponent strategy performing actions which either consume or produce too much (see Example 3 in Section 3). However, it is common practice to consider opponent having maximum power, to look for robust strategy. We give unlimited economic power to the agents in the opponent team, in the sense that at each moment they have money enough to acquire the resources they need for a move, provided that the resources are available.

Actually in [9] an EXPTIME algorithm for the model checking problem was given, along with a PSPACE lower bound. The main technical contribution here is to provide an EXPTIME lower bound for the model checking problem for PRB-ATL. This result shows that the model checking problem for this logic is EXPTIME-complete. The hardness proof is obtained by means of a reduction from the acceptance problem for *Linearly-Bounded Alternating Turing Machines* (LB-ATM), known to be EXPTIME-complete [7], to the model checking problem for PRB-ATL. More precisely, let n be the number of agents, r the number of resources, and M the maximum component occurring in the initial resource availability vector, the algorithm given in [9] runs in exponential time in n, r, and the size of the representation of M (assuming that M is represented in binary). To prove here the inherent difficulty with respect to multiple input parameters, we show two reductions: one parametric in the representation of M (the digit size), that assumes constant both n and r, and another parametric in r, and assuming constant both n and the value of M.

2 Comparison with related works

In this section we compare our approach with the existing literature underlining differences and similarities respect to [2] and [6].

In the work by Alechina et al. [2], resource bounds only appear in the formulae and are applied solely to the proponent team, but they are not represented inside the model. Indeed, agents of the proponent team are endowed with new resources at the different steps of the system execution. This means that it is possible to ask whether a team can reach a goal with a given amount of resources, but it is not possible to keep trace of the evolution of the global availability of resources. Moreover, resources are private to agents of the proponent team (not shared, as in our approach) and resource consumption due to the actions of the opponent is not controlled. Here instead, we keep trace of resource global availability, whose evolution depends on both proponent and opponent moves. In this way, it is possible to avoid undesired/unrealistic computations of the system such as, for instance, computations consuming unboundedly. Let us see a very simple example. Consider the formula $\psi = \langle\langle A^{\vec{\$}} \rangle\rangle \Box p$. Its semantics is that agents in team A have together a strategy which can guarantee that p always holds, whatever agents of the opponent team do (without consuming too many resources) and provided the expense of the agents in A does not exceed $\vec{\$}$. A loop in the structure where the joint actions of agents consume resources without producing them, cannot be a model for ψ. On the contrary, consider the formula $\psi' = \langle\langle A^{b} \rangle\rangle \Box p$, belonging to the formalism proposed in [2], expressing a similar property, with the only difference that the agents of A use an amount of resources bounded by b. A model for ψ' must contain a loop where the actions of agents in A do not consume resources, but the actions of agents in the opponent team may possibly consume resources, leading to an unlimited consumption of resources.

As a further difference, recall that in [2] actions can only consume resources. Without resource productions, the model for many formulae (for example those containing the *global* operator $\Box$) must have a loop whose actions do not consume resources (*do-nothing* actions), and a run satisfying these formulae is eventually formed by only such actions. On the contrary, by allowing resource production, we can model more complex situations when dealing with infinite games.

Finally, observe that a similarity with the cited paper is in the role of money, that could be seen as a private resource, endowed to the agents of the proponent team.

Bulling and Farwer [6] adopted an "horizontal" approach, in the sense that they explored a large number of variants of a formalism to model these complex systems. In particular, they explored the border between decidability and undecidability of the model checking problem for all such variants, and they showed how the status of a formalisms (wrt decidability of its model checking problem) is affected by (even small) changes in language, model, and semantics. Our work takes advantage of this analysis in order to propose a logic that captures several desirable properties (especially concerning the variety of natural real world scenario that is possible to express), still preserving decidability. However, our approach presents conceptual novelties that make it difficult to accomplish a direct comparisons between the formalisms presented here and the ones proposed in [6]. We are referring here to both the above mentioned idea of dealing with resources as global entities for which agents compete, and the notion of cost of resource acquisition (price of the resources) that dynamically changes depending on the global availability of that resource (thus allowing one to model the classic market law that says that getting a resource is more expensive in shortage scenario). In [6], there is no such a notion as resources are assigned to (team of) agents and proponent and opponent do not compete for their acquisition.

As regards the complexity issue, in [6], no complexity analysis (for the model checking problem) is performed, while, in [2], an upper bound is given for RB-ATL, that matches the one given in [9] for PRB-ATL. The algorithm for PRB-ATL runs in exponential time in the number n of agents, the number r of resources, and the digit size of the maximum component M occurring in the initial resource availability vector (assuming a binary reppresentation). Analogously, the model checking algorithm for RB-ATL runs in exponential time in r, in the digit size of the maximum component of resource endowment vectors b occuring in team operators $\langle\langle A^b \rangle\rangle$ of φ and in the number n of the agents (this is implicit in set of states of $|G|$). Actually, both n and r are often treated as constant [2, 3] (without this assumption, the complexity of ATL model-checking is shown to be exponential in the number of agents [11]). However, no complexity lower bound has been exhibit so far. Aim of this paper is to fill this gap, by providing an EXPTIME lower bound for PRB-ATL.

3 A logical formalization: PRB-ATL

Syntax. We start with the introduction of some notations we will use in the rest of the paper. The set of *agents* is $\mathscr{AG} = \{a_1, a_2, \dots, a_n\}$ and a *team* is any subset of $\mathscr{AG}$. The integers n and r will be used throughout the paper to denote the number of agents and *resource types* (or simply *resources*), respectively. Let $\mathscr{M} = (\mathbb{N} \cup \{\infty\})^r$ denote the set of *global availabilities of resources on the market (or in nature)* and let $\mathscr{N} = (\mathbb{N} \cup \{\infty\})^n$ denote the set of *money availabilities for the agents,* where $\mathbb{N}$ is the set of natural numbers (zero included). Given a money availability $\vec{\$} \in \mathscr{N}$, its i-th component $\vec{\$}[i]$ is the money availability of agent a_i[1]. Finally, the set Π is a finite set of *atomic propositions*.

The formulae of PRB-ATL are given by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle A^{\vec{\$}}\rangle\rangle \bigcirc \varphi \mid \langle\langle A^{\vec{\$}}\rangle\rangle \varphi \mathscr{U} \varphi \mid \langle\langle A^{\vec{\$}}\rangle\rangle \Box \varphi \mid \sim \vec{m}$$

where $p \in \Pi$, $A \subseteq \mathscr{AG}$, $\sim \in \{<, \leq, =, \geq, >\}$, $\vec{m} \in \mathscr{M}$ and $\vec{\$} \in \mathscr{N}$. Formulae of the kind $\sim \vec{m}$ test the current availability of resources on the market. As usual, other standard operators can be considered as abbreviation, e.g., the operator $\langle\langle A^{\vec{\$}}\rangle\rangle \Diamond \psi$ can be defined as $\langle\langle A^{\vec{\$}}\rangle\rangle \top \mathscr{U} \psi$, for every formula ψ.

Priced game structure. *Priced game structures* are defined by extending the definitions of concurrent game structure and resource-bounded concurrent game structure given in, respectively, [3] and [2].

Definition 1 *A* priced game structure *G is a tuple $\langle Q, \pi, d, D, qty, \delta, \rho, \vec{m}_0 \rangle$, where:*

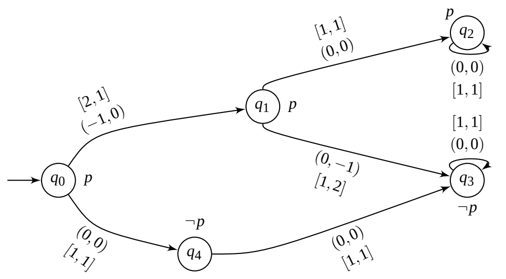
- *Q is the finite set of* locations*; $q_0 \in Q$ is called* initial location*.*
- *$\pi : Q \to 2^{\Pi}$ is the* evaluation function*, which determines the atomic propositions holding true in each location.*
- *$d : Q \times \mathscr{AG} \to \mathbb{N}$ is the* action function *giving the number $d(q,a) \geq 1$ of* actions *available to an agent $a \in \mathscr{AG}$ at a location $q \in Q$. The actions available to a at q are identified with the numbers*[2] *$1, \ldots, d(q,a)$ and a generic action is usually denoted by α. We assume that each agent has at least one available action at each location, that could be thought of as the action* do-nothing *and we assume that it is always the first.*
- *$D : Q \to 2^{\mathbb{N}^n}$ is a function that maps each location q to the set of vectors $\{1, \ldots, d(q,a_1)\} \times \ldots \times \{1, \ldots, d(q,a_n)\}$. Each vector, called* action profile *and denoted by $\vec{\alpha}$, identifies a choice among the actions available for each agent in the location q. (The action of the agent a in $\vec{\alpha}$ is $\vec{\alpha}(a)$.)*
- *$qty : Q \times \mathscr{AG} \times \mathbb{N} \to \mathbb{Z}^r$ is a partial function, where $qty(q,a,\alpha)$, with $1 \leq \alpha \leq d(q,a)$, defines at location q the amount of resources required by the a's action α. We define $qty(q,a,1) = \vec{0}$, that is the vector whose components are all equal to 0, for every $q \in Q$, $a \in \mathscr{AG}$ (doing nothing neither consumes nor produces resources).*
- *$\delta : Q \times \mathbb{N}^n \to Q$ is the* transition function*. For $q \in Q$ and $\vec{\alpha} \in D(q)$, $\delta(q, \vec{\alpha})$ defines the next location reached from q if the agents perform the actions in the action profile $\vec{\alpha}$.*
- *$\rho : \mathscr{M} \times Q \times \mathscr{AG} \to \mathbb{N}^r$ is the* price function*. It returns the* price vector *of the resources (a price for each resource), based on the current resource availability and location, and on the acting agent.*
- *$\vec{m}_0 \in \mathscr{M}$ is the initial global availability of resources. It represents the resource availability on the market at the initial state of the system.*

Note that a negative value in $qty(q,a,\vec{\alpha})$ represents a resource consumption, while a positive one represents a resource production. We also consider the extension of the function *qty*, called again with the same name, to get the amount of resources required by a given team. Thus, for a location q, a team A and an action profile $\vec{\alpha}$, $qty(q,A,\vec{\alpha}) = \sum_{a \in A} qty(q,a,\vec{\alpha}(a))$. Moreover, we will use the function $consd : Q \times \mathscr{AG} \times \mathbb{N} \to \mathbb{N}^r$ that for the tuple (q,a,α) returns the vector of the resources which are consumed by an agent a, being in state q, for an action α. This vector is obtained from $qty(q,a,\alpha)$ by replacing the positive components, representing a resource production, with zeros, and the negative components, representing a resource consumption, with their absolute values.

Example 1 *A priced game structure with two agents a_1 and a_2 and one resource R_1 is depicted in Figure 1. The only atomic proposition is p, labeling the locations q_0, q_1, q_2. The action profiles, labeling*

[1]Throughout all the paper, symbols identifying vectors are denoted with an arrow on the top (e.g., $\vec{\$}$, $\vec{m}$).

[2]No ambiguity will arise from the fact that actions of different agents are identified with the same numbers.



$\mathscr{AG} = \{a_1,a_2\}$, $R = \{R_1\}$, $Q = \{q_0,q_1,q_2,q_3,q_4\}$, $\vec{m}_0 = \langle 1\rangle$
$\pi(q_0) = \pi(q_1) = \pi(q_2) = \{p\}$, $\pi(q_3) = \pi(q_4) = \{\}$
$d(q_0,a_1) = 2$, $d(q_0,a_2) = 1$, $d(q_1,a_1) = 1$, $d(q_1,a_2) = 2$
$d(q_2,a_1) = d(q_2,a_2) = d(q_3,a_1) = d(q_3,a_2) = 1$
$d(q_4,a_1) = d(q_4,a_2) = 1$
$D(q_0) = \{[1,1],[2,1]\}$, $D(q_1) = \{[1,1],[1,2]\}$
$D(q_2) = D(q_3) = D(q_4) = \{[1,1]\}$
$qty(q_0,a_1,1) = \langle 0\rangle$, $qty(q_0,a_1,2) = \langle -1\rangle$, $qty(q_0,a_2,1) = \langle 0\rangle$
$qty(q_1,a_1,1) = \langle 0\rangle$, $qty(q_1,a_2,1) = \langle 0\rangle$, $qty(q_1,a_2,2) = \langle -1\rangle$
$qty(q,a,1) = \langle 0\rangle$, $\forall q \in \{q_2,q_3,q_4\}, a \in \mathscr{AG}$

Figure 1: Example of priced game structure $G = \langle Q,\pi,d,D,qty,\delta,\rho,\vec{m}_0\rangle$.

the transitions in the graph and depicted with square brackets, are as follows. $D(q_0) = \{[1,1],[2,1]\}$ *is due to the existence of two actions of* a_1 *and one action of* a_2 *at location* q_0, $D(q_1) = \{[1,1],[1,2]\}$ *corresponds to a single action of* a_1 *and two actions of* a_2 *at location* q_1. *In all the other locations the only action profile is* $[1,1]$ *corresponding to the existence of a single action of both the agents. The function qty is represented by parentheses. The price vector is not depicted.*

Semantics. In the following, given a resource availability $\vec{m}$, by $\mathscr{M}^{\leq\vec{m}}$ we denote the set $\{\vec{m}' \in \mathscr{M} \mid \vec{m}' \leq \vec{m}\}$. In order to give the formal semantics let us first define the following notions.

Definition 2 *A* configuration *c of a priced game graph G is a pair* $\langle q,\vec{m}\rangle \in Q \times \mathscr{M}^{\leq\vec{m}_0}$. *Given two configurations* $c = \langle q,\vec{m}\rangle$ *and* $c' = \langle q',\vec{m}'\rangle$, *and an action profile* $\vec{\alpha} \in D(q)$, *we say that* $c \rightarrow_{\vec{\alpha}} c'$ *if* $q' = \delta(q,\vec{\alpha})$ *and* $\vec{m}' = \vec{m} + qty(q,\mathscr{AG},\vec{\alpha})$. *A* computation *over G is an infinite sequence $C = c_1c_2\dots$ of configurations of G, such that for each i there is an action profile $\vec{\alpha}_i$ such that* $c_i \rightarrow_{\vec{\alpha}_i} c_{i+1}$.

Let $C = c_1c_2\dots$ be a computation. We denote by $C[i]$ the i-th configuration c_i in C and by $C[i,j]$, with $1 \leq i \leq j$, the finite sequence of configurations $c_ic_{i+1}\dots c_j$ in C. Given a configuration $c = \langle q,\vec{m}\rangle$ and a team A, a function $\vec{\alpha}_A : A \rightarrow \mathbb{N}$ is called *A-feasible in c* if there exists an action profile $\vec{\alpha} \in D(q)$ with $\vec{\alpha}_A(a) = \vec{\alpha}(a)$ for all $a \in A$ and $\vec{0} \leq qty(q,A,\vec{\alpha}) + \vec{m} \leq \vec{m}_0$. In this case we say that $\vec{\alpha}$ *extends* $\vec{\alpha}_A$.

Definition 3 *A* strategy *F_A of a team A is a function which associates to each finite sequence of configurations $c_1c_2\dots c_s$, a function $\vec{\alpha}_A : A \rightarrow \mathbb{N}$ which is A-feasible in c_s.*

In other words, a strategy F_A returns a choice of the actions of the agents in the team A, considering only those actions whose resource consumption does not exceed the available amount and whose resource production does not exceed the amount consumed so far. Clearly, this constraint will limit both proponent and opponent team.

For each strategy F_A of a team A and for each sequence of configurations $c_1c_2\dots c_s$, there are several possibilities for the next configuration c_{s+1}, depending on the different choices of the opponent team $\overline{A} = \mathscr{AG} \setminus A$. Anyway, fixed a strategy $F_{\overline{A}}$ of the opponent team, there is at most one action profile obtained according to both the strategies, that is the action profile $\vec{\alpha}$ extending both $\vec{\alpha}_A$, given by the strategy F_A, and $\vec{\alpha}_{\overline{A}}$, given by the strategy $F_{\overline{A}}$ (i.e. $\vec{\alpha}$ is such that $\vec{\alpha}(a) = \vec{\alpha}_X(a)$, for $X \in \{A,\overline{A}\}$ and $a \in X$). A computation $C = c_1,c_2\dots$, is the *outcome of the strategies F_A and $F_{\overline{A}}$ from the configuration c_1* if, for each $i \geq 1$, there is an action profile $\vec{\alpha}_i$ obtained according to both F_A and $F_{\overline{A}}$, such that $c_i \rightarrow_{\vec{\alpha}_i} c_{i+1}$. Given a strategy F_A and a configuration c, $out(c,F_A)$ denotes the set of the outcomes of F_A and $F_{\overline{A}}$ from c, for all the strategies $F_{\overline{A}}$ of the team $\overline{A}$. Observe that, given a finite sequence of configurations $C = c_1c_2\dots c_s$, if the action profile $\vec{\alpha}$ according to the two strategies is not such that $\vec{0} \leq qty(q_s,\mathscr{AG},\vec{\alpha}) + \vec{m}_s \leq \vec{m}_0$, then there is no next configuration. Thus outcome of the strategies F_A and $F_{\overline{A}}$ from a given configuration may be undefined (recall that we consider only infinite computations).

Example 2 *Consider the priced game structure in Figure 1, with teams $A = \{a_1\}$ and $B = \{a_2\}$, one resource type and initial global availability $\vec{m}_0 = \langle 1 \rangle$. Let $c = \langle q_0, \langle 1 \rangle \rangle$ be a sequence of configurations (of length* 1*). Team A has two possible strategies in c, one for each possible action of agent a_1, and team B has one strategy for the single available action of agent a_2. Suppose that, according to the strategy F_A, agent a_1 chooses to perform the action* 2 *($F_A(c)(a_1) = 2$), then the action profile $[2,1]$ is performed and one unit of the unique resource is consumed. In the obtained configuration $\langle q_1, \langle 0 \rangle \rangle$ the agent a_1 has one available action while the agent a_2 has two actions. Anyway F_B cannot return the action* 2 *for the agent a_2, since this action would require an amount of the resource greater than* 0*, which is the current availability. Thus only the configuration $\langle q_2, \langle 0 \rangle \rangle$ can be reached and the computation $C = \langle q_0, \langle 1 \rangle \rangle \langle q_1, \langle 0 \rangle \rangle \langle q_2, \langle 0 \rangle \rangle \langle q_2, \langle 0 \rangle \rangle \ldots$ is the only one that belongs to $out(c, F_A)$.*

Now we introduce the concept of consistent strategy. Two properties have to be satisfied: first, the outcomes starting from c are always defined and also the agents of the proponent team have enough money to realize the chosen actions.

Definition 4 *Let $\vec{\$} \in \mathcal{N}$, c be a configuration, $A \subseteq \mathcal{AG}$ be the proponent team, and $\overline{A} = \mathcal{AG} \setminus A$ be the opponent team. A strategy F_A of A is said to be* consistent with respect to $\vec{\$}$ and c *(*$(\vec{\$}, c)$-strategy*), if*

1. *for any strategy $F_{\overline{A}}$ of $\overline{A}$, the outcome of F_A and $F_{\overline{A}}$ from the configuration c is defined,*
2. *for every $C = c_1 c_2 \ldots \in out(c, F_A)$, with $c_i = \langle q_i, \vec{m}_i \rangle$, for every $i \geq 1$ and $a_k \in A$: $\sum_{j=1}^{i} \rho(\vec{m}_j, q_j, a_k) \cdot consd(q_j, a_k, F_A(C[1,j])(a_k)) \leq \vec{\$}[k]$.*

In the above condition the dot operator denotes the usual scalar product of vectors. Observe that only the money availability of the team A is tested. Actually, we suppose that the opponent team $\overline{A}$ always has money enough to make its choice. Notice also that the actions *producing* resources do not cause a reimbursement of money to the agents. As it is usual when dealing with temporal logics, we guarantee that priced game structures are non-blocking, in the sense that at least a $(\vec{\$}, c)$-strategy exists for a given team A. Indeed, agents of A can always jointly choose the *do-nothing* action.

A formula of PRB-ATL is evaluated with respect to a priced game structure G and a configuration $c = \langle q, \vec{m} \rangle$. The definition of the semantics is completed by the definition of the satisfaction relation $\models$:

- $(G, c) \models p$ iff $p \in \pi(q)$
- $(G, c) \models \neg\psi$ iff $(G, c) \not\models \psi$
- $(G, c) \models \psi_1 \wedge \psi_2$ iff $(G, c) \models \psi_1$ and $(G, c) \models \psi_2$
- $(G, c) \models \langle\langle A^{\vec{\$}} \rangle\rangle \bigcirc \psi$ iff there exists a $(\vec{\$}, c)$-strategy F_A such that, for all $C \in out(c, F_A)$, it holds that $(G, C[2]) \models \psi$
- $(G, c) \models \langle\langle A^{\vec{\$}} \rangle\rangle \psi_1 \mathcal{U} \psi_2$ iff there exists a $(\vec{\$}, c)$-strategy F_A such that, for all $C \in out(c, F_A)$, there exists $i \geq 0$ such that $(G, C[i]) \models \psi_2$ and, for all $1 \leq j < i$, it holds that $(G, C[j]) \models \psi_1$
- $(G, c) \models \langle\langle A^{\vec{\$}} \rangle\rangle \Box \psi$ iff there exists a $(\vec{\$}, c)$-strategy F_A such that, for all $C \in out(c, F_A)$, it holds that $(G, C[i]) \models \psi$ for all $i \geq 1$
- $(G, c) \models \sim \vec{m}'$ iff $\vec{m} \sim \vec{m}'$ where $\sim \in \{<, \leq, =, \geq, >\}$.

Given a PRB-ATL formula and a priced game srtucture G, we say that G satisfies φ, $G \models \varphi$, if $(G, c_0) \models \varphi$ where $c_0 = \langle q_0, \vec{m}_0 \rangle$. The *model checking* problem for PRB-ATL consists in verifying whether $G \models \varphi$.

Example 3 *Consider the priced game structure in Figure 1, with teams $A = \{a_1\}$ and $B = \{a_2\}$. A formula $\psi = \langle\langle \mathcal{AG}^{\vec{\$}} \rangle\rangle \bigcirc \langle\langle A^{\vec{\$}'} \rangle\rangle \Box p$ holds true in the configuration $\langle q_0, \langle 1 \rangle \rangle$, provided that $\vec{\$}$ and $\vec{\$}'$ are enough to make the move. Indeed, a_1 and a_2 together are able to force the computation to reach the*

$\langle q_1, \langle 0 \rangle\rangle$ (one unit of resource is consumed). From such a configuration, the opponent team B cannot force the computation into q_3, as the action 2 is not allowed for a_2 (no resources are available to perform the action), and thus ψ holds. Instead, ψ is false in the configuration $\langle q_0, \langle 2 \rangle\rangle$ (actually in each configuration $\langle q_0, \langle x \rangle\rangle$, with $x > 1$), because $\langle q_1, \langle 1 \rangle\rangle$ is reached after the execution of the first transition, and in that configuration action 2 for a_2 in B is allowed, leading to q_3. Finally, notice that the formula is false also when evaluated in $\langle q_0, \langle 0 \rangle\rangle$, as the only possible transition is the one leading from q_0 to q_4 (no resources are available to perform action 1 for agent a_1).

4 Complexity lower bounds for the model checking problem

In [9], the authors presented an algorithm for model checking PRB-ATL, providing an exponential upper bound for the problem. In particular, let n be the number of agents, r the number of resources, and M the maximum component occurring in the initial resource availability vector, the proposed algorithm runs in exponential time in n, r, and the size of the representation of M (assuming that M is represented in binary). In this section we prove that an algorithm that behaves asymptotically better cannot exist, thus proving that the problem is EXPTIME-complete. To prove the inherent difficulty with respect to the multiple input parameters, we show two reductions: one parametric in the representation of M (the digit size), which assumes both n and r constant, and the other parametric in r, this time assuming constant both n and the value of M. We conjecture the existence of a third EXPTIME reduction, in which r and M are constant and the parameter is n. In fact, if it was not the case, it would be possible to improve the proposed model checking algorithm in a way that its complexity would not be exponential in n.

We first recall the formalism of *linearly-bounded alternating Turing machines* (LB-ATM) and the notion of *hierarchical representation*, a succinct way of representing priced game structures inspired to the work done in [4] for classical Kripke structures. Finally, we present the two reductions from the acceptance problem for LB-ATM, known to be EXPTIME-complete [7], to the model checking problem for PRB-ATL.

4.1 Linearly-bounded alternating Turing Machines

A *linearly-bounded alternating Turing machines* (LB-ATM) is a tuple $\langle \mathscr{Q}, \Gamma, \mathscr{I}, \mathsf{q}_0, \rangle$, where $\mathscr{Q}$ is the set of *states*, partitioned in $\mathscr{Q}_\forall$ (*universal states*) and $\mathscr{Q}_\exists$ (*existential states*); Γ is the set of *tape symbols*, including the 'blank' symbol B, and two special symbols $\llcorner$ and $\lrcorner$, denoting the left and right *tape delimiters*; $\mathscr{I} \subseteq \mathscr{Q} \times \Gamma \times \mathscr{Q} \times \Gamma \times \{\leftarrow, \rightarrow\}$ is the *instruction set*; $\mathsf{q}_0 \in \mathscr{Q}$ is the *initial state*.

Symbols from Γ are stored in the *tape cells*, and the first and the last cell of the tape store, respectively, the symbols $\llcorner$ and $\lrcorner$. A *tape configuration* s is a sequence of the symbols stored in the tape cells, and keeps trace of an *head cell*. A *configuration c* is a pair (q, s) of a state q and a tape configuration s, and $\mathscr{C}$ is the set of the configurations. The initial configuration is $c = (\mathsf{q}_0, \mathsf{s}_0)$, where s_0 contains the input, possibly followed by a sequence of blanks, and its head cell stores the first input symbol.

An *instruction* $i = (\mathsf{q}, \lambda, \mathsf{r}, \nu, \sim) \in \mathscr{I}$ is also denoted $\langle \mathsf{q}, \lambda \rangle \rightarrow \langle \mathsf{r}, \nu, \sim \rangle$, where $\langle \mathsf{q}, \lambda \rangle$ is called a *full state*. Its intuitive meaning is as follows: "whenever the machine is in the state q and the symbol in the head cell is λ, then the machine switches to state r, the symbol in the head cell is replaced with ν, and the head position is moved to the left or to the right (according to $\sim$)". An *execution step* of the machine is denoted $c \xrightarrow{i} c'$, where $c, c' \in \mathscr{C}$, $i \in \mathscr{I}$ and c' is the configuration reached from c after the execution of the instruction i. Let $\mathscr{C}_{next(c)} = \{c' \in \mathscr{C} \mid c \xrightarrow{i} c'$ is an execution step, for some $i \in \mathscr{I}\}$. All the tape configurations are linear in the length of the input and we follow the common practice to only consider machines whose tape length does not vary during the computation. We can also assume that LB-ATM have no infinite computations since any LB-ATM can be transformed into another, accepting the same language and haltingin a finite number of steps. Such a LB-ATM counts the number of execution steps and rejects any computation whose number of steps exceeds the number of possible configurations.

The *acceptance condition* is defined recursively. A configuration $c = (\mathsf{q}, \mathsf{s})$ is said to be *accepting* if either one of the following conditions is verified: (i) $q \in \mathscr{Q}_\forall$ and c' is accepting for all $c' \in \mathscr{C}_{next(c)}$ or (ii) $q \in \mathscr{Q}_\exists$ and there exists $c' \in \mathscr{C}_{next(c)}$ such that c' is accepting. Notice that an universal (existential) state always accepts (rejects) if $\mathscr{C}_{next(c)} = \emptyset$. A LB-ATM *accepts on an initial input tape* s_0, if the initial configuration $(\mathsf{q}_0, \mathsf{s}_0)$ is accepting.

Hierarchical representation. In order to exhibit our encoding proposal, we make use of a hierarchical representation analogous to the one described in [4, 12, 13] for model checking, and in [14] for module checking procedures. Given a finite state machine, the idea of hierarchical representation is to replace two or more substructures of the machine that are structurally equivalent, by another (structurally equivalent) module, that is a finite state machine itself. The use of hierarchical representation results in an exponentially more succinct representation of the system, that amounts (in most cases) to more efficient model checking procedures (in the other cases, this does not yield a more efficient behavior, as the analysis requires a flattening of the machine itself, thus incurring in an exponential blow up in its size).

In our context, this idea can be suitably adapted to deal with the presence of resources, as follows. Modules do not represent structurally equivalent substructures, but substructures that have the same impact on the values of resource variables. In principle, whenever the analysis is focused on the evolution of resource variables, it makes sense to consider as equivalent two substructures that can possibly differ in their structure but whose effect on the set of resource variables is exactly the same. This approach could be thought of as a hierarchical representation based on *functional* equivalence between substructures, as opposed to the classical notion of hierarchical representation based on *structural* equivalence.

4.2 A reduction from the acceptance problem for LB-ATM

Given an LB-ATM $\mathscr{A}$ and an input tape configuration s_0, we provide a priced game structure $G_{\mathscr{A},\mathsf{s}_0}$, with two agents ag_1 and ag_2, and a formula $\phi_{\mathscr{A},\mathsf{s}_0}$ such that $G_{\mathscr{A},\mathsf{s}_0} \models \phi_{\mathscr{A},\mathsf{s}_0}$ if and only if $\mathscr{A}$ accepts on s_0.

In the following, we exhibit the game structure by using a graphical (hierarchical) representation (Figures 2-7 in Appendix). Notice that only significant information is explicitly shown in the pictures. In particular, labels on transitions (arcs) represent consumptions/productions of resources due to the execution of the joint move (proponent and opponent moves) associated to that transition. For example, the label "$-1i, +1\bar{i}, +10\mu_L, -10\overline{\mu_L}$" on the loop transition of Figure 4b means that the actions associated to the transition will consume 1 unit of the (type) resource i and 10 unit of $\overline{\mu_L}$, and will produce 1 unit of the resource $\bar{i}$ and 10 unit of μ_L. Availability of other resources is unchanged, then the relative information is omitted.

The reduction uses the three resource variables μ_L, μ, and μ_R to encode the tape configuration, plus three auxiliary resource variables i, r, and t, that will be useful during the construction. Moreover, we associate to the above set of variables the set of *counterbalanced variables* $\{\overline{\mu_L}, \overline{\mu}, \overline{\mu_R}, \bar{i}, \bar{r}, \bar{t}\}$. The idea behind the use of counterbalanced variables, that is also the key idea of the reduction, consists of designing the game structure in a way that to every consumption (resp., production) of a resource, say for instance μ, a corresponding production (resp., consumption) of its counterbalanced $\overline{\mu}$ exists. In particular, this is true inside each module of the hierarchical structure, thus the sum of the availability of a resource variable and its counterbalanced variable is kept constant along all the computation at every module's entry and exit points, equal to a value *Max*, which depends on the input of the LB-ATM. This will allow us to force the execution of specific transitions at specific availabilities of resource variables. Consider, for example, the node of Figure 4b with 2 outgoing transitions, one of which is a loop transition. The presence of 2 outgoing transitions means that either the proponent or the opponent can choose between 2 moves. But such a freedom is only potential, as in any moment of the computation the choice of the next move by the proponent/opponent is constrained by the resource availability: if the loop transition is enabled, then the availability of the resource i is greater than 0, and thus the availability of its counterbalanced variable $\bar{i}$ is less than Max, that means that the other transition, which consumes Max units of the resource $\bar{i}$, is disabled. On the contrary, if the non-loop transition is enabled, there are Max units of the resource $\bar{i}$ available, and thus the availability of the resource i is 0, that means that the loop transition is disabled. Thus, by taking advantage of the features of counterbalanced variables, we are able to force the executions to have a somehow deterministic behavior.

Encoding of the tape. Without loss of generality, we consider $\mathsf{LB\text{-}ATM}$ on input alphabet $\Sigma = \{1,2,\mathsf{B}\}$, thus Γ is the set $\{1,2,\mathsf{B},\llcorner,\lrcorner\}$. Recall that the symbols B, $\llcorner$, and $\lrcorner$ denote the 'blank' symbol, the left delimiter, and the right delimiter, respectively. Tape symbols are encoded by the digits $0,1,2,3$ and 4, in a pretty natural way: 0 encodes the 'blank' symbol, 1 and 2 encode the input symbols 1 and 2, and 3 and 4 encode the left and right delimiters. The tape configuration is encoded by means of the three resource variables μ_L, μ, and μ_R. The value of μ ranges over the set $\{0,1,2,3,4\}$ and encodes the value stored in the cell currently read by the head (according to the above encoding of tape symbols into digits). The value of μ_L encodes the tape configuration at the left of the current head position in a forward fashion. The value of μ_R encodes the tape configuration at the right of the current head position in a reverse fashion, that is, μ_R encodes the reverse of the string corresponding to the tape configuration at the right of current head position. As an example, consider the tape configuration $\mathsf{s} = \llcorner\mathsf{B112\underline{1}1B2BB}\lrcorner$, the symbol read by the head is the underlined one. Such a configuration is encoded by means of the three resource variables as follows: $\mu_L = 30112$, $\mu_L = 1$, and $\mu_R = 400201$. It can be noticed that the length of the representation of the three variables μ_L, μ, and μ_R is proportional to the length of the tape configuration which is at most linear in the size of the input, namely $O(|\mathsf{s}_0|)$. Using such an encoding, the machine operation "shift the head to the left" can be represented by means of the following operations on resource variables:

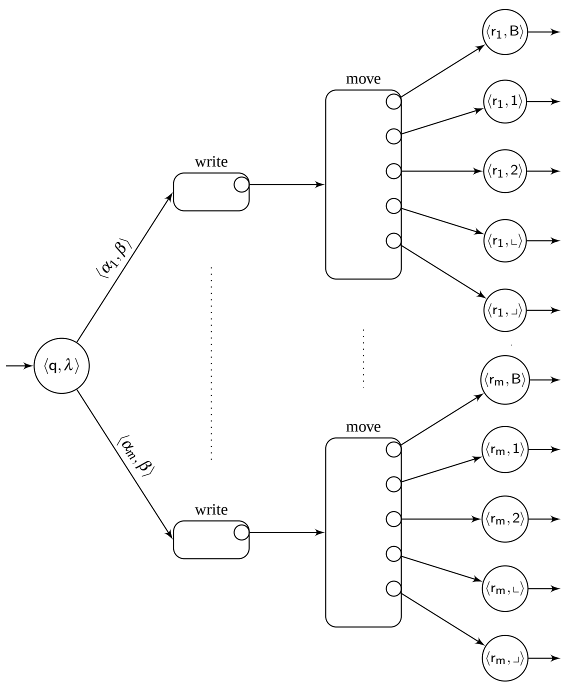
- the new value of μ_R is $\mu_R * 10 + \mu$
- the new value of μ is $\mu_L \mod 10$,
- the new value of μ_L is $\mu_L/10$ (/ is the integer division),

The operation "shift the head to the right" can be encoded analogously.

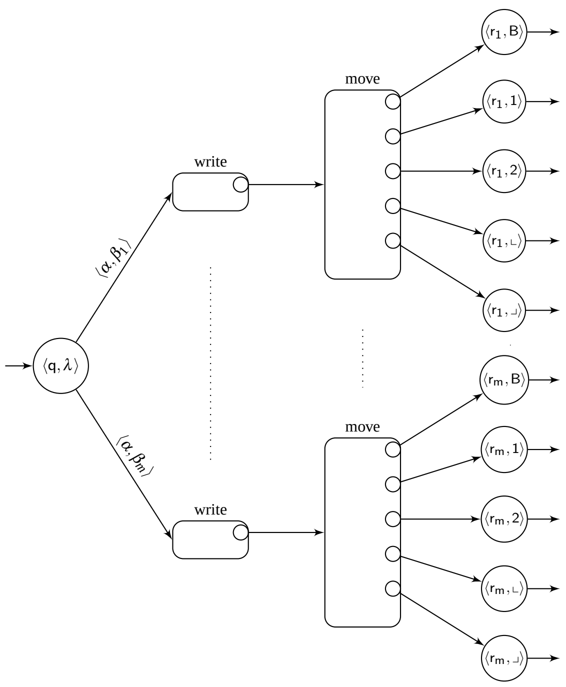
Notice that in order to encode in polynomial time the operations of shifting the head to left and right, we encode the string to the right of the current head position in a reverse order. Indeed, in this way the symbol stored on the cell immediately to the right of the head corresponds to the least significant digit of μ_R, and thus can be accessed by using the module operation ($\mu_R \mod 10$).

Encoding of the instructions. The encoding of the instructions is depicted in Figure 2. Transitions starting from a node labeled $\langle \mathsf{q},\lambda\rangle$ represent all the possible instructions matching the full state $\langle \mathsf{q},\lambda\rangle$ of the $\mathsf{LB\text{-}ATM}$, that is, all the instructions that can be possibly performed at the full state $\langle \mathsf{q},\lambda\rangle$.

More in detail, given a full state $\langle \mathsf{q},\lambda\rangle$ of the machine, with $\mathsf{q} \in \mathscr{Q}_\exists$, the encoding of the set $\{\langle \mathsf{q},\lambda\rangle \to \langle \mathsf{r}_1, \nu_1, \sim_1\rangle, \langle \mathsf{q},\lambda\rangle \to \langle \mathsf{r}_2, \nu_2, \sim_2\rangle, \ldots, \langle \mathsf{q},\lambda\rangle \to \langle \mathsf{r}_m, \nu_m, \sim_m\rangle\}$ of matching instructions is shown in Figure 2a, (recall that $\sim_i \in \{\leftarrow, \rightarrow\}$). Analogously, the encoding of the set of instructions matching the full state $\langle \mathsf{q},\lambda\rangle$, with $\mathsf{q} \in \mathscr{Q}_\forall$, is shown in Figure 2b. Let us underline that the action profiles $\langle \alpha_1,\beta\rangle, \ldots, \langle \alpha_m,\beta\rangle$ labeling transitions corresponding to an existential state are such that the first agent ag_1 has the capability to force a specific transition (instruction) to be executed, depending on the


$\langle \mathsf{q}, \lambda \rangle$
$\langle \alpha_1, \beta \rangle$
$\langle \alpha_m, \beta \rangle$
write
write
move
move
$\langle \mathsf{r}_1, \mathsf{B} \rangle$
$\langle \mathsf{r}_1, 1 \rangle$
$\langle \mathsf{r}_1, 2 \rangle$
$\langle \mathsf{r}_1, \llcorner \rangle$
$\langle \mathsf{r}_1, \lrcorner \rangle$
$\langle \mathsf{r}_m, \mathsf{B} \rangle$
$\langle \mathsf{r}_m, 1 \rangle$
$\langle \mathsf{r}_m, 2 \rangle$
$\langle \mathsf{r}_m, \llcorner \rangle$
$\langle \mathsf{r}_m, \lrcorner \rangle$

(a) Full state $\langle \mathsf{q}, \lambda \rangle$, with $\mathsf{q} \in \mathscr{Q}_\exists$.


$\langle \mathsf{q}, \lambda \rangle$
$\langle \alpha, \beta_1 \rangle$
$\langle \alpha, \beta_m \rangle$
write
write
move
move
$\langle \mathsf{r}_1, \mathsf{B} \rangle$
$\langle \mathsf{r}_1, 1 \rangle$
$\langle \mathsf{r}_1, 2 \rangle$
$\langle \mathsf{r}_1, \llcorner \rangle$
$\langle \mathsf{r}_1, \lrcorner \rangle$
$\langle \mathsf{r}_m, \mathsf{B} \rangle$
$\langle \mathsf{r}_m, 1 \rangle$
$\langle \mathsf{r}_m, 2 \rangle$
$\langle \mathsf{r}_m, \llcorner \rangle$
$\langle \mathsf{r}_m, \lrcorner \rangle$

(b) Full state $\langle \mathsf{q}, \lambda \rangle$, with $\mathsf{q} \in \mathscr{Q}_\forall$.

Figure 2: Encoding of the set of instructions matching a full state $\langle \mathsf{q}, \lambda \rangle$ of a LB-ATM.

choice of the α_i for the next action, independently from the choice β of the other agent ag_2. On the other hand, the action profiles $\langle \alpha, \beta_1 \rangle, \ldots, \langle \alpha, \beta_m \rangle$ labeling transitions corresponding to an universal state are such that the roles of the agents are exchanged.

The LB-ATM representation of Figure 2 is hierarchical and involves the modules *write* and *move*. The former encodes the rewriting of the head cell performed by $\mathscr{A}$ and, to this aim, makes use of one of the following modules (Figure 3), depending on the symbol λ read by the head, and on the symbol ν to be written:

- *inc*, depicted in Figure 3a, is used when the rewriting corresponds to an increment, for example, when the symbol 2 has to be written in place of the symbol 1;
- *double_inc*, depicted in Figure 3b, is used when the rewriting corresponds to a double increment, for example, when the symbol 2 (encoded as 2) has to be written in place of the symbol B (encoded as 0);
- *dec*, depicted in Figure 3c, is used when the rewriting corresponds to a decrement, for example, when the symbol 1 has to be written in place of the symbol 2;
- *double_dec*, depicted in Figure 3d, is used when the rewriting corresponds to a double decrement, for example, when the symbol B has to be written in place of the symbol 2.

Obviously, the module does nothing when the symbol to be written corresponds to the symbol currently stored in the head cell.

The module *move* encodes the shift (to right or to left) of the head. It is designed in a way that the only next location that can be reached by the game is consistent with the value stored on the new head cell (after the shift operation). In Figure 4 and 5 the sub-modules encoding the operation "shift to right" are depicted. The encoding of the operation "shift to left" can be realized analogously.

As an example, we describe the first two modules of Figure 4. The module *shift_right*, depicted in

Figure 4a, is performed through the following steps:

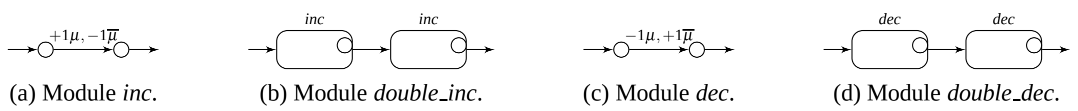
- multiply by 10 the value of μ_L (module $\mathit{times\_10}(\mu_L)$),
- increment the value of μ_L by the value of μ (module $\mathit{add}(\mu_L,\mu)$),
- divide by 10 the value of μ_R (module $\mathit{div\_10}(\mu_R)$ — the remainder of the division is stored in the resource variable r),
- assign to the resource variable μ the value of r (module $\mathit{assign}(\mu,r)$),
- suitably lead the computation to the location corresponding to the next state of the LB-ATM, depending on the value read by the head, that is, the value stored on the resource variable μ (module $\mathit{choose\_next\_state}(\mu)$).

The module $\mathit{times\_10}(\mu_L)$, that multiplies by 10 the value of μ_L (Figure 4b), is performed by storing the value of μ_L in the resource variable i, by setting the value of μ_L to 0, and then by executing a transition (the loop transition), which consumes 1 unit of i and produces 10 units of μ_L (the suitable quantity of the counterbalanced variables is produced or consumed as well, to keep the sum constant) as long as items of the resource i are available. When the availability of i goes down to 0, the other transition is executed (the last transition is needed to keep constant the sum between i and its counterbalanced variable $\bar{i}$). It is easy to convince oneself that the value of μ_L in the exit node is equal to its value in the entry node times 10, and that the sum of each variable and its counterbalanced one is constant. As a last remark, we point out that the names of some of the modules are parametric, in the sense that the arguments between parenthesis are not actual resource variables, but parameters (e.g., x, x_1, x_2) to be instantiated. We adopted this notation for modules that are used more than once, and that are instantiated with actual resource variables when they are used (e.g., the module *assign* depicted in Figure 4c is called $\mathit{assign}(x_1,x_2)$ and it is used, for instance, inside the module $\mathit{times\_10}(\mu_L)$ (Figure 4b), where x_1 (resp., x_2) is instantiated with i (resp., μ_L), and inside the module $\mathit{add}(\mu_L,\mu)$ (Figure 5a), where x_1 (resp., x_2) is instantiated with t (resp., μ).

Now, as resource productions are involved in the reduction, we need to guarantee that the availability of each resource never exceeds the initial one. To this end the values of the components of the vector $\vec{m}_0$ of initial resource availability are set to the value $Max = 322\dots224$, that is the largest number corresponding to an encoding of any tape configuration (precisely, it encodes the tape configuration $\llcorner 22\dots22 \lrcorner$). Before starting the simulation of the LB-ATM, a preliminary step, depicted in Figure 6, modifies the value of the resource variables in such a way that they correctly encode the input tape s_0 and the sum of the availability of each resource variable and its counterbalanced is equal to Max. Thus, the value of the resource variables never exceed Max.

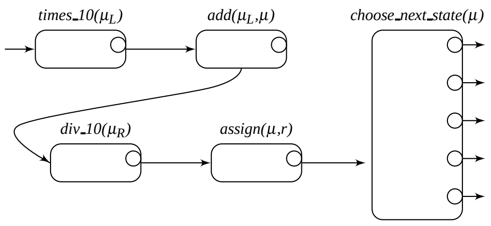
At this point, given a LB-ATM $\mathscr{A}$ and an input tape configuration s_0, the game structure $G_{\mathscr{A},\mathsf{s}_0}$ presents, among others, the following features (the other features of $G_{\mathscr{A},\mathsf{s}_0}$ are either irrelevant or represented in the graphical representation of the encoding — see Figures 2-6):

- 2 agents, ag_1 and ag_2;
- 5 locations, namely $\langle \mathsf{q},\mathsf{B}\rangle$, $\langle \mathsf{q},1\rangle$, $\langle \mathsf{q},2\rangle$, $\langle \mathsf{q},\llcorner\rangle$, $\langle \mathsf{q},\lrcorner\rangle$, for each internal state q of $\mathscr{A}$ (plus other locations — the circles in the pictures — that do not correspond to particular states of the LB-ATM, but are needed to perform the encoding);
- only one atomic proposition p, that holds true over all and only the locations having no matching

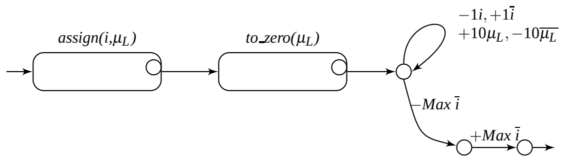


(a) Module *inc*. (b) Module *double_inc*. (c) Module *dec*. (d) Module *double_dec*.

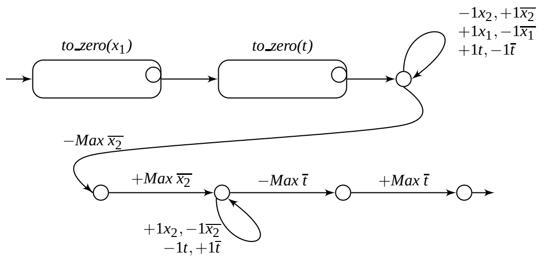
Figure 3: Encoding of the module *write*.



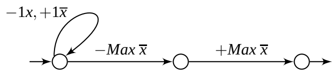
(a) Module *shift_right*.



(b) Module *times_10(μ_L)*.



(c) Module *assign(x_1, x_2)*.



(d) Module *to_zero(x)*.

Figure 4: Encoding of the module *shift_right* - part I.

instructions;

- initial global availability $\vec{m}_0$ is such that all resources are available in quantity Max, as already mentioned above; notice that Max also represents the maximum value occurring in the initial resource availability vector, that is, $M = Max$;
- initial location $\langle \mathsf{q}_0, \lambda \rangle$, where q_0 is the initial state of the LB-ATM and λ is the first input symbol.

The formula $\phi_{\mathscr{A},\mathsf{s}_0} = \langle\langle A^{\vec{\$}} \rangle\rangle \Diamond p$, with $A = \{ag_1\}$ and the value of $\vec{\$}$ being irrelevant for our purposes, is such that $G_{\mathscr{A},\mathsf{s}_0} \models \phi_{\mathscr{A},\mathsf{s}_0}$ if and only if $\mathscr{A}$ accepts on input s_0.

Notice that, for the sake of readability, the game structure used in the reduction does not respect the requirement that, in every location, the first action of every agent is the *do-nothing* action, which does not consume or produce resources. Nevertheless, this omission does not affect the correctness of our reduction, that can be easily adapted using a game structure fulfilling the above requirement.

Theorem 1 *Model checking* PRB-ATL *is EXPTIME-hard even assuming n and r constant.*

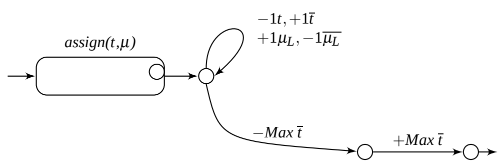
Let us stress that the above reduction makes use of a constant number of agents and resources, while the digit size of M (the maximum value occurring in $\vec{m}_0$) is linear in the size of the tape configuration. This is consistent with the complexity of the algorithm in [9], which remains exponential even if we consider a constant number of agents and resources as input.

Corollary 1 *The model checking problem for* PRB-ATL *is EXPTIME-complete.*

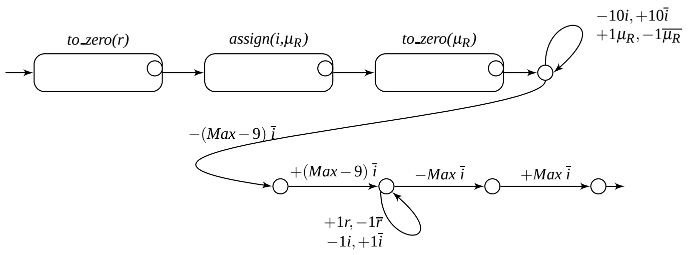
4.3 Another reduction.

As noted at the beginning of Section 4, it is possible to exhibit two more reductions according to which two parameters, out of three, are assumed constant. In the following, we briefly outline how to obtain a reduction from the acceptance problem for LB-ATM, when n and M are constant.

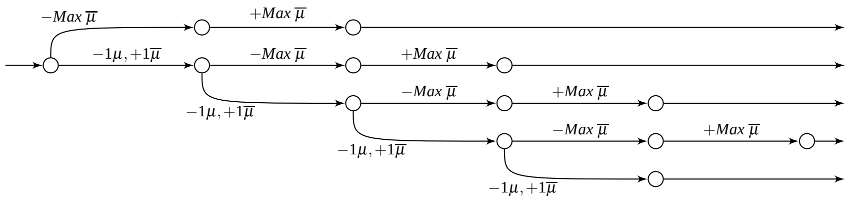
This reduction is simpler than the previous. Here the encoding of the tape is obtained using a number of resources which is linear in the length of the tape. Let $|\mathsf{s}|$ be the length of the tape, we use 2 sets of



(a) Module *add(μ_L, μ).*



(b) Module *div_10(μ_R).*



(c) Module *choose_next_state(μ).*

Figure 5: Encoding of the module *shift_right* - part II.

$|\mathsf{s}|$ resource variables, namely, $\mu_L^1, \mu_L^2, \ldots, \mu_L^{|\mathsf{s}|}$ and $\mu_R^1, \mu_R^2, \ldots, \mu_R^{|\mathsf{s}|}$, plus the resource variable μ. Each variable encodes the content of a tape cell: variable μ encodes the content of the head cell, while, for each i, the variable μ_L^i (resp., μ_R^i) encodes the content of the i-th cell on the left (resp., right) of the tape cell. Notice that, since there are finitely many possible values for a tape cell, the value of M is upper bounded. Now, the encoding of the set of instructions matching a full state $\langle \mathsf{q}, \lambda \rangle$ of a LB-ATM is the same used for the previous reduction and depicted in Figure 2. Nevertheless, the encoding of the module *move*, which encodes the shift (to right or to left) of the head, is slightly different. In Figure 7, the sub-modules encoding the operation "shift to right" are depicted. Essentially, the value of the variable representing a cell is transmitted to the variable representing the cell on the right, and the next location reached on the game structure is set according to the value stored on the current head cell (after the shift operation). The encoding of the operation "shift to left" is made analogously.

Theorem 2 *Model checking* PRB-ATL *is EXPTIME-hard even assuming n and M constant.*

5 Discussion

In this paper we have presented a formalism which is very suitable to model properties of multi-agent systems when the agents share resources and the need of avoiding an unbounded consumption of such resources is crucial. Within our framework it is possible to keep trace of a real global availability of the

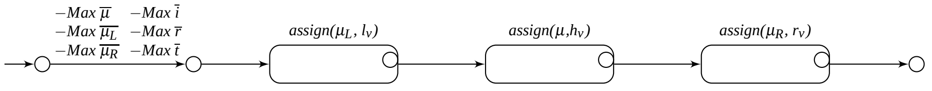


Figure 6: Preliminary step of the reduction (l_v, h_v, and r_v encode the input tape configuration).

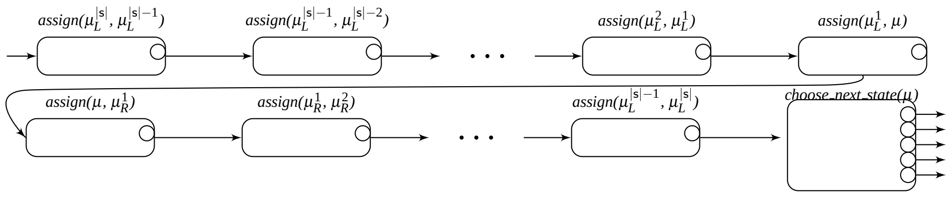

$assign(\mu_L^{|s|}, \mu_L^{|s|-1})$
$assign(\mu_L^{|s|-1}, \mu_L^{|s|-2})$
$assign(\mu_L^{2}, \mu_L^{1})$
$assign(\mu_L^{1}, \mu)$
$assign(\mu, \mu_R^{1})$
$assign(\mu_R^{1}, \mu_R^{2})$
$assign(\mu_L^{|s|-1}, \mu_L^{|s|})$
choose_next_state(μ)

Figure 7: Encoding of the module *shift_right*.

resources, used by both the proponent and opponent players, avoiding thus unrealistic situations in which an unbounded quantity of resources is used in a game.

The technical focus of the paper has been on the complexity of the model checking problem, and we proved that it is EXPTIME complete (recall that also for simpler formalism this problem is in EXPTIME, though the lower bound is not known). Other problems of interest exist in the context of multi-agents system verification. The most important one is the *reachability problem*, that is the problem of determining whether a team, with a given amount of money and a given initial global resource availability, has a strategy to force the execution of the system to reach a given location. More precisely, the reachability problem for a team A on a priced game structure G is a particular instance of the model checking problem, namely, the problem of verifying the truth at the initial configuration of G of a PRB-ATL formula of the kind $\langle\langle A^{\vec{\$}}\rangle\rangle\Diamond p$, for a team A, a money endowment $\vec{\$}$ and $p \in \Pi$. An upper bound on the complexity of this problem is clearly given by the algorithm for solving the model checking problem for PRB-ATL. Let us observe that the reductions given in section 4 apply also to the reachability problem, since the formula used there was $\phi_{\mathscr{A},\mathsf{s}_0} = \langle\langle A^{\vec{\$}}\rangle\rangle\Diamond p$, thus we have the following corollary.

Corollary 2 *The reachability problem for* PRB-ATL *is EXPTIME-complete.*

One of the novelties of our logic is that *the resource production* is allowed in the actions, though with some limitations. Model checking and reachability problems seem both to be simpler in the case one restricts our formalism by considering agent actions that cannot produce resources. The reachability problem is indeed NP-hard in this case: it immediately follows from a result in [11], when the number of agents is not constant. Anyway, we can prove the NP-hardness for just two agents using a reduction from 3-SAT (due to lack of space we omit here the proof). The model checking problem, instead, turns out to be PSPACE-hard, since the reduction from QBF problem given in [9] works also in this case, when actions cannot produce resources. Observe that PRB-ATL with this restriction is again different from other formalisms in literature, mainly for the possibility of tracking resources avalability and for considering shared resources.

Finally, we want to note that also the more general problem, called *optimal coalition problem*, is EXPTIME-complete (the upper bound was shown in [9]). It is the problem of finding optimal (with respect to a suitable cost function) coalitions that are capable to satisfy a given *parametric* PRB-ATL formula, that is, a PRB-ATL formula in which *parametric team operators* $\langle\langle X^{\vec{\$}}\rangle\rangle$ may occur in place of the classical team operators $\langle\langle A^{\vec{\$}}\rangle\rangle$. One could also investigate other optimization problems. The most interesting is, perhaps, to consider the money availability not as an input of the problem, but rather as a parameter to minimize, that is to establish how much money each agent should be provided with, to perform a given task.

Further research directions concern the study of variants of the logic. First, one can consider extensions based on the full alternating-time temporal language ATL*, as already done in [6], and its fragment ATL$^+$.

References

[1] Natasha Alechina, Brian Logan, Nguyen Hoang Nga & Abdur Rakib (2009): *A Logic for Coalitions with Bounded Resources*. In: *Proc. of the 21st International Joint Conference on Artificial Intelligence*, IJCAI '09, pp. 659–664.

[2] Natasha Alechina, Brian Logan, Nguyen Hoang Nga & Abdur Rakib (2010): *Resource-bounded alternating-time temporal logic*. In: *Proc. of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, AAMAS '10, pp. 481–488.

[3] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-time temporal logic*. *Journal of ACM* 49(5), pp. 672–713, doi:`10.1145/585265.585270`.

[4] Rajeev Alur & Mihalis Yannakakis (2001): *Model checking of hierarchical state machines*. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 23(3), pp. 273–303, doi:`10.1145/503502.503503`.

[5] Nils Bulling & Berndt Farwer (2009): *Expressing Properties of Resource-Bounded Systems: The Logics RTL* and RTL*. In Jürgen Dix, Michael Fisher & Peter Novák, editors: *Computational Logic in Multi-Agent Systems (CLIMA X)*, Springer, pp. 22–45, doi:`10.1007/978-3-642-16867-3_2`.

[6] Nils Bulling & Berndt Farwer (2010): *On the (Un-)Decidability of Model Checking Resource-Bounded Agents*. In: *Proc. of the 19th European Conference on Artificial Intelligence*, ECAI '10, pp. 567–572, doi:`10.3233/978-1-60750-606-5-567`.

[7] Ashok K. Chandra, Dexter C. Kozen & Larry J. Stockmeyer (1981): *Alternation*. *Journal of ACM* 28(1), pp. 114–133, doi:`10.1145/322234.322243`.

[8] Mehdi Dastani, Koen V. Hindriks & John-Jules Charles Meyer, editors (2010): *Specification and Verification of Multi-agent Systems*, 1st edition. Springer Publishing Company, Incorporated.

[9] D. Della Monica, M. Napoli & M. Parente (2011): *On a Logic for Coalitional Games with Priced-Resource Agents*. *Electronic Notes in Theoretical Computer Science (ENTCS)* 278, pp. 215–228, doi:`10.1016/j.entcs.2011.10.017`. Proc. of the 7th Workshop on Methods for Modalities (M4M 2011) and the 4th Workshop on Logical Aspects of Multi-Agent Systems (LAMAS 2011).

[10] Valentin Goranko (2001): *Coalition games and alternating temporal logics*. In: *Proc. of the 8th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '01, Morgan Kaufmann, pp. 259–272.

[11] Wojciech Jamroga & Jürgen Dix (2005): *Do Agents Make Model Checking Explode (Computationally)?* In: *Proc. of the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005)*, *Lecture Notes in Computer Science* 3690, Springer, pp. 398–407, doi:`10.1007/11559221_40`.

[12] Salvatore La Torre, Margherita Napoli, Mimmo Parente & Gennaro Parlato (2003): *Hierarchical and Recursive State Machines with Context-Dependent Properties*. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow & Gerhard J. Woeginger, editors: *Proc. of the 30th International Colloquium on Automata, Languages and Programming (ICALP)*, *Lecture Notes in Computer Science* 2719, Springer, pp. 776–789, doi:`10.1007/3-540-45061-0_61`.

[13] Salvatore La Torre, Margherita Napoli, Mimmo Parente & Gennaro Parlato (2008): *Verification of scope-dependent hierarchical state machines*. *Information and Computation* 206(9-10), pp. 1161–1177, doi:`10.1016/j.ic.2008.03.017`.

[14] Aniello Murano, Margherita Napoli & Mimmo Parente (2008): *Program Complexity in Hierarchical Module Checking*. In Iliano Cervesato, Helmut Veith & Andrei Voronkov, editors: *Proc. of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, *Lecture Notes in Computer Science* 5330, Springer, pp. 318–332, doi:`10.1007/978-3-540-89439-1_23`.

[15] Marc Pauly (2001): *A Logical Framework for Coalitional Effectivity in Dynamic Procedures*. *Bulletin of Economic Research* 53(4), pp. 305–324, doi:`10.1111/1467-8586.00136`.

[16] Marc Pauly (2002): *A Modal Logic for Coalitional Power in Games*. *Journal of Logic and Computation* 12(1), pp. 149–166, doi:`10.1093/logcom/12.1.149`.