Mikołaj Bojańczyk
Sławomir Lasota
Igor Potapov (Eds.)

LNCS 9328

Reachability Problems

9th International Workshop, RP 2015
Warsaw, Poland, September 21–23, 2015
Proceedings

Springer

Lecture Notes in Computer Science **9328**

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

More information about this series at http://www.springer.com/series/7407

Mikołaj Bojańczyk · Sławomir Lasota
Igor Potapov (Eds.)

Reachability Problems

9th International Workshop, RP 2015
Warsaw, Poland, September 21–23, 2015
Proceedings

Springer

Editors
Mikołaj Bojańczyk
The Institute of Informatics
University of Warsaw
Warsaw
Poland

Sławomir Lasota
The Institute of Informatics
University of Warsaw
Warsaw
Poland

Igor Potapov
University of Liverpool
Liverpool
UK

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-24536-2 ISBN 978-3-319-24537-9 (eBook)
DOI 10.1007/978-3-319-24537-9

Library of Congress Control Number: 2015949282

LNCS Sublibrary: SL1 Theoretical Computer Science and General Issues

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the papers presented at the 9th International Workshop on Reachability Problems (RP) held on September 21–23, 2015, at the University of Warsaw. Previous workshops in the series were located at: the University of Oxford (2014), Uppsala University (2013), the University of Bordeaux (2012), the University of Genoa (2011), Masaryk University Brno (2010), École Polytechnique (2009), the University of Liverpool (2008), and Turku University (2007).

The aim of the conference is to bring together scholars from diverse fields with a shared interest in reachability problems, and to promote the exploration of new approaches for the modelling and analysis of computational processes by combining mathematical, algorithmic, and computational techniques. Topics of interest include (but are not limited to): reachability for infinite state systems; rewriting systems; reachability analysis in counter/timed/cellular/communicating automata; Petri-nets; computational aspects of semigroups, groups, and rings; reachability in dynamical and hybrid systems; frontiers between decidable and undecidable reachability problems; complexity and decidability aspects; predictability in iterative maps, and new computational paradigms.

The invited speakers at the 2015 workshop were:

- Christel Baier, Technische Universität Dresden, Germany;
- Alessandro D'Innocenzo, Università degli Studi dell'Aquila, Italy;
- Jerome Leroux, LaBRI, University of Bordeaux, France;
- Peter Bro Miltersen, Aarhus University, Denmark;
- Andrey Rybalchenko, Microsoft Research, USA;
- James Worrell, University of Oxford, UK

There were 23 submissions. Each submission was reviewed by at least three Program Committee (PC) members. The members of the PC and the list of external reviewers can be found on the next two pages. The PC is grateful for the highly appreciated and high-quality work produced by these external reviewers. Based on these reviews, the PC decided to accept 14 papers, in addition to the six invited talks. The workshop also provided the opportunity to researchers to give informal presentations, prepared very shortly before the event, informing the participants about current research and work in progress.

We gratefully acknowledge the help of Eryk Kopczyński, Marlena Nowińska, and Wojciech Czerwiński in organizing the event, as well as the Warsaw Center for Mathematics and Computer Science for financial support.

It is also a pleasure to thank the team behind the EasyChair system and the Lecture Notes in Computer Science team at Springer, who together made the production of this

volume possible in time for the workshop. Finally, we thank all the authors for their high-quality contributions, and the participants for making this edition of RP 2015 a success.

September 2015

Mikołaj Bojańczyk
Sławomir Lasota
Igor Potapov

Organization

Program Committee

Mikolaj Bojanczyk	Warsaw University, Poland
Tomas Brazdil	Masaryk University, Czech Republic
Thomas Brihaye	Université de Mons, Belgium
Krishnendu Chatterjee	Institute of Science and Technology, Austria
Lorenzo Clemente	LaBRI, University of Bordeaux I, France
Javier Esparza	Technische Universität München, Germany
Kousha Etessami	University of Edinburgh, UK
Stefan Göller	LSV, ENS Cachan & CNRS, France
Christoph Haase	LSV, ENS Cachan & CNRS, France
Tero Harju	University of Turku, Finland
Raphaël Jungers	The University of Leuven, Belgium
Sławomir Lasota	Warsaw University, Poland
Richard Mayr	University of Edinburgh, UK
Pierre Mckenzie	Université de Montréal, Canada
Joel Ouaknine	Oxford University, UK
Giovanni Pighizzini	Università degli Studi di Milano, Italy
Igor Potapov	University of Liverpool, UK
Alexander Rabinovich	Tel Aviv University, Israel
Sylvain Salvati	Inria, France
Sylvain Schmitz	LSV, ENS Cachan & CNRS, France
Olivier Serre	LIAFA, CNRS & Université Paris Diderot - Paris 7, France

Additional Reviewers

Althoff, Matthias	Mahata, Pritha
Blondin, Michael	Meyer, Philipp J.
Bozzelli, Laura	Novotný, Petr
Carayol, Arnaud	Porreca, Antonio E.
Fijalkow, Nathanael	Sakai, Masahiko
Frehse, Goran	Sangnier, Arnaud
Gentilini, Raffaella	Sankur, Ocan
Hunter, Paul	Totzke, Patrick
Kiefer, Stefan	Velner, Yaron

Abstracts of Invited Talks

Modeling and Co-design of Control Tasks Over Wireless Networking Protocols: State of the Art and Challenges

Alessandro D'Innocenzo

Department of Information Engineering, Computer Science and Mathematics
Center of Excellence DEWS, University of L'Aquila, Italy
alessandro.dinnocenzo@univaq.it

Wireless networked control systems are the integration of physical processes with wireless networked computing units. The co-design of control policies and network configuration must take into account the joint dynamics of operating systems, communication protocols/media and physical devices/processes. In this talk we first give a brief overview of the state-of-the-art on wireless networked control systems. Then we present some recent advances in the co-design of control tasks over wireless networking protocols subject to long term (e.g. failures, malicious attacks, etc) and short term (e.g. dynamic routing, packet drops) networking non-idealities.

Regarding long term networking non-idealities, we address the co-design problem of controller and communication protocol when the physical plant is a MIMO LTI system and the communication nodes are subject to failures and/or malicious attacks. We first characterize by means of necessary and sufficient conditions the set of network configurations that invalidate controllability and observability of the plant. Then, we investigate the problem of detecting and isolating communication nodes affected by failures and/or malicious attacks and provide necessary and sufficient conditions for the solvability of this problem.

Regarding short term networking non-idealities, we investigate the exploitation of redundancy when routing actuation data to a LTI system connected to the controller via a wireless network. We first consider the modeling, stability analysis and controller design problems when the actuation signal is subject to switching propagation delays due to dynamic routing. We show how to model these systems as pure switching linear systems and provide an algorithm for robust stability analysis. We show that the stability analysis problem is NP-hard in general and provide an algorithm that computes in a finite number of steps the look-ahead knowledge of the dynamic routing policy necessary to achieve controllability and stabilizability. Finally, we consider the case when actuation packets can be delivered from the controller to the actuator via multiple paths, each associated with a delay and a packet loss probability: we show that the joint optimal co-design of controller gain, routing and coding parameters can tremendously improve the control performance.

Vector Addition Systems Howto

Jérôme Leroux

University of Bordeaux and CNRS, LaBRI, UMR 5800, Talence, France

Vector addition systems or equivalently Petri nets are one of the most popular formal models for the representation and the analysis of parallel processes. Many known problems for vector addition systems are shown to be decidable thanks to the theory of well-structured transition systems. Indeed, vector addition systems with configurations equipped with the classical point-wise ordering are well-structured transition systems. Based on this observation, problems like the coverability or the termination are shown to be decidable. However, the well-structured transition systems theory cannot explain the decidability of the reachability problem. In this presentation, we show that runs of vector addition systems can be equipped with a well quasi-order that satisfies an amalgamation property. This observation provides a unifying way for solving many problems for vector addition systems including the central reachability problem.

Recent Results on Concurrent Reachability Games

Peter Bro Miltersen

Aarhus University, Denmark

A finite-state *concurrent reachability game G* is a finitely presented two-player game of potentially infinite duration, played between Player 1, the *reachability* player, and Player 2, the *safety* player. The arena of the game consists of a finite set of positions $0, 1, 2, \ldots, N$. When play begins, a pebble rests at position 1, the "start position". At each stage of play, with the pebble resting at a particular "current" position k, Player 1 chooses an *action* $i \in \{1, 2, \ldots, m\}$ while Player 2 concurrently, and without knowledge of the choice of Player 1 similarly chooses an action $j \in \{1, 2, \ldots, m\}$. A fixed and commonly known *transition function* $\pi : \{1, 2, \ldots, N\} \times \{1, 2, \ldots, m\}^2 \rightarrow \{0, 1, 2, \ldots, N\}$ determines the next position of the pebble, namely $\pi(k, i, j)$. If the pebble ever reaches 0 (the "goal position"), play ends, and Player 1 wins the game. If the pebble *never* reaches the goal position, Player 2 wins.

Concurrent reachability games (or slightly more general models) were studied by the game theory community since the 1950s [3]. They were introduced to the computer science community in the seminal paper by de Alfaro, Henzinger and Kupferman [2], first appearing at FOCS'98. The last decade saw numerous new results on quantitative aspects of near-optimal strategies for playing concurrent reachability games, new algorithms and complexity results for finding such strategies, and new applications areas for the algorithms. In this talk, we survey some of these results [1, 4, 5, 6, 7, 8, 9] and present some of the open problems that remain.

References

1. Chatterjee, K., Hansen, K.A., Ibsen-Jensen, R.: Strategy complexity of concurrent stochastic games with safety and reachability objectives. CoRR, abs/1506.02434 (2015)
2. de Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. Theor. Comput. Sci. **386**(3), 188–217 (2007)
3. Everett, H.: Recursive games. In: Kuhn, H.W., Tucker, A.W. (eds.) Contributions to the Theory of Games, Vol. III, volume 39 of Annals of Mathematical Studies. Princeton University Press (1957)

P.B. Miltersen—The author acknowledges support from The Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation and from the Center for research in the Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council.

4. Frederiksen, S.K.S., Miltersen, P.B.: Approximating the value of a concurrent reachability game in the polynomial time hierarchy. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) Algorithms and Computation. LNCS, vol. 8283, pp. 457–467. Springer, Heidelberg (2013)
5. Frederiksen, S.K.S., Miltersen, P.B.: Monomial strategies for concurrent reachability games and other stochastic games. In: Abdulla, P.A., Potapov, I. (eds.) RP 2013. LNCS, vol. 8169, pp. 122–134. Springer, Heidelberg (2013)
6. Hansen, K.A., Ibsen-Jensen, R., Miltersen, P.B.: The complexity of solving reachability games using value and strategy iteration. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 77–90. Springer, Heidelberg (2011)
7. Hansen, K.A., Koucký, M., Lauritzen, N., Miltersen, P.B., Tsigaridas, E.P.: Exact algorithms for solving stochastic games: extended abstract. In: 43rd ACM Symposium on Theory of Computing, (STOC 2011), pp. 205–214 (2011)
8. Hansen, K.A., Koucky, M., Miltersen, P.B.: Winning concurrent reachability games requires doubly exponential patience. In: 24th Annual IEEE Symposium on Logic in Computer Science (LICS 2009), pp. 332–341. IEEE (2009)
9. Miltersen, P.B., Sørensen, T.B.: A near-optimal strategy for a heads-up no-limit texas hold'em poker tournament. In: 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007) (2007)

Horn Constraints with Quantifiers and Cardinalities

Andrey Rybalchenko

Microsoft Research

In this talk we look at verification of parameterized programs from the constraint solving perspective. We show how Horn constraints enhanced with universal quantification and cardinality operators 1) can be used to specify verification conditions, and 2) can be solved automatically.
Joint work with Nikolaj Bjørner and Klaus v. Gleissenthall.

Reachability Problems for Continuous Linear Dynamical Systems

James Worrell

Department of Computer Science, University of Oxford
Parks Road, Oxford OX1 3QD, UK
james.worrell@cs.ox.ac.uk

It is well understood that the interaction between discrete and continuous dynamics makes hybrid automata difficult to analyse algorithmically. However many natural verification questions concerning only the continuous dynamics of such systems are already extremely challenging. This is so even for linear dynamical systems, such as linear hybrid automata and continuous-time Markov chains, whose evolution is determined by linear differential equations. For example, one can ask to decide whether it is possible to escape a particular location of a linear hybrid automaton given initial values of the continuous variables. Likewise one can ask whether a given set of probability distributions is reachable during the evolution of continuous-time Markov chain.

This talk focusses on reachability problems for solutions of linear differential equations. A central decision problem in this area is the Continuous Skolem Problem, which asks whether a real-valued function satisfying an ordinary linear differential equation has a zero. This can be seen as a continuous analog of the Skolem Problem for linear recurrence sequences, which asks whether the sequence satisfying a given recurrence has a zero term. For both the discrete and continuous versions of the Skolem Problem, decidability is open.

We show that the Continuous Skolem Problem lies at the heart of many natural verification questions on linear dynamical systems. We describe some recent work, done in collaboration with Chonev and Ouaknine, that uses results in transcendence theory, Diophantine approximation, and real algebraic geometry to obtain decidability for certain variants of the problem. In particular, we consider a bounded version of the Continuous Skolem Problem, corresponding to time-bounded reachability. We prove decidability of the bounded problem assuming Schanuel's conjecture, one of the main conjectures in transcendence theory. We describe some partial decidability results in the unbounded case and discuss mathematical obstacles to proving decidability of the Continuous Skolem Problem in full generality.

Reasoning About Cost-Utility Constraints in Probabilistic Models

Christel Baier

Faculty of Computer Science
Technische Universität Dresden
Dresden, Germany
Christel.Baier@tu-dresden.de

Various types of automata models with weights attached to the states and/or transitions have been introduced to model and analyze the resource-awareness and other quantitative phenomena of systems. In this context, weight accumulation appears as a natural concept to reason about cost and utility measures. The accumulation of non-negative weights can, for instance, serve to formalize the total energy consumption of a given task schedule or the total penalty to be paid for missed deadlines. Weight functions with negative and positive values can be used to model the energy level in battery-operated devices or the total win or loss of a share at the stock market over one day. The conceptual similarity between accumulated weights and counter machines causes the undecidability of many verification problems for multi-weighted models and temporal logics with weight accumulation over finite paths of unbounded length. However, decidability can be achieved for verification tasks in specialized structures, such as energy games or models with non-negative weight functions. Likewise, decidability results have been established for temporal logics with restricted forms of weight accumulation, such as modalities for weight accumulation along finite windows or limit-average properties.

This extended abstract summarizes results on algorithmic problems for a cost-utility analysis in weighted Markov chains and weighted Markov decision processes.

C. Baier—The author is supported by the DFG through the collaborative research centre HAEC (SFB 912), the Excellence Initiative by the German Federal and State Governments (cluster of excellence cfAED and Institutional Strategy) and the EU-FP-7 grant MEALS (295261).

Contents

Reasoning About Cost-Utility Constraints in Probabilistic Models

Christel Baier(✉)

Faculty of Computer Science, Technische Universität Dresden, Dresden, Germany
Christel.Baier@tu-dresden.de

Various types of automata models with weights attached to the states and/or transitions have been introduced to model and analyze the resource-awareness and other quantitative phenomena of systems. In this context, weight accumulation appears as a natural concept to reason about cost and utility measures. The accumulation of non-negative weights can, for instance, serve to formalize the total energy consumption of a given task schedule or the total penalty to be paid for missed deadlines. Weight functions with negative and positive values can be used to model the energy level in battery-operated devices or the total win or loss of a share at the stock market over one day. The conceptual similarity between accumulated weights and counter machines causes the undecidability of many verification problems for multi-weighted models and temporal logics with weight accumulation over finite paths of unbounded length [5,4,3,11]. However, decidability can be achieved for verification tasks in specialized structures, such as energy games [9,10,19,25] or models with non-negative weight functions [1,21,3]. Likewise, decidability results have been established for temporal logics with restricted forms of weight accumulation, such as modalities for weight accumulation along finite windows [11,3] or limit-average properties [14,5,23].

This extended abstract summarizes results on algorithmic problems for a cost-utility analysis in weighted Markov chains and weighted Markov decision processes. The first part relies on the paper [3] and reports on results for linear temporal specifications with weight assertions. The second part addresses computation schemes for optimal weight parameters in probabilistic reachability constraints. It relies on the papers [24,2,20].

Linear Temporal Logic with Weight Assertions. In the approach of [3], linear temporal logic (LTL) has been extended by *weight assertions* of the form $\Diamond^{\mathcal{A}}(\varphi_1; \mathsf{wconstr}; \varphi_2)$ where $\mathcal{A}$ is a deterministic finite automaton (called *weight monitor*), φ_1 and φ_2 are formulas (called pre- and postcondition) and $\mathsf{wconstr}$ is a linear constraint on the accumulated values of one or more weight functions.

The author is supported by the DFG through the collaborative research centre HAEC (SFB 912), the Excellence Initiative by the German Federal and State Governments (cluster of excellence cfAED and Institutional Strategy) and the EU-FP-7 grant MEALS (295261).

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 1–6, 2015.
DOI: 10.1007/978-3-319-24537-9_1

The weight monitor $\mathcal{A}$ serves to specify the windows of paths where the weights are accumulated. The semantics of weight assertions is defined over pairs (π, j) consisting of an infinite path π in a weighted Markovian model $\mathcal{M}$ and a position $j \in \mathbb{N}$ and given by $(\pi, j) \models \Phi^{\mathcal{A}}(\varphi_1; \mathsf{wconstr}; \varphi_2)$ iff there exist positions $k \in \mathbb{N}$, $k > j$, such that (i) the path fragment $\pi[j \ldots k]$ from position j to k satisfies the weight constraint $\mathsf{wconstr}$ as well as the regular constraint imposed by the weight monitor $\mathcal{A}$ and (ii) the precondition holds for the current position j, while the postcondition holds for position k, i.e., $(\pi, j) \models \varphi_1$ and $(\pi, k) \models \varphi_2$. Pure weight constraints have the form $\Phi^{\mathcal{A}}(\mathit{true}; \mathsf{wconstr}; \mathit{true})$, briefly written as $\Phi^{\mathcal{A}}\,\mathsf{wconstr}$. For example, with $\mathcal{A}$ being a weight monitor specifying the successful processing of a request, the formulas

$$\Box(\mathit{request} \rightarrow \Phi^{\mathcal{A}}((\mathsf{utility} \geqslant u) \wedge (\mathsf{cost} \leqslant c)))$$

$$\Box(\mathit{request} \rightarrow \Phi^{\mathcal{A}}(\mathsf{utility}/\mathsf{cost} \geqslant r))$$

provide cost-utility guarantees for the processing of each request. Here, u, c, r are constants and $\mathsf{utility}$ and cost are symbols for (the accumulated values of) weight functions. Assuming that all cost values are positive, the ratio constraint $\mathsf{utility}/\mathsf{cost} \geqslant r$ can be rephrased as a linear constraint $\mathsf{utility} - r \cdot \mathsf{cost} \geqslant 0$, which meets the syntax of weight constraints.

When dealing with the full class of deterministic finite automata as weight monitors, the model-checking problem for LTL extended by weight assertions is undecidable. This even holds for Boolean combination of pure weight assertions. However, decidability results can be established for LTL extended by weight assertions over *acyclic* weight monitors. In this case, the maximal probability for a formula to hold in a given weighted Markov decision process can be computed using a reduction to the probabilistic model-checking problem for standard LTL and unweighted Markov decision processes (MDP). Although the transformation of the weighted MDP into an unweighted MDP is exponential, the well-known 2EXPTIME-completeness result [13] still holds for LTL extended by weight assertions over acyclic weight monitors. A similar reduction technique is applicable for LTL extended by *simple* weight assertions interpreted over MDPs with non-negative weight functions, called *reward functions*. In this context, simple weight assertions have the form $\Phi^{\mathcal{A}}((\mathsf{wgt}_1 \bowtie_1 c_1) \wedge \ldots \wedge (\mathsf{wgt}_n \bowtie_n c_n))$ where $\mathcal{A}$ is an arbitrary (possibly cyclic) weight monitor and $\mathsf{wgt}_1, \ldots, \mathsf{wgt}_n$ are symbols for the reward functions in the given MDP. The symbols $\bowtie_1, \ldots, \bowtie_n$ stand for comparison operators, while $c_1, \ldots c_n$ are constants.

Optimal Weight and Ratio Bounds. Although the parameter synthesis problem for LTL extended by modalities with parametric step bounds and Markov chains is undecidable [8], several algorithms have been proposed to compute optimal weight bounds for specific parametric formulas. The work on *quantiles* for probabilistic reachability constraints in Markov chains and MDPs with non-negative weight (i.e., reward) functions [24,2] can be seen as special

instances of the parameter synthesis problem. For example, the quantiles

$$r^{\exists} = \min \left\{ r \in \mathbb{N} \ : \ \Pr^{\max}_{\mathcal{M},s}(\Diamond^{\leqslant r} goal) > q \right\}$$

$$r^{\forall} = \min \left\{ r \in \mathbb{N} \ : \ \Pr^{\min}_{\mathcal{M},s}(\Diamond^{\leqslant r} goal) > q \right\}$$

for state s of an MDP $\mathcal{M}$ with a reward function and a distinguished goal state can be interpreted as the minimal cost required to ensure that the goal state will be reached from s with probability larger than q under some scheduler for $\mathcal{M}$ resp. for all schedulers for $\mathcal{M}$.[1] Quantiles with upper probability bounds "$< q$" or non-strict probability bounds "$\geqslant q$" or "$\leqslant q$" are defined accordingly. For qualitative probability bounds (i.e., for the cases where $q \in \{0, 1\}$) polynomially time-bounded computation schemes have been presented in [24] using variants of Dijkstra's shortest path algorithm. Given the EXPTIME-completeness result by [18] for checking whether $p_{s,r} \stackrel{\text{def}}{=} \Pr^{\max}_{\mathcal{M},s}(\Diamond^{\leqslant r} goal) > q$ when $\mathcal{M}, s, r$ and q are given, no polynomially time-bounded algorithms can be expected for the general case (i.e., for $q \in]0, 1[$). The idea to compute $r^{\exists}$ is a two-step approach. In the first step, we check the existence of some r satisfying $p_{s,r} > q$. For this, we can apply standard linear-programming techniques for computing the maximal probabilities $p_s = \Pr^{\max}_{\mathcal{M},s}(\Diamond goal)$ for reaching the goal from s without reward constraints. Note that thanks to the monotonicity of accumulated rewards along the prefixes of infinite paths we have $p_s = \sup_{r \in \mathbb{N}} p_{s,r}$. If $p_s \leqslant q$ then there is no r with $p_{s,r} > q$, and therefore $r^{\exists} = \infty$. Otherwise, we proceed with the second step, where we successively consider all candidates $r = 0, 1, 2, \ldots$ and compute the values $p_{t,r}$ for all states t until $p_{s,r} > q$. The values $p_{t,r}$ for all states t and fixed reward bound r are obtained by solving a linear program with one variable per state and reusing the values $p_{t,r'}$ for $0 \leqslant r' < r$ computed in previous iterations. This yields an exponentially time-bounded computation scheme for the existential quantile $r^{\exists}$. The universal quantile $r^{\forall}$ can be computed analogously.

When switching from reward to weight functions that might have positive and negative values, we cannot longer rely on the monotonicity of the accumulated weights along the prefixes of infinite paths. For single-weighted Markov chains and qualitative probability bounds, a polynomially time-bounded approach has been presented in [20]. It uses a binary search and polynomial-time decision procedures for checking qualitative probability constraints for weight assertions of the form $\Box(\mathsf{wgt} > c)$. The latter relies on computation of the expected long-run average weight in bottom strongly connected components and variants of shortest-path algorithms. More challenging are quantiles for assertions on cost-utility ratios, e.g., of the form

$$\mathrm{r}^{opt} \stackrel{\text{def}}{=} \sup \left\{ r \in \mathbb{Q} \ : \ \Pr_{\mathcal{M},s}(\Box(\mathsf{ratio} > r)) > 0 \right\}$$

where $\mathsf{ratio} = \mathsf{utility}/\mathsf{cost}$ is the quotient of two positive weight functions of a Markov chain $\mathcal{M}$. A polynomial-time algorithm for computing r^{opt} has been

[1] The notations $\Pr^{\max}_{\mathcal{M},s}(\varphi)$ and $\Pr^{\max}_{\mathcal{M},s}(\varphi)$ are used to denote the supremum resp. infimum of the probabilities for the event φ, when ranging over all schedulers. For the events considered here, optimal schedulers exists.

presented in [20] using the observation that $\mathrm{r}^{opt} \in R$ where R consists of the long-run ratios of the bottom strongly connected components of $\mathcal{M}$ and the ratios $\mathsf{ratio}(\sigma) = \mathsf{utility}(\sigma)/\mathsf{cost}(\sigma)$ along the simple finite paths σ starting in s as well as the ratios of all simple cycles that are accessible from s.[2] As R is a finite subset of $\mathbb{Q}$, r^{opt} is rational. The straight-forward approach to compute $\max\{r \in R : \Pr_{\mathcal{M},s}(\Box(\mathsf{ratio} > r)) > 0\}$ by inspecting all elements $r \in R$ has exponential time complexity. However, the *continued-fraction method* (see e.g. [17]) yields a polynomial-time computation scheme for r^{opt}. The continued-fraction method requires polynomial-time algorithms to compute a bound N for the denominator of r^{opt} (i.e., $\mathrm{r}^{opt} \in \{a/b : a \in \mathbb{Z}, b \in \{1, \ldots, N\}\}$) and an approximation of r^{opt} up to precision $\varepsilon = 1/(2N^2)$. Such a bound N can be derived in polynomial time from the local cost values and the long-run ratios of the bottom strongly connected components. An ε-approximation of r^{opt} can be obtained by a binary search in the interval $[0, \mathrm{r}^{max}]$ where r^{max} is the quotient of the maximal local utility value and the minimal local cost value.

Conclusion. The material summarized in this extended abstract is in the line of recent work by various research groups on temporal logics and algorithms for analyzing the tradeoff of different objectives in Markovian models. Besides the related work mentioned at the beginning of this article, there are various approaches that deal with multi-weighted models and multiple probability and expectation objectives (see e.g. [16,6,22]) or related synthesis problems (see e.g. [26,7,15,12]). The list of open problems in these research directions is still long and covers, e.g., a better understanding of the gaps between decidability and undecidability of model-checking problems for temporal logics with weight assertions in multi-weighted Markovian models, parameter synthesis algorithms for more complex parametric weight assertions, and algorithms to reason about quantiles with quantitative probability constraints in weighted MDPs or stochastic two-player game structures.

Acknowledgments. The presented material relies on joint work with Marcus Daum, Clemens Dubslaff, Joachim Klein, Daniel Krähmann, Sascha Klüppelholz, Jana Schubert, Michael Ummels and Sascha Wunderlich.

References

1. Andova, S., Hermanns, H., Katoen, J.-P.: Discrete-time rewards model-checked. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 88–104. Springer, Heidelberg (2004)
2. Baier, C., Daum, M., Dubslaff, C., Klein, J., Klüppelholz, S.: Energy-utility quantiles. In: Badger, J.M., Rozier, K.Y. (eds.) NFM 2014. LNCS, vol. 8430, pp. 285–299. Springer, Heidelberg (2014)

[2] A path is called simple if each state occurs at most once. Likewise, a simple cycle means a cycle that visits each state at most once, except that its first state equals its last state.

3. Baier, C., Klein, J., Klüppelholz, S., Wunderlich, S.: Weight monitoring with linear temporal logic: complexity and decidability. In: 23rd Conference on Computer Science Logic and the 29th Symposium on Logic In Computer Science (CSL-LICS), pp. 11:1–11:10. ACM (2014)
4. Bauer, S.S., Juhl, L., Larsen, K.G., Srba, J., Legay, A.: A logic for accumulated-weight reasoning on multiweighted modal automata. In: 6th International Symposium on Theoretical Aspects of Software Engineering (TASE), pp. 77–84. IEEE Computer Society (2012)
5. Boker, U., Chatterjee, K., Henzinger, T.A., Kupferman, O.: Temporal specifications with accumulative values. In: 26th Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 43–52. IEEE Computer Society (2011)
6. Brázdil, T., Brozek, V., Chatterjee, K., Forejt, V., Kucera, A.: Multiple mean-payoff objectives in Markov decision processes. Logical Methods in Computer Science **10**(1) (2014)
7. Bruyère, V., Filiot, E., Randour, M., Raskin, J.-F.: Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In: 31st International Symposium on Theoretical Aspects of Computer Science (STACS), Leibniz International Proceedings in Informatics (LIPIcs), vol. 25, pp. 199–213. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2014)
8. Chakraborty, S., Katoen, J.-P.: Parametric LTL on markov chains. In: Diaz, J., Lanese, I., Sangiorgi, D. (eds.) TCS 2014. LNCS, vol. 8705, pp. 207–221. Springer, Heidelberg (2014)
9. Chatterjee, K., Doyen, L.: Energy and mean-payoff parity markov decision processes. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 206–218. Springer, Heidelberg (2011)
10. Chatterjee, K., Doyen, L.: Energy parity games. Theoretical Computer Science **458**, 49–60 (2012)
11. Chatterjee, K., Doyen, L., Randour, M., Raskin, J.-F.: Looking at mean-payoff and total-payoff through windows. Information and Computation **242**, 25–52 (2015)
12. Chatterjee, K., Henzinger, T.A., Jobstmann, B., Singh, R.: Measuring and synthesizing systems in probabilistic environments. Journal of the ACM **62**(1), 9:1–9:34 (2015)
13. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. Journal of the ACM **42**(4), 857–907 (1995)
14. de Alfaro, L.: Formal Verification of Probabilistic Systems. Ph.D. thesis, Stanford University, Department of Computer Science (1997)
15. Dräger, K., Forejt, V., Kwiatkowska, M., Parker, D., Ujma, M.: Permissive controller synthesis for probabilistic systems. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014 (ETAPS). LNCS, vol. 8413, pp. 531–546. Springer, Heidelberg (2014)
16. Etessami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. Logical Methods in Computer Science **4**(4) (2008)
17. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer (1993)
18. Haase, C., Kiefer, S.: The odds of staying on budget. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 234–246. Springer, Heidelberg (2015)
19. Juhl, L., Guldstrand Larsen, K., Raskin, J.-F.: Optimal bounds for multiweighted and parametrised energy games. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 244–255. Springer, Heidelberg (2013)

20. Krähmann, D., Schubert, J., Baier, C., Dubslaff, C.: Ratio and weight quantiles. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) MFCS 2015. LNCS, vol. 9234, pp. 344–356. Springer, Heidelberg (2015)
21. Laroussinie, F., Sproston, J.: Model checking durational probabilistic systems. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 140–154. Springer, Heidelberg (2005)
22. Randour, M., Raskin, J.-F., Sankur, O.: Percentile queries in multi-dimensional markov decision processes. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 123–139. Springer, Heidelberg (2015)
23. Tomita, T., Hiura, S., Hagihara, S., Yonezaki, N.: A temporal logic with mean-payoff constraints. In: Aoki, T., Taguchi, K. (eds.) ICFEM 2012. LNCS, vol. 7635, pp. 249–265. Springer, Heidelberg (2012)
24. Ummels, M., Baier, C.: Computing quantiles in markov reward models. In: Pfenning, F. (ed.) FOSSACS 2013 (ETAPS 2013). LNCS, vol. 7794, pp. 353–368. Springer, Heidelberg (2013)
25. Velner, Y., Chatterjee, K., Doyen, L., Henzinger, T.A., Rabinovich, A.M., Raskin, J.-F.: The complexity of multi-mean-payoff and multi-energy games. Information and Computation **241**, 177–196 (2015)
26. von Essen, C., Jobstmann, B.: Synthesizing efficient controllers. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 428–444. Springer, Heidelberg (2012)

Integer-Complete Synthesis for Bounded Parametric Timed Automata

Étienne André[1(✉)], Didier Lime[2], and Olivier H. Roux[2]

[1] LIPN, CNRS, UMR 7030, Université Paris 13, Sorbonne Paris Cité, Villetaneuse, France
Etenne.Andre@lipn.fr
[2] IRCCyN, CNRS, UMR 6597, École Centrale de Nantes, Nantes, France

Abstract. Ensuring the correctness of critical real-time systems, involving concurrent behaviors and timing requirements, is crucial. Parameter synthesis aims at computing dense sets of valuations for the timing requirements, guaranteeing a good behavior. However, in most cases, the emptiness problem for reachability (*i.e.,* whether there exists at least one parameter valuation for which some state is reachable) is undecidable and, as a consequence, synthesis procedures do not terminate in general, even for bounded parameters. In this paper, we introduce a parametric extrapolation, that allows us to derive an underapproximation in the form of linear constraints containing all the integer points ensuring reachability or unavoidability, and all the (non-necessarily integer) convex combinations of these integer points, for general PTA with a bounded parameter domain. Our algorithms terminate and can output constraints arbitrarily close to the complete result.

1 Introduction

The verification of systems mixing time and concurrency is a notoriously difficult problem. Timed automata (TA) [1] are a powerful formalism for which many interesting problems (including the reachability of a location) are decidable. However, the classical definition of TA is not tailored to verify systems only partially specified, especially when the value of some timing constants is not yet known. Parametric timed automata (PTA) [2] leverage this problem by allowing the specification and the verification of systems where some of the timing constants are parametric. This expressive power comes at the price of the undecidability of most interesting problems.

Related Work. Parametric Timed Automata were introduced in [2]. The simple problem of the existence of a parameter valuation such that some location is reachable is undecidable in both discrete and dense-time even with only three parametric clocks (*i.e.,* clocks compared to a parameter) [7,17], and even with only strict constraints [11]. It is decidable for a single parametric clock in discrete and dense time [2], and in discrete time with two parametric clocks and

This work is partially supported by the ANR national research program "PACS" (ANR-2014).

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 7–19, 2015.
DOI: 10.1007/978-3-319-24537-9_2

one parameter (and arbitrarily many non-parametric clocks) [10], or for a single parametric clock (with arbitrarily many non-parametric clocks) [7]. More complex properties expressed in parametric TCTL have been studied in [9,18]. PTA subclasses have been studied, most notably L/U-automata, for which the problem is decidable [13], but no synthesis algorithm is provided, and there are indeed practical difficulties in proposing one [14]. When further restricting to L- or U-automata, the integer-valued parameters can be synthesized however [8]. The trace preservation problem (*i.e.,* given a reference parameter valuation whether there exists another valuation for which the discrete behavior is the same) is undecidable for both general PTA and L/U-PTA [4].

In [14], we focus on integer-valued bounded parameters (but still considering dense-time), for which many problems are obviously decidable, and we provide symbolic algorithms to compute the set of correct integer parameter values ensuring a reachability ("IEF") or unavoidability ("IAF") property. A drawback is that returning only integer points prevents designers to use the synthesized constraint to study the robustness or implementability of their system (see, *e.g.,* [16]).

Contribution. We propose terminating algorithms that compute a dense under-approximation of the set of parameter values ensuring reachability or unavoidability in bounded PTA (*i.e.,* PTA with a bounded parameter domain). These under-approximations are "integer-complete" in the sense that they are guaranteed to contain at least all the correct integer values given in the form of a finite union of polyhedra; they are also "almost-complete" in the sense that the only points that may not be included in the result are non-integer (rational) points beyond the last integer point in a convex polyhedron. To the best of our knowledge, these algorithms are the first synthesis algorithms that return an almost-complete result for a subclass of PTA (namely bounded PTA) for which the corresponding emptiness problems are undecidable; in fact, with the exception of two subclasses of L/U-PTA (namely L-PTA and U-PTA) considered over discrete parameter valuations [8], we are not aware of any terminating synthesis algorithm deriving a complete or an almost-complete result. Our algorithms also quantify the "size" of the resulting constraint, *i.e.,* they return all valuations except possibly some non-integer points beyond the "last" integer points.

While of great practical interest, our algorithms are in essence quite similar to those of [14]. We however demonstrate that while the algorithms from [14] also return a symbolic representation of the "good" integer parameter values, interpreting the result of the IAF algorithm as dense is not correct in the sense that some non-integer parameter values in that result may not ensure the unavoidability property. Furthermore, since we produce real-valued parameter values, we cannot use anymore the result from [14] ensuring termination of the algorithms, which allows to derive a bound on clock values but relies on the parameters being bounded integers. The main *technical* contribution of this paper is therefore the derivation of a maximum-constant-based parametric extrapolation operator for bounded PTA that ensures termination of our algorithms. To the best of our knowledge this operator is the first of its kind.

Finally, we have implemented the two algorithms and briefly report on them.

Outline. We first recall the necessary definitions in Section 2. We present our parametric extrapolation in Section 3. We then introduce our terminating algorithms (namely RIEF, RIAF) in Section 4. We conclude in Section 5.

2 Preliminaries

2.1 Clocks, Parameters and Constraints

Throughout this paper, we assume a set $X = \{x_1, \ldots, x_H\}$ of *clocks*, *i.e.,* real-valued variables that evolve at the same rate. A clock valuation is a function $w : X \to \mathbb{R}_+$. We identify a clock valuation w with the *point* $(w(x_1), \ldots, w(x_H))$. We write $X = 0$ for $\bigwedge_{1 \leq i \leq H} x_i = 0$. Given $d \in \mathbb{R}_+$, $w + d$ denotes the valuation such that $(w + d)(x) = w(x) + d$, for all $x \in X$.

We assume a set $P = \{p_1, \ldots, p_M\}$ of *parameters*, *i.e.,* unknown constants. A parameter *valuation* v is a function $v : P \to \mathbb{Q}_+$. We identify a valuation v with the *point* $(v(p_1), \ldots, v(p_M))$. An *integer* parameter valuation is a valuation $v : P \to \mathbb{N}$.

In the following, we assume $\sim \in \{<, \leq, \geq, >\}$. A *constraint* C (*i.e.,* a convex polyhedron) over $X \cup P$ is a conjunction of inequalities of the form $lt \sim 0$, where lt denotes a linear term over $X \cup P$ of the form $\sum_{1 \leq i \leq H} \alpha_i x_i + \sum_{1 \leq j \leq M} \beta_j p_j + d$, with $\alpha_i, \beta_j, d \in \mathbb{Z}$. Given a parameter valuation v, $v(C)$ denotes the constraint over X obtained by replacing each parameter p in C with $v(p)$. Likewise, given a clock valuation w, $w(v(C))$ denotes the expression obtained by replacing each clock x in $v(C)$ with $w(x)$. We say that v *satisfies* C, denoted by $v \models C$, if the set of clock valuations satisfying $v(C)$ is nonempty. Given a parameter valuation v and a clock valuation w, we denote by $w|v$ the valuation over $X \cup P$ such that for all clocks x, $w|v(x) = w(x)$ and for all parameters p, $w|v(p) = v(p)$. We use the notation $w|v \models C$ to indicate that $w(v(C))$ evaluates to true. We say that C is *satisfiable* if $\exists w, v$ s.t. $w|v \models C$. An *integer point* is $w|v$, where w is an integer clock valuation, and v is an integer parameter valuation. We define the *time elapsing* of C, denoted by $C^{\nearrow}$, as the constraint over X and P obtained from C by delaying all clocks by an arbitrary amount of time. We define the *past* of C, denoted by $C^{\swarrow}$, as the constraint over X and P obtained from C by letting time pass backward by an arbitrary amount of time (see [14]). Given $R \subseteq X$, we define the *reset* of C, denoted by $[C]_R$, as the constraint obtained from C by resetting the clocks in R, and keeping the other clocks unchanged. We denote by $C{\downarrow_P}$ the projection of C onto P, *i.e.,* obtained by eliminating the clock variables.

A *guard* g is a constraint over $X \cup P$ defined by inequalities of the form $x \sim z$, where z is either a parameter or a constant in $\mathbb{Z}$. Let *plt* denote a parametric linear term over P, that is a linear term without clocks ($\alpha_i = 0$ for all i). A *zone* C is a constraint over $X \cup P$ defined by inequalities of the form $x_i - x_j \sim plt$, where $x_i, x_j \in X \cup \{x_0\}$, where x_0 is the zero-clock always equal to 0.

A *parametric constraint* K is a constraint over P defined by inequalities of the form $plt \sim 0$. We denote by $\top$ (resp. $\bot$) the parametric constraint that corresponds to the set of all possible (resp. the empty set of) parameter valuations.

2.2 Parametric Timed Automata

Parametric timed automata (PTA) extend timed automata with parameters within guards and invariants in place of integer constants [2].

Definition 1. *A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, l_0, X, P, I, E)$, where: i) Σ is a finite set of actions, ii) L is a finite set of locations, iii) $l_0 \in L$ is the initial location, iv) X is a set of clocks, v) P is a set of parameters, vi) I is the invariant, assigning to every $l \in L$ a guard $I(l)$, vii) E is a set of edges $e = (l, g, a, R, l')$ where $l, l' \in L$ are the source and destination locations, $a \in \Sigma$, $R \subseteq X$ is a set of clocks to be reset, and g is a guard.*

Given a parameter valuation v, we denote by $v(\mathcal{A})$ the non-parametric timed automaton where all occurrences of a parameter p_i have been replaced by $v(p_i)$.

Definition 2 (Semantics of a TA). *Given a PTA $\mathcal{A} = (\Sigma, L, l_0, X, P, I, E)$, and a parameter valuation v, the concrete semantics of $v(\mathcal{A})$ is given by the timed transition system $(Q, q_0, \Rightarrow)$, with $Q = \{(l, w) \in L \times \mathbb{R}_+^H \mid v|w \models I(l)\}$, $q_0 = (l_0, X = 0)$, $((l, w), e, (l', w')) \in \Rightarrow$ if $\exists w'' : (l, w) \xrightarrow{e} (l', w'') \xrightarrow{d} (l', w')$, with: $(l, w) \xrightarrow{e} (l', w')$, if $(l, w), (l', w') \in Q$, there exists $e = (l, g, a, R, l') \in E$, $w' = [w]_R$, and $v|w \models g$; and $(l, w) \xrightarrow{d} (l, w + d)$, with $d \in \mathbb{R}_+$, if $\forall d' \in [0, d], (l, w + d') \in Q$.*

We refer to states of a TA as concrete states. A concrete run of a TA is an alternating sequence of (concrete) states of Q and edges of the form $s_0 \overset{e_0}{\Rightarrow} s_1 \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} s_m$, such that for all $i = 0, \ldots, m-1$, $e_i \in E$, and $(s_i, e_i, s_{i+1}) \in \Rightarrow$. Given a state $s = (l, w)$, we say that s is reachable (or that $v(\mathcal{A})$ reaches s) if s belongs to a run of $v(\mathcal{A})$.

Symbolic States. We now recall the symbolic semantics of PTA: A *symbolic state* of a PTA $\mathcal{A}$ is a pair (l, C) where $l \in L$ is a location, and C its associated zone. A state $s = (l, C)$ is v-compatible if $v \models C$. The initial state of $\mathcal{A}$ is $s_0 = (l_0, (X = 0)^{\nearrow} \wedge I(l_0))$. The symbolic semantics relies on the Succ operation. Given a symbolic state $s = (l, C)$ and an edge $e = (l, g, a, R, l')$, $\mathsf{Succ}(s, e) = (l', C')$, with $C' = \left([(C \wedge g)]_R\right)^{\nearrow} \cap I(l')$.

A symbolic run of a PTA is an alternating sequence of symbolic states and edges of the form $s_0 \overset{e_0}{\Rightarrow} s_1 \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} s_m$, such that for all $i = 0, \ldots, m-1$, $e_i \in E$, and s_{i+1} belongs to $\mathsf{Succ}(s_i, e)$. Given a state s, we say that s is reachable if s belongs to a run of $\mathcal{A}$. A maximal run is a run that is either infinite, or that cannot be extended.

Given a PTA $\mathcal{A}$ and a parameter valuation v, given a concrete state (l, w) of $v(\mathcal{A})$ and a symbolic state (l', C) of $\mathcal{A}$, we write $(l, w) \in v((l', C))$ if $l = l'$ and $w \models v(C)$.

In this paper, we will consider *bounded* PTA. A bounded parameter domain assigns to each parameter a minimum integer bound and a maximum integer bound. That is, each parameter p_i ranges in an interval $[a_i, b_i]$, with $a_i, b_i \in \mathbb{N}$. Hence, a bounded parameter domain is a hyperrectangle in M dimensions.

Integer Hulls. We briefly recall some definitions from [14]. Let C be a convex polyhedron. C is topologically closed if it can be defined using only non-strict inequalities.[1] The integer hull of a topologically closed polyhedron, denoted by $\mathsf{IH}(C)$, is defined as the convex hull of the integer vectors of C, *i.e.,* $\mathsf{IH}(C) = \mathsf{Conv}(\mathsf{IV}(C))$, where Conv denotes the convex hull, and IV the set of vectors with integer coordinates.

We treat integer hulls for finite unions of convex polyhedra in a manner similar to [14]: given a (possibly non-convex) finite union of convex polyhedra $\bigcup_i C_i$, we write $\mathsf{IH}(\bigcup_i C_i)$ for the set $\bigcup_i(\mathsf{IH}(C_i))$. Given a symbolic state $s = (l, C)$, we often write $\mathsf{IH}(s)$ for $(l, \mathsf{IH}(C))$.

Decision and Computation Problems. Given a class of decision problems $\mathcal{P}$ (reachability, unavoidability, etc.), we consider the problem of synthesizing the set (or part of it) of parameter valuations v such that $v(\mathcal{A})$ satisfies ϕ.

Here, we mainly focus on reachability (*i.e.,* does there exist a run that goes through some goal locations) called here EF, and unavoidability (*i.e.,* do all maximal runs go through some goal locations) called here AF.

3 Parametric Extrapolation

In this section, we present an extrapolation based on the classical k-extrapolation used for the zone-abstraction for timed automata, but this time in a parametric setting.

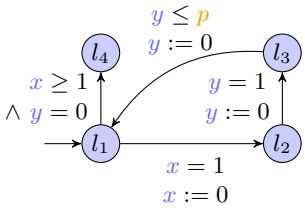


Fig. 1. Motivating PTA

First, let us motivate the use of an extrapolation. Consider the PTA in Fig. 1. After a number n of times through the loop, we get constraints in l_1 of the form $0 \le x - y \le n \times p$, with n growing without bound. Even if the parameter p is bounded (*e.g.,* in $[0, 1]$), the time necessary to reach location l_4 is unbounded. This was not the case in [14] due to the fact that parameters were integers. Hence, on this PTA, we cannot just apply the integer hull (as in [14]) to ensure termination of our algorithms.

Now, we will show that the union for all values of the parameters of the classical k-extrapolation used for the zone-abstraction for timed automata leads

[1] We only define here the integer hull of a topologically closed polyhedron. In fact, any non-closed polyhedron can be represented by a closed polyhedron with one extra dimension [12]. Direct handling of not-necessarily-closed (NNC) polyhedra raises no theoretical issue but would impair the readability of this paper (see [14]).

to a non-convex polyhedron. Let us consider the PTA in Fig. 2a with a parameter p such that $0 \leq p \leq 1$. By taking n times the loop we obtain:

$$0 \leq x \leq p \quad \wedge \quad 0 \leq y - x \leq (n+1) \times p \quad \wedge \quad 0 \leq p \leq 1$$

The greatest constant of the model is $k = 1$. After one loop, y can be greater than 1. Then, for each value of p, we can apply the classical k-extrapolation used for timed automata (as recalled in [6]) of the corresponding zone. The union for all values of p of these extrapolations, projected to the plan (y, p) is depicted by the plain blue part (light and dark blue) of Fig. 2b. The obtained polyhedron is non-convex.

(a) PTA **(b)** Extrapolation

Fig. 2. Example illustrating the non-convex parametric extrapolation

Let us now introduce our concept of (M, X)-extrapolation.

For any zone C and variable x, we denote by $\mathsf{Cyl}_x(C)$ the *cylindrification* of C along variable x, *i.e.*, $\mathsf{Cyl}_x(C) = \{w \mid \exists w' \in C, \forall x' \neq x, w'(x') = w(x')$ and $w(x) \geq 0\}$. This is a usual operation that consists in *unconstraining* variable x.

Definition 3 ((M, x)-extrapolation). *Let C be a polyhedron. Let M be a non-negative integer constant and x be a clock. The (M, x)-extrapolation of C, denoted by $\mathsf{Ext}_x^M(C)$, is defined as:*

$$\mathsf{Ext}_x^M(C) = \big(C \cap (x \leq M)\big) \cup \mathsf{Cyl}_x\big(C \cap (x > M)\big) \cap (x > M).$$

Given $s = (l, C)$, we write $\mathsf{Ext}_x^M(s)$ for $\mathsf{Ext}_x^M\big(C\big)$.

To illustrate the (M, X)-extrapolation, we go back to the example of Fig. 2a after one loop. C is the polyhedron for $n = 1$. $\mathsf{Ext}_y^1(C)$ is depicted in Fig. 2b by the plain blue part as follows: $\big(C \cap (y \leq 1)\big)$ is in light blue and $\mathsf{Cyl}_y\big(C \cap (y > 1)\big) \cap (y > 1)$ is in dark blue. Note that for this example the $(1, y)$-extrapolation gives the same result as the union for all values of the parameter p of the classical extrapolation for timed automata. Lemma 1 follows from Definition 3.

Lemma 1. *For all polyhedra C, integers $M \geq 0$ and clock variables x and x', we have $\mathsf{Ext}_x^M\big(\mathsf{Ext}_{x'}^M(C)\big) = \mathsf{Ext}_{x'}^M\big(\mathsf{Ext}_x^M(C)\big)$.*

Proof (sketch). The result comes from the following facts:

1. $\mathsf{Cyl}_x\big(\mathsf{Cyl}_{x'}(C)\big) = \mathsf{Cyl}_{x'}\big(\mathsf{Cyl}_x(C)\big)$;
2. for $x \neq x', \mathsf{Cyl}_x(C) \cap (x' \sim M) = \mathsf{Cyl}_x\big(C \cap (x' \sim M)\big)$ for $\sim \in \{<, \leq, \geq, >\}$.

We can now consistently define the (M, X)-extrapolation operator:

Definition 4 ((M, X)-extrapolation). *Let M be a non-negative integer constant and X be a set of clocks. The (M, X)-extrapolation operator Ext_X^M is defined as the composition (in any order) of all Ext_x^M, for all $x \in X$. When clear from the context we omit X and only write M-extrapolation or Ext^M.*

In the rest of this section, we prove most results on the extrapolation first on Ext_x^M. It is then straightforward to adapt them to Ext_X^M using Lemma 1.

Crucially, extrapolation preserves the projection onto P:

Lemma 2. *Let C be a constraint over $X \cup P$. Then $C{\downarrow_P} = \mathsf{Ext}_x^M(C){\downarrow_P}$.*

For the preservation of behaviors, following [6], we use a notion of simulation:

Definition 5 (Simulation [6]). *Let $\mathcal{A} = (\Sigma, L, l_0, X, I, E)$ be a TA and $\preceq$ a relation on $L \times \mathbb{R}_+^H$. Relation $\preceq$ is a (location-based) simulation if:*

- *if $(l_1, w_1) \preceq (l_2, w_2)$ then $l_1 = l_2$,*
- *if $(l_1, w_1) \preceq (l_2, w_2)$ and $(l_1, w_1) \xrightarrow{a} (l_1', w_1')$, then there exists (l_2', w_2') such that $(l_2, w_2) \xrightarrow{a} (l_2', w_2')$ and $(l_1', w_1') \preceq (l_2', w_2')$,*
- *if $(l_1, w_1) \preceq (l_2, w_2)$ and $(l_1, w_1) \xrightarrow{d} (l_1, w_1 + d)$, then there exists d' such that $(l_2, w_2) \to (l_2, w_2 + d')$ and $(l_1, w_1 + d) \preceq (l_2, w_2 + d')$.*

If $\preceq^{-1}$ is also a simulation relation then $\preceq$ is called a bisimulation.

State s_1 simulates s_2 if there exists a simulation $\preceq$ such that $s_2 \preceq s_1$. If $\preceq$ is a bisimulation, then the two states are said bisimilar.

Lemma 3 ([6, Lemma 1]). *Let M be a non-negative integer constant greater or equal to the maximum constant occurring in the time constraints of the TA. Let $\equiv_M$ be the relation defined as $w \equiv_M w'$ iff $\forall x \in X$: either $w(x) = w'(x)$ or $(w(x) > M$ and $w'(x) > M)$. The relation $\mathcal{R} = \{((l, w), (l, w'))|w \equiv_M w'\}$ is a bisimulation relation.*

Lemma 4. *For all parameter valuation v, non-negative integer constants M, clocks x and valuation set C, $v(\mathsf{Ext}_x^M(C)) = \mathsf{Ext}_x^M(v(C))$.*

We use the bounds on parameters to compute the maximum constant M appearing in all the guards and invariants of the PTA. When those constraints are parametric expressions, we compute the maximum value that the expression can take for all the bounded parameter values (it is unique since expressions are linear): *e.g.,* if a guard is $x \leq 2p_1 - p_2 + 1$ and $p_1 \in [2, 5]$, and $p_2 \in [3, 4]$ then the maximum constant corresponding to this constraint is $2 \times 5 - 3 + 1 = 8$.

Also note that bounding the parameter domain of PTA is not a strong restriction in practice – especially since the bounds can be arbitrarily large.

Lemmas 5 and 6 are instrumental in proving the preservation of all correct integer parameter values in the algorithms of Section 4, while Lemma 7 is the key to proving their termination.

Lemma 5. *Let $\mathcal{A}$ be a PTA, s be a symbolic state of $\mathcal{A}$, and M a non-negative integer constant greater than the maximal constant occurring in the PTA (including the bounds of parameters). Let x be a clock, v be a parameter valuation, and $(l, w) \in v(\mathsf{Ext}_x^M(s)))$ be a concrete state. There exists a state $(l, w') \in v(s)$ such that (l, w) and (l, w') are bisimilar.*

Extrapolation and Integer Hulls. Here, for the sake of simplicity, and similarly to [14], we consider that all polyhedra are topologically closed and, to avoid confusion, we equivalently (provided that M is (strictly) greater than the maximal constant in the PTA) define $\mathsf{Ext}_x^M(s)$ as $(s \cap (x \leq M)) \cup \mathsf{Cyl}_x(s \cap (x \geq M)) \cap (x \geq M)$.

Lemma 6. *For all integer parameter valuations v, all non-negative integer constants M, and all reachable symbolic states $s = (l, C)$, $v(\mathsf{IH}(\mathsf{Ext}_X^M(C))) = v(\mathsf{Ext}_X^M(C))$.*

Lemma 7. *In a bounded PTA, the set of constraints $\mathsf{IH}(\mathsf{Ext}_X^M(C))$ over the set of symbolic reachable states (l, C) is finite.*

4 Integer-Complete Dense Parametric Algorithms

In this section, we describe two parameter synthesis algorithms that always terminate for *bounded* PTA, and return not only all the integer points solution of the problem (à la [14]) but also all real-valued points in between integer points; that is, these algorithms return a list of convex combinations of integer points, and all rational-valued points contained in each such convex combination are also solution of the problem.

4.1 Parametric Reachability: RIEF

The goal of RIEF given in Algorithm 1 ("R" stands for robust, and "I" for integer hull) is to synthesize parameter valuations solution to the EF-synthesis problem, *i.e.,* the valuations for which there exists a run eventually reaching a location in G. It is inspired by the algorithms EF and IEF introduced in [14] that both address the same problem; however EF does not terminate in general, and IEF can only output integer valuations. In fact, if we replace all occurrences of $\mathsf{IH}(C)$ in Algorithm RIEF by C, we obtain Algorithm EF from [14]. RIEF proceeds as a post-order traversal of the symbolic reachability tree, and collects all parametric constraints associated with the target locations G. In contrast to EF, it stores in S the *integer hulls* of the states, which ensures termination due to the finite number of possible integer hulls of k-extrapolations; however,

Algorithm 1. $\mathsf{RIEF}(\mathcal{A}, s, G, S)$

input : A PTA $\mathcal{A}$, a symbolic state $s = (l, C)$, a set of target locations G, a set S of passed states on the current path
output: Constraint K over the parameters

```
1 if l ∈ G then K ← C↓_P ;
2 else
3     K ← ⊥;
4     if IH(Ext^M_X(s)) ∉ S then
5         for each outgoing e from l in A do
6             K ← K ∪ RIEF(A, Succ(s, e), G, S ∪ {IH(Ext^M_X(s))});
```

in contrast to IEF, RIEF returns the actual states (instead of their integer hull), which yields a larger result than IEF.

As a direct consequence of Lemma 7, it is clear that RIEF explore only a finite number a symbolic states. Therefore, we have the following theorem:

Theorem 1. *For any bounded PTA $\mathcal{A}$, the computation of* $\mathsf{RIEF}(\mathcal{A}, \mathsf{Init}(\mathcal{A}), G, \emptyset)$ *terminates.*

Theorem 2. *Upon termination of* RIEF*, we have:*

1. *Soundness: If $v \in \mathsf{RIEF}(\mathcal{A}, \mathsf{Init}(\mathcal{A}), G, \emptyset)$ then G is reachable in $v(\mathcal{A})$;*
2. *Integer completeness: If v is an integer parameter valuation, and G is reachable in $v(\mathcal{A})$ then $v \in \mathsf{RIEF}(\mathcal{A}, \mathsf{Init}(\mathcal{A}), G, \emptyset)$.*

Example 1. Consider the simple PTA with a unique transition from the initial location l_0 to l_1 with guard $1 \leq x \leq 2a$. To ensure the $EF\{l_1\}$ property, we just need to be able to go through the transition from l_0 to l_1. The parametric zone C_1 obtained in l_1 is $1 \leq x \wedge 1 \leq 2a$, which implies $a \geq \frac{1}{2}$. The integer hull of C_1 is $1 \leq x \wedge 1 \leq a$, which implies $a \geq 1$.

Algorithm IEF gives the result $a \geq 1 \wedge a \in \mathbb{N}$, while algorithm RIEF gives (here) the exact result $a \geq \frac{1}{2}$.

4.2 Parametric Unavoidability: RIAF

RIAF (given in Algorithm 2) synthesizes parameter valuations solution to the AF-synthesis problem. It is inspired by the algorithms AF and IAF introduced in [14]; however AF may not terminate, and IAF can only output integer valuations. Note also, as shown in Example 2 below, that interpreting the result of IAF as a dense set is incorrect in general, since it may contain non-integer values that do not ensure unavoidability. RIAF works as a post-order traversal of the symbolic reachability tree, keeping valuations that permit to go into branches reaching G and cutting off branches leading to a deadlock or looping without any occurrence of G. More precisely, RIAF uses three sets of valuations: *i)* K_{Good} contains the

Algorithm 2. $\mathsf{RIAF}(\mathcal{A}, s, G, S)$

input : A PTA $\mathcal{A}$, a symbolic state (l, C), a set of target locations G, a set S of passed states on the current path
output: Constraint K over the parameters

1 **if** $l \in G$ **then** $K \leftarrow C{\downarrow_P}$;
2 **else**
3 **if** $(l, \mathsf{IH}(Ext_X^M(C))) \in S$ **then** $K \leftarrow \bot$;
4 **else**
5 $K \leftarrow \top$; $K_{Live} \leftarrow \bot$;
6 **for** *each outgoing* $e = (l, g, a, R, l')$ *from* l *in* $\mathcal{A}$ **do**
7 $S' \leftarrow \mathsf{Succ}((l, C), e)$;
8 $K_{Good} \leftarrow \mathsf{RIAF}(\mathcal{A}, S', G, S \cup \{(l, \mathsf{IH}(\mathsf{Ext}_X^M(C)))\})$;
9 $K_{Block} \leftarrow \top \setminus S'{\downarrow_P}$;
10 $K \leftarrow K \cap (K_{Good} \cup K_{Block})$;
11 $K_{Live} \leftarrow K_{Live} \cup (C \cap g)^{\swarrow}$;
12 $K \leftarrow K \setminus (\mathbb{R}^{X \cup P} \setminus K_{Live}){\downarrow_P}$;

set of valuations that indeed satisfy AF, recursively computed by calling RIAF; *ii*) K_{Block} allows to cut off branches leading to deadlock or looping, by keeping only valuations in the complement of the first state in that branch; *iii*) K_{Live} is necessary to forbid reaching states from which no transition can be taken for any e, even after some delay.

The main difference between AF and RIAF is that we use the convergence condition of IAF, which operates on integer hulls instead of symbolic states, hence ensuring termination with the same reasoning as RIEF.

We state below the soundness and integer completeness of RIAF. The proofs are easily adapted from those of AF in [14], by using the additional arguments provided in the proof of RIEF, and in particular Lemma 7.

Theorem 3. *Upon termination of* RIAF*, we have:*

1. *Soundness: If* $v \in \mathsf{RIAF}(\mathcal{A}, \mathsf{Init}(\mathcal{A}), G, \emptyset)$ *then* G *is inevitable in* $v(\mathcal{A})$*;*
2. *Integer completeness: If* v *is an integer parameter valuation, and* G *is inevitable in* $v(\mathcal{A})$ *then* $v \in \mathsf{RIAF}(\mathcal{A}, \mathsf{Init}(\mathcal{A}), G, \emptyset)$.

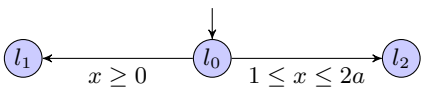


Fig. 3. Counter-example to the density of the result of IAF.

Example 2. Consider the PTA in Fig. 3. To ensure the $AF\{l_1\}$ property, we need to cut the transition from l_0 to l_2. The parametric zone C_2 obtained in l_2 is $1 \leq x \wedge 1 \leq 2a$, which implies $a \geq \frac{1}{2}$. The integer hull of C_2 is $1 \leq x \wedge 1 \leq a$, which implies $a \geq 1$. In order to block the path to l_2 in l_0, we intersect with the

complement of projection on parameters of $\mathsf{IH}(C_2)$, *i.e.,* $a < 1$. Since there is no constraint on the transition from l_0 to l_1 the final result of our former algorithm IAF is actually $a < 1$. For integer parameters this means $a = 0$, which is correct. But if we interpret the result for real parameters, we obtain that, for instance, $a = \frac{1}{2}$ should be a valuation ensuring the property, while it is obviously not.

On the same example, RIAF gives (here) the exact result $a < \frac{1}{2}$.

4.3 Implementation in Romo

The algorithms have been implemented in the tool Romo [15]; polyhedra operations (both convex and non-convex) are handled by the PPL library [5]. To illustrate this, we refer the reader to the scheduling example of [14]. It consists in three tasks τ_1, τ_2, τ_3 scheduled using static priorities ($\tau_1 > \tau_2 > \tau_3$) in a non-preemptive manner. Task τ_1 is periodic with period a and a non-deterministic duration in $[10, b]$, where a and b are parameters. Task τ_2 only has a minimal activation time of $2a$ and has a non-deterministic duration in $[18, 28]$ and finally τ_3 is periodic with period $3a$ and a non-deterministic duration in $[20, 28]$. Each task is subject to a deadline equal to its period so that it must only have one instance active at all times. We ask for the parameter values that ensure that the system does not reach a deadline violation.[2] Algorithm IEF produces the constraint $a \geq 34, b \geq 10, a - b \geq 24$ in 7.4 s on a Core i7/Linux computer, while algorithm RIEF produces the constraint $a > \frac{562}{17}, b \geq 10, a - b > \frac{392}{17}$ in 12.7 s.

As illustrated here, the results are indeed a bit more precise but the main improvement is of course the guaranteed density of the result. Also, RIEF is generally slower than IEF and profiling shows that this is due to a decreased efficiency in computing the integer hull: we start each time from the whole symbolic state instead of starting from the successor of an already computed integer hull. This could maybe be mitigated using a cache for the constraints generated in computing the integer hulls.

5 Conclusion

Summary. We introduced here an extrapolation for symbolic states that contains not only clocks but also parameters. We then proposed algorithms that always terminate for PTA with bounded parameters, and output symbolic constraints that define dense sets of parameter valuations that are guaranteed to be correct and containing at least all integer points.

Synthesizing not only the integer points but also the real-valued points is of utmost importance for the robustness or implementability of the system. In fact, one can even consider any degree of precision instead of integers (*e.g.,* a degree of precision of $\frac{1}{10}$) by appropriately resizing the constants of the PTA (*e.g.,* by multiplying all constants and all parameter bounds by 10). This makes possible the synthesis of an underapproximated result arbitrarily close to the actual solution.

[2] The result is therefore the complement of the result given by IEF and therefore an over-approximation containing no incorrect integer value.

Future Works. We proposed a first attempt to define a k-extrapolation for PTA; this can serve as a basis for further developments, *e.g.,* using better extrapolation operators such as L/U, local-L/U or local-diagonal-L/U abstractions. The approach we propose is fairly generic and could probably be adapted to more complex properties, expressed in LTL or CTL and their parametric variants. Moreover, we would like to extend in a similar manner the inverse method proposed in [3], hence ensuring termination of this algorithm with an almost-complete result. Furthermore, we use here the integer hull as an underapproximation of the result; in contrast, we could use an overapproximation using a notion (yet to be defined) of "external integer hull", and then combine both hulls to obtain two sets of "good" and "bad" parameter valuations separated by an arbitrarily small set of unknown valuations.

Acknowledgment. We would like to thank anonymous reviewers for their useful comments, especially for a meaningful remark on a preliminary version of this paper together with the suggestion of the example in Fig. 1.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994)
2. Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: STOC, pp. 592–601. ACM (1993)
3. André, É., Chatain, T., Encrenaz, E., Fribourg, L.: An inverse method for parametric timed automata. International Journal of Foundations of Computer Science **20**(5), 819–836 (2009)
4. André, É., Markey, N.: Language preservation problems in parametric timed automata. In: Sankaranarayanan, S., Vicario, E. (eds.) FORMATS 2015. LNCS, vol. 9268, pp. 27–43. Springer, Heidelberg (2015)
5. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming **72**(1–2), 3–21 (2008)
6. Behrmann, G., Bouyer, P., Larsen, K.G., Pelánek, R.: Lower and upper bounds in zone-based abstractions of timed automata. STTT **8**(3), 204–215 (2006)
7. Beneš, N., Bezděk, P., Larsen, K.G., Srba, J.: Language emptiness of continuous-time parametric timed automata. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 69–81. Springer, Heidelberg (2015)
8. Bozzelli, L., La Torre, S.: Decision problems for lower/upper bound parametric timed automata. Formal Methods in System Design **35**(2), 121–151 (2009)
9. Bruyère, V., Raskin, J.-F.: Real-time model-checking: Parameters everywhere. Logical Methods in Computer Science **3**(1:7) (2007)
10. Bundala, D., Ouaknine, J.: Advances in parametric real-time reasoning. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 123–134. Springer, Heidelberg (2014)
11. Doyen, L.: Robust parametric reachability for timed automata. Information Processing Letters **102**(5), 208–213 (2007)

12. Halbwachs, N., Proy, Y., Raymond, P.: Veriffcation of linear hybrid systems by means of convex approximations. In: LeCharlier, B. (ed.) SAS 1994. LNCS, vol. 864. Springer, Heidelberg (1994)
13. Hune, T., Romijn, J., Stoelinga, M., Vaandrager, F.W.: Linear parametric model checking of timed automata. Journal of Logic and Algebraic Programming **52–53**, 183–220 (2002)
14. Jovanović, A., Lime, D., Roux, O.H.: Integer parameter synthesis for real-time systems. IEEE Transactions on Software Engineering **41**(5), 445–461 (2015)
15. Lime, D., Roux, O.H., Seidner, C., Traonouez, L.-M.: Romeo: a parametric model-checker for petri nets with stopwatches. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 54–57. Springer, Heidelberg (2009)
16. Markey, N.: Robustness in real-time systems. In: SIES, pp. 28–34. IEEE Computer Society Press (2011)
17. Miller, J.S.: Decidability and complexity results for timed automata and semi-linear hybrid automata. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 296–309. Springer, Heidelberg (2000)
18. Wang, F.: Parametric timing analysis for real-time systems. Information and Computation **130**(2), 131–150 (1996)

Polynomial Interrupt Timed Automata

Béatrice Bérard[1,4], Serge Haddad[2,4,5(✉)], Claudine Picaronny[2,4,5], Mohab Safey El Din[1,4,5], and Mathieu Sassolas[3]

[1] Sorbonne Université, Université P. & M. Curie, LIP6, ParaSol Project, UMR 7606, Paris, France
[2] École Normale Supérieure de Cachan, LSV, UMR 8643, INRIA, Cachan, France
[3] Université Paris-Est, LACL, Créteil, France
[4] CNRS, Paris, France
[5] INRIA, Paris-Rocquencourt Center, PolSys Project, Paris, France
haddad@lsv.ems-cachan.fr

Abstract. Interrupt Timed Automata (ITA) form a subclass of stopwatch automata where reachability and some variants of timed model checking are decidable even in presence of parameters. They are well suited to model and analyze real-time operating systems. Here we extend ITA with polynomial guards and updates, leading to the class of polynomial ITA (POLITA). We prove that reachability is decidable in 2EXPTIME on POLITA, using an adaptation of the *cylindrical decomposition* method for the first-order theory of reals. Compared to previous approaches, our procedure handles parameters and clocks in a unified way. We also obtain decidability for the model checking of a timed version of CTL and for reachability in several extensions of POLITA.

1 Introduction

Hybrid Automata. Hybrid systems [14] combine continuous evolution of variables according to flow functions (described by differential inclusions) in control nodes, and discrete jumps between these nodes, where the variables can be tested by guards and updated. This class of models is very expressive and all relevant verification questions (*e.g.* reachability) are undecidable. For the last twenty years, a large amount of research was devoted to identifying subclasses with decidable properties, by restricting the continuous dynamics and/or the discrete behavior of the systems. Among these classes lie the well known Timed Automata (TA) [3], where all variables are *clocks* evolving with rate 1 w.r.t. to global time, guards are comparisons of clocks with rational constants, and updates are resets. It is proved in [15] that reachability becomes undecidable when adding one stopwatch, *i.e.*, a clock whose rate is either 0 or 1 depending on the state, to timed automata. Decidability results were also obtained for larger classes (see [2,4,5,15,17]), usually by building from the associated transition system (with uncountable state space) a finite abstraction preserving a specific class of properties, like reachability or those expressed by temporal logic formulas. In all these abstractions, a state is a pair composed of a control node and a polyhedron of variable values [15,17].

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 20–32, 2015.
DOI: 10.1007/978-3-319-24537-9_3

Interrupt Timed Automata. The class of Interrupt Timed Automata (ITA), incomparable with TA, was introduced in [8,10] as another subclass of hybrid automata with a (time-abstract) bisimulation providing a finite quotient, thus leading to decidability of reachability and some variants of timed model checking. In a basic n-dimensional ITA, control nodes are organized along n levels, with n stopwatches (also called clocks hereafter), one per level. At a given level, the associated clock is active, while clocks from lower levels are frozen and clocks from higher levels are irrelevant. Guards are linear constraints and the clocks can be updated by linear expressions (using only clocks from lower levels). The hierarchical structure of ITA makes them particularly well suited for modeling systems with interruptions, like real-time operating systems. ITA were extended with parameters in [9], while preserving decidability.

Contribution. We define the class PolITA, of polynomial ITA, where linear expressions on clocks are replaced by polynomials with rational coefficients both for guards and updates. For instance, a guard at level 2 with clock x_2 can be of the form $P_1(x_1)x_2^2 + P_2(x_1) \geq 0$, where P_1 and P_2 are polynomials with single variable x_1, the clock of level 1. Thus, guards are more expressive than in the whole class of linear hybrid automata and classical polyhedron-based abstractions [1,12] are not sufficient to deal with these constraints. Since linear constraints are not always sufficient for modeling purposes, such guards can be useful. In addition, such guards can simulate irrational (algebraic) constraints, a case that becomes undecidable in the setting of timed automata [19]. Similar polynomials of variables for programs were considered in [20], although in an untimed setting.

We establish that reachability is decidable in 2EXPTIME for PolITA by adapting the cylindrical decomposition [6,13] related to the first order theory of reals. Observe however that not any decision procedure would be appropriate for our goal. Indeed this decomposition produces a finite partition of the state space, which is the basis for the construction of a finite bisimulation quotient. The first order theory of reals has already been used in several works on hybrid automata [4,17] but it was restricted to the dynamical part, with discrete jumps that must reinitialize the variables. Our adaptation consists in an on-the-fly construction avoiding to build the whole decomposition.

The construction can also be adapted to model checking of a timed extension of CTL. From an expressiveness point of view, we show that (contrary to ITA) PolITA are incomparable with stopwatch automata (SWA). We also prove that the decidability result still holds with several extensions: adding auxiliary clocks and parameters, and enriching the possible updates. In particular, parametric ITA [9] can be seen as a subclass of PolITA, and the complexity of our reachability algorithm is better than [9] (2EXPSPACE).

Outline. We describe the model of polynomial ITA in Section 2, with an example and the presentation of the verification problems. In Section 3 we informally present the cylindrical decomposition and the decision procedures for PolITA. Then in section 4, we detail these constructions with a special focus on the data structures and algorithmic schemes. Finally, we discuss expressiveness, describe extensions and conclude in Section 5. All missing proofs and constructions can be found in [11].

2 Polynomial ITA

We denote respectively by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ the sets of natural numbers, integers, rational and real numbers, with $\mathbb{R}_{\geq 0}$ for the set of non negative real numbers. Let $X = \{x_1, \ldots, x_n\}$ be a finite set of n variables called clocks. We write $\mathbb{Q}[x_1, \ldots, x_n]$ for the set of polynomials with n variables and rational coefficients.

A *polynomial constraint* is a conjunction of constraints of the form $P \bowtie 0$ where $P \in \mathbb{Q}[x_1, \ldots, x_n]$ and $\bowtie \in \{<, \leq, =, \geq, >\}$, and we denote by $\mathcal{C}(X)$ the set of polynomial constraints. We also define $\mathcal{U}(X)$, the set of *polynomial updates* over X, by: $\mathcal{U}(X) = \{\wedge_{x \in X} x := P_x \mid \forall x \ P_x \in \mathbb{Q}[x_1, \ldots, x_n]\}$.

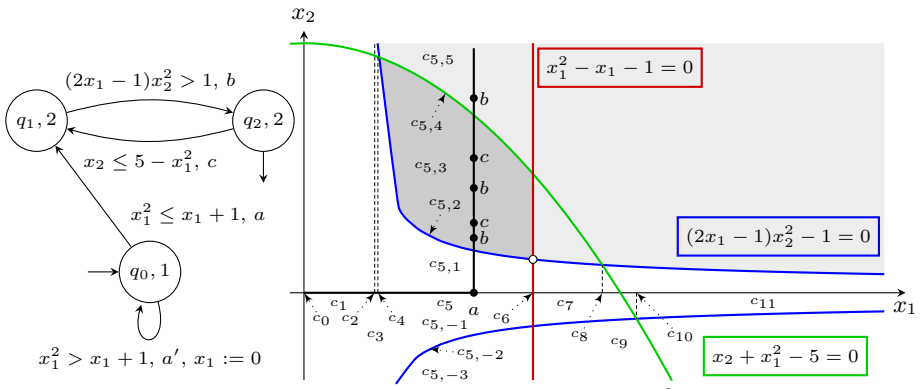
A valuation for X is a mapping $v \in \mathbb{R}^X$, also identified to the n-dimensional vector $(v(x_1), \ldots, v(x_n)) \in \mathbb{R}^n$. The valuation where $v(x) = 0$ for all $x \in X$ is denoted by $\mathbf{0}$. For $P \in \mathbb{Q}[x_1, \ldots, x_n]$ and v a valuation, the value of P at v is $P(v) = P(v(x_1), \ldots, v(x_n))$. A valuation v satisfies the constraint $P \bowtie 0$, written $v \models P \bowtie 0$, if $P(v) \bowtie 0$. The notation is extended to a polynomial constraint: $v \models \varphi$ with $\varphi = \bigwedge_i P_i \bowtie_i 0$ if $v \models P_i \bowtie_i 0$ for every i.

An update of valuation v by $u = \wedge_{x \in X} x := P_x$ in $\mathcal{U}(X)$ is the valuation $v[u]$ defined by $v[u](x) = P_x(v)$ for each $x \in X$. Hence an update is atomic in the sense that all variables are assigned simultaneously. For valuation v, delay $d \in \mathbb{R}_{\geq 0}$ and $k \in [1..n]$, the valuation $v' = v +_k d$, corresponding to *time elapsing of d for x_k*, is defined by $v'(x_k) = v(x_k) + d$ and $v'(x) = v(x)$ for $x \neq x_k$.

Definition 1 (PolITA). *A* polynomial interrupt timed automaton *(*PolITA*) is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, F, X, \lambda, \Delta \rangle$, where:*

- *Σ is a finite alphabet, with ε the empty word in Σ^*, the set of words over Σ;*
- *Q is a finite set of states, q_0 is the initial state, $F \subseteq Q$ is the set of final states;*
- *$X = \{x_1, \ldots, x_n\}$ consists of n interrupt clocks;*
- *the mapping $\lambda : Q \to \{1, \ldots, n\}$ associates with each state its level and $x_{\lambda(q)}$ is called the* active clock *in state q;*
- *$\Delta \subseteq Q \times \mathcal{C}(X) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{U}(X) \times Q$ is the set of transitions. Let $q \xrightarrow{\varphi, a, u} q'$ in Δ be a transition with $k = \lambda(q)$ and $k' = \lambda(q')$. The guard φ is a conjunction of constraints $P \bowtie 0$ with $P \in \mathbb{Q}[x_1, \ldots, x_k]$ (P is a polynomial over clocks from levels less than or equal to k). The update u is of the form $\wedge_{i=1}^{n} x_i := C_i$ with:*
  - *if $k > k'$, i.e. the transition decreases the level, then for $1 \leq i \leq k'$, $C_i = x_i$ and for $i > k'$, $C_i = 0$;*
  - *if $k \leq k'$ then for $1 \leq i < k$, $C_i = x_i$, $C_k = P$ for some $P \in \mathbb{Q}[x_1, \ldots, x_{k-1}]$ or $C_k = x_k$, and for $i > k$, $C_i = 0$.*

Example 1. PolITA $\mathcal{A}_0$ of Fig. 1a has alphabet $\{a, a', b, c\}$, two levels, with q_0 at level 1 and q_1, q_2 at level 2. The single final state is q_2. At level 1, only x_1 appears in guards and updates (here the only update is the reset of x_1 by action a'), while at level 2 guards use polynomials in both x_1 and x_2. In the sequel, the polynomials of $\mathcal{A}_0$ are denoted by $A = x_1^2 - x_1 - 1$, $B = (2x_1 - 1)x_2^2 - 1$ and $C = x_2 + x_1^2 - 5$.



(a) A sample PolITA $\mathcal{A}_0$. **(b)** Sample trajectory of $\mathcal{A}_0$ in $\mathbb{R}^2$. The axes are not orthonormal.

Fig. 1. A PolITA and an example of a trajectory.

A *configuration* (q, v) of $\mathcal{A}$ consists of a state q and a clock valuation v.

Definition 2. *The semantics of a* PolITA $\mathcal{A}$ *is defined by the (timed) transition system* $\mathcal{T}_\mathcal{A} = (S, s_0, \to)$*, where* $S = \{(q, v) \mid q \in Q,\ v \in \mathbb{R}^X\}$ *is the set of configurations, with initial configuration* $s_0 = (q_0, \mathbf{0})$*. The relation* $\to$ *on* S *consists of two types of steps:*
Time steps: *Only the active clock in a state can evolve, all other clocks are frozen. For a state* q *with active clock* $x_{\lambda(q)}$*, a time step of duration* $d \in \mathbb{R}_{\ge 0}$ *is defined by* $(q, v) \xrightarrow{d} (q, v')$ *with* $v' = v +_{\lambda(q)} d$*.*
Discrete steps: *There is a discrete step* $(q, v) \xrightarrow{a} (q', v')$ *if there exists a transition* $q \xrightarrow{\varphi, a, u} q'$ *in* Δ *such that* $v \models \varphi$ *and* $v' = v[u]$*.*

A run of a PolITA $\mathcal{A}$ is a path in the graph $\mathcal{T}_\mathcal{A}$ alternating time and discrete steps. For a given run ρ, the *trace* of ρ is the sequence of letters (or word) appearing in the path and the *timed word* of ρ is the sequence of letters along with the absolute time of the occurrence, *i.e.* the sum of all delays appearing before the letter. A run is accepting if it ends in a state of F. The *language* (resp. *timed language*) of $\mathcal{A}$ is the set of traces (resp. timed words) of accepting runs.

Example 2. In $\mathcal{A}_0$, the transition from q_0 to q_1 can only be fired before (or when) x_1 reaches $\frac{1+\sqrt{5}}{2}$, *i.e.* at the point labeled c_6 on Fig. 1b. Then, transition b from q_1 to q_2 can only be taken once x_2 reaches the grey areas. Transition c cannot be taken once the green curve has been crossed. Hence the loop bc can occur as long as the clock values remain in the dark gray area $c_{5,3}$, or on the green curve $c_{5,4}$. In the sequel, we show how to symbolically compute these sets, called *cells*. Since $q_2 \in F$, the run depicted in Fig. 1b is accepted by $\mathcal{A}$. The associated timed word (resp. trace) is $(a, 1.2)(b, 2.3)(c, 2.6)(b, 3.3)(c, 3.9)(b, 5.1)$ (resp. *abcbcb*).

Given a PoLITA $\mathcal{A}$, the *reachability problem* asks, given a state q, whether there exists a valuation v and a path from $(q_0, \mathbf{0})$ to (q, v) in $\mathcal{T}_\mathcal{A}$.

The reachability procedure given in Section 3 relies on a finite abstraction of $\mathcal{T}_\mathcal{A}$. This abstraction needs to be refined enough to capture time elapsing, discrete jumps through the crossing of a transition, and keep constant the truth value of constraints $P \bowtie 0$. In the resulting model, a state will consist of an automaton state coupled with a *cell* of an appropriate *cylindrical decomposition*.

3 Cylindrical Decomposition and Reachability

3.1 Definition

The *cylindrical decomposition* is the basis of the first elementary decision procedure (more precisely 2EXPTIME) for the satisfiability of the first-order logic over reals [13][1]. A cylindrical decomposition of $\mathbb{R}^n$ consists of finite partitions of $\mathbb{R}, \mathbb{R}^2, \ldots, \mathbb{R}^n$ into *cells* such that the cells for $\mathbb{R}$ are open intervals or points and cells of $\mathbb{R}^{k+1}$ are obtained by lifting cells of $\mathbb{R}^k$ on the $k+1^{th}$ axis and then partitioning this axis with intervals and points in a "similar" way for all the points of the original cell.

Example 3. Fig. 1b partly depicts a cylindrical decomposition of $\mathbb{R}^2$. The cells of $\mathbb{R}_{\geq 0}$ are denoted by $c_0, \ldots, c_{11}$ (those of the negative part of the x_1 axis are not represented). The lifting of cell c_5 is $c_5 \times \mathbb{R}$ and is partitioned into cells $c_{5,-3}, c_{5,-2}, \ldots, c_{5,5}$. Given any $z \in c_5$, $\{z\} \times \mathbb{R}$ is partitioned in an open interval $c_{5,-3} \cap \{z\} \times \mathbb{R}$ followed by a point $c_{5,-2} \cap \{z\} \times \mathbb{R}$, etc. Observe that the mapping $z \mapsto c_{5,-2} \cap \{z\} \times \mathbb{R}$ is continuous.

Definition 3. *A cell of level k is a subset of $\mathbb{R}^k$ inductively defined as follows.*

- *When $k = 1$, it is either a point or an open interval.*
- *A cell C of level $k+1$ is based on a cell C' of level k. It has one of the following shapes.*
  1. *$C = \{(x, f(x)) \mid x \in C'\}$ with f a continuous function from C' to $\mathbb{R}$;*
  2. *$C = \{(x, y) \mid x \in C' \wedge l(x) < y < u(x)\}$ with $l < u$ continuous functions from C' to $\mathbb{R}$, possibly with $l = -\infty$ and/or $u = +\infty$.*

We are interested in a cylindrical decomposition *adapted* to finite families of polynomials $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ with $\mathcal{P}_k \subseteq \mathbb{Q}[x_1, \ldots, x_k]$: in a cell of level k, the sign $(-, 0, +)$ of each polynomial in $\mathcal{P}_k$ is constant. Due to the definition of cells, a cylindrical decomposition is appropriately represented by a tree.

Definition 4. *A cylindrical decomposition of $\mathbb{R}^n$ adapted to $\mathcal{P} = \{\mathcal{P}_k\}_{k \leq n}$ such that $\mathcal{P}_k \subseteq \mathbb{Q}[x_1, \ldots, x_k]$, is a tree of cells inductively defined as follows:*

- *The root of the tree is the only cell of level 0, that is $\mathbb{R}^0$;*

[1] Later on, an EXPSPACE procedure was proposed in [7].

– *Let C be a cell of level $k < n$ in the tree. There exists some $r \in \mathbb{N}$ and continuous functions f_i, for $1 \leq i \leq r$, with $-\infty = f_0 < f_1 < \ldots < f_r < f_{r+1} = +\infty$, such that the (ordered) children of C at level $k+1$ in the tree are the cells $C_0 = \{(x,y) \mid x \in C \wedge f_0(x) < y < f_1(x)\}$, $C_1 = \{(x, f_1(x)) \mid x \in C\}$, $C_2 = \{(x,y) \mid x \in C \wedge f_1(x) < y < f_2(x)\}$, $\ldots$, $C_{2r} = \{(x,y) \mid x \in C \wedge f_r(x) < y < f_{r+1}(x)\}$.*
For all $P \in \mathcal{P}_{k+1}$, for all $i \in \{0, \ldots, 2r\}$, for all $z, z' \in C_i$, $sign(P(z)) = sign(P(z'))$.

Example 4. For the PolITA of Fig. 1a, the relevant polynomials in $\mathbb{Q}[x_1]$ are those related to level 1: the clock x_1 itself and the polynomial $A = x_1^2 - x_1 - 1$ used in both guards from q_0, hence $\mathcal{P}_1 = \{x_1, A\}$. The relevant polynomials in $\mathbb{Q}[x_1, x_2]$ are those from level 2: x_2 and $B = (2x_1 - 1)x_2^2 - 1, C = x_2 + x_1^2 - 5$ associated with the guards from q_1 and q_2, so $\mathcal{P}_2 = \{x_2, B, C\}$. For the cells of level 1, c_4, c_8, c_{10} correspond to intersection points of graphs $B = 0$ and $C = 0$ projected on the x_1 axis, while c_2 corresponds to $\frac{1}{2}$, the root of the coefficient $2x_1 - 1$ of B. Other cells like c_1, c_3 correspond to intervals between roots. In cell $c_{5,3}$ of level 2, the guards of the transitions between q_1 and q_2 are satisfied.

The main elements for the effective construction of a cylindrical decomposition are given in Section 4. For the moment, we recall the result of [13]:

Theorem 1 ([13]). *For any family $\mathcal{P} = \{\mathcal{P}_k\}_{k \leq n}$ such that $\mathcal{P}_k$ is a finite subset of $\mathbb{Q}[x_1, \ldots, x_k]$, one can build a cylindrical decomposition of $\mathbb{R}^n$ adapted to $\mathcal{P}$ in 2EXPTIME, more precisely in $(|\mathcal{P}| \cdot d)^{2^{O(n)}}$ where d is the maximal degree of a polynomial of $\mathcal{P}$.*

3.2 Reachability for PolITA

We now use this decomposition to build a finite abstraction of the set of configurations of a PolITA, which leads to the decidability of the reachability problem.

Theorem 2. *Reachability for* PolITA *is decidable in time $(d|\mathcal{A}|)^{2^{O(n)}}$ where n is the number of clocks in $\mathcal{A}$ and d the maximal degree of polynomials appearing in $\mathcal{A}$; thus in polynomial time when the number of clocks is fixed.*

Let $\mathcal{A} = \langle \Sigma, Q, q_0, F, X, \lambda, \Delta \rangle$ be a PolITA with $X = \{x_1, \ldots, x_n\}$. We define *Poly*$(\mathcal{A})$ as the set of all polynomials appearing in guards and updates of $\mathcal{A}$ (including all clocks) as follows: P belongs to *Poly*$(\mathcal{A})$ iff (1) P is a clock, (2) P occurs in a guard $P \bowtie 0$, or (3) $P = x_i - P_i$ where $x_i := P_i$ is an update.

We denote by $\mathcal{D}_\mathcal{A}$ a cylindrical decomposition adapted to *Poly*$(\mathcal{A})$, with $\mathcal{D}^1_\mathcal{A}, \ldots, \mathcal{D}^n_\mathcal{A}$ for the set of cells at the respective levels $1, \ldots, n$ so that for $1 \leq k \leq n$, $\mathcal{D}^k_\mathcal{A}$ is a decomposition of $\mathbb{R}^{\{x_1, \ldots, x_k\}}$.

We define a finite transition system $\mathcal{R}_\mathcal{A}$ with states in $Q \times \mathcal{D}_\mathcal{A}$. The states can also be partitioned according to levels as $\bigcup_{k=1}^n \lambda^{-1}(k) \times \mathcal{D}^k_\mathcal{A}$. Indeed, given a configuration (q, v) with $\lambda(q) = k$, the clocks of level $i > k$ are irrelevant and

so v can be identified as a point in $\mathbb{R}^{\{x_1,\ldots,x_k\}}$. We now define the transitions of $\mathcal{R}_\mathcal{A}$ as follows.

Time Successors. Let $succ \notin \Sigma$ be a letter representing time elapsing. Let (q, C) be a state of $\mathcal{R}_\mathcal{A}$, with $\lambda(q) = k$, and let $\underline{C} \in \mathcal{D}_\mathcal{A}^{k-1}$ be the projection of C onto $\mathbb{R}^{k-1}$ and $-\infty = f_0 < \cdots < f_{r+1} = +\infty$ be the functions dividing $\underline{C}$ as in Definition 4. The *succ* transitions are defined as follows:

- if $C = \{(x, f_i(x)) \mid x \in \underline{C}\}$ for some $i \in \{1, \ldots, r\}$, then there is a transition $(q, C) \xrightarrow{succ} (q, C')$ where $C' = \{(x, y) \mid x \in \underline{C}, f_i(x) < y < f_{i+1}(x)\}$;
- if $C = \{(x, y) \mid x \in \underline{C}, f_{i-1}(x) < y < f_i(x)\}$ for some $i \in \{1, \ldots, r\}$, then there is a transition $(q, C) \xrightarrow{succ} (q, C')$ where $C' = \{(x, f_i(x)) \mid x \in \underline{C}\}$;
- otherwise, $C = \{(x, y) \mid x \in \underline{C}, f_r(x) < y < f_{r+1}(x)\}$, and there is a self-loop labeled by *succ*: $(q, C) \xrightarrow{succ} (q, C)$.

In all the above cases, C' is called the *time successor* of C (in the last case, C is its own time successor). Since the decomposition is *cylindrical*, time elapsing according to the current clock corresponds to moving to the "next" cell.

Proposition 1 (Correctness w.r.t. Time Elapsing). *Let v be a valuation belonging to a cell C of level k.*

- *There exists $d > 0$ such that the elapsing of d time units for x_k yields a valuation $v +_k d \in C'$, the time successor of C.*
- *For any $0 < d' < d$, the elapsing of d' time units for x_k yields a valuation $v +_k d$ that is either in C or in C'.*

Discrete Successors. Since $\mathcal{D}_\mathcal{A}$ is adapted to $Poly(\mathcal{A})$ which contains all guards and updates we can write $C \models \varphi$ whenever $v \models \varphi$ for some $v \in C$ and $C[u]$ for the unique cell $C' \in \mathcal{D}_\mathcal{A}^k$ such that for any valuation $v \in C$, $v[u] \in C'$. Discrete transitions of $\mathcal{A}$ are translated as follows into $\mathcal{R}_\mathcal{A}$: if $(q, \varphi, a, u, q') \in \Delta$ and $C \models \varphi$, there is a transition $(q, C) \xrightarrow{a} (q', C[u])$. Since the decomposition provides *sign-invariant* cells with respect to the polynomials of $\mathcal{A}$, we have:

Proposition 2 (Correctness w.r.t. Discrete Steps).

- *If $(q, v) \xrightarrow{a} (q', v') \in \mathcal{T}_\mathcal{A}$, then $(q, C) \xrightarrow{a} (q', C') \in \mathcal{R}_\mathcal{A}$ with $v \in C$ and $v' \in C'$.*
- *If $(q, C) \xrightarrow{a} (q', C') \in \mathcal{R}_\mathcal{A}$ then for all $v \in C$ there exists $v' \in C'$ such that $(q, v) \xrightarrow{a} (q', v') \in \mathcal{T}_\mathcal{A}$.*

Since the number of cells in a cylindrical decomposition is doubly exponential in the number of clocks and polynomial in the number and maximal degree of polynomials to which it is adapted [6], we obtain the complexity stated in Theorem 2. By setting $\{(q, C) \mid q \in F\}$ as the set of final states of $\mathcal{R}_{\mathcal{A},\psi}$, this construction establishes that the untimed language of a PolITA is regular.

4 Effective Construction and on-the-fly Algorithm

4.1 Construction of a Cylindrical Decomposition

Building a cylindrical decomposition consists in two stages: the elimination stage that enlarges $\mathcal{P}$ and the lifting stage that builds the cylindrical decomposition using symbolic representations of sample points (one per cell).

Elimination Stage. Starting from a cell C at level k, in order to get a partition at level $k+1$ adapted to $\mathcal{P}_{k+1}$, any two points $z, z' \in C$ should trigger a similar bevahiour for polynomials of $\mathcal{P}_{k+1}$, that we consider for our discussion as univariate polynomials of $\mathbb{Q}[x_1, \ldots, x_k][x_{k+1}]$ with variable x_{k+1}. More precisely, the properties we are looking for are:

- For all $P \in \mathcal{P}_{k+1}$ and for all z, z' in C, the number of real roots (counted with multiplicities) of the polynomials $P(z)$ and $P(z')$ in $\mathbb{R}[x_{k+1}]$ are equal (say μ_P). For $1 \leq i \leq \mu_P$ and $z \in C$, we denote by $r_{P,i}(z)$ the i^{th} real root of polynomial $P(z)$ (in increasing order) ;
- For all $P, Q \in \mathcal{P}_{k+1}$, for all $1 \leq i \leq \mu_P$ and $1 \leq j \leq \mu_Q$, for all z, z' in C, $r_{P,i}(z) \leq r_{Q,j}(z)$ implies $r_{P,i}(z') \leq r_{Q,j}(z')$.

These properties are analytical and do not provide insights on how to ensure them. Fortunately, it turns out that a simple effective sufficient condition exists: there is a finite subset of polynomials of $\mathbb{Q}[x_1, \ldots, x_k]$ denoted by $Elim_{x_{k+1}}(\mathcal{P}_{k+1})$ such that if z, z' satisfy $sign(R(z)) = sign(R(z'))$ for all $R \in Elim_{x_{k+1}}(\mathcal{P}_{k+1})$, then the above properties are satisfied.

To define $Elim_{x_{k+1}}(\mathcal{P}_{k+1})$, we need some notations. For $P = \sum_{i \leq p} a_i x_{k+1}^i$ with $a_i \in \mathbb{Q}[x_1, \ldots, x_k]$ for all i, $lcof(P)$ denotes the *leading coefficient* a_p. Since this leading coefficient is a polynomial and could be null for some $P(z)$, the set of truncations of P contains the possible "realizations" of P: $Tru(P) = \{\sum_{i \leq h} a_i x_{k+1}^i \mid \forall i > h \ a_i \notin \mathbb{R} \setminus \{0\} \wedge a_h \neq 0\}$. For instance, if $P = x_1 x_2^3 + (3x_1+1)x_2^2 + 5x_2 - 2$, then $Tru(P) = \{P, (3x_1+1)x_2^2 + 5x_2 - 2, 5x_2 - 2\}$. Given another polynomial, $Q = \sum_{i \leq q} b_i x_{k+1}^i \in \mathbb{Q}[x_1, \ldots, x_k][x_{k+1}]$, the *subresultants* $(sRes_i(P,Q))_{i \leq \max(p,q)}$ are polynomials of $\mathbb{Q}[x_1, \ldots, x_k]$ obtained as determinants of matrices whose items are coefficients of P and Q (see [6,11] for a formal definition of subresultants, a polynomial time computation and their properties).

Definition 5. *Let $\mathcal{P}_k$ be a finite subset of $\mathbb{Q}[x_1, \ldots, x_{k-1}][x_k]$ for $k > 1$. Then $Elim_{x_k}(\mathcal{P}_k)$ is the subset of $\mathbb{Q}[x_1, \ldots, x_{k-1}]$ defined for all $P, Q \in \mathcal{P}_k, R \in Tru(P)$, $T \in Tru(Q)$ by:*

- *If $lcof(R)$ does not belong to $\mathbb{Q}$ then $lcof(R) \in Elim_{x_k}(\mathcal{P}_k)$;*
- *If $deg(R) \geq 2$ then for all $sRes_j(R, \frac{\partial R}{\partial x_k})$ that are defined and do not belong to $\mathbb{Q}$, $sRes_j(R, \frac{\partial R}{\partial x_k}) \in Elim_{x_k}(\mathcal{P}_k)$;*
- *for all $sRes_j(R, T)$ that are defined and do not belong to $\mathbb{Q}$, $sRes_j(R, T) \in Elim_{x_k}(\mathcal{P})$.*

Using the properties of subresultants, one gets the following theorem whose implementation is the elimination stage of the cylindrical decomposition. Due to the quadratic blow up at each level of elimination the final number of polynomials is doubly exponential w.r.t. the original number.

Theorem 3. *Let $\mathcal{P} = \{\mathcal{P}_k\}_{k \leq n}$ be a family of finite set of polynomials such that $\mathcal{P}_k \subseteq \mathbb{Q}[x_1, \ldots, x_k]$. Define $\mathcal{Q}_n = \mathcal{P}_n$ and inductively $\mathcal{Q}_{k-1} = \mathcal{P}_{k-1} \cup Elim_{x_k}(\mathcal{Q}_k)$ for $k > 1$. Then there exists a cylindrical decomposition adapted to $\mathcal{Q}$ (and thus to $\mathcal{P}$).*

Example 5. Consider again the polynomials $B = (2x_1 - 1)x_2^2 - 1$ and $C = x_2 + x_1^2 - 5$ from the POLITA of Fig. 1a. Their subresultant of index 0 is $F = -2x_1^5 + x_1^4 + 20x_1^3 - 10x_1^2 - 50x_1 + 26$ which has precisely three real roots c_4, c_8, c_{10}: the x_1-coordinates of intersection points of graphs $B = 0$ and $C = 0$ mentioned previously.

Lifting Stage. The starting point of the lifting stage is the family $\mathcal{P}$ appropriately enlarged by the elimination stage. In the cylindrical decomposition that we build, every cell C of level k is represented by a *sample point* inside the cell and the values of signs of all polynomials of set $\mathcal{P}_k$ on this point.

We consider representations of real subrings of the form $\mathbb{D} = \mathbb{Q}[\alpha_1, \ldots, \alpha_k]$ where the α_i's are algebraic numbers, *i.e.*, roots of polynomials in $\mathbb{Q}[x]$. Any real algebraic number α can be represented by a pair (n, P) where P is a non null polynomial in $\mathbb{Q}[x]$ such that $P(\alpha) = 0$ and n is the index of α in the ordered set of real roots of P. This representation is extended for real algebraic points $(\alpha_1, \ldots, \alpha_k)$ with the notion of *triangular systems*: α_1 is the n_1^{th} root of $P_1 \in \mathbb{Q}[x_1]$, α_2 is the n_2^{th} root of $P_2(\alpha_1)$ with $P_2 \in \mathbb{Q}[x_1][x_2]$, etc.

Definition 6 (Triangular System). *For $k \geq 1$, let $(\alpha_1, \ldots, \alpha_k)$ be a sequence of reals and let $\{(n_i, P_i)\}_{i=1}^k$ be such that for all i, n_i is a positive integer and $P_i \in \mathbb{Q}[x_1, \ldots, x_{i-1}][x_i]$. Then $\{(n_i, P_i)\}_{i=1}^k$ is a triangular system of level k for $(\alpha_1, \ldots, \alpha_k)$ if:*

- *P_1 is non null and α_1 is its n_1^{th} real root;*
- *For $1 \leq i < k$, $P_{i+1}(\alpha_1, \ldots, \alpha_i)$ is a non null polynomial of $\mathbb{Q}[\alpha_1, \ldots, \alpha_i][x_{i+1}]$ and α_{i+1} is its n_{i+1}^{th} real root.*

Example 6. Let us consider the point (α_1, α_2) depicted as a circle in Fig. 1b. This point is represented by the triangular system $((2, A), (2, B))$ where $A = x_1^2 - x_1 - 2$ and $B = (2x_1 - 1)x_2^2 - 1$. This means that α_1 is the 2^{nd} root of A and α_2 is the 2^{nd} root of $B(\alpha_1)$.

The interest of such a representation is its effectiveness: in a ring $\mathbb{D} = \mathbb{Q}[\alpha_1, \ldots, \alpha_k]$ associated with a triangular system one can compute (1) the sign of an item of $\mathbb{Q}[\alpha_1, \ldots, \alpha_k]$, (2) the number of real roots of $P(\alpha_1, \ldots, \alpha_k)$ with $P \in \mathbb{Q}[x_1, \ldots, x_k][x_{k+1}]$, (3) the sign realizations of a polynomial $Q(\alpha_1, \ldots, \alpha_k)$ on the real roots of a polynomial $P(\alpha_1, \ldots, \alpha_k)$, and one can order (with merge)

the roots of $P(\alpha_1, \ldots, \alpha_k)$ and $Q(\alpha_1, \ldots, \alpha_k)$. All these procedures are performed in polynomial time (see for instance [11]).

The tree corresponding to the cylindrical decomposition is built top-down so that a triangular system is associated with a sample point of every cell and its sign realizations on the appropriate polynomials. Let us describe how, given a sample point $(\alpha_1, \ldots, \alpha_k)$, the partition over axis x_{k+1} can be built w.r.t. $\mathcal{P}_{k+1}$. First for all $P \in \mathcal{P}_{k+1}$, the number of roots of $P(\alpha_1, \ldots, \alpha_k)$ is determined. Then the roots of these polynomials are sorted and merged; their triangular system is the one associated with $(\alpha_1, \ldots, \alpha_k)$ extended by the polynomial for which they are roots. Then the open intervals between these roots or beyond these roots must be specified, to yield the *completed line partitioning*. Let (r, P) and (s, Q) be the borders of an open interval, then one selects as sample point, a root of $\frac{\partial(PQ)}{\partial x_{k+1}}$ located in the interval. Let (r, P) and $+\infty$ (resp. $-\infty$ and $(1, P)$) be the borders of the last (resp. first) open interval, then one selects $(r, P[x_{k+1} := x_{k+1} - 1])$ (resp. $(r, P[x_{k+1} := x_{k+1} + 1])$) as sample point. To achieve this step it remains to compute the sign realizations of $P(\alpha_1, \ldots, \alpha_k)$ for all $P \in \mathcal{P}_{k+1}$ on these sample points. Theorem 1 results from these two construction steps.

4.2 On-the-fly Algorithm

The abstraction from Section 3 provides decidability of the reachability problem, by the algorithm that builds the finite graph $\mathcal{R}_{\mathcal{A}}$. However, building the complete graph is not efficient in practice, since it requires to build the set of all cells beforehand, even though usually most of them are unreachable. In the sequel, we show an on-the-fly construction of $\mathcal{R}_{\mathcal{A}}$ that reduces complexity in practice.

The key to the on-the-fly algorithm is to store only the part of the tree corresponding to the current sample point and its time successors. This construction relies on executing the lifting phase only when the level is increased and then only for the current sample point. As an illustration, in Fig. 1b, only the lifting for x_2 above c_5 has been represented, since it is the only relevant one with respect to the given trajectory. Note that liftings over sample points c_0 to c_6 have to be computed in order to build the reachable part of $\mathcal{R}_{\mathcal{A}_0}$. On the other hand, liftings over c_7 to c_{11} and over unrepresented cells to the left of c_0, need not, since level 2 is not reachable from these cells. As a result, we do not keep the whole tree but only part of it.

We show that this information is sufficient to compute the successors through time elapsing and transition firing. Although this pruning yields better performances in practice, the computational complexity in the worst case is not improved.

Definition 7 (Pruned Tree). *Let $\{\mathcal{P}_k\}_{k \leq n}$ be the polynomials obtained by the elimination phase. The* pruned tree *for sample point $(\alpha_1, \ldots, \alpha_k)$ is the sequence of completed line partitionings for sample points $\{(\alpha_1, \ldots, \alpha_i)\}_{1 \leq i \leq k}$. The pruned tree for the empty sample point ($k = 0$) is the line partitioning at level 1.*

A valuation $(v_1, \ldots, v_k, 0, \ldots, 0)$ at level k is represented by a sample point $(\alpha_1, \ldots, \alpha_k)$, or, equivalently, by a pruned tree for sample point $(\alpha_1, \ldots, \alpha_{k-1})$

and the index m of α_k in the line partitioning for $(\alpha_1, \ldots, \alpha_{k-1})$. In this representation, computing the time successors of $(\alpha_1, \ldots, \alpha_k)$ is simply done by incrementing m (if it is not the maximal index in the line partitioning).

The set of enabled discrete transitions can be generated by computing the signs of polynomials appearing in guards. When a discrete transition $q \xrightarrow{g,a,u} q'$ is chosen, there are three cases w.r.t. the level of states q and q'.

- The level decreases, *i.e.* $\lambda(q') < \lambda(q)$. Then the pruned tree corresponding to the new configuration is the truncation of the original pruned tree up to height $\lambda(q')$. Otherwise said, we "forget" line partitionings for levels above $\lambda(q')$; however, the partitionings are kept in memory to avoid redundant computations. The new index is the index of $\alpha_{\lambda(q')}$ in the partitioned line for this level.
- The level is unchanged, *i.e.* $\lambda(q') = \lambda(q) = k$. The only possible change of clock values is through an update $x_k := P$ with $P \in \mathbb{Q}[x_1, \ldots, x_{k-1}]$. The polynomial of degree 1 $R = x_k - P$ was added to $Poly(\mathcal{A})$ and its unique root α'_k appears in the line partitioning of level k. Note that in the triangular system representing $(\alpha_1, \ldots, \alpha'_k)$ it may appear as $((n_1, P_1), \ldots, (n_k, P_k))$ with $(n_k, P_k) \neq (1, R)$. Hence to determine the index in the partitioned line the algorithm must actually determine the sign of R for all sample points of the line until 0 is found.
- The level increases, *i.e.* $\lambda(q') > \lambda(q)$. If there is an update of x_k, the same computations as above must be performed in order to find the new sample point corresponding to the valuation of clocks up to $\lambda(q)$. Then the pruned tree of height $\lambda(q')$ has to be computed (or retrieved). This is done by $\lambda(q') - \lambda(q)$ lifting steps. These lifting steps are applied on sample points of the form $(\alpha_1, \ldots, \alpha_{\lambda(q)}, 0, \ldots, 0)$, since all clocks are null for levels above $\lambda(q)$.

The on-the-fly algorithm builds the reachable part of $\mathcal{R}_\mathcal{A}$ as follows: the elimination phase is computed and the line for x_1 is partitioned. It starts with a queue containing q_0 with index corresponding to the root of x_1 (*i.e.* 0). Then until the queue is empty, it computes all (new) successors through time and discrete transitions, building the pruned tree as described above. As noted above, a line partitioning only needs to be computed once. In addition, and this also holds for the complete construction of $\mathcal{R}_\mathcal{A}$, the *triangular* structure of triangular systems enables a sharing of line partitioning at lower levels.

5 Conclusion and Discussion

We extend ITA with polynomial expressions on clocks, and prove that reachability is decidable using the cylindrical decomposition. We also show that an on-the-fly construction of a class automaton is possible during the lifting phase of this decomposition.

We now mention several additional results proved in [11] but omitted here. The first one concerns the decidability of the model checking of $\mathsf{TCTL}_{\mathrm{int}}$, a variant of TCTL [1], where only local clocks can be used in the formulas.

The PoLITA is equipped with atomic propositions that hold in states. Another direction was to investigate the expressive power of the model and try to extend it while keeping decidability of reachability. We first established that stopwatch automata and PoLITA are incomparable. Then we proved that reachability is still decidable when including parameters in the expressions of guards and updates, with a better complexity than obtained in [9] (2EXPSPACE). We also extend the model by adding at each level i, a set of *auxiliary* clocks Y_i in addition to the *main* clock x_i. With several restrictions, we still obtain a decidability result for reachability. A last extension allows updates for clocks of levels lower than the current one. Again with some restrictions, decidability for reachability is preserved via a translation into a basic PoLITA, similarly to [10] for ITA. Finally, as also presented in [10] for ITA, it is possible to extend the model of PoLITA by adding timed automata at a lower level 0, producing a class that is stricly more expressive than timed automata.

An implementation is in progress to experiment the practical efficiency of the decision procedures. Since the construction still suffers from the doubly exponential complexity of the cylindrical decomposition, we plan to investigate if recent methods [16] with a lower complexity could be used to achieve reachability, possibly for a restricted version of PoLITA. Another direction would be to enlarge the class of functions (like those studied in [18]) labelling guards and updates, still ensuring a finite bisimulation quotient.

References

1. Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking in dense real-time. Information and Computation **104**, 2–34 (1993)
2. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. TCS **138**, 3–34 (1995)
3. Alur, R., Dill, D.L.: A theory of timed automata. TCS **126**, 183–235 (1994)
4. Alur, R., Henzinger, T.A., Lafferriere, G., Pappas, G.J.: Discrete abstractions of hybrid systems. Proceedings of the IEEE **88**(7), 971–984 (2000)
5. Asarin, E., Maler, O., Pnueli, A.: Reachability analysis of dynamical systems having piecewise-constant derivatives. TCS **138**(1), 35–65 (1995)
6. Basu, S., Pollack, R., Roy, M.F.: Algorithms in Real Algebraic Geometry. Springer (2006)
7. Ben-Or, M., Kozen, D., Reif, J.: The complexity of elementary algebra and geometry. In: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, STOC 1984, pp. 457–464. ACM (1984)
8. Bérard, B., Haddad, S.: Interrupt timed automata. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 197–211. Springer, Heidelberg (2009)
9. Bérard, B., Haddad, S., Jovanović, A., Lime, D.: Parametric interrupt timed automata. In: Abdulla, P.A., Potapov, I. (eds.) RP 2013. LNCS, vol. 8169, pp. 59–69. Springer, Heidelberg (2013)
10. Bérard, B., Haddad, S., Sassolas, M.: Interrupt timed automata: Verification and expressiveness. Formal Methods in System Design **40**(1), 41–87 (2012)
11. Bérard, B., Haddad, S., Picaronny, C., Safey El Din, M., Sassolas, M.: Polynomial interrupt timed automata. CoRR abs/1504.04541, April 2015

12. Cassez, F., Larsen, K.G.: The impressive power of stopwatches. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 138–152. Springer, Heidelberg (2000)
13. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In: Brakhage, H. (ed.) Automata Theory and Formal Languages 2nd GI Conference, vol. 33, pp. 134–183. Springer, Heidelberg (1975)
14. Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.): HS 1991 and HS 1992. LNCS, vol. 736. Springer, Heidelberg (1993)
15. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998)
16. Hong, H., Din, M.S.E.: Variant quantifier elimination. Journal of Symbolic Computation **47**(7), 883–901 (2012)
17. Lafferriere, G., Pappas, G.J., Sastry, S.: O-minimal hybrid systems. MCSS **13**(1), 1–21 (2000)
18. Miller, D.J.: Constructing o-minimal structures with decidable theories using generic families of functions from quasianalytic classes. ArXiv e-prints 1008.2575, August 2010
19. Miller, J.S.: Decidability and complexity results for timed automata and semi-linear hybrid automata. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 296–309. Springer, Heidelberg (2000)
20. Müller-Olm, M., Seidl, H.: Computing polynomial program invariants. Inf. Process. Lett. **91**(5), 233–244 (2004)

Irregular Behaviours for Probabilistic Automata

Nathanaël Fijalkow[1,2] and Michał Skrzypczak[1,2](✉)

[1] LIAFA, Université Denis Diderot - Paris 7, Paris, France
[2] University of Warsaw, Warsaw, Poland
mskrzypczak@mimuw.edu.pl

Abstract. We consider probabilistic automata over finite words. Such an automaton defines the language consisting of the set of words accepted with probability greater than a given threshold. We show the existence of a universally non-regular probabilistic automaton, *i.e.* an automaton such that the language it defines is non-regular for every threshold. As a corollary, we obtain an alternative and very simple proof of the undecidability of determining whether such a language is regular.

1 Introduction

Rabin introduced probabilistic automata in 1963 [Rab63]. They have been studied ever since, with applications to different areas, such as Computational Linguistics and Biology. Despite its simplicity, this computational model is very powerful, and many decision problems for probabilistic automata are known to be undecidable.

A probabilistic automaton defines a probabilistic language through a threshold semantics, as defined by Rabin [Rab63]. The algorithmic properties of these languages are well-understood; we refer to the book of Paz [Paz71] and the survey of Condon [Con01] for a wealth of results about them.

The class of probabilistic languages strictly subsumes the class of regular languages; in this paper, we consider the decision problem of determining whether a probabilistic language is regular. This problem has been considered by Bertoni [Ber74], and proved undecidable. The aim of this paper is to give a different and simple proof of this result.

2 Preliminaries

Let Q be a finite set of states. A distribution over Q is a function $\delta : Q \to [0, 1]$ such that $\sum_{q \in Q} \delta(q) = 1$. We denote $\mathcal{D}(Q)$ the set of distributions over Q.

Definition 1 (Probabilistic Automaton). *A probabilistic automaton $\mathcal{A}$ is given by a finite set of states Q, a transition function $\phi : A \to (Q \to \mathcal{D}(Q))$, an initial state $q_0 \in Q$, and a set of final states $F \subseteq Q$.*

The Research Leading to These Results Has Received Funding from the European Union's Seventh Framework Programme (FP7/2007-2013) Under Grant Agreement 259454 (GALE).

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 33–36, 2015.
DOI: 10.1007/978-3-319-24537-9_4

All the numbers in the transition function of a probabilistic automaton are assumed to be rational numbers.

In a transition function ϕ, the quantity $\phi(a)(s,t)$ is the probability to go from the state $s \in Q$ to the state $t \in Q$ reading the letter a. A transition function naturally induces a morphism $\phi : A^* \to (Q \to \mathcal{D}(Q))$. We denote $\mathbb{P}_{\mathcal{A}}(s \xrightarrow{w} t)$ the probability to go from a state s to a state t reading w on the automaton $\mathcal{A}$, *i.e.* $\phi(w)(s,t)$.

The *acceptance probability* of a word $w \in A^*$ by $\mathcal{A}$ is $\sum_{t \in F} \phi(w)(q_0,t)$, which we denote $\mathbb{P}_{\mathcal{A}}(w)$.

The following threshold semantics was introduced by Rabin [Rab63].

Definition 2 (Probabilistic Language). *Let $\mathcal{A}$ be a probabilistic automaton and $x \in (0,1)$. This induces the probabilistic language*

$$L^{>x}(\mathcal{A}) = \{w \in A^* \mid \mathbb{P}_{\mathcal{A}}(w) > x\}.$$

The *emptiness problem* was considered by Rabin: given a probabilistic automaton $\mathcal{A}$, determine whether $L^{>\frac{1}{2}}(\mathcal{A})$ is non-empty, *i.e.* whether there exists a word w such that $\mathbb{P}_{\mathcal{A}}(w) > \frac{1}{2}$.

Theorem 1 ([Paz71]). *The emptiness problem is undecidable.*

A simple undecidability proof was given by Gimbert and Oualhadj in [GO10].

3 A Universally Non-regular Probabilistic Automaton

Theorem 2. *There is a probabilistic automaton $\mathcal{C}$ such that for every number x in $(0,1)$, the language $L^{>x}(\mathcal{C})$ is non-regular.*

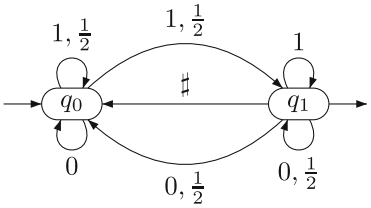


Fig. 1. A universally non-regular probabilistic automaton.

In the original paper introducing probabilistic automata, Rabin [Rab63] gave an example of a probabilistic automaton $\mathcal{A}$ such that $L^{>x}(\mathcal{A})$ is non-regular, for all irrational numbers x. The alphabet of the Rabin's automaton $\mathcal{A}$ is $\{0,1\}$. The automaton $\mathcal{A}$ computes the binary decomposition function denoted bin, *i.e.* $\mathbb{P}_{\mathcal{A}}(u) = \mathrm{bin}(u)$, defined by $\mathrm{bin}(a_1 \cdots a_n) = \frac{a_1}{2^n} + \cdots + \frac{a_n}{2^1}$. We show that adding

one letter and one transition to this probabilistic automaton makes it universally non-regular.

The automaton $\mathcal{C}$ is represented in Figure 1. The alphabet is $C = \{0, 1, \sharp\}$. The only difference between the automaton $\mathcal{A}$ proposed by Rabin [Rab63] and this one is the only transition over $\sharp$. As observed by Rabin, a simple induction shows that for u in $\{0,1\}^*$, we have $\mathbb{P}_{\mathcal{C}}(u) = \mathrm{bin}(u)$.

We show that for all numbers x in $(0,1)$, the language $L^{>x}(\mathcal{C})$ is non-regular. Let u, v in $\{0,1\}^*$, observe that $\mathbb{P}_{\mathcal{C}}(u \cdot \sharp \cdot v) = \mathrm{bin}(u) \cdot \mathrm{bin}(v)$.

Fix x in $(0,1)$. For every u, v in $\{0,1\}^*$ such that $\mathrm{bin}(u) < \mathrm{bin}(v) < x$, there exists w in $\{0,1\}^*$ such that $u \cdot \sharp \cdot w \in L^{>x}(\mathcal{C})$ and $v \cdot \sharp \cdot w \notin L^{>x}(\mathcal{C})$; it suffices to choose w such that $\mathrm{bin}(w)$ is in $\left(\frac{x}{\mathrm{bin}(v)}, \frac{x}{\mathrm{bin}(u)}\right)$. It follows that the left quotients $u^{-1} \cdot L^{>x}(\mathcal{C})$ and $v^{-1} \cdot L^{>x}(\mathcal{C})$ are distinct, so $L^{>x}(\mathcal{C})$ has an infinite number of pairwise distinct left quotients, hence it is not regular.

4 Main Result

Theorem 3. *(Undecidability of the regularity problem) The regularity problem is undecidable for probabilistic automata.*

This result was originally proved in [Ber74].

Roughly speaking, the idea is to use the universally non-regular automaton given in Section 3 to "amplify" an irregular behaviour.

Proof. We construct a reduction from the emptiness problem to the regularity problem. Then, undecidability of the latter follows from Theorem 1.

Let $\mathcal{A}$ be a probabilistic automaton over an alphabet A. We construct a probabilistic automaton $\mathcal{B}$ such that:

$$L^{>\frac{1}{2}}(\mathcal{A}) \text{ is empty} \quad \text{if and only if} \quad L^{>\frac{1}{2}}(\mathcal{B}) \text{ is regular.}$$

The automaton $\mathcal{B}$ is over the alphabet $B = A \uplus C$ where $C = \{0, 1, \sharp\}$, and uses the automaton $\mathcal{C}$ from Section 3. It is obtained as the sequential composition of $\mathcal{A}$ and $\mathcal{C}$: it starts in $\mathcal{A}$ and from every final state of $\mathcal{A}$ moves by $\sharp$ to the initial state of $\mathcal{C}$. The initial state of $\mathcal{B}$ is the initial state of $\mathcal{A}$, the only final state of $\mathcal{B}$ is the final state of $\mathcal{C}$.

For $u \in A^*$ and $v \in C^*$, we have $\mathbb{P}_{\mathcal{B}}(u \cdot \sharp \cdot v) = \mathbb{P}_{\mathcal{A}}(u) \cdot \mathbb{P}_{\mathcal{C}}(v)$. A word which is not in $A^* \cdot \sharp \cdot C^*$ has no accepting run, so is accepted with probability 0.

- Assume that $L^{>\frac{1}{2}}(\mathcal{A})$ is empty. Thanks to the above observation we have that $L^{>\frac{1}{2}}(\mathcal{B})$ is empty, so in particular it is regular.
- Conversely, assume that $L^{>\frac{1}{2}}(\mathcal{A})$ is non-empty. Let u be a word such that $\mathbb{P}_{\mathcal{A}}(u) > \frac{1}{2}$. Observe that $L^{>\frac{1}{2}}(\mathcal{B}) \cap (u \cdot \sharp \cdot C^*) = u \cdot \sharp \cdot L^{>x}(\mathcal{C})$, where $x = \frac{1}{2 \cdot \mathbb{P}_{\mathcal{A}}(u)}$ is in $(0,1)$. By Theorem 2, the language $L^{>x}(\mathcal{C})$ is non-regular, hence so is $u \cdot \sharp \cdot L^{>x}(\mathcal{C})$, implying that $L^{>\frac{1}{2}}(\mathcal{B})$ is also non-regular.

∎

References

[Ber74] Bertoni, A.: Mathematical methods of the theory of stochastic automata. In: Blikle, A. (ed.) Mathematical Foundations of Computer Science. IFMBE Proceedings, vol. 28, pp. 9–22. Springer, Heidelberg (1974)

[Con01] Condon, A.: Bounded error probabilistic finite state automata, chapter 1. In: Handbook on Randomized Computing, vol. II, pp. 509–532. Kluwer (2001)

[GO10] Gimbert, Hugo, Oualhadj, Youssouf: Probabilistic automata on finite words: decidable and undecidable problems. In: Abramsky, Samson, Gavoille, Cyril, Kirchner, Claude, Meyer auf der Heide, Friedhelm, Spirakis, Paul G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 527–538. Springer, Heidelberg (2010)

[Paz71] Paz, A.: Introduction to Probabilistic Automata. Academic Press (1971)

[Rab63] Rabin, M.O.: Probabilistic automata. Information and Control **6**(3), 230–245 (1963)

Reachability in Succinct One-Counter Games

Paul Hunter(✉)

Département d'Informatique, Université Libre de Bruxelles (U.L.B.),
Brussels, Belgium
paul.hunter@ulb.ac.be

Abstract. We consider two-player games with reachability objectives played on transition systems of succinct one-counter machines, that is, machines where the counter is incremented or decremented by a value given in binary. We show that the winner-determination problem is EXPSPACE-complete regardless of whether transitions are guarded by constraints on the counter or if the counter is restricted to non-negative values.

1 Introduction

Transition graphs of one-counter machines are some of the most simple infinite state systems, and have been extensively studied [1,3,5,6,8–10,13,14], often under the guise of more general formalisms such as pushdown automata, Petri nets, and vector addition systems with states. A *one-counter system* consists of a finite state machine equipped with an integer-valued counter that can be incremented or decremented thereby adding functionality, for example by enabling certain transitions when the counter is within a certain range. Common characteristics for differentiating classes of one-counter systems include: whether the counter is incremented/decremented by 1 or if it can be changed by larger amounts (called "short-range" and "long-range" dynamics in [13]); whether the counter can take negative values or is restricted to the non-negative integers; and the scope of counter-based constraints that can be placed on transitions.

Two player games are a natural tool for modelling interaction-based processes. They are commonly used in Computer Science to represent interactions such as System vs Environment, or Model vs Specification. Such games played on one-counter systems are particularly useful for modelling the behaviour of resource-based systems, or for model-checking simple, but infinite, state systems.

In this paper we consider two player games on one-counter systems where the objective of one of the players (Eve) is to reach a particular configuration (or a particular counter value). Many variations of this reachability problem have been studied: most closely, the reachability problem for VASS [3,13]; but also the reachability problem for one-counter automata (which can be seen as single player games) [5,9]; countdown games [11] and robot games [2] (which can be seen as restrictions of our problem); and the closely related problems of parity games on one-counter systems [14], and CTL [8] model checking.

This work was supported by the ERC inVEST (279499) project.

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 37–49, 2015.
DOI: 10.1007/978-3-319-24537-9_5

Unlike the majority of the previous work, we consider both integer and non-negative integer counter systems, as well as systems with counter-constrained transitions and minimally constrained systems – the two player analogue of one-counter nets [1]. In some cases, however, we can establish some complexity bounds based on previous work. It was shown in [3,13] that the reachability problem for one-dimensional VASS was PSPACE-complete, and the counter reachability problem is in PTIME. Our work differs from [3] and [13] in that we consider one-counter systems where the counter is updated by adding integer constants (given in binary) rather than simple incrementing and decrementing by 1, however their results imply an EXPSPACE upper bound for some cases of our problem (namely, minimally constrained integer and non-negative integer counter systems). The games we consider also generalize the games considered in [2,11] giving us an EXPTIME-hard lower bound for all cases.

Our Contribution. We consider the winner determination problem for reachability and counter reachability games on several classes of succinct one-counter systems. As mentioned, the related work establish an EXPTIME-hardness lower bound (for some classes) and an EXPSPACE upper bound (for some classes). We show that the problems are equivalent for all classes (in contrast to the non-succinct case) and that they are EXPSPACE-complete. As a side-effect of our work, we also obtain an EXPSPACE-completeness result for the winner determination problem for Büchi games on one-counter systems where the Büchi condition is applied to configurations rather than states as in [14] (and implicitly in [8]).

2 Preliminaries

2.1 Arenas, Plays, and Strategies

The two-player games we are interested in for this paper are played on an *arena* – a (possibly infinite) directed graph with vertices partitioned into two sets, one set belonging to Eve and one to Adam. The two players move a token around the graph, the player that owns the location of the token choosing a successor to move to. In this way, the two players generate a (possibly infinite) path in the graph, and we use this path to determine the winner (see Section 2.3).

More formally, an *arena* is a tuple $(V, V_\exists, E, v_I)$ where (V, E) is a directed graph, $v_I \in V$ is the *initial vertex*, and $V_\exists \subseteq V$ is the set of vertices *belonging to Eve*. For convenience, we often write $V_\forall$ for $V \setminus V_\exists$, the set of vertices *belonging to Adam*. In the sequel when depicting arenas, we use squares for vertices in $V_\exists$ and circles for vertices in $V_\forall$ (diamonds represent arbitrary vertices).

A *play (from $v \in V$)* in an arena is a (possibly infinite) sequence of vertices $v_0, v_1, \ldots$ where $v_0 = v$ and for all i, $(v_i, v_{i+1}) \in E$. If the starting vertex of a play is omitted it is assumed to be v_I. A *strategy for Eve (Adam)* is a function $\sigma : V^*V_\exists \to V$ ($\tau : V^*V_\forall \to V$) satisfying $(v, \sigma(w)) \in E$ ($(v, \tau(w)) \in E$) for all

finite plays w ending in v. A play $v_0, v_1, \ldots$ is *consistent* with a strategy σ for Eve (Adam) if for all i such that $v_i \in V_\exists$ ($v_i \in V_\forall$) we have $\sigma(v_0 v_1 \ldots v_i) = v_{i+1}$.

A *winning condition* (see Section 2.3) defines a set of plays which are *winning for Eve*, any other play is *winning for Adam*. We assume that a player loses if they cannot move, so a winning condition includes all finite plays ending in a vertex in $V_\forall$ with no outgoing edges, and excludes all finite plays ending in a vertex in $V_\exists$ with no outgoing edges. A *game* is given by an arena and a winning condition. A strategy for a player is *winning* if all plays consistent with the strategy are winning for that player, and a player *wins a game* if they have a winning strategy.

2.2 One-Counter Systems

A one-counter system is a finite presentation of an infinite arena. It is given by a finite arena with *weights* and *guards* on the transitions (for clarity in figures, edges with no given weight are assumed to have weight 0, and edges with no given guard are assumed to have guard $\mathtt{true}$). Intuitively, the game is played as before, with the two players moving a token around the vertices, but in addition there is an (integer-valued) counter which constrains the transitions available. The weights indicate how much the counter is incremented/decremented at each step and the guards, which define a set of counter values, indicate which transitions are available for the current counter value.

Formally, a *succinct one-counter system (with counter values in $\mathbb{Z}$)* [$\mathsf{SOCS}_\mathbb{Z}$] is a tuple $(V, V_\exists, E, v_I, w, \gamma)$ where

- $(V, V_\exists, E, v_I)$ is an arena with V a finite set;
- $w : E \to \mathbb{Z}$ is the weight function; and
- γ is a function that maps each edge to a guard – a conjunction of counter constraints of the form $(\mathtt{c} \sim M)$, $\mathtt{bit}_i(\mathtt{c})$ and $\overline{\mathtt{bit}_i}(\mathtt{c})$ where $M, i \in \mathbb{Z}$, $i \geq 0$ and $\sim \in \{=, \neq, <, \leq, >, \geq\}$.

The numerical values appearing in w and λ are assumed to be given in binary, with the exception of the subscripts in the $\mathtt{bit}$ and $\overline{\mathtt{bit}}$ predicates, which are given in unary. Note, a counter value $c \in \mathbb{Z}$ satisfies $\mathtt{bit}_i(c)$ ($\overline{\mathtt{bit}_i}(c)$) if and only if the i-th bit of $|c|$ is 1 (0).

Remark 1. In the literature the guard constraints are typically more restrictive, for example the counter inequalities may only involve 0 (sign-testing), or just $=$ and 0 (zero-testing), and the $\mathtt{bit}$-predicate is not included. In the next section we will show that even the more expressive constraints considered here can be simulated by games without constraints, so for convenience we adopt the more expressive form. Note also that by using multiple transitions one can obtain a disjunction of the guarding conditions, so having conjunctions of constraints is sufficient to obtain any[1] Boolean combination of constraints.

[1] In order to avoid an exponential blow-up, rather than converting an arbitrary Boolean combination to DNF one should adopt a more direct approach using intermediate nodes and the antagonism of the players (e.g. Eve = $\vee$, Adam = $\wedge$).

A $\mathsf{SOCS}_{\mathbb{Z}}$, $\mathcal{G} = (V, V_\exists, E, v_I, w, \gamma)$, defines an arena $\mathcal{A}_\mathcal{G} = (V', V'_\exists, E', v'_I)$ where $V' = V \times \mathbb{Z}$, $V'_\exists = V_\exists \times \mathbb{Z}$, $v'_I = (v_I, 0)$, and $\big((v,c),(v',c')\big) \in E'$ iff $e = (v, v') \in E$, $c' = c + w(e)$ and $\gamma(e)$ is true at $\mathsf{c} = c$.

A *succinct one-counter process (with counter values in* $\mathbb{Z}$*)* [$\mathsf{SOCP}_{\mathbb{Z}}$] is a $\mathsf{SOCS}_{\mathbb{Z}}$ where $\gamma(e) = \mathsf{true}$ for all $e \in E$. A *succinct one-counter system (with counter values in* $\mathbb{N}$*)* [$\mathsf{SOCS}_{\mathbb{N}}$] is a $\mathsf{SOCS}_{\mathbb{Z}}$ where the positivity constraint $(\mathsf{c} \geq -w(e))$ is included in $\gamma(e)$ for all $e \in E$, and a *succinct one-counter process (with counter values in* $\mathbb{N}$*)* [$\mathsf{SOCP}_{\mathbb{N}}$] is a $\mathsf{SOCS}_{\mathbb{N}}$ where the positivity constraint is the only constraint in $\gamma(e)$. Note that the positivity constraint ensures the counter value never goes below 0 (assuming it starts on 0), so the arena associated with a $\mathsf{SOCS}_{\mathbb{N}}$ (or $\mathsf{SOCP}_{\mathbb{N}}$) can be restricted to $V \times \mathbb{N}$.

2.3 Winning Conditions

Reachability Conditions. In this paper we focus on reachability games, that is, games where the winning condition consists of all plays that contain at least one vertex from a given set of vertices. We are particularly interested in two problems: the *reachability problem* – Can Eve reach a particular vertex (or any of a set of vertices) with a particular counter value?; and the *counter reachability problem* – Can Eve reach a particular counter value (in any vertex)? Without loss of generality we can assume that 0 is the target counter value in both problems.[2] To avoid trivialities we assume the counter reachability problem applies to all vertices except the initial vertex. We do not consider the third natural reachability problem, *state reachability* (Can Eve reach a particular vertex with any counter value?) – as observed in [9] this problem is easily seen to be log-space equivalent to the reachability problem.

More formally, we define the problem Reach($\mathsf{SOCS}_{\mathbb{Z}}$) as: Given a $\mathsf{SOCS}_{\mathbb{Z}}$, $\mathcal{G} = (V, V_\exists, E, v_I, w, \gamma)$, and a *target set* of vertices $T \subseteq V$, does Eve have a strategy (in $\mathcal{A}_\mathcal{G}$) to ensure that all consistent plays contain a vertex in $T \times \{0\}$? That is, does Eve win the *one-counter reachability game* defined by $(\mathcal{G}, T)$? The problem C-Reach($\mathsf{SOCS}_{\mathbb{Z}}$) is the same as Reach($\mathsf{SOCS}_{\mathbb{Z}}$) with the target set $T = V \setminus \{v_I\}$. The problems Reach($*$) and C-Reach($*$) for $* \in \{\mathsf{SOCP}_{\mathbb{Z}}, \mathsf{SOCS}_{\mathbb{N}}, \mathsf{SOCP}_{\mathbb{N}}\}$ are defined similarly.

In the sequel, when depicting a one-counter reachability game, we use a double border for vertices in the target set T.

Remark 2. Without loss of generality, we can assume $T \subseteq V_\exists$: for any vertex $v \in T \cap V_\forall$ we can add a new vertex $v' \in V_\exists$ and an edge (v', v) with weight 0 and replace all edges (u, v) by (u, v') (with the same weight and guard conditions). We observe there is a natural correspondence between (winning) plays in this new game and (winning) plays in the original game, and replacing v with v' in T does not change this correspondence.

[2] It is straightforward to modify a $\mathsf{SOCS}_{\mathbb{Z}}$ to change the target counter value. For the other models it is less clear, but it follows from the equivalences we establish in the next section.

Büchi Conditions. We are also interested in an analogue of reachability extended to infinite plays, the *Büchi condition*. This is also specified by a target set $T \subseteq V_\exists$ and an (infinite) play is winning for Eve if there is some vertex $v \in T$ which occurs infinitely often in the play. In the case of one-counter games there are two variants: the *Büchi condition*, where only some vertex in T needs to be visited infinitely often – that is, the counter value is irrelevant for the winning condition (formally, some $v \in T$ occurs infinitely often as a first component in the play in $\mathcal{A}_\mathcal{G}$); and the *strong Büchi condition* where the vertex has to be visited infinitely often with a particular counter value, again, assumed without loss of generality to be 0 (formally, for some $v \in T$, $(v, 0)$ occurs infinitely often in the play in $\mathcal{A}_\mathcal{G}$). Note that unlike the state reachability and reachability problems, it is not immediately clear that these problems are equivalent. We denote by BÜCHI$(*)$ (STBÜC$(*)$) for $* \in \{\mathsf{SOCS}_\mathbb{Z}, \mathsf{SOCP}_\mathbb{Z}, \mathsf{SOCS}_\mathbb{N}, \mathsf{SOCP}_\mathbb{N}\}$ the decision problem: given a one-counter system $\mathcal{G}$ of type $*$, and $T \subseteq V_\exists$, does Eve win the (strong) one-counter Büchi game given by $(\mathcal{G}, T)$?

3 Equivalence of Models and Problems

In this section we establish the following result:

Theorem 1. *The problems* REACH$(\mathsf{SOCS}_\mathbb{Z})$, REACH$(\mathsf{SOCP}_\mathbb{Z})$, REACH$(\mathsf{SOCS}_\mathbb{N})$, REACH$(\mathsf{SOCP}_\mathbb{N})$, C-REACH$(\mathsf{SOCS}_\mathbb{Z})$, C-REACH$(\mathsf{SOCP}_\mathbb{Z})$, C-REACH$(\mathsf{SOCS}_\mathbb{N})$, *and* C-REACH$(\mathsf{SOCP}_\mathbb{N})$ *are all equivalent under log-space reductions.*

As a consequence of our constructions, we obtain a similar result for strong Büchi games.

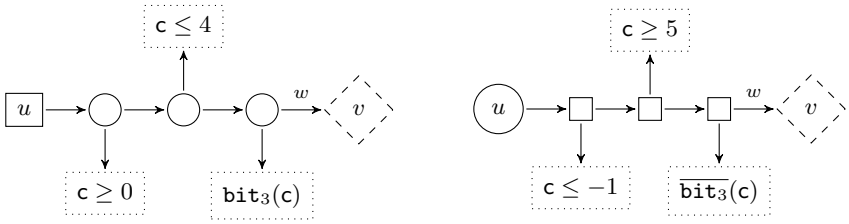
Corollary 1. *The problems* STBÜC$(\mathsf{SOCS}_\mathbb{Z})$, STBÜC$(\mathsf{SOCP}_\mathbb{Z})$, STBÜC$(\mathsf{SOCS}_\mathbb{N})$, *and* STBÜC$(\mathsf{SOCP}_\mathbb{N})$ *are equivalent under log-space reductions.*

These results follow from Lemmas 2, 4, and 5, established in the remainder of this section.

3.1 Removing the Guards

In this section we demonstrate how we can remove the guarding conditions, showing that reachability games on one-counter systems are equivalent to games on one-counter processes. For this section we focus only on the reachability condition.

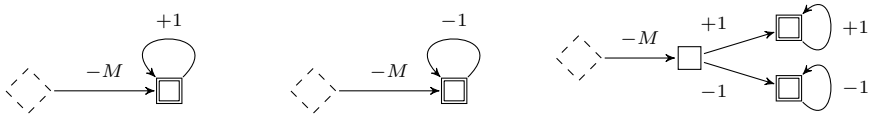
We first observe that because the counter takes on integral values, we can assume the inequality constraints are of the form $\mathsf{c} \sim M$ where $\sim \in \{\leq, \geq, \neq\}$. The idea of the construction is given in Figure 1. We replace each guarding condition with a series of gadgets (see Figures 2 and 3) which collectively check the condition. The opponent of the player that can choose the original transition controls whether these condition gadgets are entered, and therefore acts as the Verifier – if the transition is taken and the guard not satisfied, Verifier can move to the appropriate gadget and win; if the guard is satisfied, then any such



(a) Construction for $u \in V_\exists$ **(b)** Construction for $u \in V_\forall$

Fig. 1. Constructions for the edge (u, v) with guard $(0 \leq \mathtt{c} < 5) \wedge \mathtt{bit}_3(\mathtt{c})$ and weight w

move by Verifier will be losing and so the token ends up at the original vertex corresponding to the end of the original transition. Note that although the bit and $\overline{\mathtt{bit}}$ gadgets (Figure 3) contain guarded transitions, they are only inequality constraints and can be replaced in a similar manner.



(a) Gadget for $\mathtt{c} \leq M$ **(b)** Gadget for $\mathtt{c} \geq M$ **(c)** Gadget for $\mathtt{c} \neq M$

Fig. 2. Gadgets for inequality guards (Eve's perspective)

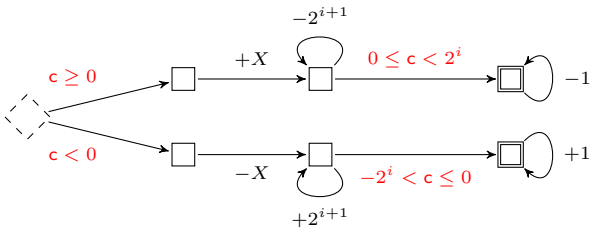


Fig. 3. Gadget for $\mathtt{bit}_i(\mathtt{c})$ $(X = 2^i)$ and $\overline{\mathtt{bit}_i}(\mathtt{c})$ $(X = 0)$ guards

Key to the correctness of the construction is the following easily checked result:

Lemma 1. *Suppose a token is at the diamond vertex in each gadget in Figures 2 and 3 and the counter has value c. Then*

- *Eve wins the gadget in Figure 2 (a) iff* $c \leq M$*;*
- *Eve wins the gadget in Figure 2 (b) iff* $c \geq M$*;*
- *Eve wins the gadget in Figure 2 (c) iff* $c \neq M$*;*
- *If* $X = 2^i$ *then Eve wins the gadget in Figure 3 iff* $\mathtt{bit}_i(c)$*; and*
- *If* $X = 0$ *then Eve wins the gadget in Figure 3 iff* $\overline{\mathtt{bit}_i}(c)$*.*

We observe that in Figures 2 and 3 the "name" of the gadget corresponds to the condition necessary and sufficient for *Eve* to win. Thus when Eve is Verifier in the construction (e.g. Figure 1 (b)) we need to use the gadgets corresponding to the complements of the guarding conjuncts – that is, we use the gadgets for the constraints that would make the guarding condition false.

This construction gives a reduction from Reach($\mathsf{SOCS}_{\mathbb{Z}}$) to Reach($\mathsf{SOCP}_{\mathbb{Z}}$) and from Reach($\mathsf{SOCS}_{\mathbb{N}}$) to Reach($\mathsf{SOCP}_{\mathbb{N}}$), and it is straightforward to check that the reduction is in log-space. As reductions in the other direction are trivial:

Lemma 2. *We have the following log-space equivalences:*

- Reach($\mathsf{SOCS}_{\mathbb{Z}}$) $\equiv_L$ Reach($\mathsf{SOCP}_{\mathbb{Z}}$)*, and*
- Reach($\mathsf{SOCS}_{\mathbb{N}}$) $\equiv_L$ Reach($\mathsf{SOCP}_{\mathbb{N}}$)

3.2 Removing Negative Counter Values

In this section we show how to simulate negative counter values with a counter than only takes non-negative values. We use a similar idea to the construction in [13] for the same problem on non-succinct one-counter games: storing the negation in the state space of the arena. That is, we duplicate the one-counter system, using one copy (with its weights and inequality constraints negated) to represent the behaviour with a negative counter value. The main difficulty with this construction is in the transition between the copies. In the non-succinct case this step was relatively straightforward because there is only counter value (0) where the counter changes from non-negative values to negative values. In the succinct case there are potentially exponentially many such points so we have to be more careful. The solution is to store the current counter value in a sufficiently high bit range so that we can perform the necessary arithmetic to set the correct counter value for the "negative" side. To do this, we use the gadget in Figure 4.

More precisely, given a $\mathsf{SOCS}_{\mathbb{Z}}$, $\mathcal{G} = (V, V_\exists, E, v_I, w, \gamma)$, we transform it into a $\mathsf{SOCS}_{\mathbb{N}}$, $\hat{\mathcal{G}}$, as follows:

1. Let $\mathcal{G}' = (V', V'_\exists, E', v'_I, w', \gamma')$ be a disjoint copy of $\mathcal{G}$ where $w'(e') = -w(e)$ and constraints in γ of the form $(\mathsf{c} \sim M)$ are replaced in γ' with $(-M \sim \mathsf{c})$.
2. For every edge $(u, v) \in E$ with weight w, add the gadget of Figure 4, where N is such that $-w < 2^N$, between $u \in V$ and v', the vertex in V' corresponding to v. Likewise for every edge $(u, v) \in E'$.
3. For every edge (u, v) with weight w add the constraint $(\mathsf{c} \geq -w)$.

The main result for establishing correctness of this construction is the following:

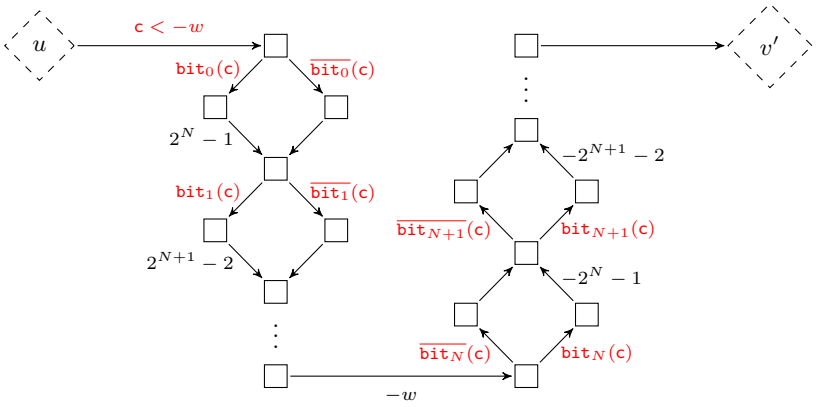


Fig. 4. Gadget for reduction from $\mathsf{SOCS}_{\mathbb{Z}}$ to $\mathsf{SOCS}_{\mathbb{N}}$

Lemma 3. *Suppose a token is on the vertex labelled u in Figure 4 with (non-negative) counter value $c < -w < 2^N$. Then the token can reach v' with counter value $-(c+w)$, without having a negative counter value.*

Following Lemma 3, it is easy to see that a reachability game on $\mathcal{G}$ specified by $T \subseteq V_\exists$ reduces to a reachability game on $\hat{\mathcal{G}}$ specified by $T \cup T'$, where $T' \subseteq V'_\exists$ is the set of vertices in V' which correspond to vertices in T. Furthermore, this reduction holds for the counter reachability problem. As the reverse reductions are trivial:

Lemma 4. *We have the following log-space equivalences:*

- $\textsc{Reach}(\mathsf{SOCS}_{\mathbb{Z}}) \equiv_L \textsc{Reach}(\mathsf{SOCS}_{\mathbb{N}})$*; and*
- $\text{C-}\textsc{Reach}(\mathsf{SOCS}_{\mathbb{Z}}) \equiv_L \text{C-}\textsc{Reach}(\mathsf{SOCS}_{\mathbb{N}})$.

3.3 From Reachability to Counter Reachability

For the final step toward Theorem 1, we reduce the reachability problem to the counter reachability problem. Our reduction utilizes the "long-range" property of succinct one-counter systems and is consequently not valid in the non-succinct case[3] and therefore does not contradict the separation established in [3].

Given a $\mathsf{SOCS}_{\mathbb{Z}}$ (or $\mathsf{SOCP}_{\mathbb{Z}}$, $\mathsf{SOCS}_{\mathbb{N}}$, $\mathsf{SOCP}_{\mathbb{N}}$) $\mathcal{G} = (V, V_\exists, E, v_I, w, \gamma)$ and a target set $T \subseteq V_\exists$, we construct a $\mathsf{SOCS}_{\mathbb{Z}}$ ($\mathsf{SOCP}_{\mathbb{Z}}$, $\mathsf{SOCS}_{\mathbb{N}}$, $\mathsf{SOCP}_{\mathbb{N}}$, respectively) $\mathcal{G}'$ as follows:

- Replace all $\texttt{bit}$ and $\overline{\texttt{bit}}$ predicates with the gadgets from Section 3.1;
- Double the weights of the edges in $\mathcal{G}$;

[3] We note that the reduction does work with transition systems with *unary-encoded* weights, but not to the more restrictive "short-range" semantics of non-succinct systems which only permit transitions of weight 0,±1.

- Replace all constraints of the form $(\mathsf{c} \sim M)$ with $(\mathsf{c} \sim 2M+1)$;
- Add a new initial vertex v'_I and a new sink (with 0-weighted loops) v'_f;
- Add an edge of weight $+1$ from v'_I to v_I; and
- Add edges of weight -1 from T to v'_f.

Due to parity arguments the counter in $\mathcal{G}'$ can only have value 0 at v'_I and v'_f. There is a natural one-to-one correspondence between plays in $\mathcal{G}$ and plays in $\mathcal{G}'$ that have not reached v'_f, and the counter in $\mathcal{G}$ has value c if and only if the corresponding counter value in $\mathcal{G}'$ is $2c+1$. It follows that v'_f can be reached with counter value 0 in $\mathcal{G}'$ if and only if the target set T can be reached with counter value 0 in $\mathcal{G}$. Thus Eve wins the counter reachability game on $\mathcal{G}$' if and only if she has a winning strategy in the original reachability game. Again, as the reverse reduction is trivial:

Lemma 5. *We have the following log-space equivalences:*

- $\textsc{Reach}(\mathsf{SOCS}_{\mathbb{Z}}) \equiv_L \text{C-}\textsc{Reach}(\mathsf{SOCS}_{\mathbb{Z}})$*;*
- $\textsc{Reach}(\mathsf{SOCP}_{\mathbb{Z}}) \equiv_L \text{C-}\textsc{Reach}(\mathsf{SOCP}_{\mathbb{Z}})$*;*
- $\textsc{Reach}(\mathsf{SOCS}_{\mathbb{N}}) \equiv_L \text{C-}\textsc{Reach}(\mathsf{SOCS}_{\mathbb{N}})$*; and*
- $\textsc{Reach}(\mathsf{SOCP}_{\mathbb{N}}) \equiv_L \text{C-}\textsc{Reach}(\mathsf{SOCP}_{\mathbb{N}})$.

4 EXPSPACE-completeness of Succinct One-Counter Games

In this section we establish the tight complexity bounds for the one-counter reachability and Büchi games. That is,

Theorem 2. *For* $* \in \{\mathsf{SOCS}_{\mathbb{Z}}, \mathsf{SOCP}_{\mathbb{Z}}, \mathsf{SOCS}_{\mathbb{N}}, \mathsf{SOCP}_{\mathbb{N}}\}$ *the problems* $\textsc{Reach}(*)$, $\text{C-}\textsc{Reach}(*)$ *and* $\textsc{StBüc}(*)$ *are* EXPSPACE*-complete.*

The upper bound is established in Lemma 8, and the matching lower bound is established in Lemma 11.

4.1 Upper Bound

Critical to the EXPSPACE membership of the reachability problems is the following result which follows almost directly from [14] and [8].

Theorem 3 ([8,14]). $\textsc{Büchi}(\mathsf{SOCP}_{\mathbb{N}})$ *is* EXPSPACE*-complete.*

To establish the upper bound for reachability games, we have the following log-space reduction:

Lemma 6. $\textsc{Reach}(\mathsf{SOCP}_{\mathbb{N}}) \leq_L \textsc{Büchi}(\mathsf{SOCP}_{\mathbb{N}})$.

Proof. Given a $\mathsf{SOCP}_{\mathbb{N}}$ and a target set $T \subseteq V_\exists$, we add to each vertex $v \in T$ the gadget in Figure 5. The target set for the Büchi condition consists of the new target vertices occurring in these gadgets. It is straightforward that in a $\mathsf{SOCS}_{\mathbb{N}}$ the gadget allows Eve to reach a target vertex (and remain there indefinitely) if and only if the counter value at v is 0. Thus Eve wins the new game with a Büchi condition if and only if she wins the original reachability game.

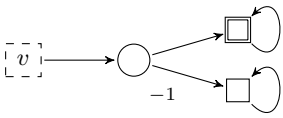


Fig. 5. Gadget for reduction from Reach($\mathsf{SOCP}_{\mathbb{N}}$) to Büchi($\mathsf{SOCP}_{\mathbb{N}}$)

For strong Büchi games we use the following log-space equivalence:

Lemma 7. StBüc($\mathsf{SOCS}_{\mathbb{N}}$) $\equiv_L$ Reach($\mathsf{SOCS}_{\mathbb{N}}$).

Proof. We give the proof for StBüc($\mathsf{SOCS}_{\mathbb{N}}$) $\leq_L$ Reach($\mathsf{SOCS}_{\mathbb{N}}$) as the reverse reduction is trivial. Given a $\mathsf{SOCS}_{\mathbb{N}}$, $\mathcal{G} = (V, V_\exists,$, with target set $T \subseteq V_\exists$, we construct a new one-counter reachability game as follows:

- The $\mathsf{SOCS}_{\mathbb{N}}$ consists of $|T|+1$ copies of $\mathcal{G}$ with a $(\mathsf{c}=0)$-guarded edge from (v,i) to $(v,i+1)$ for all $v \in T$ and $1 \leq i \leq |T|$,
- The initial vertex is $(v_0, 1)$, and
- The target set is $\{(v, |T|+1) : v \in T\}$.

Clearly Eve wins this game if and only if in the original game she can reach T with counter value 0 $|T|+1$ times. Hence if she wins the Büchi game she has a winning strategy in the reachability game. We now show the converse, that is if she can reach T $|T|+1$ times then she can reach some vertex in Y with counter value 0 infinitely often. More precisely we will show how to defeat any positional (w.r.t. the current state and counter value) strategy for Adam in the original Büchi game. It is well known [15] that such strategies are sufficient for winning strategies, thus this is sufficient for our result. Such a strategy has a natural interpretation in the reachability game, so Eve has a counterstrategy to ensure T is visited with counter value 0 $|T|+1$ times against this strategy. By the pigeon-hole principle there is some vertex $v \in T$ visited at least twice in the play. Hence Eve has a strategy (against Adam's strategy) to reach v with counter value 0 from both v_0 and v itself. Hence Eve can visit v with counter value 0 infinitely often in the original Büchi game.

These results, together with Theorem 1 and Corollary 1 establish the upper bounds for the reachability and strong Büchi conditions.

Lemma 8. *For* $* \in \{\mathsf{SOCS}_{\mathbb{Z}}, \mathsf{SOCP}_{\mathbb{Z}}, \mathsf{SOCS}_{\mathbb{N}}, \mathsf{SOCP}_{\mathbb{N}}\}$ *the problems* Reach($*$), C-Reach($*$) *and* StBüc($*$) *are in* EXPSPACE.

4.2 Lower Bounds

For the lower bound it would appear that we could use Theorem 3 together with a reduction similar to Lemma 7. Unfortunately, it is not clear how to do this without an exponential blow-up – it is not difficult to construct examples where

a vertex cannot be reached in fewer than a super-exponential number of steps, so counting the number of times the target set is visited is not practical. Instead, we revisit the hardness proof given in [8] and adapt it for reachability games. In fact, it follows from [8] that it is sufficient to give gadgets for computing the mod operator (where the arguments are stored in the counter value), and for simulating a polynomial-space bounded Turing Machine. We establish these results for $\mathsf{SOCS}_{\mathbb{N}}$ in the next two lemmas.

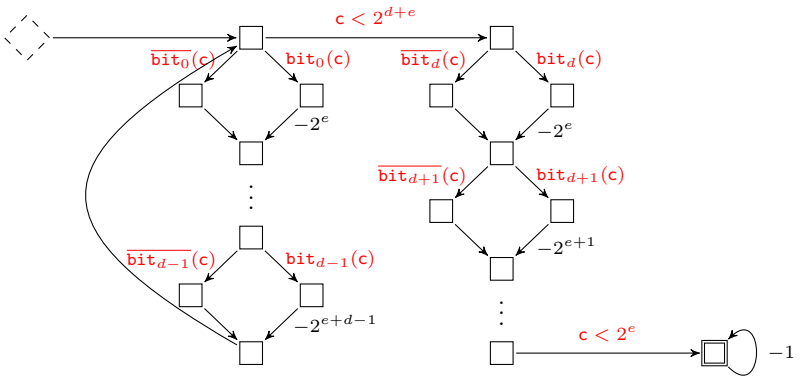


Fig. 6. Gadget for the mod operator

Lemma 9. *Suppose a token is at the diamond vertex in the gadget of Figure 6 with counter value $c = M + 2^d.C + 2^e.B$ where $M < 2^d$, $C < 2^{e-d}$ and $C < M$. Then Eve wins the gadget iff $C \equiv B \pmod{M}$.*

The next result follows directly from the standard reduction of space-bounded Turing Machines to one-counter machines (see e.g. [7]).

Lemma 10. *Let $\mathcal{M}$ be an (Alternating) Turing Machine which uses at most $q(n)$ space on any input of size n for some polynomial q. Then, for any $n \in \mathbb{N}$ there exists a reachability game $\mathcal{G}_n$, constructible in log-space, such that Eve wins with initial counter value $c < 2^n$ if and only if $\mathcal{M}$ accepts c.*

With these gadgets we can use a similar argument to [8] to give a reduction from any language in EXPSPACE to the reachability problem on $\mathsf{SOCS}_{\mathbb{N}}$.

Corollary 2. Reach($\mathsf{SOCS}_{\mathbb{N}}$) *is* EXPSPACE*-hard.*

Combining Theorem 1, Corollaries 1 and 2, and Lemma 7 gives us the necessary lower bound:

Lemma 11. *For $* \in \{\mathsf{SOCS}_{\mathbb{Z}}, \mathsf{SOCP}_{\mathbb{Z}}, \mathsf{SOCS}_{\mathbb{N}}, \mathsf{SOCP}_{\mathbb{N}}\}$ the problems* Reach($*$), C-Reach($*$) *and* StBüc($*$) *are* EXPSPACE*-hard.*

5 Conclusion and Further Work

We have shown the reachability problem for succinct one-counter systems is EXPSPACE-complete. This result is independent of whether the counter can take any integer value or just non-negative values, and whether transitions include counter constraints or not. We have also shown the counter reachability problem is EXPSPACE-complete under the same conditions, in contrast to the non-succinct case. In addition, we established EXPSPACE-completeness for Büchi games on one-counter systems when the Büchi condition is applied to configurations rather than states.

In this paper we considered $\mathsf{SOCS}_{\mathbb{N}}$ with *blocking semantics* – that is, transitions that would make the counter negative are forbidden. Another class to consider are $\mathsf{SOCS}_{\mathbb{N}}$ with *non-blocking semantics*: transitions that would make the counter negative are permitted, but the counter is set to 0. Such games are closely related to energy games [4]. The pseudo-polynomial time algorithm of that paper shows that the reachability problem for such one-counter systems is in EXPTIME, but it appears the complexity is much lower. Indeed, it is not difficult to show that Adam requires only memoryless, counter-free strategies, however this does not immediately yield a polynomial certificate for coNP-membership, as the single-player complexity is not known.

References

1. Abdulla, P.A., Cerans, K.: Simulation is decidable for one-counter nets (extended abstract). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 253–268. Springer, Heidelberg (1998)
2. Arul, A., Reichert, J.: The complexity of robot games on the integer line. In: QAPL, pp. 132–148 (2013)
3. Brázdil, T., Jančar, P., Kučera, A.: Reachability games on extended vector addition systems with states. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part II. LNCS, vol. 6199, pp. 478–489. Springer, Heidelberg (2010)
4. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003. LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)
5. Demri, S., Gascon, R.: The effects of bounding syntactic resources on presburger LTL. In: TIME, pp. 94–104 (2007)
6. Demri, S., Lazić, R., Sangnier, A.: Model checking freeze LTL over one-counter automata. In: Amadio, R. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 490–504. Springer, Heidelberg (2008)
7. Fearnley, J., Jurdziński, M.: Reachability in two-clock timed automata Is PSPACE-complete. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 212–223. Springer, Heidelberg (2013)
8. Göller, S., Haase, C., Ouaknine, J., Worrell, J.: Model checking succinct and parametric one-counter automata. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part II. LNCS, vol. 6199, pp. 575–586. Springer, Heidelberg (2010)

9. Haase, C., Kreutzer, S., Ouaknine, J., Worrell, J.: Reachability in succinct and parametric one-counter automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 369–383. Springer, Heidelberg (2009)
10. Jančar, P., Kučera, A., Moller, F.: Simulation and bisimulation over one-counter processes. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 334–345. Springer, Heidelberg (2000)
11. Jurdzinski, M., Sproston, J., Laroussinie, F.: Model checking probabilistic timed automata with one or two clocks. Logical Methods in Computer Science **4**(3) (2008)
12. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM **47**(2), 312–360 (2000)
13. Reichert, J.: On the complexity of counter reachability games. In: Abdulla, P.A., Potapov, I. (eds.) RP 2013. LNCS, vol. 8169, pp. 196–208. Springer, Heidelberg (2013)
14. Serre, O.: Parity games played on transition graphs of one-counter processes. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 337–351. Springer, Heidelberg (2006)
15. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science **200**, 135–183 (1998)

On Reachability-Related Games on Vector Addition Systems with States

Petr Jančar(✉)

Department Computer Science, FEI, Technical University of Ostrava (VŠB-TUO),
17. Listopadu 15, 70833 Ostrava, Czech Republic
petr.jancar@vsb.cz
http://www.cs.vsb.cz/jancar

Abstract. A new research result in this paper shows the decidability, and the TOWER upper bound on complexity, of solving parity multi-energy games (with given initial credit) in the framework of extended vector addition systems with states (where some components in the change vectors are not fixed but can be made arbitrarily large). The result is not deep w.r.t. the state-of-the-art, since it can be shown by a simple reduction to the version without the parity condition that was solved by Brázdil, Jančar, and Kučera (ICALP 2010). Besides giving the reduction, the main aim of the paper is to highlight the crucial ideas of a direct (self-contained) proof of the result; a particular novelty here is a natural attractor construction that seems to have not been used in this context so far.

1 Introduction

We deal with certain two-player turn-based *games* on *vector addition systems with states* (VASSs), which is a well-known structure (tightly related to classical place/transition Petri nets) that occurs in various contexts in computer science. We aim at highlighting the crucial ideas that lead to an algorithm solving a particular type of these games, namely the *parity multi-energy games* in the framework of extended VASSs. This yields a new research result, though surely not a deep one w.r.t. the state-of-the-art; the main aim is to explain the used techniques, which includes a natural attractor construction that seems to have not been used in this context so far.

Fig. 1 shows a 3-dimensional VASS $\mathcal{M}$, with 5 *control states* and 12 *transitions*; in fact, it is an *extended* VASS, an eVASS, in the sense of [8], since we also use ω (which can be instantiated by any nonnegative integer) in the change vectors. A d-dimensional eVASS is viewed as operating on d integer *counters*; it generates a transition system on the set of *configurations* $\textsc{Conf} = Q \times \mathbb{Z}^d$, as well as its restriction to the set $\textsc{Conf}_+ = Q \times \mathbb{N}^d$ of *nonnegative configurations* (where $\mathbb{Z}$, $\mathbb{N}$ are the sets of integers and of nonnegative integers, respectively).

The standard *reachability problem* asks if there is a path inside $\textsc{Conf}_+$ from one given configuration to another. (Referring to Fig. 1, we have,

Supported by the Grant Agency of the Czech Rep., project GAČR:15-13784S.

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 50–62, 2015.
DOI: 10.1007/978-3-319-24537-9_6

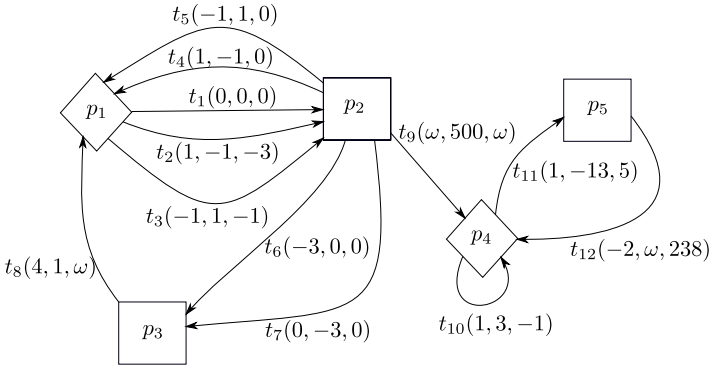


Fig. 1. Example of a 3-dim eVASS game $\mathcal{M}$

e.g., $p_1(4,8,12) \overset{t_2}{\mapsto} p_2(5,7,9) \overset{t_4}{\mapsto} p_1(6,6,9) \overset{t_1}{\mapsto} p_2(6,6,9)$.) This is an intriguing decidable problem that is still not sufficiently understood; it is a subject of an ongoing research as testified by, e.g., the recent papers [4,19] and the references therein.

In the contexts like controller synthesis, a natural model is *two-player turn-based games*, played by *System* and *Environment* (or Adam and Eve, or P0 and P1, etc.); here we refer to *Player* $\Box$ (or just $\Box$, she), and to *Player* $\Diamond$ (or just $\Diamond$, he), similarly as in [8]. In this case we have a partition $Q = Q_\Diamond \uplus Q_\Box$ of the set of control states ($Q_\Diamond = \{p_1, p_4\}$ and $Q_\Box = \{p_2, p_3, p_5\}$ in our example). The players create *a play*, i.e. an infinite path, *from a given configuration* $p\mathbf{u}$ so that in the current configuration $q\mathbf{v}$ Player $\Diamond$ chooses the next transition if $q \in Q_\Diamond$ and Player $\Box$ chooses it if $q \in Q_\Box$. (We assume that each control state has at least one outgoing transition.)

In this setting, the "alternating reachability problem", or the *two-player game with a reachability objective*, where Player $\Diamond$ tries to force reaching (an element of) a given *set of target configurations* while $\Box$ tries to keep the play outside this ("dangerous") set forever, is undecidable in general; i.e., the problem asking which of the players has a winning strategy from a given configuration is undecidable. This follows from the fact that we can easily construct a game in which Player $\Box$ can force Player $\Diamond$ to correctly simulate a given Minsky machine. We can refer to [1] where the authors studied a general framework of monotonic games of which eVASS-games are examples. We note that [1] belongs to the works that deal with a general framework of *well structured transition systems*. The paper [15] is another example, where also the natural *completions with limits* are studied; here we will use a standard example of such a completion, namely the set of *symbolic configurations* $\mathrm{SCONF} = Q \times (\mathbb{Z} \cup \{\omega\})^d$.

There are natural variants of VASS-games that are decidable though they do not belong to the decidable fragment shown in [1]; we can name [21] as an example. The most relevant for us here are so called *multi-energy games* where we ask if $\Box$ can maintain the counters nonnegative, i.e., if she can force moving inside CONF_+ forever, when starting from a given configuration $p\mathbf{u} \in \mathrm{CONF}_+$.

The complementary question asks if $\Diamond$ can force reaching an element of the set of configurations in which at least one counter is negative (i.e., the respective "type of energy" has been exhausted and is missing). In the terms of energy games we speak about the *given initial credit* (GIC) *problem* when starting from a concrete $p\mathbf{u}$; the *arbitrary initial credit* (AIC) *problem* asks for a given $p \in Q$ if there is an initial credit $\mathbf{u} \in \mathbb{N}^d$ such that Player $\Box$ wins from $p\mathbf{u}$.

An algorithm solving the GIC problem (for extended VASSs) was given in [8]; solving the AIC problem served as the basic case. The problem was shown to belong to TOWER (the "smallest" nonelementary complexity class); it was shown, in principle, to be solvable in d-exponential time for dimension d.

There is now a row of papers in the literature that deal with various variants of multi-energy games and/or related multi-min-payoff games and/or related issues; we can name [2,3,6,7,10–12,14,16,22], without any claim on the completeness of this list. The most recent, and indeed remarkable, result has been achieved by Jurdziński, Lazić, and Schmitz [17], who have shown the pseudo-polynomiality of multi-energy games when the dimension is fixed, and 2-ExpTime-completeness in general (in the framework of standard VASSs). We can mention that a first step in the "pseudo-polynomiality direction" was presented at the Workshop on Reachability Problems by Chaloupka in 2010 [9].

It would be surely useful to have *an exposition of the main research ideas* in the above (and related) papers; as usual, the same ideas are often hidden in the various technical formalisms, they recur in various contexts, etc. Nevertheless, to give such a survey would be a difficult task. This paper is meant just as an attempt to make *a modest step in this direction*, by highlighting the crucial ideas that lead to an algorithm solving a problem in this area. The shown result is new, though not deep w.r.t. the state-of-the-art; it serves us mainly for an illustration.

Concretely, we show an algorithm for solving the GIC problem on eVASS games where the $\Box$-objective is *extended by a parity condition*. Each control state $q \in Q$ has an attached *rank* $\text{RANK}(q) \in \mathbb{N}$. The objective of $\Box$ is not only to keep the play in Conf_+ but, moreover, to guarantee that the *least rank* of control states *visited infinitely often* is *even*. In [10] the problem was studied for the dimension 1; this version belongs to a class of problems that are known to be in **NP** and co-**NP** but for which the question of polynomiality is open. The AIC problem for (general) parity-VASS games was studied in [11]; the decidability of this problem was then used in [3] to show the decidability of the respective GIC problem. The concluding remarks of [17] make clear that the new approach has not been so far applied to the case with the parity condition; the authors only mention that a TOWER upper bound could be derived.

The authors of [17] also mention that they improve the complexity results of [8] for VASS-games, not mentioning the case of eVASS-games. Here, in Section 2, we show that the GIC problem for eVASS-games extended with the parity condition, *the GIC-parity-eVASS problem*, can be easily reduced to the normal ("non-parity") GIC-eVASS problem; the possibility of using ω (i.e., an arbitrary nonnegative value) for a counter-change is especially convenient in this

reduction, even when we start with standard VASSs. This easy reduction thus also yields a (more general) alternative to the decidability proof in [3]; moreover, the result of [8] then immediately puts the GIC-parity-eVASS problem in TOWER. (We can also note the opposite direction, namely that the GIC-eVASS problem is easily reducible to the GIC-parity-VASS problem: to arrange that $\Box$ can add an arbitrarily large value to a counter, we add an appropriate odd-ranked control state that has a loop that increments the counter.)

For the sake of highlighting several research ideas and techniques, in Section 3 we give a direct proof yielding the mentioned TOWER membership. The section has three subsections, roughly corresponding to three main ideas.

Section 3.1 highlights the notion of *mixing two* $\Box$*-strategies* (the game-strategies of Player $\Box$) in the context of (VASS- or) eVASS-games. This also underpins an inductive proof of a crucial fact of *pruning*, saying that for each $q \in Q_\Diamond$ that is universally $\Diamond$-winning (i.e., $\Diamond$ has a winning strategy from every configuration $q\mathbf{u}$) we can fix just one outgoing transition (and prune away the others) without changing $Win_\Diamond$, where $Win_\Diamond \subseteq \textsc{Conf}$ is the set of configurations for which $\Diamond$ has winning strategies.

Section 3.2 recalls some standard results on *detecting nonnegative cycles* in VASSs (and eVASSs), and on *exponential bounds on the lengths of such shortest cycles*. The results entail that deciding if a given $q \in Q$ is universally $\Diamond$-winning (which is the complement of the AIC problem) is in **NP**; it is **NP**-complete, in fact, for which we can refer to [22]. The strategy-mixing from Section 3.1 then yields an exponentially bounded initial credit that suffices for $\Box$-winning in any given $q \in Q$ if q is not universally $\Diamond$-winning.

Section 3.3 then describes *an algorithm* for solving the GIC-parity-eVASS problem, *based* on solving the AIC problem and *on attractor constructions*. This natural use of (alternating-reachability) attractors does not seem to have appeared in the literature in this context so far (while some methods of forward analysis or of bounded counters have been used instead).

1.1 Strategies, Winning Strategies, Winning Regions $Win_\Box$, $Win_\Diamond$

We make some notions and notation more formal, to the extent that should suffice for avoiding possible confusions; we refer to Fig. 1 when giving concrete examples. In particular we make precise the (standard) notion of strategies of the players, in the form that is technically convenient for our reasoning.

A *d-dim eVASS game* $\mathcal{M}$ is given by a finite directed multigraph (Q, T) where the nodes $p \in Q$ are also called the *control states* and the edges $t \in T$ are called the *transitions*; moreover, we have a partition $Q = Q_\Diamond \uplus Q_\Box$, and each $t \in T$ has an associated *change-vector* $\Delta(t) \in \mathbb{Z}^d_\omega$, where $\mathbb{Z}^d_\omega = (\mathbb{Z} \cup \{\omega\})^d$ (e.g., $\Delta(t_{12}) = (-2, \omega, 238)$). More precisely, we allow ω only in $\Delta(t)$ for $\Box$*-transitions* t, which are the transitions outgoing from $\Box$*-states*, i.e., from the control states $q \in Q_\Box$; for $\Diamond$*-transitions* t we have $\Delta(t) \in \mathbb{Z}^d$. We also require that *each control state* $q \in Q$ *has at least one outgoing transition*; we would add a loop t on q with $\Delta(t) = \mathbf{0}$ otherwise, where $\mathbf{0}$ stands for the d-dimensional zero vector.

An *instantiated path* (in a d-dim eVASS game $\mathcal{M}$) is a (finite or infinite) path

$$q_0 \xrightarrow{(t_1,\delta_1)} q_1 \xrightarrow{(t_2,\delta_2)} q_2 \xrightarrow{(t_3,\delta_3)} \dots \tag{1}$$

where $q_0 \xrightarrow{t_1} q_1 \xrightarrow{t_2} q_2 \xrightarrow{t_3} \dots$ is a path in the graph of $\mathcal{M}$ (or a "walk", since we allow $q_i = q_j$ for $i \neq j$) and δ_i is an instance of $\Delta(t_i)$, i.e., $\delta_i \in \mathbb{Z}^d$ arises from $\Delta(t_i)$ by replacing each ω with a nonnegative integer. (We have $\delta_i = \Delta(t_i)$ when $\Delta(t_i) \in \mathbb{Z}^d$, which is always the case when $q_{i-1} \in Q_\Diamond$.) Each control state $q \in Q$ is also viewed as a zero-length path (from q to q).

A *$\Diamond$-strategy* is a function that attaches, to each finite instantiated path

$$q_0 \xrightarrow{(t_1,\delta_1)} q_1 \xrightarrow{(t_2,\delta_2)} q_2 \cdots \xrightarrow{(t_m,\delta_m)} q_m \tag{2}$$

where $q_m \in Q_\Diamond$, the pair $(t, \Delta(t))$ for an outgoing transition t of q_m. A *$\Box$-strategy* attaches, to each path (2) where $q_m \in Q_\Box$, a pair (t, δ) where t is an outgoing transition of q_m and δ is an instance of $\Delta(t)$.

An infinite instantiated path of the form (1) is also called a *play*, or a *play from* q_0; the *play* is *consistent with a $\Diamond$-strategy* (*a $\Box$-strategy*) σ if for any $q_i \in Q_\Diamond$ ($q_i \in Q_\Box$) we have $(t_{i+1}, \delta_{i+1}) = \sigma(q_0 \xrightarrow{(t_1,\delta_1)} q_1 \xrightarrow{(t_2,\delta_2)} q_2 \cdots \xrightarrow{(t_i,\delta_i)} q_i)$. A *play from a configuration* $q_0\mathbf{u}$ is a play from q_0.

We consider the *parity eVASS games*, where we also have $\text{RANK} : Q \to \mathbb{N}$. A *$\Box$-strategy σ is winning from* $q_0\mathbf{u}$ if any play (1) that is consistent with σ satisfies the $\Box$-objective, i.e.: $\mathbf{u} + \sum_{j=1}^{i} \delta_j$ is nonnegative for all $i \in \mathbb{N}$ and the least $\text{RANK}(q_i)$ visited infinitely often is even. We observe that if a $\Box$-strategy σ is winning from $q_0\mathbf{u}$ then it is winning from every $q_0\mathbf{v}$ where $\mathbf{u} \leq \mathbf{v}$ (where $\leq$ is taken component-wise). Hence the set $Win_\Box$ of configurations for which $\Box$ has winning strategies is upward closed, when $p\mathbf{u} \leq q\mathbf{v}$ is defined as $p = q$ and $\mathbf{u} \leq \mathbf{v}$. Therefore $Win_\Box$ is the upward closure of $Min_\Box$, the set of its minimal elements, which is finite by Dickson's Lemma.

Similarly, $Win_\Diamond$ is the downward closure of $Max_\Diamond$ in the completion $\text{SCONF} = Q \times \mathbb{Z}_\omega^d$ of *symbolic configurations*; we include $p\mathbf{u} \in \text{SCONF}$ in $Win_\Box$ iff there is an instance $p\mathbf{v}$ of $p\mathbf{u}$ (where $\mathbf{v}$ arises from $\mathbf{u}$ by replacing all ω's with some nonnegative integers) such that $p\mathbf{v} \in Win_\Box$; otherwise $p\mathbf{u}$ is in $Win_\Diamond$. By standard determinacy results we have $\text{SCONF} = Win_\Diamond \uplus Win_\Box$.

E.g., the symbolic configurations inside the nodes of the graph in Fig. 2 (which will be discussed later) constitute $Max_\Diamond$ for $\mathcal{M}$ in Fig. 1, when assuming $\text{RANK}(p_1) = 0$ and $\text{RANK}(p_i) = i$ for all $i \in \{2, 3, 4, 5\}$. We can check that the $\Diamond$-strategy described as "if the last performed transition was t_4, then perform t_3; if the last performed transition was t_5, then perform t_2; perform t_{10} in p_4 (and otherwise do whatever)" is winning from $p_2(2, 2, x)$ for any $x \in \mathbb{Z}$.

The result that we will show can be formulated as follows:

Theorem 1. *There is an algorithm that, given a parity eVASS game $\mathcal{M}$, constructs $Min_\Box$, $Max_\Diamond$, and a description of (finite-memory) strategies winning for Player $\Box$ and Player $\Diamond$ in $Win_\Box$ and $Win_\Diamond$, respectively. The complexity of the algorithm can be bounded by the function* TOWER*, defined by* $\text{TOWER}(0) = 1$ *and* $\text{TOWER}(n+1) = 2^{\text{TOWER}(n)}$.

2 Reducing GIC-parity-eVASS to GIC-eVASS

Using the results from [8], Theorem 1 in principle follows from the next lemma.

Lemma 1. *The GIC-parity-eVASS problem is polynomial-time reducible to the GIC-eVASS problem.*

The lemma constitutes a natural case for applying the general method of encoding a parity condition into a safety condition. An instance of this method was used in [11] in the context of the AIC-problem for VASSs but the general method was already used in some previous works, as is also indicated in [11].

We now describe a relevant reduction, which conveniently uses the possibility of ω's in $\Delta(t)$ of $\Box$-transitions. For an instance $\mathcal{M}, p_0\mathbf{u}$ of the GIC-parity-eVASS problem we construct an instance $\mathcal{M}', p_0'\mathbf{u}'$ of the GIC-eVASS problem as follows:

1. Starting with $\mathcal{M}$, for each odd rank r (where $\textsc{rank}(q) = r$ for some $q \in Q$) we add a new counter c_r, and modify the change-vectors $\Delta(t)$ of transitions so that each entering q with $\textsc{rank}(q) = r$ entails decrementing c_r.
2. We add a fresh control state p_0' in $Q_\Box$ and a transition $p_0' \xrightarrow{t} p_0$ where the vector $\Delta(t)$ has ω in the components corresponding to all above counters c_r, and 0 in the other components. The new initial configuration will be $p_0'\mathbf{u}'$ where $\mathbf{u}'$ arises from $\mathbf{u}$ by filling zeros in all c_r. (Hence Player $\Box$ can put any nonnegative numbers in all c_r in the first step.)
3. For each q with an even $\textsc{rank}(q)$ we add a fresh state $q' \in Q_\Box$ and the transition $q' \xrightarrow{t'} q$ where $\Delta(t')$ has ω in the components corresponding to all c_r with $r > \textsc{rank}(q)$ and 0 elsewhere; all original transitions leading to q are redirected to q'. (When entering an even-rank state q, Player $\Box$ gets a possibility to increase the counters c_r for all odd ranks r bigger than $\textsc{rank}(q)$.)

The construction thus increases the dimension: if $\mathcal{M}$ is a d-dim parity-eVASS game with ℓ odd ranks, then $\mathcal{M}'$ is a $(d+\ell)$-dim eVASS game. Before proving the correctness of the reduction we note a general fact that any winning strategy of $\Box$ can be made "cycle-free" as explained below.

We say that an instantiated path π of the form (2) is a *good cycle* if $m \geq 1$, $q_0 = q_m$, $\Delta(\pi) = \sum_{i=1}^{m} \delta_i \geq \mathbf{0}$, and the least $\textsc{rank}(q_i)$ (for $i \in \{0, 1, \ldots, m\}$) is even. If an instantiated path π of the form (2) contains a good cycle, then by $\textsc{trim}(\pi)$ we mean the path $\pi_1\pi_3$ where $\pi = \pi_1\pi_2\pi_3$ and $\pi_2 = q_i \xrightarrow{(t_{i+1},\delta_{i+1})} q_{i+1} \cdots \xrightarrow{(t_j,\delta_j)} q_j$ is the first good cycle in π (for the least j primarily and the least i secondarily). Generally by $\pi_1\pi_2$ we refer to the natural concatenation of paths, which assumes that the end-state of π_1 and the start-state of π_2 are the same. (Recall that we also allow the trivial zero-length paths of the form q.) If π does not contain a good cycle, then $\textsc{trim}(\pi) = \pi$.

A $\Box$-strategy σ is *cycle-free* if for any $\pi \in \textsc{dom}(\sigma)$ we have $\sigma(\pi) = \sigma(\textsc{trim}(\pi))$. We have already noted that if a $\Box$-strategy wins from $q\mathbf{u}$, then it wins from $q\mathbf{v}$ for any $\mathbf{v} \geq \mathbf{u}$. We can thus derive that for any configuration $q_0\mathbf{u} \in \mathit{Win}_\Box$ there is a cycle-free $\Box$-strategy winning from $q_0\mathbf{u}$.

Now we show the correctness of the above reduction:

a/ Suppose that $\Box$ has a winning strategy in the (parity) game given by $\mathcal{M}$ and $p_0\mathbf{u}$; let σ be such a cycle-free strategy. By Dickson's Lemma (and the finite branching of $\Diamond$'s choices) there is $\ell \in \mathbb{N}$ that bounds the lengths of the prefixes with no good cycles in the plays from p_0 that are consistent with σ. Hence $\Box$ can use the analogous strategy in the "non-parity" game $\mathcal{M}'$ from $p'_0\mathbf{u}'$, while always adding $\ell{+}1$, say, to the respective counters c_r when performing the c_r-increasing transitions; she thus keeps all counters nonnegative.

b/ If $\Diamond$ has a winning strategy σ in the game $\mathcal{M}$ from $p_0\mathbf{u}$, then any play consistent with σ either enters a configuration with a negative counter (i.e., in the respective play $\mathbf{u} + \sum_{j=1}^{i} \delta_j$ has a negative component for some i) or the least rank visited infinitely often is odd. The analogous $\Diamond$-strategy is then winning in the "non-parity" game $\mathcal{M}'$ from $p'_0\mathbf{u}'$, since in this case any consistent play obviously enters a configuration with a negative counter (which might be c_r for some odd rank r).

3 A Direct Proof of Theorem 1

We now give a self-contained proof that can be viewed as based on three main ideas, explained in the following three subsections. We do not describe all technical details when the respective proof steps should be clear.

3.1 Mixing $\Box$-Strategies, and Pruning $\Diamond$-Transitions

An important step for constructing $Max_\Diamond$ (which naturally entails constructing $Min_\Box$) is to decide which states $q \in Q$ are $\Diamond$-*universal*, i.e., which satisfy $q(\omega, \ldots, \omega) \in Win_\Diamond$. We will base our reasoning on an induction on the number D of $\Diamond$'s choices, defined as $D = |\{t \mid t \text{ is a } \Diamond\text{-transition}\}| - |Q_\Diamond|$; hence $D = 0$ iff each $q \in Q_\Diamond$ has exactly one outgoing transition.

Consider a parity-eVASS game $\mathcal{M}$ and a control state $q^s \in Q_\Diamond$ (the superscript s refers to "splitting") with two different outgoing transitions t_1^s, t_2^s (and maybe some others). Let $\mathcal{M}_1$ arise from $\mathcal{M}$ by removing t_2^s, and let $\mathcal{M}_2$ arise from $\mathcal{M}$ be removing t_1^s. We now suggest a way how to combine two $\Box$-strategies, σ_1 for $\mathcal{M}_1$ and σ_2 for $\mathcal{M}_2$, to yield a strategy $\sigma = \textsc{mix}(\sigma_1, \sigma_2)$ for $\mathcal{M}$.

This combination of strategies is based on the fact that any (finite or infinite) path $\pi = q_0 \xrightarrow{t_1} q_1 \xrightarrow{t_2} \cdots$ in (the graph of) $\mathcal{M}$ can be naturally viewed as *a merge of a path from* q_0 *in* $\mathcal{M}_1$ *and a path from* q^s *in* $\mathcal{M}_2$.

E.g., if in Fig. 1 we put $q^s = p_1$, $t_1^s = t_2$, $t_2^s = t_3$, then the path

$$\pi = p_2 \xrightarrow{t_6} p_3 \xrightarrow{t_8} \mathbf{p_1} \xrightarrow{\mathbf{t_3}} p_2 \xrightarrow{t_6} p_3 \xrightarrow{t_8} p_1 \xrightarrow{t_3} p_2 \xrightarrow{t_6} p_3 \xrightarrow{t_8} \mathbf{p_1} \xrightarrow{\mathbf{t_2}} p_2$$

can be split into the path $\pi_1 = p_2 \xrightarrow{t_6} p_3 \xrightarrow{t_8} \mathbf{p_1} \xrightarrow{\mathbf{t_2}} p_2$ in $\mathcal{M}_1$ and the path $\pi_2 = \mathbf{p_1} \xrightarrow{\mathbf{t_3}} p_2 \xrightarrow{t_6} p_3 \xrightarrow{t_8} p_1 \xrightarrow{t_3} p_2 \xrightarrow{t_6} p_3 \xrightarrow{t_8} \mathbf{p_1}$ in $\mathcal{M}_2$. We have shown in boldface the first occurrence of a transition that is not present in $\mathcal{M}_1$, then after it the first occurrence of a transition that is not present in $\mathcal{M}_2$, and we would continue in this alternation if π was longer. We now formalize this intuitive idea.

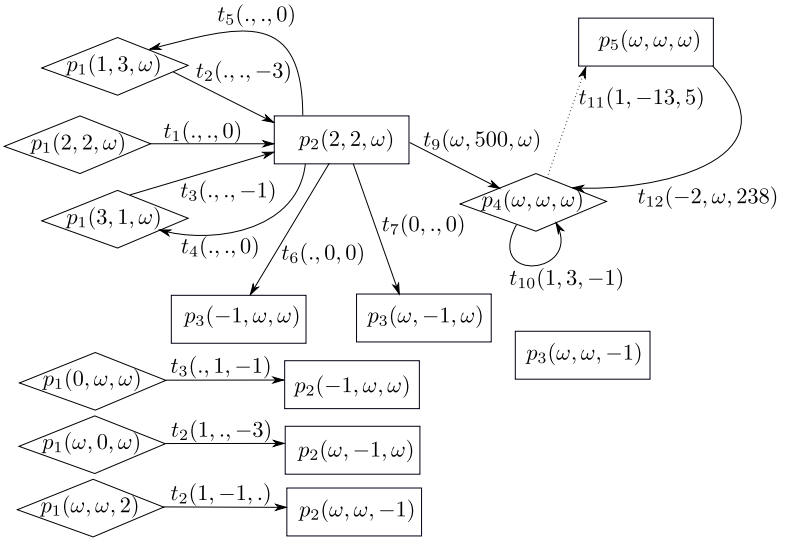


Fig. 2. A 3-dim eVASS game $\mathcal{S}$ ($\Diamond$-Strategy) related to $\mathcal{M}$ from Fig. 1

Consider a path $\pi = q_0 \xrightarrow{t_1} q_1 \xrightarrow{t_2} \cdots$ in $\mathcal{M}$, and a path π_2 in $\mathcal{M}_2$ that finishes in q^s; we can have that π_2 is just q^s (a trivial path from q^s to q^s). We transform the pair (π, π_2) to $\textsc{Trans}(\pi, \pi_2) = (\pi', \pi_2')$ as follows:

1. If t_2^s does not occur in π, then $\textsc{Trans}(\pi, \pi_2) = (\pi, \pi_2)$. (Hence (π, π_2) is a fixpoint of the mapping $\textsc{Trans}$.)
2. If $\pi = \rho_1\rho_2$ where $\rho_2 = q^s \xrightarrow{t_2^s} \ldots$ starts with the first occurrence of t_2^s in π, then:
   a) if ρ_2 does not contain t_1^s, then $\textsc{Trans}(\pi, \pi_2) = (\rho_1, \pi_2\rho_2)$, and $(\rho_1, \pi_2\rho_2)$ is defined as a fixpoint of $\textsc{Trans}$;
   b) if $\rho_2 = \rho_2'\rho_2''$ where $\rho_2'' = q^s \xrightarrow{t_1^s} \ldots$ starts with the first occurrence of t_1^s in ρ_2, then $\textsc{Trans}(\pi, \pi_2) = (\rho_1\rho_2'', \pi_2\rho_2')$.

For a (finite or infinite) path π in $\mathcal{M}$, we put $\textsc{split}(\pi) = (\pi_1, \pi_2)$ where (π_1, π_2) is the fixpoint of $\textsc{Trans}$ reached when repeatedly applying $\textsc{Trans}$ to the pair (π, q^s). The fixpoint is "reached" after infinitely many iterations when π is infinite and contains infinitely many occurrences of both t_1^s and t_2^s. We note that π_1 is path in $\mathcal{M}_1$ and π_2 is a path in $\mathcal{M}_2$ starting in q^s. The function $\textsc{split}(\pi)$ is naturally extended to the instantiated paths $\pi = q_0 \xrightarrow{(t_1,\delta_1)} q_1 \xrightarrow{(t_2,\delta_2)} \cdots$ (by simply ignoring δ_i).

Now we define the *mixed* $\Box$*-strategy* $\sigma = \textsc{mix}(\sigma_1, \sigma_2)$ in $\mathcal{M}$ as follows: For an instantiated path π finishing in some $q \in Q_\Box$ we consider $\textsc{split}(\pi) = (\pi_1, \pi_2)$; if π_1 finishes in q (and thus π_2 in q^s, where $q \neq q^s$ since $q \in Q_\Box$ and $q^s \in Q_\Diamond$), then we put $\sigma(\pi) = \sigma_1(\pi_1)$; if π_2 finishes with q (and thus π_1 with q^s), then we put $\sigma(\pi) = \sigma_2(\pi_2)$.

We note that any (infinite) play consistent with σ is a merge of two plays consistent with σ_1 and σ_2, respectively; more precisely, it is a merge of a play with

a prefix of another play (which might be finite). The nature of the $\Box$-objective, including the parity condition, is such that merging two plays satisfying the objective yields a play that also satisfies the objective, on condition that each play operates on its "own" counter values (that are summed up in the combined play). We thus easily observe:

Lemma 2. *If σ_1 is a $\Box$-strategy in $\mathcal{M}_1$ that is winning from $q_0\mathbf{u}$, and σ_2 is a $\Box$-strategy in $\mathcal{M}_2$ that is winning from $q^s\mathbf{v}$, then $\sigma = \text{MIX}(\sigma_1, \sigma_2)$ is a $\Box$-strategy in $\mathcal{M}$ that is winning from $q_0(\mathbf{u}+\mathbf{v})$; if, moreover, $q_0 = q^s$, then σ is winning from $q^s(\mathbf{u}+\mathbf{v})$.*

This also entails that if q^s is $\Diamond$-universal (i.e., $q^s(\omega, \ldots, \omega) \in \mathit{Win}_\Diamond$), then q^s must be $\Diamond$-universal in at least one of $\mathcal{M}_1$, $\mathcal{M}_2$. In this case at least one of the transitions t_1^s, t_2^s can be removed from $\mathcal{M}$ so that $\mathit{Win}_\Diamond$ (and thus also $\mathit{Win}_\Box$) does not change. We thus get:

Lemma 3. *(Pruning Lemma.) For each $q \in Q_\Diamond$ that is $\Diamond$-universal there is one outgoing transition t of q such that $\mathit{Win}_\Diamond$ does not change when we remove ("prune away") the other outgoing transitions of q.*

After such pruning at each $\Diamond$-universal $q \in Q_\Diamond$, the set $Q^{D\text{-}univ} = \{q \in Q \mid q(\omega, \ldots, \omega) \in \mathit{Win}_\Diamond\}$ is "transition-closed", i.e., there is no transition $q \xrightarrow{t} q'$ where $q \in Q^{D\text{-}univ}$ and $q' \notin Q^{D\text{-}univ}$. (In our example, $Q^{D\text{-}univ} = \{p_4, p_5\}$.)

3.2 Detecting Nonnegative Cycles, and Their (Exponential) Lengths

The question if $q \in Q^{D\text{-}univ}$ is complementary to the question if the answer to the AIC problem for q is positive. Lemma 3 and the fact that $Q^{D\text{-}univ}$ is transition-closed after the respective pruning show that there is a restriction of $\mathcal{M}$ arising by removing all but one outgoing transition for each $q \in Q_\Diamond$ such that $Q^{D\text{-}univ}$ remains unchanged (though $\mathit{Win}_\Diamond$ might shrink for non-$\Diamond$-universal states).

This suggests to start with considering the basic case, where $\Diamond$ has no choice, i.e., $D = |\{t \mid t \text{ is a } \Diamond\text{-transition}\}| - |Q_\Diamond| = 0$. This is equivalent to the case with $Q_\Diamond = \emptyset$, i.e., to the one-player game where Player $\Box$ freely constructs a play. In this case a sufficient initial credit for a given control state q_0 exists iff there is a path from q_0 to some q which is on a nonnegative cycle $q \xrightarrow{t_1} q_1 \xrightarrow{t_2} q_2 \cdots \xrightarrow{t_m} q_m$, where $m \geq 1$, $q_m = q$, and $\sum_{i=1}^{m} \Delta(t_i) \geq \mathbf{0}$ (we put $z + \omega = \omega + z = \omega + \omega = \omega$ for any $z \in \mathbb{Z}$), and, moreover, the least $\text{RANK}(q_i)$, $i \in \{1, 2, \ldots, m\}$, is even.

The existence and the structure of such cycles can be found in polynomial time, by using linear programming (LP). Let us recall the idea from [18]. We introduce a (nonnegative real) variable x_t for each transition t, and the linear constraints $x_t \geq 0$, $\sum_{t \in T} x_t \geq 1$, $\sum_{t \in T} x_t \cdot \Delta(t) \geq \mathbf{0}$, and the "Kirchhoff's Law" constraint $\sum_{t, target(t)=p} x_t = \sum_{t, source(t)=p} x_t$ for each control state p. The Parikh image of a nonnegative cycle $C = t_1 t_2 \ldots t_m$ (i.e. the vector of dimension $|T|$ where the component corresponding to t gives the number of occurrences of t in C) is obviously an integer solution (of our set of constraints).

On the other hand, a polynomial time algorithm for linear programming finds a rational solution, if there is one, which easily yields an integer solution (by multiplying with a common denominator). We aim at getting an Eulerian path but the transitions t with positive x_t (in the solution) can create a nonnegative multicycle (a set of separated cycles in the graph of $\mathcal{M}$) instead of one cycle. By a repeated use we can compute all transitions that can appear in a nonnegative multicycle. On the graph restricted to such transitions we solve the problem for each connected component separately.

The possible presence of ω's in $\Delta(t)$ can be handled easily; moreover, we can also easily concentrate on the cycles where the least visited rank is even. Hence already the existence of a polynomial-time algorithm solving LP entails that the lengths of such shortest cycles can be written in polynomial space, w.r.t. the size of $\mathcal{M}$, denoted $||\mathcal{M}||$, where the numbers are presented in binary; the lengths are thus exponential in $||\mathcal{M}||$. We can thus easily derive:

Lemma 4. *1. The problem asking if $q(\omega,\ldots,\omega) \in Win_\Diamond$ for a given parity-eVASS game $\mathcal{M}$ and a control state q is in* **NP**.
2. For a parity-eVASS game $\mathcal{M}$ there is a (polynomially computable) bound $B_0 \in \mathbb{N}$, exponential in $||\mathcal{M}||$, such that for any control state q we have that $q(\omega,\ldots,\omega) \in Win_\Box$ implies $q(B_0,\ldots,B_0) \in Win_\Box$.

Proof. (Sketch.)
1. We guess a restriction of $\mathcal{M}$ with just one outgoing transition for each $p \in Q_\Diamond$, and then verify (in polynomial-time) that no respective cycle is reachable from the given q. (The idea to combine the analogue of Lemma 3 in [8] with the cycle detecting in [18] to get the **NP**-membership first appeared in the FSTTCS'10 publication mentioned at [22]; also **NP**-hardness was shown there.)
2. As already discussed, we have such an exponential bound B'_0 in the case with no $\Diamond$-choice, i.e. with $D = |\{t \mid t \text{ is a } \Diamond\text{-transition}\}| - |Q_\Diamond| = 0$. In the general case it suffices to put $B_0 = 2^D \cdot B'_0$, as can be verified by a repeated use of Lemma 2. □

Remark. By Rackoff's approach [20] concentrating on the effects of simple cycles, and by the bounds from [5], it is straightforward to show that the lengths of our shortest nonnegative cycles are pseudopolynomial when the dimension is fixed. It is the multiplication by 2^D that had to be circumvented in [17].

3.3 Attractor-Based Algorithm for the GIC-parity-eVASS Problem

Assume a d-dim parity-eVASS game $\mathcal{M}$. We aim to give an algorithm constructing $Max_\Diamond$, the set of maximal elements in $Win_\Diamond \subseteq \text{SConf} = Q \times \mathbb{Z}^d_\omega$, and a description of a (generalized) $\Diamond$-strategy winning in $Win_\Diamond$. (A description relevant to $\mathcal{M}$ in Fig. 1 is depicted in Fig. 2.) By $p\mathbf{u}$, $q\mathbf{v}$, etc. we further refer to the symbolic configurations (elements of SConf). By $\mathbf{u}_\ell$ we refer to the ℓ-th component of $\mathbf{u}$, and we define the *support* of $p\mathbf{u}$ as $\text{supp}(p\mathbf{u}) = \{\ell \in \{1,2,\ldots,d\} \mid \mathbf{u}_\ell \neq \omega\}$. The algorithm uses a "program variable" Solved containing the (stepwise increasing) set of the symbolic configurations that have

been already shown to belong to the set $Win_\Diamond$ (which is downward closed); initially we have $\textsc{Solved} = \emptyset$.

Algorithm (whose input is a d-dim parity-eVASS game $\mathcal{M}$):

1. Process the support $\emptyset$, i.e., divide the set $\mathscr{C}_\emptyset = \{p(\omega, \ldots, \omega) \mid p \in Q\}$ between $Win_\Diamond$ and $Win_\Box$, while also fixing one appropriate outgoing transition for each $p \in Q_\Diamond$ where $p(\omega, \ldots, \omega) \in Win_\Diamond$ (which can done by Lemma 4(1) and its proof). Add (the downward closure of) all $p(\omega, \ldots, \omega) \in Win_\Diamond$ to $\textsc{Solved}$.
2. If all supports have been processed, then halt. Otherwise choose an unprocessed support $\textsc{sp} \subseteq \{1, 2, \ldots, d\}$ such that all inclusion-lesser supports $\textsc{sp}' \subset \textsc{sp}$ have been processed. Let $\mathscr{C}_{\textsc{sp}} = \{p\mathbf{u} \mid \textsc{supp}(p\mathbf{u}) = \textsc{sp}\}$.
   Some of $p\mathbf{u} \in \mathscr{C}_{\textsc{sp}}$ can be already in $\textsc{Solved}$, since bounded by lesser-support elements of $Win_\Diamond$; in the case $|\textsc{sp}| = 1$ we add all $p\mathbf{u} \in \mathscr{C}_{\textsc{sp}}$ whose single non-ω component is negative to $\textsc{Solved}$.
   The construction maintains the *invariant* that replacing any finite component in $p\mathbf{u} \in \mathscr{C}_{\textsc{sp}} \setminus \textsc{Solved}$ with ω yields an already established element of $Win_\Box$.
   i. Apply the standard attractor-construction on $\mathscr{C}_{\textsc{sp}}$, constructing the subset $C' \subseteq \mathscr{C}_{\textsc{sp}} \setminus \textsc{Solved}$ *from which* $\Diamond$ *can force reaching* $\textsc{Solved}$; in this construction we ignore the ω-components (outside $\textsc{sp}$), i.e., each transition leaves ω-components unchanged.
   (Start with $C' = \emptyset$ and keep extending C' with unsolved $p\mathbf{u}$ for which $p \in Q_\Diamond$ and $p\mathbf{u} \mapsto q\mathbf{v} \in C' \cup \textsc{Solved}$, in which case fix the respective transition for $p\mathbf{u}$, or for which $p \in Q_\Box$ and $q\mathbf{v} \in C' \cup \textsc{Solved}$ for all $p\mathbf{u} \mapsto q\mathbf{v}$. The construction is finite due to the above invariant.)
   ii. Extend $\textsc{Solved}$ with C', and let C be the set of the *minimal* currently unsolved elements of $\mathscr{C}_{\textsc{sp}}$; let $\mathscr{C}_{\textsc{sp}} \setminus \textsc{Solved} = C \uplus \textsc{Above}$. By another attractor-construction (now from the $\Box$-viewpoint) construct $\mathcal{D} \subseteq C$ from where $\Box$ can force reaching $\textsc{Above}$.
   iii. If $\mathcal{D} = C$, then finish processing $\textsc{sp}$ (claiming that $\mathscr{C}_{\textsc{sp}} \setminus \textsc{Solved} \subseteq Win_\Box$) and go to 2. Otherwise (if $\mathcal{D} \subset C$) consider the respective $(d - |\textsc{sp}|)$-dim parity-eVASS game (with the counters outside $\textsc{sp}$), where $C \setminus \mathcal{D}$ serves as the set of control states.
   (In our example, at some moment we might have $\textsc{sp} = \{1, 2\}$ and consider the 1-dim game with control states $p_1(1, 3, \omega)$, $p_1(2, 2, \omega)$, $p_1(3, 1, \omega)$, $p_2(2, 2, \omega)$, where all these control states happen to be $\Diamond$-universal.)
   Find $\Diamond$-universal states in this $(d - |\textsc{sp}|)$-dim game. If there are some, then add the respective symbolic configurations to $\textsc{Solved}$ (which is obviously sound), and go to i.; otherwise finish processing $\textsc{sp}$, and go to 2.

To show the termination and the correctness of the algorithm (which includes showing the mentioned invariant), we need to show that at the end of processing $\textsc{sp}$ we have $\mathscr{C}_{\textsc{sp}} \setminus \textsc{Solved} \subseteq Win_\Box$. This surely holds for $\textsc{sp} = \emptyset$; thus also the invariant holds when $|\textsc{sp}| = 1$. For the sake of contradiction, suppose that at the end of processing $\textsc{sp}$ (for which the invariant holds) there is some (minimal unsolved) $p\mathbf{u} \in C$ that belongs to $Win_\Diamond$. Hence $p\mathbf{u} \leq p\mathbf{v}$ for some $p\mathbf{v} \in Max_\Diamond$, where $\textsc{supp}(p\mathbf{v}) = \textsc{supp}(p\mathbf{u}) = \textsc{sp}$. We can verify that there is a $\Diamond$-strategy

σ that is winning from every instance of $p\mathbf{v}$ (imagine that ω's outside SP are replaced with very large integers) and the respective consistent plays never visit an instance of $p\mathbf{v}'$ where $\text{SUPP}(p\mathbf{v}') = \text{SP}$ and $\mathbf{v}' > \mathbf{v}$ (which means $\mathbf{v}' \geq \mathbf{v}$ and $\mathbf{v}' \neq \mathbf{v}$). But the same strategy σ is winning from all instances of $p\mathbf{u}$ as well; the consistent plays then never visit an instance of (a symbolic configuration in) $\{p\mathbf{w} \mid \text{SUPP}(p\mathbf{w}) = \text{SP}, \mathbf{w} > \mathbf{u}\} \subseteq \text{ABOVE}$. If a consistent play from an instance of $p\mathbf{u}$ visits an instance of $\{q\mathbf{w} \mid \text{SUPP}(q\mathbf{w}) = \text{SP}, \mathbf{w} \geq \mathbf{v}\}$ for another $q\mathbf{v} \in C$, then there is a $\Diamond$-strategy winning from all instances of $q\mathbf{v}$ that avoids visiting instances of both $\{p\mathbf{w} \mid \text{SUPP}(p\mathbf{w}) = \text{SP}, \mathbf{w} > \mathbf{u}\} \subseteq \text{ABOVE}$ and $\{q\mathbf{w} \mid \text{SUPP}(q\mathbf{w}) = \text{SP}, \mathbf{w} > \mathbf{v}\} \subseteq \text{ABOVE}$. By continuing this reasoning we deduce that there is a $\Diamond$-strategy winning from all instances of some $p'\mathbf{u}' \in C$ that avoids visiting ABOVE; this contradicts with the condition that $\mathcal{D} = C$ or that the $(d-|\text{SP}|)$-dim game with the control states $C \setminus \mathcal{D}$ has no $\Diamond$-universal states.

Using the bound B_0 from Lemma 4(2) iteratively (also for bounding the possible numbers $|C \setminus \mathcal{D}|$ of control states in the $(d-|\text{SP}|)$-dim games), it is a routine to derive a d-exponential upper bound on the complexity of our algorithm.

We leave implicit the produced presentations of (finite-memory) $\Diamond$-strategies winning in $Win_\Diamond$, in the form of (large) finite automata given as eVASS-games where the control states are some symbolic configurations and each $\Diamond$-state has just one outgoing transition (which is exemplified by Fig. 2 when we remove t_{11}). Similarly we leave implicit the presentations of the (finite-memory) $\Box$-strategies.

Remark. In any VASS game where $\Diamond$ has no choice, the GIC problem is equivalent to the *nontermination problem* in VASSs, which is EXPSPACE-complete, similarly as the coverability and boundedness problems, as follows from the classical results by Lipton and Rackoff (cf. [20] and, e.g., the survey [13]). Hence ALTEXPSPACE-completeness, i.e., 2-EXPTIME-completeness, in the case of two-player games, which was shown in [17], looks natural, but it requires a (much) deeper insight for analysing the complexity of the above algorithm.

Acknowledgment. I thank the anonymous reviewers for helpful comments.

References

1. Abdulla, P., Bouajjani, A., d'Orso, J.: Monotonic and downward closed games. Journal of Logic and Computation **18**(1), 153–169 (2008). (a preliminary version appeared at CSL/KGC 2003)
2. Abdulla, P.A., Atig, M.F., Hofman, P., Mayr, R., Kumar, K.N., Totzke, P.: Infinite-state energy games. In: Proc. of CSL-LICS 2014, pp. 7:1–7:10. ACM Press (2014)
3. Abdulla, P.A., Mayr, R., Sangnier, A., Sproston, J.: Solving parity games on integer vectors. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013 – Concurrency Theory. LNCS, vol. 8052, pp. 106–120. Springer, Heidelberg (2013)
4. Blondin, M., Finkel, A., Göller, S., Haase, C., McKenzie, P.: Reachability in two-dimensional vector addition systems with states is PSPACE-complete. In: Proc. of LICS 2015. ACM Press (2015)
5. Borosh, I., Flahive, M., Treybig, B.: Small solutions of linear Diophantine equations. Discrete Mathematics **58**, 215–220 (1986)

6. Brázdil, T., Chatterjee, K., Kučera, A., Novotný, P.: Efficient controller synthesis for consumption games with multiple resource types. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 23–38. Springer, Heidelberg (2012)
7. Brázdil, T., Kiefer, S., Kučera, A., Novotný, P., Katoen, J.P.: Zero-reachability in probabilistic multi-counter automata. In: Proc. of CSL-LICS 2014, pp. 22:1–22:10. ACM Press (2014)
8. Brázdil, T., Jančar, P., Kučera, A.: Reachability games on extended vector addition systems with states. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 478–489. Springer, Heidelberg (2010)
9. Chaloupka, J.: Z-reachability problem for games on 2-dimensional vector addition systems with states is in P. Fundam. Inform. **123**(1), 15–42 (2013). (a preliminary version appeared at the Workshop on Reachability Problems 2010)
10. Chatterjee, K., Doyen, L.: Energy parity games. Theor. Comput. Sci. **458**, 49–60 (2012). (a preliminary version appeared at ICALP 2010)
11. Chatterjee, K., Randour, M., Raskin, J.: Strategy synthesis for multi-dimensional quantitative objectives. Acta Inf. **51**(3–4), 129–163 (2014). (a preliminary version appeared at CONCUR 2012)
12. Courtois, J.-B., Schmitz, S.: Alternating vector addition systems with states. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 220–231. Springer, Heidelberg (2014)
13. Esparza, J.: Decidability and complexity of Petri net problems - an introduction. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998)
14. Fahrenberg, U., Juhl, L., Larsen, K.G., Srba, J.: Energy games in multiweighted automata. In: Cerone, A., Pihlajasaari, P. (eds.) ICTAC 2011. LNCS, vol. 6916, pp. 95–115. Springer, Heidelberg (2011)
15. Finkel, A., Goubault-Larrecq, J.: The theory of WSTS: the case of complete WSTS. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 3–31. Springer, Heidelberg (2012)
16. Juhl, L., Guldstrand Larsen, K., Raskin, J.-F.: Optimal bounds for multiweighted and parametrised energy games. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 244–255. Springer, Heidelberg (2013)
17. Jurdziński, M., Lazić, R., Schmitz, S.: Fixed-dimensional energy games are in pseudo-polynomial time. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 260–272. Springer, Heidelberg (2015)
18. Kosaraju, S., Sullivan, G.: Detecting cycles in dynamic graphs in polynomial time. In: Proceedings of STOC 1988, pp. 398–406. ACM Press (1988)
19. Leroux, J., Schmitz, S.: Reachability in vector addition systems demystified. In: Proc. of LICS 2015. ACM Press (2015)
20. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theoretical Computer Science **6**, 223–231 (1978)
21. Raskin, J., Samuelides, M., van Begin, L.: Games for counting abstractions. Electr. Notes Theor. Comput. Sci. **128**(6), 69–85 (2005)
22. Velner, Y., Chatterjee, K., Doyen, L., Henzinger, T.A., Rabinovich, A.M., Raskin, J.: The complexity of multi-mean-payoff and multi-energy games. Inf. Comput. **241**, 177–196 (2015). (based on versions appearing at FSTTCS 2010 and FoSSaCS 2011)

A Topological Method for Finding Invariant Sets of Continuous Systems

Laurent Fribourg[2], Eric Goubault[1(✉)], Sameh Mohamed[1,2], Marian Mrozek[3], and Sylvie Putot[1]

[1] LIX, Ecole Polytechnique, 91128 Palaiseau Cedex, France
{goubault,putot}@lix.polytechnique.fr
[2] LSV, ENS Cachan and CNRS, Cachan, France
{fribourg,mohamed}@lsv.ens-cachan.fr
[3] Division of Computational Mathematics, Jagiellonian University, Kraków, Poland
marian.mrozek@uj.edu.pl

Abstract. A usual way to find positive invariant sets of ordinary differential equations is to restrict the search to predefined finitely generated shapes, such as linear templates, or ellipsoids as in classical quadratic Lyapunov function based approaches. One then looks for generators or parameters for which the corresponding shape has the property that the flow of the ODE goes inwards on its border. But for non-linear systems, where the structure of invariant sets may be very complicated, such simple predefined shapes are generally not well suited. The present work proposes a more general approach based on a topological property, namely Ważewski's property. Even for complicated non-linear dynamics, it is possible to successfully restrict the search for isolating blocks of simple shapes, that are bound to contain non-empty invariant sets. This approach generalizes the Lyapunov-like approaches, by allowing for inwards and outwards flow on the boundary of these shapes, with extra topological conditions. We developed and implemented an algorithm based on Ważewski's property, SOS optimization and some extra combinatorial and algebraic properties, that shows very nice results on a number of classical polynomial dynamical systems.

1 Introduction

This paper describes a new method for proving the existence of a *positive invariant* set (generally called invariant set in computer science - we will stick to the former terminology, classical in the dynamical systems community) of a dynamical system, inside some region of the state space. Positive invariant sets are central to control theory and to validation of systems, such as programs (when considering discrete dynamical systems), physical systems (considering continuous dynamical systems), or hybrid systems. In this paper, we are focusing on continuous dynamical systems, but part of the method described here makes sense in a discrete setting - in particular, Conley's index theory [20] can be developed for discrete systems, only the differential conditions we are giving in

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 63–75, 2015.
DOI: 10.1007/978-3-319-24537-9_7

this paper have to be replaced by different conditions, which will be developed elsewhere.

Let us consider an autonomous polynomial differential equation, for the rest of this article :

$$\frac{dx}{dt} = f(x) \tag{1}$$

where x is a vector $(x_1, \ldots, x_n)$ of $\mathbb{R}^n$ and f is a vector of n polynomials in $x_1, \ldots, x_n$, e.g. for all $i = 1, \ldots, n$, $f_i \in \mathbb{R}[x_1, \ldots, x_n]$, the multivariate polynomial ring on n variables. As a polynomial function is locally Lipschitz, we know by the Cauchy-Lipschitz theorem that the vector field f generates a flow $\varphi : U \to \mathbb{R}^n$, where U is an open subset of $\mathbb{R} \times \mathbb{R}^n$ in such a way that $t \mapsto \varphi(x, t)$ is a solution of the differential equation.

A *positive invariant set* is a subset of the state-space such that if the initial state of the system belongs to this set, then the state of the system remains inside the set for all future time instances. An *invariant set* is a subset of the state-space which is positively invariant under the flow, and positively invariant under the opposite flow (i.e. it is also negatively invariant). The classical approach to find positive invariant sets of dynamical systems is through the determination of a Lyapunov function, generally polynomial, which decreases along trajectories of the dynamical system, positive in a neighborhood of the equilibrium point (the only point on which its value is zero). This approach is particularly well-suited to linear dynamical systems, where quadratic Lyapunov functions prove to be the right class of functions, but in the case of non-linear systems [15,30] the shape of the invariant set itself may be very complicated, in fact, far too complicated to be easily found in general by polynomial Lyapunov functions. Some authors, including the authors of the present article, have shown how to find, in some cases, rational functions [30], and even functions in the differential field extension of rational functions by some logarithms and exponentials, which could be candidate Lyapunov functions [11]. This is of course highly costly in computational power.

Recently, some authors have proposed to use piecewise linear [24] or piecewise quadratic [2] Lyapunov functions inspired both by abstract interpretation of programs [25] and recent results in hybrid systems theory, most notably in switched systems theory [3,26]. If these methods, based on "templates", are computationally tractable, they are limited by the fact that they can only find very specific shaped positive invariant sets ; they must be in particular convex, which is not always the case, even for simple classical systems.

We propose in this paper a method that also builds on templates. But instead of considering more complicated Lyapunov functions (or shapes of candidate invariant sets) as in [11], we relax the classical condition that, for a template (given by a piecewise linear, quadratic, and even more general polynomial Lyapunov function) to be positively invariant, the flow of the differential system must go inwards, condition that can be expressed as a negativity condition on a certain Lie derivative (a notion defined in Section 3). We relax this classical condition by asking only some parts of the boundary of the template to have inwards flows, relying on some simple techniques of Conley's index theory [8],

and in particular Ważewski's property, to show the existence of a positive invariant set within this template. The positive invariant set itself may be very complicated, but we do not need to precisely describe it ; the template serves as an outer approximation of this positive invariant set.

Let us for instance consider the case of Example 1, which will be our running example throughout this paper.

Example 1. (Ex 2.8 of [20]): $\dot{x} = y, \dot{y} = y + (x^2 - 1)\left(x + \frac{1}{2}\right)$. This system has several invariant sets in $B = [-2, 2] \times [-2, 2] \subset \mathbb{R}^2$: there are in particular 3 fixed points $(-1, 0)$, $\left(-\frac{1}{2}, 0\right)$ and $(1, 0)$ for this system within this box. On this system, it would be difficult to find a linear template on which we can prove the inward flows property (in fact, there is a "natural" degree 4 polynomial Lyapunov function, see [20]), whereas we will see that boxes such as B can be easily shown to contain a positive invariant set, using our approach.

Claims and Contents of this Article. The main idea of our article is that even though invariant sets for nonlinear dynamics may be very complicated to represent, and thus to find (e.g. by an explicit Lyapunov function), there exist *topological* criteria to deduce that there exists a non-empty positive invariant set inside some region of the state space. Note that not all positive invariant sets contain a point onto which the dynamical system converges. There might be a limit cycle, or a more complicated recurrent sets, for instance. For more complicated asymptotic behaviours, within invariant sets, we rely on notions from the Conley index theory [20], which we quickly state in Section 2. The key useful notions here, are that of an *isolating block* and the Ważewski's theorem, which gives a sufficient condition to the existence of a non-empty (positive) invariant set within an isolating block.

Several approaches have been developed over the years to algorithmically compute such isolating sets and index pairs, but most of them have been derived for discrete time dynamical systems [21,29]. Most approaches for continuous-time systems reduce the problem to the discrete-time setting by constructing rigorous outer approximations of a map for the flow, which of course involves approximating the solution of the ordinary differential equation which describes the system.

In this work, we generalize the template based approach of [24], designed originally for linear systems, to derive some algebraic conditions for a template to be an isolating block. This is done in Section 3, in which we give a sufficient condition for a polynomial template to be an isolating block, expressed as conditions on the Lie derivatives of the polynomial functions involved, on the faces of the template, that generalize the conditions given in [28]. The main difficulty is in fact of a topological nature : most of Section 3 is concerned with proving that the so-called *exit set* on the template, under the flow given by Equation 1, i.e. the set of states leaving the template, on its faces, is closed. This is necessary for the template to be an isolating block. Note that our templatized isolating blocks are particular C^∞ isolating-block-with-corners of [14], that are as powerful as (generalized) Lyapunov functions for finding invariant sets (Theorem 2.4 of [14]); but these isolating blocks

are "robust": they are still isolating blocks for nearby flows (Theorem 3.5 of [14]), which make them more robust numerically. They are also close to the polyfacial sets of [23] attributed there to the original paper of Ważewski [31].

We remark then, that the conditions we gave for a template to be an isolating block can be solved, in particular, by Sum of Squares programming [17] using Stengle's nichtnegativstellensatz and even, most often, just Putinar's positivstellensatz [22], which is computationally tractable using a SdP (Semi Definite Programming) relaxation. This makes a second major difference with [28] where an interval-based method is used instead. Finally, an isolating block may only contain an empty invariant set, unless the conditions for Ważewski's principle are satisfied. For the purpose of this paper, we use a simpler condition, of a purely combinatorial nature, in Section 4. We end up by discussing the algorithm on simple examples from the literature, in Section 5. The first simple experiments obtained with our *Matlab* implementation are still quite costly, but we propose in the conclusion a number of possible algorithmical improvements.

2 Some Basics of Dynamical Systems Theory

The following definitions come from Conley index theory, and the qualitative description of nonlinear dynamics [20]. Let us call $\varphi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ the *flow* function, such that $\varphi(., x_1, \ldots, x_n)$ is the unique solution to the differential equation system (1) starting, at time 0, at state $(x_1, \ldots, x_n) \in \mathbb{R}^n$, meaning that $\varphi(0, x_1, \ldots, x_n) = (x_1, \ldots, x_n)$ and $\frac{d\varphi}{dt}(t, x_1, \ldots, x_n) = f \circ \varphi(t, x_1, \ldots, x_n)$. (Positive) invariant sets are invariant under the flow φ, for all (resp. positive) times, i.e. they are sets S such that (resp. $\varphi(\mathbb{R}^+, S) \subset S$) $\varphi(\mathbb{R}, S) \subset S$.

A subtle point is that the method is designed to find invariant sets S within a compact set N, but not quite positive invariant sets. But in fact, it is well-known that when we have one, we will have the other [5,23].

For non-linear dynamics, the shape of invariant sets can be very complicated. A central notion to our method is that of *isolating block*, that isolates invariant sets, meaning that invariant sets therein, if ever they exist, are necessarily in the *interior* of isolating blocks.

Definition 1. *(Isolating Block). A compact set B is an* isolating block *if*

(a) $B^- = \{x \in B \mid \varphi([0,T), x) \not\subseteq B, \forall T > 0\}$ *is closed*
(b) $\forall T > 0, \{x \in B \mid \varphi([-T,T], x) \subseteq B\} \subseteq intB$

Condition (a) imposes that the *exit set* B^-, i.e. the set of states of B which leave B under flow φ, is closed in the topology of $\mathbb{R}^n$. When condition (b) is satisfied, B is called an isolating neighborhood. The combination of (a) and (b) guarantees that no trajectory is inner tangential to the boundary ∂B of B. A fundamental difficulty in computational topological dynamics is that isolating neighborhoods are generally much easier to construct than isolating blocks.

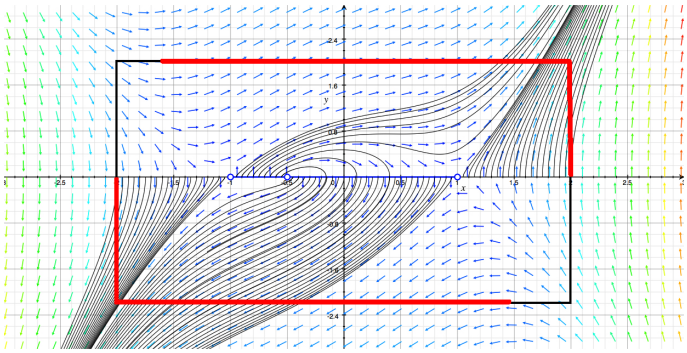


Fig. 1. The three fixed points and the exit set (in red) of the system of Example 1.

Example 2. For Example 1, we will see that for $B = [-2, 2] \times [-2, 2]$ the box of Figure 2, B^- is the set of four red segments, one on each face, on the same Figure, and we will prove (Example 5.) that B^- is closed, so that B is an isolating block for this system. Note that this is a robust notion : all $B = [-a, a] \times [-b, b]$ with $a > 1$ and $b > 1$, in particular, are isolating blocks.

Still, isolating blocks may not contain interesting (meaning non-empty) invariant sets. There is a simple topological condition on isolating blocks that implies the existence of a non empty invariant set therein.

Theorem 1. *(Ważewski Property [20]). If B is an isolating block and B^- is not a deformation retract of B then there exists a not-empty invariant set S in the interior of B.*

We will not define formally a deformation retract, for the sake of simplicity. Let us just say that a deformation retract of a topological space B is a subspace which is an "elastic" deformation of it, so that it retains its essential topological features. For the method we are developing here, we will content ourselves with the much weaker statement that among the topological features that are retained in a deformation retract, is the number of connected components.

Example 3. We saw in Example 2 that we had a square B with closed exit set B^- made of two connected components. Clearly B^- is not a deformation retract of B, as B^- is made of two connected components, and B of only one.

3 Isolating Blocks: Algebraic Conditions

We are now giving a simple criterion for a compact set B, given as a general polynomial template, to be an isolating block for the dynamics given by Equation 1. Note that although the method can be defined for general polynomial templates, which is what we describe here, isolating blocks are robust properties

that permit the use of very simple templates in general, which will be the case in the experiments presented here. The set $B \subseteq \mathbb{R}^n$ is defined, for some vector $c = (c_1, \ldots, c_m) \in \mathbb{R}^m$, by the m polynomial inequalities :

$$(P) \quad \begin{cases} p_1(x_1, \ldots, x_n) \leq c_1 \\ \cdots \\ p_m(x_1, \ldots, x_n) \leq c_m \end{cases}$$

We call P_i^c the face of template B given by $\{(x_1, \ldots, x_n) | p_i(x_1, \ldots, x_n) = c_i\} \cap B$ which might be proper (non-empty) or not. In what follows, we suppose that each face of B is proper.

We call *minimal polynomial templates*, the templates B whose border ∂B is equal (and not just included as would be generally the case) to $\bigcup_{i=1}^{s} \{x | p_i(x) = c_i, p_j(x) \leq c_j \ \forall j \neq i\}$. For $x \in \partial B$, we note $I(x)$ the non-empty and maximal set of indices in $1, \ldots, m$ such that for all $i \in I(x)$, $p_i(x) = c_i$.

Let us now define Lie derivatives, that we will use hereafter.

Definition 2. *(Lie Derivative and Higher-order Lie Derivatives). The Lie derivative of $h \in \mathbb{R}[x]$ along the vector field $f = (f_1, \ldots, f_n)$ is defined by*

$$\mathcal{L}_f(h) = \sum_{i=1}^{n} \frac{\partial h}{\partial x_i} f_i = \langle f, \nabla h \rangle$$

Higher-order derivatives are defined by $\mathcal{L}_f^{k+1}(h) = \mathcal{L}_f(\mathcal{L}_f^{(k)}(h))$ with $\mathcal{L}_f^0(h) = h$.

For polynomial dynamical systems, only a finite number of Lie derivatives are necessary to generate all higher-order Lie derivatives. Indeed, let $h \in \mathbb{R}[x_1, \ldots, x_n]$, we recursively construct an ascending chain of ideals of $\mathbb{R}[x_1, \ldots, x_n]$ by appending successive Lie derivatives of h to the list of generators:

$$< h > \subseteq < h, \mathcal{L}_f^1(h) > \subseteq \cdots \subseteq < h, \mathcal{L}_f^1(h), \ldots, \mathcal{L}_f^{(N)}(h) >$$

Since the ring $\mathbb{R}[x]$ is Noetherian [16], this increasing chain of ideals has necessarily a finite length: the maximal ideal is called the differential radical ideal of h and will be noted $\sqrt[\mathcal{L}]{< h >}$. Its order is the smallest N such that:

$$\mathcal{L}_f^{(N)}(h) \in < h, \mathcal{L}_f^{(1)}(h), \ldots, \mathcal{L}_f^{(N-1)}(h) > \tag{2}$$

Not surprisingly, this is a notion that has already been used to characterize algebraic positive invariant sets of dynamical systems [10].

This N is computationally tractable. If we note N_i the order of the differential radical ideal $\sqrt[\mathcal{L}]{< p_i >}$, then for face i we should compute the successive Lie derivatives until N_i. This can be done by testing if the *Gröbner basis* spanned by the derivatives changes. Indeed, two ideals are equal if they have the same reduced Gröbner basis (usually a Gröbner basis software produces reduced bases) [1]. If we denote by $\mathcal{G}(\{g_1, \cdots, g_n\})$ the Gröbner basis of $\{g_1, \cdots, g_n\}$, the first n s.t. $\mathcal{G}(\{\mathcal{L}_f^{(0)}(p_i), \cdots, \mathcal{L}_f^{(n)}(p_i)\}) = \mathcal{G}(\{\mathcal{L}_f^{(1)}(p_i), \cdots, \mathcal{L}_f^{(n+1)}(p_i)\})$ is equal to N_i. It can be observed that upper bounds for N_i could be used instead of computing Gröbner bases in some cases, see [27].

Example 4. If we take the first face P_1^c of the template $\{p_1 = -x, p_2 = x, p_3 = -y, p_4 = y\}$ with $\{c_1 = 2, c_2 = 2, c_3 = 2, c_4 = 2\}$ i.e $P_1^c = \{-x = 2, x \leq 2, -y \leq 2, y \leq 2\}$ then $\mathcal{L}_f^{(1)}(p_1) = -y$, $\mathcal{G}(\{\mathcal{L}_f^{(0)}(p_1), \mathcal{L}_f^{(1)}(p_1)\}) = \{-x, -y\}$; $\mathcal{L}_f^{(2)}(p_1) = -y - (x^2 - 1)\left(x + \frac{1}{2}\right)$, $\mathcal{G}(\{\mathcal{L}_f^{(0)}(p_1), \mathcal{L}_f^{(1)}(p_1), \mathcal{L}_f^{(2)}(p_1)\}) = \mathbf{1}$. We can already deduce from this that $N_1 = 3$.

Now, in order to find isolating blocks, we need to find and prove some topological properties on the exit sets, see Definition 1. For polynomial templates, we rely on Lemma 1:

Lemma 1. *Let x_0 be a point on the border ∂B of a minimal polynomial template B. Then x_0 is in the exit set B^- of B if and only if, for some $i_0 \in I(x)$,*

$$\exists k_0 > 0 \text{ such } \mathcal{L}_f^{(k_0)}(p_{i_0}) > 0 \text{ and } \forall 0 < k < k_0 \ \mathcal{L}_f^{(k)}(p_{i_0}) = 0$$

For a template B to be an isolating block, we know from Definition 1 that we need to check first that the exit set B^- is closed :

Lemma 2. *Let B be a compact minimal polynomial template defined by the set of inequalities (P) and let N_i be the order of the differential radical ideal $\sqrt[\infty]{< p_i >}$ (i.e. the index defined by Equation 2) for the dynamical system of Equation 1.*

If for each face P_i^c of template B, for all $k \in \{1, \cdots, N_i - 2\}$,

$$(H_k^i) \ : \ \begin{cases} \{x \in P_i^c \mid \mathcal{L}_f^{(1)}(p_i)(x) = 0, \ \cdots, \ \mathcal{L}_f^{(k)}(p_i)(x) = 0, \\ \qquad \mathcal{L}_f^{(k+1)}(p_i)(x) < 0\} = \emptyset \end{cases}$$

then B^- is closed and is equal to $\bigcup_{i=1}^m \{x \in P_i^c \mid \mathcal{L}_f^{(1)}(p_i)(x) \geq 0\}$.

Proof. We begin to show that if for each face P_i^c of template B, for all $k \in \{1, \cdots, N_i - 2\}$, we have $\{x \in P_i^c \mid \mathcal{L}_f^{(1)}(p_i)(x) = 0, \ \cdots, \ \mathcal{L}_f^{(N_i-1)}(p_i)(x) = 0\} = \emptyset$ as well as (H_k^i) as above, then B^- is closed and is equal to $\bigcup_{i=1}^m \{x \in P_i^c \mid \mathcal{L}_f^{(1)}(p_i)(x) \geq 0\}$. If we set, for $k = 0, \ldots, N_i - 1$:

$$\begin{aligned} P_k &= \{x \mid \mathcal{L}_f^{(1)}(p_i)(x) = 0, \ \cdots, \ \mathcal{L}_f^{(k)}(p_i)(x) = 0, \mathcal{L}_f^{(k+1)}(p_i)(x) \geq 0\} \\ Q_k &= \{x \mid \mathcal{L}_f^{(1)}(p_i)(x) = 0, \ \cdots, \ \mathcal{L}_f^{(k)}(p_i)(x) = 0, \mathcal{L}_f^{(k+1)}(p_i)(x) > 0\} \\ R_k &= \{x \mid \mathcal{L}_f^{(1)}(p_i)(x) = 0, \ \cdots, \ \mathcal{L}_f^{(k)}(p_i)(x) = 0, \mathcal{L}_f^{(k+1)}(p_i)(x) < 0\} \end{aligned}$$

We then have, for $k = 1, \ldots, N_i - 2$, $P_k = Q_k \cup P_{k+1} \cup R_{k+1}$. Given that N_i is the index of the differential ideal $\sqrt[\infty]{< p_i >}$, we know that $Q_{N_i-1} = \emptyset$, $R_{N_i-1} = \emptyset$, since $\mathcal{L}_f^{(1)}(p_i)(x) = 0, \ \cdots, \ \mathcal{L}_f^{(N_i-1)}(p_i)(x) = 0$ implies $\mathcal{L}_f^{(N_i)}(p_i)(x) = 0$. Given the hypotheses, we know also that for all $k = 1, \ldots, N_i - 2$, $R_k = \emptyset$, and $P_{N_i-1} = \emptyset$. This means that $P_k = \bigcup_{i=k}^{N_i-2} Q_i$.

By Lemma 1, x is in $B_- \cap P_i^c$ if and only if it is in $\cup_{i=0}^{\infty} Q_i \cup = \cup_{i=0}^{N_i-2} Q_i$. This last set is, by the equation above, equal to P_0, which is the inverse image by a continuous function (the higher Lie derivative) of the closed set $[0, max \ \mathcal{L}_f^{(1)}(p_i)(B)]$ (since B is compact in $\mathbb{R}^n$).

We can then notice that the exit set B_- is the union of $B_- \cap P_i^c$, for all $i = 1, \ldots, m$, each one of which is closed, hence is closed.

Note now that $p_i = c \wedge \mathcal{L}_f^1(p_i) \ \ldots \ \mathcal{L}_f^{N_i-1}(p_i) = 0$ is equivalent to saying that all solutions to the ODE are constant on face p_i. This implies that the face P_i^c is not an exit set, and the exit set relative to P_i^c is trivially closed (since empty). This means that, to check that a template is an isolating block, we only need to check that for all $k \in \{1, \cdots, N_i - 2\}$, :

$$p_i = c_i \wedge (p_j \leq c_j)_{j \neq i} \wedge \mathcal{L}_f^{(1)}(p_i)(x) = 0, \ \cdots, \ \mathcal{L}_f^{(k)}(p_i)(x) = 0, \ \Rightarrow \mathcal{L}_f^{(k+1)}(p_i)(x) \geq 0$$

Note that the criterion used in [28] strictly implies all (H_k^i).

Finally, for B a polynomial template to be an isolating block, we need to show (see Definition 1) that there is no inner tangential flow within it. It is easy to see that not having any inner tangential flow is equivalent to asking that for each of its faces P_i^c, for all $k \in \{0, \cdots, N_i - 1\}$, no $x \in P_i^c$ can satisfy the following set of equalities and inequalities :

$$\mathcal{L}_f^{(1)}(p_i)(x) = 0, \ \cdots, \ \mathcal{L}_f^{(2k-1)}(p_i)(x) = 0, \ \mathcal{L}_f^{(2k)}(p_i)(x) < 0 \tag{3}$$

This is clearly satisfied when the condition of Lemma 2 is satisfied.

The algorithm we are going to develop now thus relies on checking the condition of Lemma 2. This can be checked using Sum of Squares optimization [17] and Stengle's nichtnegativstellensatz, for increasing k from 1 to $N_i - 2$, for each face i of the template. This is done as follows. We determine polynomials α_j $(j = 0, \ldots, k)$, SoS polynomials $\beta_{S,\mu}$ $(S \subseteq \{1, \ldots, i-1, i+1, \ldots, m\}, \mu \in \{0, 1\})$ and an integer l, such that

$$\sum_{j=0}^{k} \alpha_j \mathcal{L}_f^{(j)} \ + \sum_{S \subset \{1,\ldots,i-1,i+1,\ldots,m\}} \beta_{S,\mu} G_{S,\mu} + \left(\mathcal{L}_f^{(k+1)}\right)^{2l} = 0 \tag{4}$$

where $G_{S,\mu} = (-\mathcal{L}_f^{(k+1)})^{\mu} \Pi_{s \in S}(c_s - p_s)$ for any $S \subseteq \{1, \ldots, i-1, i+1, \ldots, m\}$ and $\mu \in \{0, 1\}$ and the convention that $\mathcal{L}_f^0(p_i) = c_i - p_i$. Practically speaking, this is done by bounding the degrees of the polynomials α_j and $\beta_{S,\mu}$ we are looking for, and taking low values for l (in all our examples, we took $l = 1$). Hence we get the following Proposition, at the heart of our algorithm :

Proposition 1. *For each face P_i^c, if for all $k = 1, \ldots, N_i - 2$, there exist polynomials α_j $(j \in \{0, \ldots, k\})$ and sum-of-squares polynomials $\beta_{S,\mu}$ $(S \subseteq \{1, \ldots, i-1, i+1, \ldots, m\}$ and $\mu \in \{0, 1\})$ such that Equation 4 holds, then the template P_i^c is an isolating block.*

To provide for faster results, in most cases, we begin, for a given k (and i), instead of solving (H_k^i) by Equation 4, by solving the simpler property $p_i = c_i \wedge (p_j \leq c_j)_{j \neq i} \wedge \mathcal{L}_f^{(1)}(p_i)(x) = 0, \ \cdots, \ \mathcal{L}_f^{(k)}(p_i)(x) = 0, \ \Rightarrow \mathcal{L}_f^{(k+1)}(p_i)(x) > 0$. If so, we can stop testing (H_k^i) for higher values of k since they are then trivially satisfied. We can test whether this equation above is true using Putinar's *positivstellensatz* [22] which is much less computationally demanding than Stengle's

nichtnegativstellensatz, and which also stops the algorithm potentially before reaching $k = N_i - 2$. A sufficient condition for this to be true is to find polynomials α_l $(l = 1, \ldots, k)$, γ_i and sum-of-squares polynomials β_j $j = 1, \ldots, m$, $j \neq i$ such that $\mathcal{L}_f^{(k+1)}(p_i) = \sum_{j=1}^{k} \alpha_j \mathcal{L}_f^{(j)}(p_i) + \beta_0 + \sum_{j=1, j\neq i}^{m} \beta_j(c_j - p_j) + \gamma_i(p_i - c_i)$. For each fixed integer $D > 0$, which we choose as a bound on the degree of polynomials α_l, γ_i and β_j, this can be tested by semidefinite programming (see [18] and the improvement of [19] for a discussion on the maximal degree for these problems).

Example 5. We take again the face P_1^c, and try to prove (H_1^1) for example. A sufficient condition is to find polynomials α, γ (for equality conditions) and sum-of-squares polynomials $\beta_0, \beta_1, \beta_2, \beta_3$ (for inequality conditions) such that

$$\mathcal{L}_f^{(2)}(p_1) = \alpha \mathcal{L}_f^{(1)}(p_1) + \beta_0 + \beta_1(c_2 - p_2) + \beta_2(c_3 - p_3) + \beta_3(c_4 - p_4) + \gamma(p_1 - c_1)$$

which is trivially satisfied with $\alpha = 1$, $\beta_0 = \frac{9}{2}$, $\beta_1 = \beta_2 = \beta_3 = 0$ and $\gamma = \left(\left(\frac{1}{2} + x\right)(2 - x) - 3\right)$. Using *SOSTools* with the SdP solver *SeDuMi* under *Matlab* gives more complicated solutions.

4 A Simple Combinatorial Condition for Proving the Existence of (Non-empty) Invariant Sets

Even though we found a compact minimal polynomial template B with closed exit set, i.e. an isolating block, it can be the case that the inner invariant set is empty. We use Ważewski's property, Theorem 1, to ensure that it is not empty.

It is difficult, in general, to test whether B^- is a deformation retract of B or not. We will use sufficient conditions to guarantee that B^- is not a deformation retract of B, in the simpler case where B is contractible (i.e. there is a deformation retract of B onto any of its points).

Using Lemma 2, we know that exit sets on each of the faces P_i^c is given as the set of points x on P_i^c such that $\mathcal{L}_f^{(1)}(p_i)(x) \geq 0$. Define G as the graph whose nodes are the P_i^c for which $P_i^c \cap B^-$ is non-empty and whose edges are given by pairs P_i^c, P_j^c of faces of B, such that $\mathcal{L}_f^{(1)}(p_i)(x) \geq 0 \ \wedge \ \mathcal{L}_f^{(1)}(p_j)(x) \geq 0$ is satisfiable on $P_i^c \cap P_j^c$ (when non-empty).

If G is not connected, then the exit set B^- is trivially not connected either, because G has a number of components less or equal than that of B^- (this can be stricly less if some $P_i^c \cap B^-$ is not connected). But B is connected because it is in particular contractible. Thus B^- cannot be a deformation retract of B. This is what we used in Example 3 to prove that there is a positive invariant set within B. Note that we can do the same for the complement of the exit set (i.e. the entrance set), combined, and that, by Alexander duality [13], these two connectedness tests are rather fine tests : the connected components of the entrance set give information on the first cohomology group of the exit set in dimension $n = 3$.

This leads to Proposition 2, that uses positivstellensatz once again as an algorithmic method to determine connectivity of (an abstraction of) graph G.

Proposition 2. *Let $G^\sharp$ be the graph whose nodes are given by the faces P_i^c of the template considered such that there exists an x with $p_i(x) = c_i$ and $\mathcal{L}_f^{(1)}(p_i)(x) = \beta_0 + \sum_{k=1,k\neq i}^m \beta_k(c_k - p_k) + \gamma_i(c_i - p_i)$ (where β_0, β_k are SoS polynomials and γ_i is any polynomial), and whose edges are given by pair of faces (P_i^c, P_j^c) such that there exists an x with $p_i(x) = c_i$, $p_j(x) = c_j$, and $-\mathcal{L}_f^{(1)}(p_i)(x) \times \mathcal{L}_f^{(1)}(p_j)(x) = \beta'_0 + \sum_{k=1,k\neq i,k\neq j}^m \beta'_k(c_k - p_k) + \gamma_i(c_i - p_i) + \gamma_j(c_j - p_j)$ (where β'_0, β'_k are SoS polynomials and γ_i is any polynomial). Then if $G^\sharp$ is disconnected and the template is an isolating block then its invariant subset is non-empty.*

Algorithmically, on top of the classical SdP relaxation for solving positivstellensatz, we use a simple depth-first traversal of the graph to compute the set of connected components of $G^\sharp$.

Example 6. We consider again Example 1. Each of the four faces of B is a node in $G^\sharp$. The faces are respectively given by $\{(-2, y) | -2 \leq y \leq 2\}$ (face P_1^c), $\{(2, y) | -2 \leq y \leq 2\}$ (face P_2^c), $\{(x, -2) | -2 \leq x \leq 2\}$ (face P_3^c) and $\{(x, 2) | -2 \leq x \leq 2\}$ (face P_4^c). We thus have non-empty intersections $P_1^c \cap P_3^c = \{(-2, -2)\}$, $P_1^c \cap P_4^c = \{(-2, 2)\}$, $P_2^c \cap P_3^c = \{(2, -2)\}$ and $P_3^c \cap P_4^c = \{(2, 2)\}$. Therefore we have an edge from P_1^c to P_3^c if and only if $\mathcal{L}_f^{(1)}(p_1)(-2, -2) = 2$ and $\mathcal{L}_f^{(1)}(p_3)(-2, -2) = \frac{13}{2}$ are both positive - which is true ; we have an edge from P_1^c to P_4^c if and only if $\mathcal{L}_f^{(1)}(p_1)(-2, 2) = -2$ and $\mathcal{L}_f^{(1)}(p_4)(-2, 2) = \frac{5}{2}$ are both positive - which is false ; and similarly, we obtain that there is no edge between P_2^c and P_3^c, and there is an edge between P_2^c and P_4^c.

We conclude that B^- has (at least) two connected components, and that there is a non empty invariant set within the square B.

5 Experiments

The algorithm was implemented in Matlab, with the *Symbolic Math Toolbox* to compute the Lie derivatives, and *MuPaD* for *Gröbner basis* manipulations. Sum of square problems are solved with the semi-definite programs solver *SeDumi*. Timings of the execution of our algorithm on classical examples are given for a MacBook Air (Processor) 1,3 GHz Intel Core i5 , (Memory) 4 GB 1600 MHz DDR3 and expressed in seconds as follows : t Gröbner is the time needed to find the order of the differential radical for a given face, t SoS optim is the time taken to solve the SoS optimization problems for each face. We also indicate in Figure 2 the order of the differential radical in column "N_i" and the maximal degree of polynomials in the corresponding Gröbner base.

Example A. It is our running example, Example 1 from Section 1. Just as a matter of comparison, if we had applied directly Stengle's nichtnegativstellensatz, the time it would have taken to prove closedness of the exit set, for each face, would have been around 130 seconds, in sharp contrast with the 4 seconds, using Putinar's positivstellensatz.

Example B. It is defined by $\left(\dot{x} = y, \dot{y} = -y - x + \frac{1}{3}x^3\right)$ (Example 4 of [7], with a quadratic Lyapunov function), with as template, the box defined by $c = (2.4, 2.4, 2.4, 2.4)^t$. We are able to show, using the method of Proposition 2, that this box contains a non-trivial invariant.

Example C. It is defined by $\left(\dot{x} = -\frac{1}{10}x + y - x^3, \dot{y} = -x - \frac{1}{10}y, \dot{z} = 5z\right)$ (Example 4.1, page 21, of [28]). There are three fixed points $p_0 = (-1, 0, 0)$, $p_1 = (1, 0, 0)$, and $p_2 = (0, 1, -1)$, and rather complicated dynamics between neighborhoods of these points. The only face we are considering is the sphere of radius $\frac{1}{5}$ centered at p_2, which is defined by the template $x^2 + (y-1)^2 + (z+1)^2 = \frac{1}{25}$. The exit set can be shown to have two connected components, but Proposition 2 fails to see that, because we have only one face. This can be solved by considering the two hemispheres, one with $z \geq -1$, the other with $z \leq -1$. Our method then finds one connected component in one of the hemispheres and the other in the opposite one : $G^\sharp$ is disconnected, hence contains a non-empty invariant. Note that the direct SoS approach seems to find only degree $\geq$3 polynomials [28].

Ex.	Face	t Gröbner	d^o G-base	N_i	t SoS optim
Example A	Face 1 $(-x)$	0.39	3	3	4.7
	Face 2 (x)	0.43	3	3	4.67
	Face 3 $(-y)$	0.45	3	3	5.01
	Face 4 (y)	0.45	3	3	5.07
Example B	Face 1 (x)	0.38	2	3	4.8
	Face 2 $(-x)$	0.43	2	3	4.7
	Face 3 (y)	0.36	2	3	4.9
	Face 4 $(-y)$	0.39	2	3	5.0
Example C	Face 1	0.51	3	4	14.6
Example D	Face 1	1.83	7	4	158.42

Fig. 2. Some benchmarks

Example D. It is the same as Example C (i.e. Example 4.1 of [28]), but using as template a 4-norm ellipsoid centered at the point $(0, \frac{1}{2}, -\frac{1}{2})$ whose principal axes are pointing in the coordinate directions x, y, and z and have lengths $\frac{3}{2}$, 1, and 1, respectively. This is described by the only face $\left(\frac{2}{3}x\right)^4 + \left(y - \frac{1}{2}\right)^4 + \left(z + \frac{1}{2}\right)^4 = 1$. The exit set is connected but is not simply connected : this can be seen by noticing that the entrance set has two components. But as for Example C, Proposition 2 cannot help distinguish them right away as we have encoded the template by just one face.

6 Conclusion and Future Work

This paper is a first step towards more involved criteria for finding (positive) invariant sets. First, this can be generalized to switched systems, a particular

class of hybrid systems which have regained recent interest in the control community. Also, the nature of the invariant sets isolated by our method can be more precisely characterized, using further the Conley index theory, making for instance the difference between a stable point or a limit cycle. We can also generalize the combinatorial criterion of Section 4 : we used the first step of a general nerve lemma [32], there might be an interest in going one step further.

Another direction of improvement concerns the choice of templates. For instance, classical linear templates are quite hard to use for Van der Pol's equation, but the results of [6] seem to indicate that refining them should be possible.

There are also numerous algorithmic improvements over the costly Gröbner base and SoS computations. Instead of SoS methods, we could think of using simpler but still precise inner [12] and outer [9] approximations of the image of a polynomial function on a box. Some quantifier elimination methods might also been useful, using Cylindrical Algebraic Decomposition [4].

Finally, turning the SoS problems that we used for finding solutions to some polynomial inequalities into real optimization problems will provide a way to find the vector c defining the faces of a template isolating block, instead of merely checking the property for a given c. For finite or regular control problems, a finite set of parameters defining a stabilizing control may also be found this way.

Acknowledgments. The authors were partially supported by Digiteo Project SIMS 2013-0544D, by ANR projects CAFEIN, ANR-12-INSE-0007 and MALTHY, ANR-13-INSE-0003, by iCODE and by the academic chair "Complex Systems Engineering" of École polytechnique-ENSTA-Télécom-Thalès-Dassault Aviation-DCNS-DGA-FX-Fondation ParisTech-FDO ENSTA.

References

1. Adams, W.W., Loustaunau, P.: American mathematical society. An introduction to Grobner bases, Graduate studies in mathematics (1994)
2. Adjé, A., Garoche, P.-L.: Automatic synthesis of piecewise linear quadratic invariants for programs. In: D'Souza, D., Lal, A., Larsen, K.G. (eds.) VMCAI 2015. LNCS, vol. 8931, pp. 99–116. Springer, Heidelberg (2015)
3. Bacciotti, A., Mazzi, L.: Stability of dynamical polysystems via families of Lyapunov functions. Jour. Nonlin. Analysis (2007)
4. Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry. Springer (2006)
5. Bhatia, N.P., Hájek, O.: Local Semi-dynamical Systems, vol. 90. Lecture Notes in Mathematics. Springer (1969)
6. Boczko, E., Kalies, W.D., Mischaikow, K.: Polygonal approximation of flows. Topology and its Applications 154 (2007)
7. Bouissou, O., Chapoutot, A., Djaballah, A., Kieffer, M.: Computation of parametric barrier functions for dynamical systems using interval analysis. In: IEEE CDC 2014 (2014)
8. Conley, C.: Isolated invariant sets and the Morse index. Amer. Math. Soc., CBMS Regional Conf. Series Math. 38 (1978)
9. Dang, T., Testylier, R.: Reachability analysis for polynomial dynamical systems using the Bernstein expansion. Reliable Computing **17**, 128–152 (2012)

10. Ghorbal, K., Platzer, A.: Characterizing algebraic invariants by differential radical invariants. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014 (ETAPS). LNCS, vol. 8413, pp. 279–294. Springer, Heidelberg (2014)
11. Goubault, E., Jourdan, J.-H., Putot, S., Sankaranarayanan, S.: Finding non-polynomial positive invariants and Lyapunov functions for polynomial systems through Darboux polynomials. In: American Control Conference (2014)
12. Goubault, E., Mullier, O., Putot, S., Kieffer, M.: Inner approximated reachability analysis. In: HSCC, pp. 163–172 (2014)
13. Hazewinkel, M.: Alexander duality. Encyclopaedia of Mathematics (2002)
14. Wesley Wilson Jr., F., Yorke, J.A.: Lyapunov functions and isolating blocks. Journal of Differential Equations (1973)
15. Khali, H.K.: Nonlinear Systems. Prentice Hall (2002)
16. Lang, S.: Algebra. Graduate Texts in Mathematics. Springer, New York (2002)
17. Lasserre, J.-B.: Moments, positive polynomials and their applications, vol. 1. World Scientific (2009)
18. Lombardi, H.: Une borne sur les degrés pour le theorème des zéros rééls effectifs. In Real Algebraic Geometry, vol. 1524. Lecture Notes in Mathematics, pp. 323–345. Springer (1992)
19. Lombardi, H., Perrucci, D., Roy, M.-F.: An elementary recursive bound for effective positivstellensatz and Hilbert 17th problem (2014)
20. Mischaikow, K., Mrozek, M.: Conley index theory. In: Fiedler, B. (ed.) Handbook of Dynamical Systems II: Towards Applications, North-Holland (2002)
21. Mrozek, M.: Index pairs algorithms. Foundations of Computational Mathematics **6**(4), 457–493 (2006)
22. Putinar, M.: Positive polynomials on compact semi-algebraic sets. Indiana University Mathematics Journal **42**(3), 969–984 (1993)
23. Srzednicki, R.: Ważewski Method and Conley Index (2004)
24. Sankaranarayanan, S., Dang, T., Ivančić, F.: A policy iteration technique for time elapse over template polyhedra. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 654–657. Springer, Heidelberg (2008)
25. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 25–41. Springer, Heidelberg (2005)
26. Shorten, R., Wirth, F., Mason, O., Wulff, K., King, C.: Stability criteria for switched and hybrid systems. SIAM Rev. 49(4), 545–592
27. Guillermo Moreno Socías: Length of polynomial ascending chains and primitive recursiveness. **71**, 181–205 (1992)
28. Stephens, T., Wanner, T.: Rigorous validation of isolating blocks for flows and their Conley indices. Technical report, IMA Preprint Series #2424 (May 2014)
29. Szymczak, A.: A combinatorial procedure for finding isolating neighbourhoods and index pairs. Proceedings of the Royal Society of Edinburgh: Section A Mathematics **127**, 1075–1088 (1997)
30. Vidyasagar, M.: Nonlinear Systems Analysis, Networks Series. Prentice-Hall (1978)
31. Ważewski, T.: Sur un principe topologique de l'examen de l'allure asymptotique des intégrales des équations différentielles ordinaires. Ann. Soc. Polon. Math. **10**(25), 279–313 (1947)
32. Welker, V., Ziegler, G.M., Živaljević, R.T.: Homotopy colimits - comparison lemmas for combinatorial applications. J. Reine Angew. Math. **509**, 117–149 (1999)

The Ideal View on Rackoff's Coverability Technique

Ranko Lazić[1] and Sylvain Schmitz[1,2(✉)]

[1] DIMAP, Department of Computer Science, University of Warwick, Coventry, UK
[2] LSV, ENS Cachan and CNRS and INRIA, Cachan, France
schmitz@lsv.ens-cachan.fr

Abstract. Rackoff's small witness property for the coverability problem is the standard means to prove tight upper bounds in vector addition systems (VAS) and many extensions. We show how to derive the same bounds directly on the computations of the VAS instantiation of the generic backward coverability algorithm. This relies on a dual view of the algorithm using ideal decompositions of downwards-closed sets, which exhibits a key structural invariant in the VAS case. The same reasoning readily generalises to several VAS extensions.

1 Introduction

Checking safety properties in infinite transition systems can often be reduced to *coverability* checks. The coverability problem asks, given a transition system and two configurations x and y and a quasi-ordering $\leq$ over configurations, whether x might *cover* y, i.e. reach some configuration $y' \geq y$ in finitely many steps. The problem is decidable for the large class of (effective) *well-structured transition systems* (WSTS) where $\leq$ is a *well-quasi-ordering* (wqo) compatible with the transition relation [1,9]. The algorithm to that end is a generic *backward coverability* procedure, which computes successively the sets of configurations that can cover y in at most $0, 1, 2, \ldots$ steps. Those sets are upwards-closed and since $\leq$ is a wqo they can be represented through their finitely many minimal elements.

Nevertheless, the naive complexity upper bounds one can extract directly from the termination argument of the backward coverability algorithm—which also relies on $\leq$ being a wqo—are sometimes very far from the optimal ones. A striking illustration is provided by vector addition systems (VAS): the complexity bounds offered e.g. by [8] are in Ackermann, whereas coverability in VAS has long been known to be ExpSpace-complete thanks to a lower bound by Lipton [14] and an upper bound by Racko [16].

Work funded in part by the Leverhulme Trust Visiting Professorship VP1-2014-041, and the EPSRC grant EP/M011801/1.

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 76–88, 2015.
DOI: 10.1007/978-3-319-24537-9_8

Rackoff's Technique is essentially combinatorial in nature: he shows by induction on the dimension of the VAS that, if x can reach one such $y' \geq y$, then there exists a small (doubly-exponential) run in the VAS witnessing this fact. A non-deterministic algorithm can then simply look for such a witness using only exponential space. The same general technique has since been extended to prove tight complexity upper bounds for coverability in numerous extensions of VASs [3,6,7,12,13]. It is however less clear how to adapt the technique for more general systems, where for instance the notion of dimension is absent or more involved.

Remarkably, Bozzelli and Ganty [5] showed that Rackoff's small witness property can be applied to the backward coverability algorithm for VAS to obtain a 2ExpTime upper bound.[1] However, their proof uses Rackoff's analysis as a black box, and does not work directly with the structures manipulated by the backward coverability algorithm. As such, it is again unclear how this result could be translated to further classes of well-structured transition systems.

Contributions. In this paper, we revisit the backward coverability algorithm for VAS, and extract directly a 2ExpTime upper bound for its running time. We take for this in Sec. 3 a dual view on the backward coverability algorithm, by considering successively the sets of configurations that do *not* cover y in 0, 1, 2, ... or fewer steps. Such sets are downwards-closed, and enjoy a (usually effective) canonical representation as finite unions of *ideals* [4,10,11]. We show in Sec. 4 that, in the case of VAS, this dual view exhibits an additional structural property of *ω-monotonicity*, which allows to derive the desired doubly-exponential bound.

Our purpose is above all pedagogical, as we hope to see this type of reasoning applied more broadly where the simple proof argument of Rackoff fails. As illustrations of the versatility of the framework, we consider in the full version of the paper (available from https:// hal. inria. fr/ hal-01176755) the top-down and bottom-up coverability problems in *alternating branching* VAS. In each case, we provide an instance of the generic backward algorithm that solves the problem, and show that its running time matches the known optimal complexities [6,7,13].

We start with some preliminaries on WSTS and ideals in Sec. 2.

2 Preliminaries

We first recall the necessary background on well-quasi-orders, well-structured transition systems, and ideal decompositions, while illustrating systematically the definitions on VAS and reset VAS.

2.1 Well-Structured Transition Systems

A *well-quasi-order* (wqo) $(X, \leq)$ is a set X equipped with a transitive reflexive relation $\leq$ such that, along any infinite sequence $x_0, x_1, \ldots$ of elements from X,

[1] In the same spirit, Majumdar and Wang [15] show that the 'expand, enlarge, and check' algorithm for bottom-up coverability in branching VASs runs in 2ExpTime, using the combinatorial analysis of Demri et al. [7].

one can find two indices $i < j$ such that $x_i \leq x_j$. A finite or infinite sequence without such pair of indices is *bad*, and necessarily finite over a wqo. See for instance [18] for more background on wqos.

Example 2.1 (Dickson's Lemma). The set $\mathbb{N}^d$ of d-dimensional vectors of natural numbers forms a wqo when endowed with the product ordering $\sqsubseteq$, defined by $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ if $\boldsymbol{u}(i) \leq \boldsymbol{v}(i)$ for all $1 \leq i \leq d$.

A *well-structured transition system* (WSTS) [1,9] is a triple $(X, \to, \leq)$ where X is a set of configurations, $\to \subseteq X \times X$ is a transition relation, and $(X, \leq)$ is a wqo with the following *compatibility* condition: if $x \leq x'$ and $x \to y$, then there exists $y' \geq y$ with $x' \to y'$. In other words, $\leq$ is a simulation relation on the transition system $(X, \to)$. We write as usual $\to^{\leq 0} \stackrel{\text{def}}{=} \{(x, x) \mid x \in X\}$ and $\to^{\leq k+1} \stackrel{\text{def}}{=} \to^{\leq k} \cup \{(x, y) \mid \exists z \in X \,.\, x \to z \to^{\leq k} y\}$ for the reachability relation in at most k steps, and $\to^* \stackrel{\text{def}}{=} \bigcup_k \to^{\leq k}$ for the reflexive transitive closure of $\to$.

Example 2.2 (VAS are WSTS). A d-dimensional *vector addition system* (VAS) is a finite set $\boldsymbol{A}$ of vectors in $\mathbb{Z}^d$. It defines a WSTS $(\mathbb{N}^d, \to, \sqsubseteq)$ with configurations space $\mathbb{N}^d$ and $\boldsymbol{u} \to \boldsymbol{u} + \boldsymbol{a}$ for all $\boldsymbol{u}$ in $\mathbb{N}^d$ and $\boldsymbol{a}$ in $\boldsymbol{A}$ such that $\boldsymbol{u} + \boldsymbol{a}$ is in $\mathbb{N}^d$.

For instance, the 2-dimensional VAS $\boldsymbol{A}_{\div 2} = \{(-2, 1)\}$ can be seen as weakly computing the halving function: from any configuration $(n, 0)$, it can reach $(n \bmod 2, \lfloor n/2 \rfloor)$ and all its reachable configurations (n', m) satisfy $m \leq n/2$.

Example 2.3 (Reset VAS are WSTS). A d-dimensional *reset VAS* is a finite subset $\boldsymbol{A}$ of $\mathbb{Z}^d \times \mathbb{P}(\{1, \ldots, d\})$. Given $R \subseteq \{1, \ldots, d\}$ and a vector $\boldsymbol{u}$, we define the vector $R(\boldsymbol{u})$ by $R(\boldsymbol{u})(i) = 0$ if $i \in R$, and $R(\boldsymbol{u})(i) = \boldsymbol{u}(i)$ otherwise. A reset VAS defines a WSTS $(\mathbb{N}^d, \to, \sqsubseteq)$ where $\boldsymbol{u} \to R(\boldsymbol{u} + \boldsymbol{a})$ if there exists $(\boldsymbol{a}, R)$ in $\boldsymbol{A}$ such that $\boldsymbol{u} + \boldsymbol{a}$ is in $\mathbb{N}^d$.

For instance, the 5-dimensional reset VAS

$$\boldsymbol{A}_{\log} = \left\{ \begin{array}{c} (0, 0, -2, 1, 0, \emptyset), (0, 0, 1, -1, 0, \emptyset), \\ (-1, 1, -2, 1, 0, \{3\}), (1, -1, 1, -1, 1, \{4\}) \end{array} \right\}$$

is a weak computer for the logarithm function: from any configuration of the form $(1, 0, 2^n, 0, 0)$, it can reach $(1, 0, 1, 0, n)$, and all its reachable configurations of the form $(1, 0, n', m, l)$ satisfy $l \leq n$.

2.2 Ideal Decompositions

The *downward-closure* of a subset $S \subseteq X$ over a wqo $(X, \leq)$ is $\downarrow X \stackrel{\text{def}}{=} \{x \in X \mid \exists s \in S \,.\, x \leq s\}$. A subset $D \subseteq X$ is *downwards-closed* if $\downarrow D = D$. We write $\downarrow x$ for the downward-closure of the singleton set $\{x\}$. Well-quasi-orders can also be characterised by the *descending chain condition*: a quasi-order $(X, \leq)$ is a wqo if and only if every descending sequence $D_0 \supsetneq D_1 \supsetneq D_2 \supsetneq \cdots$ of downwards-closed subsets $D_i \subseteq X$ is finite.

An *ideal* of X is a non-empty downwards-closed subset $I \subseteq X$, which is *directed*: if x, x' are two elements of I, then there exists y in I with $x \leq y$ and

$x' \leq y$. Over a wqo $(X, \leq)$, any downwards-closed set $D \subseteq X$ has a unique *decomposition* as a finite union of ideals $D = I_1 \cup \cdots \cup I_n$, where the I_j's are mutually incomparable for inclusion [4,10]. Alternatively, ideals are characterised as *irreducible* downwards-closed sets: an ideal is a non-empty downwards-closed set I with the property that, if $I \subseteq D_1 \cup D_2$ for two downwards-closed sets D_1 and D_2, then $I \subseteq D_1$ or $I \subseteq D_2$.

Example 2.4 (Vector Ideals). Over $(\mathbb{N}^d, \sqsubseteq)$, observe that $\downarrow \boldsymbol{u}$ is an ideal for every $\boldsymbol{u}$ in $\mathbb{N}^d$. Those are however not the only ideals, e.g. $I \stackrel{\text{def}}{=} \{(0, n, 0) \mid n \in \mathbb{N}\}$ is also an ideal. Write $\mathbb{N}_\omega \stackrel{\text{def}}{=} \mathbb{N} \uplus \{\omega\}$ where ω is a new top element; the product ordering $\sqsubseteq$ extends naturally to $\mathbb{N}_\omega^d$. Then the ideals of $(\mathbb{N}^d, \sqsubseteq)$ are exactly the downward-closures $\downarrow \boldsymbol{u}$ inside $\mathbb{N}^d$ of vectors $\boldsymbol{u}$ from $\mathbb{N}_\omega^d$. For the previous example, $\downarrow(0, \omega, 0) = I$.

Although ideals provide finite representations for manipulating downwards-closed sets, some additional effectiveness assumptions are necessary to employ them in algorithms. In this paper, we will say that a wqo $(X, \leq)$ has *effective* ideal representations [see 10,11, for more stringent requisites] if every ideal can be represented, and there are algorithms on those representations:

(CI) to check $I \subseteq I'$ for two ideals I and I',
(II) to compute the ideal decomposition of $I \cap I'$ for two ideals I and I',
(CU') to compute the ideal decomposition of the residual $X/x \stackrel{\text{def}}{=} \{x' \in X \mid x \not\leq x'\}$ for any x in X.

Example 2.5 (Effective Representations of Vector Ideals). We shall use vectors in $\mathbb{N}_\omega^d$ as representations. For (CI), given two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ in $\mathbb{N}_\omega^d$, $\downarrow \boldsymbol{u} \subseteq \downarrow \boldsymbol{v}$ if and only if $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$. Furthermore, for (II), $\downarrow \boldsymbol{u} \cap \downarrow \boldsymbol{v} = \downarrow \boldsymbol{w}$ where $\boldsymbol{w}(i) \stackrel{\text{def}}{=} \min_{\leq}(\boldsymbol{u}(i), \boldsymbol{v}(i))$ for all $1 \leq i \leq d$. Finally, for (CU'), if $\boldsymbol{u}$ is in $\mathbb{N}^d$, then $\mathbb{N}^d/\boldsymbol{u} = \bigcup_{1 \leq j \leq d \,|\, \boldsymbol{u}(j) > 0} \downarrow \boldsymbol{u}_{/j}$ where $\boldsymbol{u}_{/j}(i) = \omega$ if $i \neq j$ and $\boldsymbol{u}_{/j}(j) \stackrel{\text{def}}{=} \boldsymbol{u}(j) - 1$ otherwise.

Crucially for the applicability of our approach, effective ideal representations exist for most wqos of interest [10,11].

3 Backward Coverability

Let us recall in this section the generic backward coverability algorithm for well-structured transition systems [1,9]. We take a dual view on this algorithm, by considering downwards-closed sets represented through their ideal decompositions, instead of the usual view using upwards-closed sets represented through their minimal elements. We present the generic algorithm, but will illustrate all the notions using the case of VAS and reset VAS in Sec. 3.2, and derive naive upper bounds for both in Sec. 3.3—which will turn out optimal for reset VAS.

3.1 Generic Algorithm

Consider a WSTS $(X, \to, \leq)$ and a target configuration y from X to be covered. Define $D_* \stackrel{\text{def}}{=} \{x \in X \mid \forall y' \geq y \,.\, x \not\to^* y'\}$ as the set of configurations that do not cover y. The purpose of the backward coverability algorithm is to compute D_*; solving a coverability instance with source configuration x_0 then amounts to checking whether x_0 belongs to D_*. The idea of the algorithm is to compute successively for every k the set D_k of configurations that do *not* cover y in k steps or fewer:

$$D_* = \bigcap_k D_k \,, \qquad D_k \stackrel{\text{def}}{=} \{x \in X \mid \forall y' \geq y \,.\, x \not\to^{\leq k} y'\} \,. \tag{1}$$

These over-approximations D_k can be computed inductively on k by

$$D_0 = X/y \,, \qquad D_{k+1} = D_k \cap \mathrm{Pre}_\forall(D_k) \,, \tag{2}$$

where for any set $S \subseteq X$,

$$\mathrm{Pre}_\forall(S) \stackrel{\text{def}}{=} \{x \in X \mid \forall z \in X \,.\, (x \to z \implies z \in S)\} \,. \tag{3}$$

The algorithm terminates as soon as $D_k \subseteq D_{k+1}$ (and thus $D_{k+j} = D_k = D_*$ for all j). This is guaranteed to arise eventually by the descending chain condition, since otherwise we would have an infinite descending chain of downwards-closed sets $D_0 \supsetneq D_1 \supsetneq D_2 \supsetneq \cdots$.

Correctness. The correctness of the algorithm hinges on the following claim:

Claim 3.1 (Correctness). Equations (1) and (2) define the same D_k.

Proof. By induction on k. For the base case, $x \not\to^{\leq 0} y'$ for all $y' \geq y$, if and only if $x \not\geq y$, i.e. if and only if x is in X/y. For the induction step and for all $y' \geq y$, $x \not\to^{\leq k+1} y'$ if and only if $x \not\to^{\leq k} y'$ and there does not exist any z with $x \to z$ and $z \to^{\leq k} y'$. The former is equivalent to x belonging to D_k by induction hypothesis. The latter occurs if and only if for all z in X, if $x \to z$ then $z \not\to^{\leq k} y'$, i.e. if and only if x belongs to $\mathrm{Pre}_\forall(D_k)$ by induction hypothesis. □

Effective Ideal Representations. The algorithm as presented above relies on the effectiveness of Eq. (2). We are going to use effective representations of the ideal decompositions of the D_k to this end. Let us first check that we are indeed dealing with downwards-closed sets:

Claim 3.2 (Downward-closure). For all k, D_k is downwards-closed.

Proof. By induction on k. For the base case, $D_0 = X/y$ is downwards-closed. For the induction step, first observe that, if D is downwards-closed, then $\mathrm{Pre}_\forall(D)$ is also downwards-closed. Indeed, let $x \leq x'$ for some x' in $\mathrm{Pre}_\forall(D)$. Consider any z such that $x \to z$. Then by WSTS compatibility, there exists $z' \geq z$ such that $x' \to z'$. Since x' belongs to $\mathrm{Pre}_\forall(D)$, z' belongs to D. Because D is downwards-closed, z also belongs to D. This shows x in $\mathrm{Pre}_\forall(D)$ as desired. We conclude by noting that downwards-closed sets are closed under intersection, hence $D_{k+1} = D_k \cap \mathrm{Pre}_\forall(D_k)$ is downwards-closed by applying the induction hypothesis to D_k. □

The only additional effectiveness assumption we make is that:

(Pre) the ideal decomposition of $\mathrm{Pre}_\forall(D)$ is computable for all downwards-closed D.

This is sufficient to compute the ideal decompositions of all the D_k. Indeed, initially D_0 is computed using (CU'). Later, $\mathrm{Pre}_\forall(D_k)$ is computable by (Pre), and its intersection with D_k is also computable by (II). Finally, recall that, by ideal irreducibility, $I_1 \cup \cdots \cup I_n \subseteq J_1 \cup \cdots \cup J_m$ for ideals $I_1, \ldots, I_n$ and downwards-closed $J_1, \ldots, J_m$ if and only if for all $1 \leq i \leq n$ there exists $1 \leq j \leq m$ such that $I_i \subseteq J_j$. Therefore, the termination check $D_k \subseteq D_{k+1}$ is effective by (CI).

3.2 Coverability for VAS and Reset VAS

In order to instantiate the backward coverability algorithm for VAS and reset VAS, we merely need to prove that they also satisfy the (Pre) effectiveness assumption: given a downwards-closed $D = {\downarrow}\boldsymbol{u}_1 \cup \cdots \cup {\downarrow}\boldsymbol{u}_m$ for some $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$ in $\mathbb{N}_\omega^d$, compute a finite set of vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ from $\mathbb{N}_\omega^d$ such that $\mathrm{Pre}_\forall(D) = {\downarrow}\boldsymbol{v}_1 \cup \cdots \cup {\downarrow}\boldsymbol{v}_n$. Using (CI) we can then select the maximal such $\boldsymbol{v}_j$ to obtain incomparable ideals.

Universal Predecessors in VAS. Thanks to (II) and the fact that $\boldsymbol{A}$ is finite (VAS are finitely branching), we start by reducing our computation to that of predecessors along a specific action $\boldsymbol{a}$ from $\boldsymbol{A}$: $\mathrm{Pre}_\forall(D) = \bigcap_{\boldsymbol{a} \in \boldsymbol{A}} \mathrm{Pre}_{\boldsymbol{a}}(D)$ where

$$\mathrm{Pre}_{\boldsymbol{a}}(D) \stackrel{\text{def}}{=} \{\boldsymbol{v} \in \mathbb{N}^d \mid \boldsymbol{v} + \boldsymbol{a} \in \mathbb{N}^d \implies \boldsymbol{v} + \boldsymbol{a} \in D\} \tag{4}$$

$$= \{\boldsymbol{v} \in \mathbb{N}^d \mid \boldsymbol{v} + \boldsymbol{a} \notin \mathbb{N}^d\} \cup \{\boldsymbol{v} \in \mathbb{N}^d \mid \boldsymbol{v} + \boldsymbol{a} \in D\} \tag{5}$$

$$= \mathbb{N}^d / \theta(\boldsymbol{a}) \cup \{\boldsymbol{v} \in \mathbb{N}^d \mid \boldsymbol{v} + \boldsymbol{a} \in D\}\,, \tag{6}$$

where $\theta(\boldsymbol{a}) \stackrel{\text{def}}{=} \min_{\sqsubseteq}\{\boldsymbol{v} \in \mathbb{N}^d \mid \boldsymbol{v} + \boldsymbol{a} \in \mathbb{N}^d\}$ is called the *threshold* of $\boldsymbol{a}$ and can be computed for all $1 \leq i \leq d$ by

$$\theta(\boldsymbol{a})(i) = \begin{cases} 0 & \text{if } \boldsymbol{a}(i) \geq 0 \\ -\boldsymbol{a}(i) & \text{otherwise.} \end{cases} \tag{7}$$

Thus by (CU') it only remains to compute a representation for the decomposition of $\{\boldsymbol{v} \in \mathbb{N}^d \mid \boldsymbol{v} + \boldsymbol{a} \in D\} = \bigcup_{1 \leq j \leq m} \{\boldsymbol{v} \in \mathbb{N}^d \mid \boldsymbol{v} + \boldsymbol{a} \in {\downarrow}\boldsymbol{u}_j\}$. For each ideal ${\downarrow}\boldsymbol{u}_j$ in the decomposition of D, $\{\boldsymbol{v} \in \mathbb{N}^d \mid \boldsymbol{v} + \boldsymbol{a} \in {\downarrow}\boldsymbol{u}_j\}$ is either the empty set if $\boldsymbol{u}_j \not\geq \theta(-\boldsymbol{a})$, or ${\downarrow}(\boldsymbol{u} - \boldsymbol{a})$ otherwise, where addition is extended with $\omega + z = \omega$ for all z in $\mathbb{Z}$.

Example 3.3. Recall the VAS $\boldsymbol{A}_{\div 2} = \{(-2, 1)\}$ from Example 2.2. Setting $D_0 = {\downarrow}(\omega, 4)$, the backward coverability algorithm computes the set of all configurations from which $\boldsymbol{A}_{\div 2}$ cannot compute at least 5 in its second component; see Fig. 1.

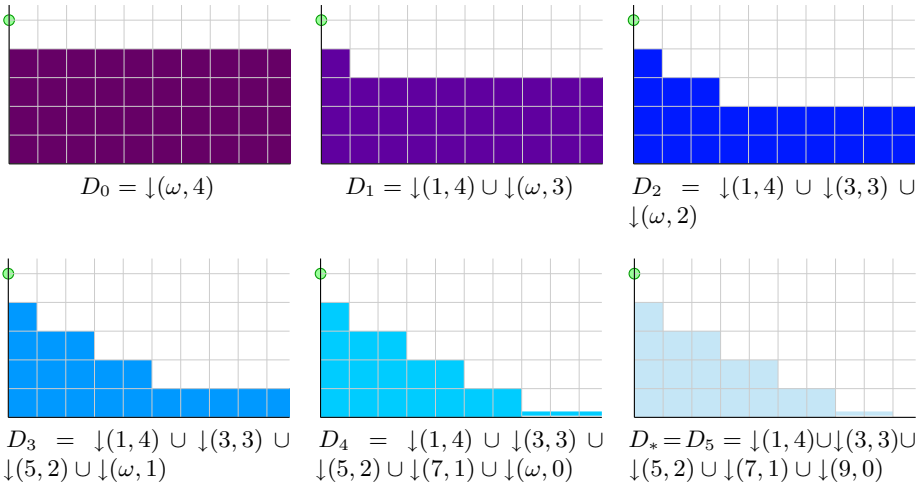


Fig. 1. The successive D_k for $\boldsymbol{A}_{\div 2}$ with target $\boldsymbol{t} = (0, 5)$.

Universal Predecessors in Reset VAS. The same reasoning holds for reset VAS as for VAS: $\mathrm{Pre}_\forall(D) = \bigcap_{(\boldsymbol{a},R)\in\boldsymbol{A}} \left(\mathbb{N}^d/\theta(\boldsymbol{a}) \cup \bigcup_{1\le j\le m}\{\boldsymbol{v} \in \mathbb{N}^d \mid R(\boldsymbol{v}+\boldsymbol{a}) \in \downarrow\boldsymbol{u}_j\}\right)$. In order to compute a representation for this last set, given a vector $\boldsymbol{v}$ in $\mathbb{N}^d_\omega$ and $R \subseteq \{1, \ldots, d\}$, define $\overline{\boldsymbol{v}}^R$ as the vector in $\mathbb{N}^d_\omega$ with ω's in the components of R:

$$\overline{\boldsymbol{v}}^R(i) \stackrel{\text{def}}{=} \begin{cases} \omega & \text{if } i \in R \\ \boldsymbol{v}(i) & \text{otherwise.} \end{cases} \tag{8}$$

Then $\{\boldsymbol{v} \in \mathbb{N}^d \mid R(\boldsymbol{v}+\boldsymbol{a}) \in \downarrow\boldsymbol{u}_j\}$ (where $R(\boldsymbol{v}+\boldsymbol{a})$ is defined as in Example 2.4) is either the empty set if $\overline{\boldsymbol{u}_j}^R \not\sqsupseteq \theta(-\boldsymbol{a})$, or $\downarrow\left(\overline{\boldsymbol{u}_j}^R - \boldsymbol{a}\right)$ otherwise.

Example 3.4. Recall the reset VAS $\boldsymbol{A}_{\log}$ from Example 2.3, in which the first two vector components are used to encode two control states. Setting

$$D_0 = \downarrow(1, 0, \omega, \omega, 1) \cup \downarrow(0, 1, \omega, \omega, 0)\ ,$$

the backward coverability algorithm computes as follows the set of all configurations from which $\boldsymbol{A}_{\log}$ cannot compute in its last component either at least 2 in state $(1, 0)$ or at least 1 in state $(0, 1)$. The interesting part of the computation for the subsequent discussion occurs from $k = 2$ to $k = 4$:

$$\begin{aligned} D_2 &= \downarrow(0,0,\omega,\omega,1) \cup \downarrow(1,0,1,0,1) \cup \downarrow(1,0,0,\omega,1) \cup \downarrow(1,0,\omega,\omega,0) \cup \\ &\quad \downarrow(0,1,\omega,0,0) \cup \downarrow(0,1,0,\omega,0)\ , \\ D_3 &= \downarrow(0,0,\omega,\omega,1) \cup \downarrow(1,0,1,0,1) \cup \downarrow(1,0,0,1,1) \cup \downarrow(1,0,\omega,\omega,0) \cup \\ &\quad \downarrow(0,1,2,0,0) \cup \downarrow(0,1,0,1,0)\ , \\ D_4 &= \downarrow(0,0,\omega,\omega,1) \cup \downarrow(1,0,1,0,1) \cup \downarrow(1,0,0,1,1) \cup \downarrow(1,0,1,\omega,0) \cup \\ &\quad \downarrow(1,0,\omega,0,0) \cup \downarrow(0,1,2,0,0) \cup \downarrow(0,1,0,1,0)\ . \end{aligned}$$

3.3 Ackermann Upper Bounds

Let us finally show how to bound the running time of the backward coverability algorithm on VAS and reset VAS. The main ingredient to that end is a combinatorial statement on the length of *controlled* descending chains of downwards-closed sets.

Controlled Descending Chains. Consider some set X with a norm $\|.\|: X \to \mathbb{N}$. Given a monotone *control* function $g: \mathbb{N} \to \mathbb{N}$ and an *initial norm* $n \in \mathbb{N}$, we say that a sequence $x_0, x_1, \dots$ of elements from X is (g,n)-*controlled* if $\|x_i\| \leq g^i(n)$ the ith iterate of g applied to n. In particular, $\|x_0\| \leq n$ initially.

This notion can be applied to the descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ constructed by the backward coverability algorithm for a d-dimensional VAS or reset VAS $\boldsymbol{A}$ and target vector $\boldsymbol{t} \in \mathbb{N}^d$. We define for this $\|.\|$ as the infinity norm on elements and finite subsets of $\mathbb{Z}_\omega^d \stackrel{\text{def}}{=} (\mathbb{Z} \uplus \{\omega\})^d$, i.e. the maximum absolute value of any occurring integer. For instance, $\|\{(1,\omega,5),(0,\omega,\omega)\}\| = 5$, and in Example 2.2 $\|\boldsymbol{A}_{\div 2}\| = 2$. When considering a downwards-closed set D with decomposition $\downarrow\boldsymbol{u}_1 \cup \cdots \cup \downarrow\boldsymbol{u}_m$, we define $\|D\| \stackrel{\text{def}}{=} \|\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_m\}\|$. Hence what is controlled in a descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ is its ideal representation.

Claim 3.5 (Control for VAS and Reset VAS). The descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ is (g,n)-controlled for $g(x) \stackrel{\text{def}}{=} x + \|\boldsymbol{A}\|$ and $n \stackrel{\text{def}}{=} \|\boldsymbol{t}\|$.

Proof. The fact that $\|D_0\| \leq \|\boldsymbol{t}\|$ follows from (CU'). Regarding the control function g, observe that taking unions and intersections of ideals using (II) cannot increase the norm. Hence it suffices to show that $\|\mathrm{Pre}_\forall(D)\| \leq \|D\| + \|\boldsymbol{A}\|$ for all $D = \downarrow\boldsymbol{u}_1 \cup \cdots \cup \downarrow\boldsymbol{u}_m$. Note that for reset VAS, $\|\overline{\boldsymbol{u}_j}^R - \boldsymbol{a}\| \leq \|\boldsymbol{u}_j - \boldsymbol{a}\|$. Hence for both VAS and reset VAS, $\|\mathrm{Pre}_\forall(D)\| \leq \max_{\boldsymbol{a}} \max_{1 \leq j \leq m}(\|\mathbb{N}^d/\theta(\boldsymbol{a})\|, \|\boldsymbol{u}_j - \boldsymbol{a}\|)$. We conclude by observing that $\|\mathbb{N}^d/\theta(\boldsymbol{a})\| \leq \|\boldsymbol{a}\| \leq \|\boldsymbol{A}\|$ by (CU') and $\|\boldsymbol{u}_j - \boldsymbol{a}\| \leq \|\boldsymbol{u}_j\| + \|\boldsymbol{a}\| \leq \|D\| + \|\boldsymbol{A}\|$. □

Upper Bound. Consider a computation $D_0 \supsetneq D_1 \supsetneq \cdots \supsetneq D_\ell = D_{\ell+1}$ of the backward coverability algorithm for a VAS or a reset VAS. At each step $0 \leq k \leq \ell$, the cost of computing D_{k+1} from D_k and of checking for termination is polynomial in $\|A\|$ and $\|D_k\|$. The difficulty is to evaluate how large ℓ can be.

The idea here is that, at every step $0 \leq k < \ell$, there is at least one *proper* ideal $\downarrow\boldsymbol{v}_k$: an ideal appearing in the representation of D_k but not in that of D_{k+1}; then $\downarrow\boldsymbol{v}_k \subseteq D_k$ but $\downarrow\boldsymbol{v}_k \not\subseteq D_{k+1}$. Note that for all $0 \leq j < k < \ell$, $\boldsymbol{v}_j \not\sqsubseteq \boldsymbol{v}_k$, since the contrary would entail $\downarrow\boldsymbol{v}_j \subseteq \downarrow\boldsymbol{v}_k \subseteq D_k \subseteq D_{j+1}$. Hence the sequence $(\boldsymbol{v}_k)_{0 \leq k < \ell}$ is a *bad* sequence, which is also controlled by (g,n) according to Claim 3.5. Using the combinatorial results from [18, Cor. 2.25 and Thm. 2.34] on such bad sequences, we obtain (see the full paper for details):

Theorem 3.6. (Length Function Theorem for Descending Chains). *Let $n > 0$. Any (g,n)-controlled descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ of downwards-closed subsets of $\mathbb{N}^d$ is of length at most $h_{\omega^{d+1}}(n \cdot d!)$, where $h(x) \stackrel{\text{def}}{=} d \cdot g(x)$.*

Here h_α for an ordinal α and base function h denotes the αth Cichoń function [18]. Each of the ℓ steps of computation can furthermore be performed in time polynomial in $g^\ell(n)$.

Since g is primitive-recursive according to Claim 3.5, the overall complexity for an instance of size n is bounded by ackermann$(p(n))$ for some primitive-recursive function p, which lies within the complexity class Ackermann [17]. Such an upper bound is overly pessimistic for VAS, but is actually tight for reset VAS: coverability for reset VAS is indeed complete for Ackermann [18,19].

4 Complexity for VAS

We know from Bozzelli and Ganty's 2ExpTime upper bound [5] for the backward coverability algorithm that the Ackermann upper bound from the previous section is far from tight in the case of VAS. We show in this section that the descending chains $D_0 \supsetneq D_1 \supsetneq \cdots$ computed by the backward coverability algorithm for VAS enjoy a structural invariant, which we dub *ω-monotonicity*, and which is absent from the chains computed for reset VAS. In turn, we show in Example 4.4, that controlled decreasing chains that are ω-monotone are much shorter, allowing us to derive the desired 2ExpTime bound in Cor. 4.6.

4.1 Transitions Between Proper Ideals

The proof of ω-monotonicity in the case of VAS can be shown directly, but reflects a more general *proper transition sequence* property of the generic backward coverability algorithm, which we are going to show in the general setting.

Let us first lift the transition relation $\to$ of a WSTS $(X, \to, \leq)$ to work over ideals. Define for any ideal I of X

$$\mathrm{Post}_\exists(I) \stackrel{\text{def}}{=} \{z \in X \mid \exists x \in I \,.\, x \to z\} \,. \tag{9}$$

Then $\downarrow\mathrm{Post}_\exists(I)$ is downwards-closed with a unique decomposition into maximal ideals. We follow Blondin et al. [2] and write '$I \to J$' if J is an ideal from the decomposition of $\downarrow\mathrm{Post}_\exists(I)$.

Example 4.1 (Transitions over Vector Ideals). In the case of a VAS $\boldsymbol{A}$, observe that, if $\boldsymbol{v}$ is a vector from $\mathbb{N}_\omega^d$, then $\mathrm{Post}_\exists(\downarrow\boldsymbol{v}) = \bigcup_{\boldsymbol{a}\in\boldsymbol{A}} \downarrow(\boldsymbol{v}+\boldsymbol{a})$. Each such $\downarrow(\boldsymbol{v}+\boldsymbol{a})$ is already an ideal. In the case of a reset VAS $\boldsymbol{A}$, we have similarly $\mathrm{Post}_\exists(\downarrow\boldsymbol{v}) = \bigcup_{(\boldsymbol{a},R)\in\boldsymbol{A}} \downarrow R(\boldsymbol{v}+\boldsymbol{a})$.

We can now state the result that motivates this subsection:

Claim 4.2 (Proper Transition Sequence). If I_{k+1} is a proper ideal of D_{k+1}, then there exist an ideal J and a proper ideal I_k of D_k such that $I_{k+1} \to J \subseteq I_k$.

Proof. An ideal is proper in D_k if and only if it intersects the set of elements *excluded* between steps k and $k+1$: by basic set operations, first observe that (2) is equivalent to

$$D_{k+1} = D_k \setminus \{x \in D_k \mid \exists z \notin D_k \,.\, x \to z\} \,. \tag{10}$$

Moreover, noting $D_{-1} \stackrel{\text{def}}{=} X$, z in (10) must belong to D_{k-1}, or x would have already been excluded before step k. We have therefore $D_{k+1} = D_k \setminus E_k$ where

$$E_{-1} \stackrel{\text{def}}{=} \{x \in X \mid x \geq y\} , \qquad E_k \stackrel{\text{def}}{=} \{x \in D_k \mid \exists z \in E_{k-1} \,.\, x \to z\} . \tag{11}$$

Consider now a proper ideal I_{k+1} of D_{k+1}: this means $I_{k+1} \cap E_{k+1} \neq \emptyset$. This implies in turn $\downarrow \mathrm{Post}_\exists(I_{k+1}) \cap E_k \neq \emptyset$ by (11), thus there exists J such that $I_{k+1} \to J$ and $J \cap E_k \neq \emptyset$.

Since $I_{k+1} \subseteq D_{k+1} \subseteq \mathrm{Pre}_\forall(D_k)$ by (2), we also know that $\mathrm{Post}_\exists(I_{k+1}) \subseteq D_k$. By ideal irreducibility, it means that $J \subseteq I_k$ for some ideal I_k from the decomposition of D_k. Observe finally that $I_k \cap E_k \neq \emptyset$, i.e. that I_k is proper. □

4.2 ω-Monotonicity

For $\boldsymbol{u}$ in $\mathbb{N}_\omega^d$, its *ω-set* is the subset $\omega(\boldsymbol{u})$ of $\{1, \ldots, d\}$ such that $\boldsymbol{u}(i) = \omega$ if and only if $i \in \omega(\boldsymbol{u})$. We say that a descending chain $D_0 \supsetneq D_1 \supsetneq \cdots \supsetneq D_\ell$ of downwards-closed subsets of $\mathbb{N}^d$ is *ω-monotone* if for all $0 \leq k < \ell - 1$ and all proper ideals $\downarrow \boldsymbol{v}_{k+1}$ in the decomposition of D_{k+1}, there exists a proper ideal $\downarrow \boldsymbol{v}_k$ in the decomposition of D_k such that $\omega(\boldsymbol{v}_{k+1}) \subseteq \omega(\boldsymbol{v}_k)$.

Claim 4.3 (VAS Descending Chains are ω-Monotone). The descending chains computed by the backward coverability algorithm for VAS are ω-monotone.

Proof. Let $D_0 \supsetneq D_1 \supsetneq \cdots \supsetneq D_\ell$ be the descending chain computed for a VAS $\boldsymbol{A}$. Suppose $0 \leq k < \ell - 1$ and $\downarrow \boldsymbol{v}_{k+1}$ is a proper ideal in the decomposition of D_{k+1}. By Claim 4.2, there exists a proper ideal $\downarrow \boldsymbol{v}_k$ in the decomposition of D_k such that $\boldsymbol{v}_{k+1} + \boldsymbol{a} \sqsubseteq \boldsymbol{v}_k$. We conclude that $\omega(\boldsymbol{v}_{k+1}) \subseteq \omega(\boldsymbol{v}_k)$. □

As we can see with Example 3.4 however, the descending chains computed for reset VAS are in general *not* ω-monotone: $(1, 0, \omega, \omega, 0)$ is proper in D_3 and has a proper transition to $(0, 1, 0, \omega, 0)$ in D_2 using $(-1, 1, -2, 1, 0, \{3\})$ from $\boldsymbol{A}_{\log}$, but no ideal with $\{3, 4\}$ as ω-set is proper in D_2.

4.3 Upper Bound

We are now in position to state a refinement of Thm. 3.6 for ω-monotone controlled descending chains. For a control function $g \colon \mathbb{N} \to \mathbb{N}$, define the function $\widetilde{g} \colon \mathbb{N}^2 \to \mathbb{N}$ by induction on its first argument:

$$\widetilde{g}(0, n) \stackrel{\text{def}}{=} 1 , \qquad \widetilde{g}(m+1, n) \stackrel{\text{def}}{=} \widetilde{g}(m, n) + (g^{\widetilde{g}(m,n)}(n) + 1)^{m+1} . \tag{12}$$

Theorem 4.4 (Length Function Theorem for ω-Monotone Descending Chains). *Any (g, n)-controlled ω-monotone descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ of downwards-closed subsets of $\mathbb{N}^d$ is of length at most $\widetilde{g}(d, n)$.*

Proof. Note that D_ℓ the last element of the chain has the distinction of not having any proper ideal, hence it suffices to bound the index k of the last set D_k with a proper ideal $\downarrow \boldsymbol{v}_k$, and add one to get a bound on ℓ. We are going to establish by induction on $d - |I|$ that if $\downarrow \boldsymbol{v}_k$ is a proper ideal from the decomposition of D_k and its ω-set is I, then $k < \widetilde{g}(d - |I|, n)$, which by monotonicity of $\widetilde{g}$ in its first argument entails $k < \widetilde{g}(d, n)$ as desired.

For the base case, $|I| = d$ implies that $\boldsymbol{v}_k$ is the vector with ω's in every coordinate, which can only occur in D_0. The inductive step is handled by the following claim, when setting $k < \widetilde{g}(d - |I| - 1, n)$ by induction hypothesis for the last index with a proper ideal whose ω-set strictly includes I:

Claim 4.5. Let I and $k < k'$ be such that:

(i) for all $j \in \{k+1, \dots, k'-1\}$, the decomposition of D_j does not contain a proper ideal whose ω-set strictly includes I;
(ii) the decomposition of $D_{k'}$ contains a proper ideal whose ω-set is I.

Then we have $k' - k \leq (\|D_{k+1}\| + 1)^{(d-|I|)}$.

For a proof, from assumption (ii), by applying the ω-monotonicity for $j = k' - 1, k' - 2, \dots, k+1$ and due to assumption (i), there exists a proper ideal $\downarrow \boldsymbol{v}_j$ in the decomposition of D_j and such that $\omega(\boldsymbol{v}_j) = I$ for all $j \in \{k+1, \dots, k'\}$. Since they are proper, those $k' - k$ vectors are mutually distinct.

Consider any such $\boldsymbol{v}_j$. Since $D_{k+1} \supseteq D_j$, by ideal irreducibility there exists a vector $\boldsymbol{u}_j$ in the decomposition of D_{k+1} such that $\boldsymbol{v}_j \sqsubseteq \boldsymbol{u}_j$. We have that $\omega(\boldsymbol{u}_j) = I$, since otherwise $\boldsymbol{u}_j$ would be proper at $D_{j'}$ for some $j' \in \{k+1, \dots, j-1\}$, which would contradict assumption (i). Hence $\|\boldsymbol{v}_j\| \leq \|\boldsymbol{u}_j\| \leq \|D_{k+1}\|$.

To conclude, note that there can be at most $(\|D_{k+1}\| + 1)^{(d-|I|)}$ mutually distinct vectors in $\mathbb{N}^d_\omega$ with I as ω-set and norm bounded by $\|D_{k+1}\|$. □

Finally, putting together Claim 3.5 (control for VAS), Claim 4.3 (ω-monotonicity), and Thm. 4.4 (lengths of controlled ω-monotone descending chains), we obtain that the backward coverability algorithm for VAS runs in 2ExpTime, and in pseudo-polynomial time if d is fixed.

Corollary 4.6. *For any d-dimensional VAS $\boldsymbol{A}$ and target vector $\boldsymbol{t}$, the backward coverability algorithm terminates after at most $((\|\boldsymbol{A}\| + 1)(\|\boldsymbol{t}\| + 2))^{(d+1)!}$ steps.*

Proof. Let $h(m, n) = \widetilde{g}(m, n)(\|\boldsymbol{A}\| + 1)(n + 2)$ where $g(x) = x + \|\boldsymbol{A}\|$. We have $h(m+1, n) \leq (h(m, n))^{m+2}$, so $\widetilde{g}(m, n) \leq h(m, n) \leq ((\|\boldsymbol{A}\|+1)(n+2))^{(m+1)!}$. □

5 Concluding Remarks

Rackoff's technique has successfully been employed to prove tight upper bounds for the coverability problem in VAS and extensions [3,6,7,12,13]. However, the technique does not readily generalise to more complex classes of well-structured transition systems, e.g. where configurations are not vectors of natural numbers.

We have shown that the same complexity bounds can be extracted in a principled way, by considering the ideal view of the backward coverability algorithm for VAS, and by noticing a structural invariant on its computations. Essentially the same arguments suffice to re-prove several recent upper bounds [6,7,13].

This paves the way for future investigations on coverability problems with large complexity gaps (where different structural invariants will need to be found).

References

1. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.K.: Algorithmic analysis of programs with well quasi-ordered domains. Inform. and Comput. **160**(1/2), 109–127 (2000)
2. Blondin, M., Finkel, A., McKenzie, P.: Handling infinitely branching WSTS. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part II. LNCS, vol. 8573, pp. 13–25. Springer, Heidelberg (2014)
3. Bonnet, R., Finkel, A., Praveen, M.: Extending the Rackoff technique to affine nets. FSTTCS 2012. LIPIcs, vol. 18, pp. 301–312. LZI (2012)
4. Bonnet, R.: On the cardinality of the set of initial intervals of a partially ordered set. Infinite and finite sets: to Paul Erdős on his 60th birthday, vol. 1, pp. 189–198. Coll. Math. Soc. János Bolyai, North-Holland (1975)
5. Bozzelli, L., Ganty, P.: Complexity analysis of the backward coverability algorithm for VASS. In: Delzanno, G., Potapov, I. (eds.) RP 2011. LNCS, vol. 6945, pp. 96–109. Springer, Heidelberg (2011)
6. Courtois, J.-B., Schmitz, S.: Alternating vector addition systems with states. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 220–231. Springer, Heidelberg (2014)
7. Demri, S., Jurdziński, M., Lachish, O., Lazić, R.: The covering and boundedness problems for branching vector addition systems. J. Comput. Syst. Sci. **79**(1), 23–38 (2013)
8. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermannian and primitive-recursive bounds with Dickson's Lemma. In: LICS 2011, pp. 269–278. IEEE Computer Society (2011)
9. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere!. Theor. Comput. Sci. **256**(1–2), 63–92 (2001)
10. Finkel, A., Goubault-Larrecq, J.: Forward analysis for WSTS, part I: Completions. In: Proc. STACS 2009. LIPIcs, vol. 3, pp. 433–444. LZI (2009)
11. Goubault-Larrecq, J., Karandikar, P., Narayan Kumar, K., Schnoebelen, P.: The ideal approach to computing closed subsets in well-quasi-orderings (in preparation) (2015)
12. Kochems, J., Ong, C.H.L.: Decidable models of recursive asynchronous concurrency (2015) (preprint) http://arxiv.org/abs/1410.8852
13. Lazić, R., Schmitz, S.: Non-elementary complexities for branching VASS, MELL, and extensions. ACM Trans. Comput. Logic 16(3:20), 1–30 (2015)
14. Lipton, R.: The reachability problem requires exponential space. Tech. Rep. 62, Yale University (1976)
15. Majumdar, R., Wang, Z.: Expand, Enlarge, and Check for Branching Vector Addition Systems. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013 – Concurrency Theory. LNCS, vol. 8052, pp. 152–166. Springer, Heidelberg (2013)

16. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theor. Comput. Sci. **6**(2), 223–231 (1978)
17. Schmitz, S.: Complexity hierarchies beyond Elementary (2013) (preprint) http://arxiv.org/abs/1312.5686
18. Schmitz, S., Schnoebelen, P.: Algorithmic aspects of WQO theory. Lecture notes (2012). http://cel.archives-ouvertes.fr/cel-00727025
19. Schnoebelen, P.: Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 616–628. Springer, Heidelberg (2010)

Synthesis Problems for One-Counter Automata

Antonia Lechner(✉)

Department of Computer Science, University of Oxford, Oxford, UK
antonia.lechner@cs.ox.ac.uk

Abstract. We consider the LTL synthesis problem for one-counter automata with integer-valued parameters, where counter values range over the nonnegative integers and counter updates are encoded in binary. This problem asks whether for a given parametric one-counter automaton and LTL formula there exist values for the parameters such that all computations from the initial configuration satisfy the formula. We show that LTL synthesis is decidable by translating it to a formula of a decidable fragment of Presburger arithmetic with divisibility.

1 Introduction

Counter automata are an important type of infinite-state computing device with applications especially in formal verification. They are often used as computational models for infinite-state systems, and in general they are equivalent to Turing machines [15]. However, various problems have been shown to be decidable for some special classes of counter automata, for example reversal-bounded counter automata [10] and one-counter automata. Many applications have been found for one-counter automata, ranging from modelling resource-bounded systems and programs with lists to XML query evaluation, see e.g. [2,4].

Generally, counter automata can be considered as acceptors with a read-only input tape [8]. In a similar vein one can consider counter automata with parameters (or read-only input registers) and ask for which parameter values a given automaton has an accepting computation [9]. In particular, linear time and branching time model checking problems for parametric one-counter automata were studied in [5]. These model checking problems treat the parameters *universally*: one asks that a specification be satisfied for all parameter values. In other words, one asks for correctness in all contexts.

The equivalent synthesis problems, where parameters are treated *existentially*, were shown to be undecidable in the case of computation tree logic, and stated as an open problem in the case of linear time logic in [5]. In the present paper, we study these LTL synthesis problems and obtain decidability with an upper complexity bound of **3NEXPTIME** for the general problem of deciding whether there exist parameter values in a given automaton such that all infinite computations from the initial configuration satisfy a given LTL specification, by reduction to a decidable fragment of Presburger arithmetic with divisibility. Intuitively, LTL synthesis generalises a variety of situations where an infinite-state system is only partly given, and we wish to complete it to ensure that it is guaranteed to satisfy certain requirements.

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 89–100, 2015.
DOI: 10.1007/978-3-319-24537-9_9

Presburger arithmetic with divisibility (PAD) is the first-order logic over the integers with an addition operation and ordering and divisibility relations. PAD has been shown to be undecidable [16], even when restricted to at most one quantifier alternation [13]. However, several decidable subsets have been identified, which have proved to be important tools to encode various computational problems and establish complexity bounds for them. In particular, several decidability results for model checking parametric counter machines have been obtained by reduction to the existential fragment of PAD. The latter was shown to be decidable in [12] and an upper bound of **NEXPTIME** was shown in [11].

For the synthesis problems considered in this paper the relevant fragment of PAD is the set of formulas of the form $\forall \boldsymbol{z} \exists \boldsymbol{x} \varphi(\boldsymbol{x}, \boldsymbol{z})$, where the only variables allowed to occur on the left side of a divisibility in the quantifier-free formula φ are $\boldsymbol{z}$. We will call this language $\forall\exists_R$PAD. Using similar methods to those described in [3], we will show that this language is decidable in **co-2NEXPTIME**.

This result will then be used to show decidability of linear time synthesis problems for parametric one-counter automata. The basic synthesis problem we consider, which we call Büchi synthesis, is whether there exist parameter values for a given one-counter automaton such that all infinite computations that start from the initial configuration satisfy a Büchi acceptance condition. We give a **3NEXPTIME** bound for this problem by translating its negation to an equivalent $\forall\exists_R$PAD formula.[1]

The translation is based on a symbolic encoding of computations of the automaton, following the idea from [7] of representing any given computation by a constant number of flows in a network, which serve as polynomial-size certificates of reachability. Intuitively, in the negation of the Büchi synthesis problem (is it true that for all possible values of the parameters, there is at least one computation that visits each accepting state only finitely often), the integer-valued parameters in the automaton correspond to universally quantified variables in the resulting $\forall\exists_R$PAD formula, and the existence of a computation with the desired property is encoded via existential quantifiers. We derive the same upper bound for the synthesis problem for LTL specifications by reducing it to Büchi synthesis. We also prove $\mathbf{NP^{NP}}$-hardness of Büchi synthesis using a reduction from a generalised version of the subset sum problem.

2 Complexity of $\forall\exists_R$PAD

Presburger arithmetic (PA) is the first-order logic over $\langle \mathbb{Z}, +, <, 0, 1 \rangle$, where $+$ and $<$ are the standard addition and ordering of integers. Presburger arithmetic with divisibility (PAD) is an extension of PA which also includes a binary division symbol $|$, i.e., it is the first-order logic over $\langle \mathbb{Z}, +, |, <, 0, 1 \rangle$. The full language of PAD is undecidable [16], and so is the subset of PAD which is restricted to at

[1] The reason why we chose to encode the negation of the Büchi synthesis problem, rather than the problem in its positive form, is that the details of the calculations are somewhat simpler and easier to follow.

most one quantifier alternation [13], but the purely existential subset (∃PAD) is known to be decidable [12] and in **NEXPTIME** [11]. Correspondingly, the purely universal subset (∀PAD) is in **co-NEXPTIME**.

In this section, we establish an upper bound for the decision problem for a logic which we call $\forall\exists_R$PAD. This language extends ∀PAD by augmenting it with a restricted form of existential quantification. We will later use this bound to derive upper bounds for the Büchi synthesis problem and the more general problem of LTL synthesis.

The quantifier elimination procedure in this section follows the ideas from [3], where decidability of a more general family of languages was shown, but an upper complexity bound was not given for the language of interest in this paper. For clarity, it is worth going through the main steps of the elimination again here, to use the precise form of the resulting formula to obtain an upper bound for the decision problem for $\forall\exists_R$PAD using the new result from [11] of ∀PAD being in **co-NEXPTIME**. We will also need a different encoding of greatest common divisors than the one detailed in [3], introducing only new universally quantified variables.

We denote the greatest common divisor of a and b as (a, b), and we write $a|b$ to express that a divides b. A result that we will need for the following translation is a generalised version of the Chinese Remainder Theorem:

Theorem 1 (Generalised Chinese Remainder Theorem (CRT), [14]). *For any $m_i \in \mathbb{N}$, $a_i, r_i \in \mathbb{Z}$, where $i \in \{1, 2, \ldots, n\}$,*

$$\exists x \bigwedge_{i=1}^{n} m_i | (a_i x - r_i) \longleftrightarrow \bigwedge_{1 \le i < j \le n} (a_i m_j, a_j m_i) | (a_i r_j - a_j r_i) \wedge \bigwedge_{i=1}^{n} (a_i, m_i) | r_i.$$

The solution for x is unique modulo $\operatorname{lcm}(m'_1, \ldots, m'_n)$*, where* $m'_i = \frac{m_i}{(a_i, m_i)}$.

We define the language $\forall\exists_R$PAD to be the set of formulas of the form

$$\forall z_1 \ldots \forall z_n \exists x_1 \ldots \exists x_m \varphi(\boldsymbol{x}, \boldsymbol{z}) \tag{1}$$

where φ is a quantifier-free PAD formula and all divisibilities are of the form $f(\boldsymbol{z}) | g(\boldsymbol{x}, \boldsymbol{z})$ for some linear terms f and g, i.e., the existentially quantified variables $\boldsymbol{x}$ do not appear on the left side of any divisibility. The aim is to eliminate all the existential quantifiers to obtain a formula in ∀PAD.

W.l.o.g. we can assume that all divisibilities in φ are positive, since every negated divisibility $\neg f | g$ can be rewritten as the equivalent formula

$$\begin{aligned}(f = 0 \wedge \neg(g = 0)) \vee \exists x \exists y \big(&(g = x + y) \wedge (f \mid x) \\ &\wedge ((1 \le y \le f - 1) \vee (1 \le y \le -f - 1)) \big),\end{aligned}$$

which introduces new existentially quantified variables which only occur on the right side of divisibilities.

Every formula of type (1) can then be written in the form

$$\forall \boldsymbol{z} \exists \boldsymbol{x} \bigvee_{i=1}^{N} \left(\bigwedge_{j=1}^{M_i} f_{ij}(\boldsymbol{z}) \mid g_{ij}(\boldsymbol{x}, \boldsymbol{z}) \wedge \bigwedge_{j=1}^{T_i} \varphi_{ij}(\boldsymbol{x}, \boldsymbol{z}) \right) \tag{2}$$

where the f_{ij} and g_{ij} are linear terms and the φ_{ij} are Presburger formulas. W.l.o.g. we assume that each φ_{ij} is of the form $\exists y_{ij}\, (y_{ij} \geq 0 \wedge h_{ij}(\boldsymbol{x}, \boldsymbol{z}) + y_{ij} = 0)$ for some linear term h_{ij}. The first step is to eliminate $\boldsymbol{x}$ from all the Presburger conjuncts. Suppose some term h_{ij} is of the form $ax_m + k_{ij}(\boldsymbol{x}, \boldsymbol{z})$, where $a \neq 0$ and k_{ij} does not depend on x_m. Then we multiply all coefficients in all other equalities and divisibilities in the i-th disjunct by a, replace every occurrence of ax_m with $-k_{ij} - y_{ij}$, and delete the equality $h_{ij}(\boldsymbol{x}, \boldsymbol{z}) + y_{ij} = 0$. This can be repeated until all the remaining equalities depend only on $\boldsymbol{y}$ and $\boldsymbol{z}$, and by rewriting them as inequalities we can ensure that they depend only on $\boldsymbol{z}$. The formula is then of the form

$$\forall \boldsymbol{z} \bigvee_{i=1}^{N} \exists \boldsymbol{x} \exists \boldsymbol{y} \left(\bigwedge_{j=1}^{M_i} f_{ij}(\boldsymbol{z}) | g_{ij}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \wedge \bigwedge_{j=1}^{T_i} h_{ij}(\boldsymbol{z}) \geq 0 \wedge \bigwedge_{y_j \in \boldsymbol{y}} y_j \geq 0 \right). \tag{3}$$

Note that the existentially quantified variables $\boldsymbol{x}$ and $\boldsymbol{y}$ can be written to the right of the disjunction. We can now eliminate these variables by applying Theorem 1 (CRT) to each of them in turn, while regarding all other variables as fixed. The CRT states that if a system of congruences has a solution, then the set of solutions forms an arithmetic progression containing infinitely many (positive and negative) integers, which means that the inequalities $y_j \geq 0$ can be omitted. As a result we get a formula of the form

$$\forall \boldsymbol{z} \bigvee_{i=1}^{N} \left(\bigwedge_{j=1}^{M_i} (\{f_{ijk}(\boldsymbol{z})\}_{k=1}^{P_{ij}}) \mid g_{ij}(\boldsymbol{z}) \wedge \bigwedge_{j=1}^{T_i} h_{ij}(\boldsymbol{z}) \geq 0 \right). \tag{4}$$

Every subformula $(\{f_k(\boldsymbol{z})\}_{k=1}^{P}) | g(\boldsymbol{z})$ involving the gcd of P linear terms can be replaced with the equivalent formula $\forall d \left(\bigwedge_{k=1}^{P} d | f_k(\boldsymbol{z}) \right) \rightarrow d | g(\boldsymbol{z})$. The resulting formula is in the universal fragment of PAD.

Given a $\forall\exists_R$PAD formula φ of the form (1), in general, converting it to (2) will cause an exponential blow-up in the size of the formula. Eliminating $\boldsymbol{x}$ and $\boldsymbol{y}$ from the Presburger conjuncts could lead to one squaring operation on coefficients per variable in the worst case. When applying the CRT to eliminate $\boldsymbol{x}$ and $\boldsymbol{y}$ from the divisibility conjuncts, each elimination of a variable leads to a quadratic blow-up of the size of the formula. So the running time of eliminating m existential quantifiers, as well as the size of the resulting formula, is $O(2^{|\varphi| 2^m})$. The final step of translating the gcd relations to PAD does not cause any additional complexity, as we just replace divisibilities involving P functions on the left side by P divisibilities involving one function each. So the quantifier elimination algorithm takes overall doubly exponential time. Combining this with

the result from [11] that every quantifier-free PAD formula that has a solution has one of size at most exponential in the number of variables and polynomial in the size of the formula (and hence ∀PAD is in **co-NEXPTIME**), we obtain our first new result:

Proposition 1. *The language $\forall\exists_R PAD$ is in **co-2NEXPTIME**.*

A lower bound of **co-NEXPTIME** follows from [6], where it was shown that ∀∃PA, the subset of Presburger arithmetic (without divisibilities) with one quantifier alternation is **co-NEXPTIME**-complete.

3 Decidable Properties of One-Counter Automata

3.1 Weighted Graphs and Flow Networks

A *weighted graph* is a tuple $G = (V, E, w)$, where V is a finite set of vertices, $E \subseteq V \times V$ is a set of directed edges, and $w : E \to \mathbb{Z}$ assigns an integer weight to each edge. For $s, t \in V$, a *path* π from s to t is a sequence of vertices $\pi = v_0 v_1 \dots v_n$ with $v_0 = s$, $v_n = t$ and $(v_i, v_{i+1}) \in E$ for $0 \leq i \leq n-1$. Such a path is a *cycle* if $s = t$. The *weight* of a path π, denoted $weight(\pi)$, is the sum of the weights of the edges that make up π. A path π is said to be *positive* if $weight(\pi) > 0$ and *negative* if $weight(\pi) < 0$.

Given a weighted graph $G = (V, E, w)$ and vertices $s, t \in V$, a *flow* from s to t is a function $f : E \to \mathbb{N}$ satisfying the flow conservation condition

$$\sum_{(v,u)\in E} f(v,u) = \sum_{(u,v)\in E} f(u,v)$$

for all $u \in V \backslash \{s, t\}$. The *value* $|f|$ of a given s-t flow f is defined as

$$|f| = \sum_{(s,u)\in E} f(s,u) - \sum_{(u,s)\in E} f(u,s).$$

The *weight* of a flow f is

$$\sum_{e\in E} f(e)w(e).$$

The *support* of a flow f is $E_f = \{e \in E : f(e) > 0\}$.

Every s-t path π induces an s-t flow f_π: for each edge $e \in E$, $f_\pi(e)$ is the number of times that e is taken in π. A flow is called a *path flow* if it corresponds to a path in such a way. It is easy to see, by a variation of the Euler path theorem, that an s-t flow f is a path flow if and only if $|f| = 1$ and the subgraph induced by the set of edges $E_f \cup \{(t, s)\}$ is strongly connected. If π is the sequential composition of π_1 and π_2, then the flow corresponding to π is the sum of the flows corresponding to π_1 and π_2: $f_\pi(e) = f_{\pi_1}(e) + f_{\pi_2}(e)$ for all $e \in E$.

3.2 One-Counter Automata

A *one-counter automaton* (1-CA) is a finite automaton with an associated counter that can store a single value. Every transition can perform a read or write operation on the counter. We consider 1-CA whose counter ranges over the nonnegative integers, with the counter values and updates encoded in binary, and with each transition being an addition of an integer or integer variable to the counter, or a zero test on the current counter value.

Formally, a *parametric one-counter automaton* is a tuple $\mathcal{C} = (V, E, X, \lambda)$, where V is a finite set of states, $E \subseteq V \times V$ is a set of transitions between states, X is a set of integer parameters, and $\lambda : E \to Op$ labels each transition with an operation on the counter, where $Op = \{zero?\} \cup \{add(a),\ add(ax) : a \in \mathbb{Z},\ x \in X\}$. A transition labelled $zero?$ can only be taken if the current counter value is zero, and it leaves the counter value unchanged. A transition labelled $add(a)$ (or $add(ax)$) adds a (or ax) to the counter and can only be taken if the resulting value is nonnegative. Substituting integer values for the parameters results in a *succinct one-counter automaton*.

A *configuration* of a 1-CA $\mathcal{C}$ is a pair (v, c), for some $v \in V$ and $c \in \mathbb{N}$. After substituting integer values for all the parameters, the transition relation E between states induces an unlabelled transition relation between configurations: every transition $v \to v'$ with label $zero?$ corresponds to the transition $(v, 0) \to (v', 0)$, and every transition $v \to v'$ with label $add(a)$ corresponds to the set of transitions $\{(v, c) \to (v', c') \mid c, c' \geq 0,\ c' - c = a\}$.

A *computation* π of $\mathcal{C}$ is a (finite or infinite) sequence of transitions between configurations $\pi = (v_0, c_0) \to (v_1, c_1) \to (v_2, c_2) \to \cdots$.

The reachability problem for parametric 1-CA asks, given a 1-CA $\mathcal{C}$ and two configurations (v, c) and (v', c'), if there are values for the parameters such that there is a computation from (v, c) to (v', c') in $\mathcal{C}$. The *Büchi synthesis* problem for parametric 1-CA asks, given a 1-CA $\mathcal{C}$ with a set of accepting states $V_{Acc} \subseteq V$ and an initial configuration (v, c), if there are parameter values for which on all infinite computations starting from (v, c), some state in V_{Acc} is visited infinitely often. Finally, the *LTL synthesis* problem for parametric 1-CA is to decide, for a given 1-CA $\mathcal{C}$ with initial configuration (v, c) and a set $V_p \subseteq V$ for each atomic proposition $p \in P$, and for an LTL formula φ over the set of propositions P, whether there exist parameter values for which φ holds on all infinite computations from (v, c).

Reductions in both directions between reachability for parametric 1-CA and the decision problem for ∃PAD have been shown in [7]. The outline of how to translate an instance of reachability to a ∃PAD formula omits some details and does not address the complexity of the procedure, so we will go through a more precise explanation including a detailed complexity assessment in this paper, making use of a symbolic encoding of the Bellman-Ford algorithm to improve the resulting complexity. For reasons of space, we will leave this detailed analysis for the appendix and only give the general idea in the following section.

3.3 Reachability for Parametric 1-CA

Our analysis relies on a symbolic representation of computations in terms of *reachability certificates*, developed in [7]. We first recall the definition and then give a symbolic encoding of these certificates in PAD.

Any parametric 1-CA $\mathcal{C}$ without zero tests is syntactically equivalent to a parametric weighted graph $G_{\mathcal{C}}$, where a transition labelled $add(a)$ (or $add(ax)$) is seen as having weight a (or ax).

Given a path $\pi = v_0 v_1 \dots v_n$ in $G_{\mathcal{C}}$, the vertex v_k is the *minimum* of π if $\pi' = v_0 v_1 \dots v_k$ has minimal weight among all the prefixes of π. In this case we define $drop(\pi) = weight(\pi')$. Then π corresponds to a valid computation from (v_0, c) to (v_n, c') if and only if $weight(\pi) = c' - c$ and $drop(\pi) \geq -c$.

For a path π from v to v' in $G_{\mathcal{C}}$, if there is a computation over π from (v, c) to (v', c') in $\mathcal{C}$, we say that π can be *taken from* (v, c) and *taken to* (v', c') in $\mathcal{C}$. Note that when compared to paths in a weighted graph, computations in a 1-CA are more restricted in the sense that transitions can only be taken if the counter value remains nonnegative.

Let f be a flow in the weighted graph $G_{\mathcal{C}}$. A cycle in f is a cycle l in $G_{\mathcal{C}}$ such that every edge of l lies in the support of f. A flow f is a *reachability certificate* for two configurations (v, c) and (v', c') if there is a computation from (v, c) to (v', c') over a path π such that $f = f_\pi$ and one of the following conditions holds: (i) f has no positive cycles, (ii) f has no negative cycles, (iii) there exists a positive cycle l that can be taken from (v, c) and a negative cycle l' that can be taken to (v', c'). We will call the three cases type-1, type-2 and type-3 reachability certificates, respectively.

Proposition 2 ([7]). *If (v', c') is reachable from (v, c) in $\mathcal{C}$, then there is a computation $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ that starts in (v, c) and ends in (v', c'), where π_1, π_2 and π_3 each yield a polynomial size (in $|\mathcal{C}|$ and the bit lengths of c and c') reachability certificate, of type 1, 3 and 2, respectively.*

The first step in the translation of reachability to the decision problem for ∃PAD is to encode the existence of each type of reachability certificate as a ∃PAD formula. We will call these formulas $Type1$, $Type2$ and $Type3$, respectively.

The main idea for all three translations is the same: for a reachability certificate f from (v, c) to (v', c') we need to encode the fact that f is a path flow from v to v', which we can do using linear constraints corresponding to flow conservation properties and setting $|f|$ to 1. It makes sense to encode f as a vector of variables $\boldsymbol{y}$, with one variable for each transition in the automaton. We further need to express that $weight(f) = c' - c$ and that $drop(\pi_f) \geq -c$, where π_f is the path that corresponds to f. The main difficulty here is that a formula representing $weight(f)$ will generally contain products of variables. One way to deal with this problem is to make a change of variables and introduce divisibilities to express these products. Finally, we need to express the existence or non-existence of positive or negative cycles. Existence can be handled simply by using a nondeterministic algorithm. For non-existence, we can use a symbolic

encoding of the Bellman-Ford algorithm, which checks for negative cycles in a weighted graph.

Once we have formulas for reachability certificates, we can use these to define formulas $\mathit{Reach\_notest}(v, c, v', c', \boldsymbol{x})$, expressing that (v', c') is reachable from (v, c) without taking any zero tests, $\mathit{Reach\_test}(v, c, v', c', \boldsymbol{x})$, expressing reachability via at least one zero test, and $\mathit{Reach}(v, c, v', c', \boldsymbol{x})$, which is general reachability in 1-CA, where the variables $\boldsymbol{x}$ encode the parameters of the automaton. The details of all the constructions can be found in the appendix. The main result is the following:

Proposition 3. *Reachability without zero tests, reachability via zero tests, and general reachability in 1-CA can all be encoded in nondeterministic polynomial time as formulas of ∃PAD with free variables $\boldsymbol{x}$ corresponding to the parameters of the 1-CA, where $\boldsymbol{x}$ are the only variables that occur on the left side of any divisibilities.*

3.4 Translating Büchi and LTL Synthesis to $\forall\exists_R$PAD

The negation of the Büchi synthesis problem asks whether for all possible parameter values, there is some infinite computation starting in (v, c) that visits every accepting state only finitely often. This can be expressed in terms of the reachability formulas from the previous section.

To be precise, we need a formula *Reach* which, as above, expresses general reachability in the whole given 1-CA $\mathcal{C}$, as well as formulas $\mathit{Reach\_notest}'$ and $\mathit{Reach\_test}'$, which express reachability without zero tests and via zero tests, respectively, in the sub-CA of $\mathcal{C}$ with state set $V \backslash V_{Acc}$. Using these formulas, we can express that it is possible to reach some configuration (u, k) from (v, c), where u is not accepting, and from where a cycle traversing only non-accepting states can be taken back to u infinitely times. We need to distinguish between two cases: either the cycle contains a zero test, or it does not. In the first case it needs to return to the exact configuration of (u, k), to guarantee that the zero test(s) could be taken again infinitely many times. In the second case it suffices to assert that the cycle is not negative so it can be taken infinitely many times without the counter ever becoming negative. Quantifying over all parameter values, we get the following formula for the negation of Büchi synthesis:

$$\begin{aligned}\forall\boldsymbol{x}\exists k\exists k' \bigvee_{u\in V\backslash V_{Acc}} & (k \le k') \wedge \mathit{Reach}(v, c, u, k, \boldsymbol{x}) \\ & \wedge(\mathit{Reach\_test}'(u, k, u, k, \boldsymbol{x}) \vee \mathit{Reach\_notest}'(u, k, u, k', \boldsymbol{x}))\end{aligned} \tag{5}$$

Since the reachability formulas have no universally quantified variables, and the universally quantified variables $\boldsymbol{x}$ are the only ones that occur on the left side of any divisibilities, it follows that (5) is in $\forall\exists_R$PAD.

It is well known that every LTL formula can be translated in polynomial space to an equivalent (finite-state) Büchi automaton of size exponential in the size of the formula, see e.g. [18]. Similarly to standard LTL model checking algorithms

for Kripke structures, we can solve the LTL synthesis problem for a given LTL formula φ and 1-CA $\mathcal{C}$ by constructing a Büchi automaton $\mathcal{C}_\varphi$ equivalent to φ, and a 1-CA with a set of accepting states obtained by constructing the product of $\mathcal{C}$ and $\mathcal{C}_\varphi$, and then solving the Büchi synthesis problem for this product automaton.

3.5 Complexity

All flow supports and zero tests in Section A.1 can be guessed in polynomial time in the size of $\mathcal{C}$ and the bit length of c and c', so the full construction of *Reach*, *Reach_test* and *Reach_notest* is in nondeterministic polynomial time. Note that the reachability problem itself asks if there exist parameter values for which (v', c') is reachable from (v, c), which we can write as the ∃PAD formula $\exists \boldsymbol{x} \mathit{Reach}(v, c, v', c', \boldsymbol{x})$. Since the decision problem for ∃PAD was shown to be in **NEXPTIME** in [11], our reduction shows that reachability for parametric 1-CA is in **NEXPTIME**. In fact, [7] also provides a proof of **NP**-hardness of reachability by reducing ∃PAD to it in nondeterministic polynomial time, so we can conclude that reachability in parametric 1-CA and the decision problem for ∃PAD are equivalent in terms of complexity, meaning that any improved complexity bounds for one problem would give us the same bounds for the other.

The negation of the Büchi synthesis problem can be expressed in terms of *Reach*, *Reach_test* and *Reach_notest*, which results in a formula in $\forall\exists_R$PAD. Since the reachability formulas can be deterministically constructed in exponential time, it follows that the Büchi synthesis problem is in **3NEXPTIME**. For the LTL synthesis problem, the Büchi automaton $\mathcal{C}_\varphi$ which is equivalent to a given LTL formula φ has exponential size in the size of φ in the worst case, but the product of $\mathcal{C}_\varphi$ and a 1-CA $\mathcal{C}$ still has the same number of parameters as $\mathcal{C}$, which is the same as the number of variables in the resulting $\forall\exists_R$PAD formula. Since in the **NEXPTIME** upper bound for the decision problem for ∃PAD, the exponential depends only on the number of variables, we get a **3NEXPTIME** upper bound for LTL synthesis as a main result of this paper.

Theorem 2. *The LTL synthesis problem for 1-CA is decidable in **3NEXPTIME**.*

4 A Lower Bound for Büchi Synthesis

In this section we will establish a $\mathbf{NP^{NP}}$ lower complexity bound for the Büchi synthesis problem by a reduction from a generalisation of subset sum, which takes as input two sets of integers $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$, as well as an integer goal g. The problem is to decide whether the following statement holds:

$$\exists S \subseteq A \forall T \subseteq B \left(\sum_{s \in S} s + \sum_{t \in T} t \neq g \right). \tag{6}$$

This problem is known to be $\mathbf{NP^{NP}}$-complete [1].

We need to construct a 1-CA $\mathcal{C} = (V, E, X, \lambda)$ with a set of parameters $X = \{x_1, x_2, \ldots, x_m\}$ and an accepting subset of states $V_{acc} \subseteq V$ such that (6) holds if and only if there is a way of substituting integer values for the parameters that forces all computations from the initial configuration to always stay in V_{acc}. The parameters X encode the choice of elements in the set S, while the elements in T will be represented by paths that can be taken in the automaton.

If for any $1 \leq i \leq m$, x_i is not either 0 or a_i, there is a path from the initial configuration $(v_0, 0)$ to a "bad cycle", i.e., a cycle that only contains non-accepting states. So for all computations to be guaranteed to never enter such a cycle, the parameters need to be assigned values such that each x_i is either 0 or a_i. Intuitively $x_i = a_i$ means that the element a_i is included in the set S.

W.l.o.g. we can assume that the elements of the sets A and B are sorted in decreasing order, and that $g \geq 0$. Let $\{a_1, a_2, \ldots, a_k\}$ be the positive elements of A, and let $\{b_1, b_2, \ldots, b_l\}$ be the positive elements of B. For simplicity, in graphs we will write $+x$ for $add(x)$, $-x$ for $add(-x)$, etc.

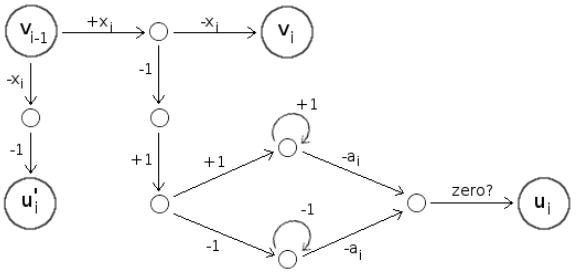


Fig. 1. The sub-automaton $\mathcal{C}_i$ for positive a_i. One of u_i and u_i' is reachable from $(v_{i-1}, 0)$ iff $x_i \neq 0$ and $x_i \neq a_i$.

Figure 1 shows a sub-automaton that allows a computation from $(v_{i-1}, 0)$ to one of the states u_i and u_i' if and only if $x_i \neq 0$ and $x_i \neq a_i$, assuming a_i is positive. If $x_i \leq 0$, it can take the first downwards transition, and only if $x_i < 0$ it can take the second transition to u_i'. If $x_i \geq 0$ then the transition to the right can be taken, storing the value of x_i as the new counter value. The following two downwards transitions can only be taken if $x_i > 0$. From here, some positive integer is either added or subtracted to reach the value a_i. Clearly, if $x_i = a_i$ then the zero test to u_i can never be taken. We will call this sub-automaton $\mathcal{C}_i$. If a_i is negative, then we can simply reverse the sign of the labels of the transitions involving x_i and a_i to obtain an automaton with the same effect.

The next step is to put these m automata in series, as shown in Figure 2. On the path $v_0 \ldots v_m$, as the initial configuration was $(v_0, 0)$, the counter value will be 0 in each state v_i for $0 \leq i \leq m$.

From $(v_m, 0)$, there is a series of k transitions that load the sum of the x_i that correspond to positive elements of A onto the counter. As can be seen

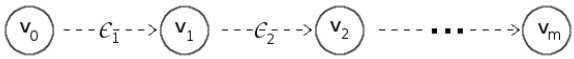


Fig. 2. The series of automata $\mathcal{C}_1, \ldots, \mathcal{C}_m$.

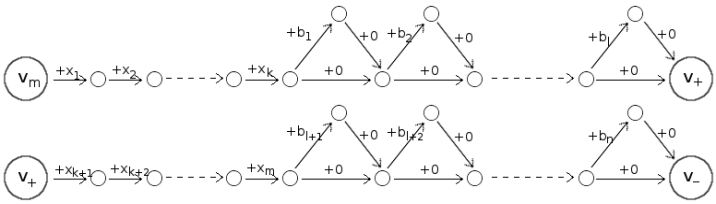


Fig. 3. Adding positive and negative values to the counter.

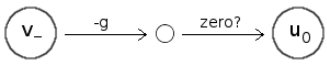


Fig. 4. The final "decision part" of the automaton.

in Figure 3, this is followed by l triangle shaped sub-automata that allow the options of either adding b_i or 0, for $1 \leq i \leq l$. This pattern is repeated for the negative values in A and B. The state v_- has a series of two transitions to u_0. Finally, we complete the 1-CA by adding a loop labelled $add(0)$ on every state. For the two states in each $\mathcal{C}_i$ that already have a loop, we add an additional $add(0)$ transition from them to a new state with a $add(0)$ loop.

We now set $V_{acc} = V \backslash \{u_0, u_1, \ldots, u_m, u'_1, \ldots, u'_m\}$. If any of the x_i are assigned a value different from 0 and a_i, one of the bad states u_i and u'_i is reachable from $(v_0, 0)$. If every x_i is either 0 or a_i, then the only bad state that could still be reached is u_0. This will happen if for some subset $T \subseteq B$, the sum of all the x_i and all the elements of T equals g.

5 Conclusion

The decidability of the LTL synthesis problem for one-counter automata was stated as an open problem in [5]. The same paper includes a proof that the corresponding synthesis problem for CTL is undecidable. The proof is by reduction from Hilbert's Tenth Problem and relies on the branching structure of CTL. The fact that this problem turned out to be decidable for LTL but undecidable for CTL confirms the general observation from [17] that synthesis problems tend to be harder for branching time than for linear time logic.

A lower bound of PSPACE for the LTL synthesis problem immediately follows from the fact that LTL model checking for Kripke structures is PSPACE-complete. An interesting open problem would be to improve this lower bound.

Following the reductions in this paper, the upper bound of both the LTL and the more restricted Büchi synthesis problem crucially depends on the upper bound for the decision problem of ∃PAD, whose precise complexity has been an open problem for decades and which was recently shown to be in **NEXPTIME** [11].

References

1. Berman, P., Karpinski, M., Larmore, L.L., Plandowski, W., Rytter, W.: On the complexity of pattern matching for highly compressed two-dimensional texts. In: Apostolico, A., Hein, J. (eds.) CPM 1997. LNCS, vol. 1264, pp. 40–51. Springer, Heidelberg (1997)
2. Bouajjani, A., Bozga, M., Habermehl, P., Iosif, R., Moro, P., Vojnar, T.: Programs with lists are counter automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 517–531. Springer, Heidelberg (2006)
3. Bozga, M., Iosif, R.: On decidability within the arithmetic of addition and divisibility. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 425–439. Springer, Heidelberg (2005)
4. Chitic, C., Rosu, D.: On validation of XML streams using finite state machines. In: WEBDB 2004, pp. 85–90. ACM Press (2004)
5. Göller, S., Haase, C., Ouaknine, J., Worrell, J.: Model checking succinct and parametric one-counter automata. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 575–586. Springer, Heidelberg (2010)
6. Haase, C.: Subclasses of Presburger arithmetic and the weak EXP hierarchy. In: Proceedings of CSL-LICS, pp. 47–56. ACM (2014)
7. Haase, C., Kreutzer, S., Ouaknine, J., Worrell, J.: Reachability in succinct and parametric one-counter automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 369–383. Springer, Heidelberg (2009)
8. Ibarra, O.H., Dang, Z.: On two-way finite automata with monotonic counters and quadratic Diophantine equations. Theor. Comput. Sci. **312**(2–3), 359–378 (2004)
9. Ibarra, O.H., Jiang, T., Trân, N., Wang, H.: New decidability results concerning two-way counter machines and applications. In: Lingas, A., Karlsson, R., Carlsson, S. (eds) ICALP 1993. LNCS, vol. 700. Springer, Heidelberg (1993)
10. Ibarra, O.H., Su, J., Dang, Z., Bultan, T., Kemmerer, R.A.: Counter machines and verification problems. Theor. Comput. Sci. **289**(1), 165–189 (2002)
11. Lechner, A., Ouaknine, J., Worrell, J.: On the complexity of linear arithmetic with divisibility. In: Proceedings of LICS (2015)
12. Lipshitz, L.: The Diophantine problem for addition and divisibility. Transactions of the American Mathematical Society **235**, 271–283 (1976)
13. Lipshitz, L.: Some remarks on the Diophantine problem for addition and divisibility. Bull. Soc. Math. Belg. Sér. B **33**(1), 41–52 (1981)
14. Mahler, K.: On the Chinese remainder theorem. Math. Nach. **18**, 120–122 (1958)
15. Minsky, M.: Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. Annals of Mathematics **74**, 437–455 (1961)
16. Robinson, J.: Definability and decision problems in arithmetic. Journal of Symbolic Logic **14**(2), 98–114 (1949)
17. Vardi, M.Y.: Branching vs. Linear time: final showdown. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 1–22. Springer, Heidelberg (2001)
18. Wolper, P.: Constructing automata from temporal logic formulas: a tutorial. In: Brinksma, E., Hermanns, H., Katoen, J.-P. (eds.) EEF School 2000 and FMPA 2000. LNCS, vol. 2090, pp. 261–277. Springer, Heidelberg (2001)

On Boundedness Problems for Pushdown Vector Addition Systems

Jérôme Leroux[1], Grégoire Sutre[1(✉)], and Patrick Totzke[2]

[1] University of Bordeaux and CNRS, LaBRI, UMR 5800, Talence, France
gregoire.sutre@labri.fr
[2] Department of Computer Science, University of Warwick, Coventry, UK

Abstract. We study pushdown vector addition systems, which are synchronized products of pushdown automata with vector addition systems. The question of the boundedness of the reachability set for this model can be refined into two decision problems that ask if infinitely many counter values or stack configurations are reachable, respectively. Counter boundedness seems to be the more intricate problem. We show decidability in exponential time for one-dimensional systems. The proof is via a small witness property derived from an analysis of derivation trees of grammar-controlled vector addition systems.

1 Introduction

Pushdown vector addition systems are finite automata that can independently manipulate a pushdown stack and several counters. They are defined as synchronized products of vector addition systems with pushdown automata. Vector addition systems, shortly *VAS*, are a classical model for concurrent systems and are computationally equivalent to Petri nets. Formally, a k-dimensional *vector addition system* is a finite set $\boldsymbol{A} \subseteq \mathbb{Z}^k$ of vectors called *actions*. Each action $a \in \boldsymbol{A}$ induces a binary relation $\xrightarrow{\boldsymbol{a}}$ over $\mathbb{N}^k$, defined by $\boldsymbol{c} \xrightarrow{\boldsymbol{a}} \boldsymbol{d}$ if $\boldsymbol{d} = \boldsymbol{c} + \boldsymbol{a}$.

A k-dimensional *pushdown vector addition system*, shortly *PVAS*, is a tuple $(Q, \Gamma, q_{init}, \boldsymbol{c}_{init}, w_{init}, \Delta)$ where Q is a finite set of *states*, Γ is a finite *stack alphabet*, $q_{init} \in Q$ is an *initial state*, $\boldsymbol{c}_{init} \in \mathbb{N}^k$ is an *initial assignment of the counters*, $w_{init} \in \Gamma^*$ is an *initial stack content*, and $\Delta \subseteq Q \times \mathbb{Z}^k \times Op(\Gamma) \times Q$ is a finite set of *transitions* where $Op(\Gamma) \stackrel{\text{def}}{=} \{\mathsf{push}(\gamma), \mathsf{pop}(\gamma), \mathsf{nop} \mid \gamma \in \Gamma\}$ is the set of stack operations. The *size* of VAS, PVAS (and GVAS introduced later) are defined as expected with numbers encoded in binary.

Example 1.1. Consider the program on the left of Figure 1, that doubles the value of the global variable x. The $\star$ expression non-deterministically evaluates to a Boolean, as it is often the case in abstraction of programs [1]. On the right is a 1-dimensional PVAS that models this procedure: states correspond to lines in the program code, operations on the variable x are directly applied, and the call stack is reflected on the pushdown stack. □

This work was partially supported by ANR project ReachHard (ANR-11-BS02-001).

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 101–113, 2015.
DOI: 10.1007/978-3-319-24537-9_10

```
1: x ← n
2: procedure DoubleX
3:     if (⋆ ∧ x > 0) then
4:         x ← (x − 1)
5:         DoubleX
6:     end if
7:     x ← (x + 2)
8: end procedure
```

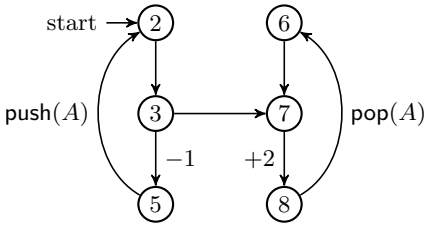


Fig. 1. A PVAS modeling a recursive program.

The semantics of PVAS is defined as follows. A *configuration* is a triple $(q, \boldsymbol{c}, w) \in Q \times \mathbb{N}^k \times \Gamma^*$ consisting of a state, a vector of natural numbers, and a stack content. The binary *step* relation $\rightarrow$ over configurations is defined by $(p, \boldsymbol{c}, u) \rightarrow (q, \boldsymbol{d}, v)$ if there is a transition $(p, op, \boldsymbol{a}, q) \in \Delta$ such that $\boldsymbol{c} \xrightarrow{\boldsymbol{a}} \boldsymbol{d}$ and one of the following conditions holds: either $op = \mathsf{push}(\gamma)$ and $v = u\gamma$, or $op = \mathsf{pop}(\gamma)$ and $u = v\gamma$, or $op = \mathsf{nop}$ and $u = v$. The reflexive and transitive closure of $\rightarrow$ is denoted by $\xrightarrow{*}$.

The *reachability set* of a PVAS is the set of configurations $(q, \boldsymbol{c}, w)$ such that $(q_{init}, \boldsymbol{c}_{init}, w_{init}) \xrightarrow{*} (q, \boldsymbol{c}, w)$. The reachability problem asks if a given configuration $(q, \boldsymbol{c}, w)$ is in the reachability set of a given PVAS. The decidability of this problem is open. Notice that for vector addition systems, even though the reachability problem is decidable [6,12], no primitive upper bound of complexity is known (see [9] for a first upper bound). However, a variant called the coverability problem is known to be ExpSpace-complete [11,13]. Adapted to PVAS, the coverability problem takes as input a PVAS and a state $q \in Q$ and asks if there exists a reachable configuration of the form $(q, \boldsymbol{c}, w)$ for some $\boldsymbol{c}$ and w. The decidability of the coverability problem for PVAS is also open. In fact, coverability and reachability are inter-reducible (in logspace) for this class [7,10]. In dimension one, we recently proved that coverability is decidable [10].

Both coverability and reachability are clearly decidable for PVAS with finite reachability sets. These PVAS are said to be *bounded*. In [8], this class is proved to be recursive, i.e. the boundedness problem for PVAS is decidable. The complexity of this problem is known to be Tower-hard [7]. The decidability is obtained by observing that if the reachability set of a PVAS is finite, its cardinality is at most hyper-Ackermannian in the size of the PVAS. Even though this bound is tight [8], the exact complexity of the boundedness problem is still open. Indeed, it is possible that there exist small certificates that witness infinite reachability sets. For instance, in the VAS case, the reachability set can be finite and Ackermannian. But when it is infinite, there exist small witnesses of this fact [13]. This yields an optimal [11] exponential-space algorithm for the VAS boundedness problem. Extending this technique to PVAS is a challenging problem.

The boundedness problem for PVAS can be refined in two different ways. In fact, the infiniteness of the reachability set may come from the stack or the

counters. We say that a PVAS is *counter-bounded* if the set of vectors $\boldsymbol{c} \in \mathbb{N}^k$ such that $(q, \boldsymbol{c}, w)$ is reachable for some q and w, is finite. Symmetrically, a PVAS is called *stack-bounded* if the set of words $w \in \Gamma^*$ such that $(q, \boldsymbol{c}, w)$ is reachable for some q and $\boldsymbol{c}$, is finite. The following lemma shows that the two associated decision problems are at least as hard as the boundedness problem.

Lemma 1.2. *The boundedness problem is reducible in logarithmic space to the counter-boundedness problem and to the stack-boundedness problem (the dimension k is unchanged by the reduction).*

The stack-boundedness problem can be solved by adapting the algorithm introduced in [8] for the PVAS boundedness problem. Informally, this algorithm explores the reachability tree and stops as soon as it detects a cycle of transitions whose iteration produces infinitely many reachable configurations. If this cycle increases the stack, we can immediately conclude stack-unboundedness. Otherwise, at least one counter can be increased to an arbitrary large number. By replacing the value of this counter by ω and then resuming the computation of the tree from the new (extended) configuration, we obtain a Karp&Miller-like algorithm [5] deciding the stack-boundedness problem. We deduce the following result.

Lemma 1.3. *The stack-boundedness problem for PVAS is decidable.*

Concerning the counter-boundedness problem, adapting the algorithm introduced in [8] in a similar way seems to be more involved. Indeed, if we detect a cycle that only increases the stack, we can iterate it and represent its effect with a regular language. However, we do not know how to effectively truncate the resulting tree to obtain an algorithm deciding the counter-boundedness problem.

Contributions. In this paper we solve the counter-boundedness problem for the special case of dimension one. We show that in a grammar setting, PVAS counter-boundedness corresponds to the boundedness problem for prefix-closed, grammar-controlled vector addition systems. We show that in dimension one, this problem is decidable in exponential time. Our proof is based on the existence of small witnesses exhibiting the unboundedness property. This complexity result improves the best known upper bound for the classical boundedness problem for PVAS in dimension one. In fact, as shown by the following Example 1.4, the reachability set of a bounded 1-dimensional PVAS can be Ackermannian large. In particular, the worst-case running time of the algorithm introduced in [8] for solving the boundedness problem is at least Ackermannian even in dimension one.

Example 1.4. The Ackermann functions $A_m : \mathbb{N} \to \mathbb{N}$, for $m \in \mathbb{N}$, are defined by induction for every $n \in \mathbb{N}$ by:

$$A_m(n) \stackrel{\text{def}}{=} \begin{cases} n+1 & \text{if } m = 0 \\ A_{m-1}^{n+1}(1) & \text{if } m > 0 \end{cases}$$

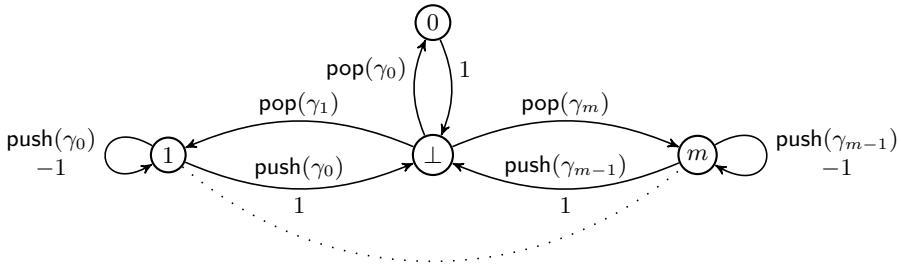


Fig. 2. One-dimensional PVAS that weakly compute Ackermann functions.

These functions are *weakly computable* by the (family of) PVAS depicted in Figure 2, in the sense that:

$$A_m(n) \;=\; \max\{c \mid (\bot, n, \gamma_m) \xrightarrow{*} (\bot, c, \varepsilon)\} \tag{1}$$

for every $m, n \in \mathbb{N}$. Indeed, an immediate induction on $k \in \{0, \ldots, m\}$ shows that $(\bot, c, \gamma_k) \xrightarrow{*} (\bot, A_k(c), \varepsilon)$ for every $c \in \mathbb{N}$. For the converse inequality, let us introduce, for each configuration $(\bot, c, w)$, the number $\theta(c, w)$ defined by

$$\theta(c, \gamma_{i_1} \cdots \gamma_{i_k}) \;\stackrel{\text{def}}{=}\; A_{i_1} \circ \cdots \circ A_{i_k}(c)$$

An immediate induction on the number of times a run come back to the state $\bot$ shows that $(\bot, c, w) \xrightarrow{*} (\bot, c', w')$ implies $\theta(c, w) \geq \theta(c', w')$. Since $\theta(c, \varepsilon) = c$, we derive that $A_m(n) \geq c$ for every c such that $(\bot, n, \gamma_m) \xrightarrow{*} (\bot, c, \varepsilon)$. This concludes the proof of Equation 1.

Notice that the reachability set of this PVAS is finite for any initial configuration. Indeed, $(\bot, c, w) \xrightarrow{*} (\bot, c', w')$ implies $\theta(c, w) \geq \theta(c', w') \geq c' + |w'|$. Therefore, there are only finitely many reachable configurations in state $\bot$. It follows that the same property holds for the other states. □

Outline. We recall some necessary notations about context-free grammars and parse trees in the next section. In Section 3, we present the model of grammar-controlled vector addition systems (GVAS) as previously introduced in [10], and reduce the counter boundedness problem for PVAS to the boundedness problem for the subclass of *prefix-closed* GVAS. We show in Section 4 that unbounded systems exhibit certificates of a certain form. Section 5 proves a technical lemma used later on and finally, in section 6, we bound the size of minimal certificates and derive the claimed exponential-time upper bound.

2 Preliminaries

We let $\overline{\mathbb{Z}} \stackrel{\text{def}}{=} \mathbb{Z} \cup \{-\infty, +\infty\}$ denote the extended integers, and we use the standard extensions of $+$ and $\leq$ to $\overline{\mathbb{Z}}$. Recall that $(\overline{\mathbb{Z}}, \leq)$ is a complete lattice.

Words. Let A^* be the set of all finite *words* over the alphabet A. The *empty word* is denoted by ε. We write $|w|$ for the *length* of a word w in A^* and $w^k \stackrel{\text{def}}{=} ww\cdots w$ for its k-fold concatenation. The *prefix* partial order $\preceq$ over words is defined by $u \preceq v$ if $v = uw$ for some word w. We write $u \prec v$ if u is a proper prefix of v. A *language* is a subset $L \subseteq A^*$. A language L is said to be *prefix-closed* if $u \preceq v$ and $v \in L$ implies $u \in L$.

Trees. A *tree* T is a finite, non-empty, prefix-closed subset of $\mathbb{N}^*$ satisfying the property that if tj is in T then ti in T for all $i < j$. Elements of T are called *nodes*. Its *root* is the empty word ε. An *ancestor* of a node t is a prefix $s \preceq t$. A *child* of a node t in T is a node tj in T with j in $\mathbb{N}$. A node is called a *leaf* if it has no child (i.e., $t0 \notin T$), and is said to be *internal* otherwise. The *size* of a tree T is its cardinal $|T|$, its *height* is the maximal length $|t|$ of its nodes $t \in T$. We let $\preceq_{\text{lex}}$ denote the lexicographic order on words in $\mathbb{N}^*$.

Context-free Grammars. A *context-free grammar* is a quadruple $G = (V, A, R, S)$, where V and A are disjoint finite sets of *nonterminal* and *terminal* symbols, $S \in V$ is a *start symbol*, and $R \subseteq V \times (V \cup A)^*$ is a finite set of *production rules*. We write

$$X \vdash \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_k$$

to denote that $(X, \alpha_1), \ldots, (X, \alpha_k) \in R$. For all words $w, w' \in (V \cup A)^*$, the grammar admits a *derivation step* $w \Longrightarrow w'$ if there exist two words u, v in $(V \cup A)^*$ and a production rule (X, α) in R such that $w = uXv$ and $w' = u\alpha v$. Let $\stackrel{*}{\Longrightarrow}$ denote the reflexive and transitive closure of $\Longrightarrow$. The *language* of a word w in $(V \cup A)^*$ is the set $L^G_w \stackrel{\text{def}}{=} \{z \in A^* \mid w \stackrel{*}{\Longrightarrow} z\}$. The *language* of G is defined as L^G_S, and it is denoted by L^G. A nonterminal $X \in V$ is called *productive* if $L^G_X \neq \emptyset$. A context-free grammar $G = (V, A, R, S)$ is in *Chomsky normal form*[1] if, for every production rule (X, α) in R, either $(X, \alpha) = (S, \varepsilon)$ or $\alpha \in V^2 \cup A$.

Parse Trees. A *parse tree* for a context-free grammar $G = (V, A, R, S)$ is a tree T equipped with a labeling function $sym : T \to (V \cup A \cup \{\varepsilon\})$ such that the root is labeled by $sym(\varepsilon) = S$ and R contains the production rule $sym(t) \vdash sym(t0) \cdots sym(tk)$ for every internal node t with children $t0, \ldots, tk$. In addition, each leaf $t \neq \varepsilon$ with $sym(t) = \varepsilon$ is the only child of its parent. Notice that $sym(t) \in V$ for every internal node t. A parse tree is called *complete* when $sym(t) \in (A \cup \{\varepsilon\})$ for every leaf t. The *yield* of a parse tree (T, sym) is the word $sym(t_1) \cdots sym(t_\ell)$ where $t_1, \ldots, t_\ell$ are the leaves of T in lexicographic order (informally, from left to right). Observe that for every word w in $(V \cup A)^*$, it holds that $S \stackrel{*}{\Longrightarrow} w$ if, and only if, w is the yield of some parse tree.

[1] To simplify the presentation, we consider a weaker normal form than the classical one, as we allow to reuse the start symbol.

3 Grammar-Controlled Vector Addition Systems

In this section we recall the notion of GVAS from [10] and show that the boundedness problem for the subclass of *prefix-closed* GVAS is inter-reducible to the counter-boundedness problem for pushdown vector addition systems.

Definition 3.1 (GVAS). *A k-dimensional* grammar-controlled vector addition system *(shortly,* GVAS*) is a tuple $G = (V, \boldsymbol{A}, R, S, \boldsymbol{c}_{init})$ where $(V, \boldsymbol{A}, R, S)$ is a context-free grammar, $\boldsymbol{A} \subseteq \mathbb{Z}^k$ is a VAS, and $\boldsymbol{c}_{init} \in \mathbb{N}^k$ is an initial vector.*

The semantics of GVAS is given by extending the relations $\xrightarrow{\boldsymbol{a}}$ of ordinary VAS to words over $V \cup \boldsymbol{A}$ as follows. Define $\xrightarrow{\varepsilon}$ to be the identity on $\mathbb{N}^k$ and let $\xrightarrow{z\boldsymbol{a}} \stackrel{\text{def}}{=} \xrightarrow{\boldsymbol{a}} \circ \xrightarrow{z}$ for $z \in \boldsymbol{A}^*$ and $\boldsymbol{a} \in \boldsymbol{A}$. Finally, let $\xrightarrow{w} \stackrel{\text{def}}{=} \bigcup_{z \in L_w^G} \xrightarrow{z}$ for $w \in (V \cup \boldsymbol{A})^*$. For a word $z = \boldsymbol{a}_1 \boldsymbol{a}_2 \cdots \boldsymbol{a}_n \in \boldsymbol{A}^*$ over the terminals, we shortly write $\sum z$ for the sum $\sum_{i=1}^n a_i$. Observe that $\boldsymbol{c} \xrightarrow{z} \boldsymbol{d}$ implies $\boldsymbol{d} - \boldsymbol{c} = \sum z$.

Ultimately, we are interested in the relation $\xrightarrow{S}$, that describes the reachability relation via sequences of actions in L_S^G, i.e., those that are derivable from the starting symbol S in the underlying grammar. A vector $\boldsymbol{d} \in \mathbb{N}^k$ is called *reachable* from a vector $\boldsymbol{c} \in \mathbb{N}^k$ if $\boldsymbol{c} \xrightarrow{S} \boldsymbol{d}$. The *reachability set* of a GVAS is the set of vectors reachable from $\boldsymbol{c}_{init}$.

A GVAS is said to be *bounded* if its reachability set is finite. The associated boundedness problem for GVAS is challenging since the coverability problem for PVAS, whose decidability is still open, is logspace reducible to it. However, the various boundedness properties that we investigate on PVAS (see Section 1) consider all reachable configurations, without any acceptance condition. So they intrinsically correspond to context-free languages that are prefix-closed. It is therefore natural to consider the same restriction for GVAS. Formally, we call a GVAS $G = (V, \boldsymbol{A}, R, S, \boldsymbol{c}_{init})$ *prefix-closed* when the language L_S^G is prefix-closed. Concerning the counter-boundedness problem for PVAS, the following lemma shows that it is sufficient to consider the special case of prefix-closed GVAS.

Lemma 3.2. *The counter-boundedness problem for PVAS is logspace inter-reducible with the prefix-closed GVAS boundedness problem (the dimension k is unchanged by both reductions).*

In this paper, we focus on the counter-boundedness problem for PVAS of dimension one. We show that this problem is decidable in exponential time. The proof is by reduction, using Lemma 3.2, to the boundedness problem for prefix-closed 1-dimensional GVAS. Our main technical contribution is the following result.

Theorem 3.3. *The prefix-closed 1-dimensional GVAS boundedness problem is decidable in exponential time.*

For the remainder of the paper, we restrict our attention to the dimension one, and shortly write GVAS instead of 1-dimensional GVAS.

Example 3.4. Consider again the Ackermann functions A_m introduced in Example 1.4. These can be expressed by the GVAS with nonterminals $X_0, \ldots, X_m$ and with production rules $X_0 \vdash 1$ and $X_i \vdash -1\ X_i\ X_{i-1} \mid 1 X_{i-1}$ for $1 \leq i \leq m$. It is routinely checked that $\max\{d \mid c \xrightarrow{X_m} d\} = A_m(c)$ for all $c \in \mathbb{N}$. □

Every GVAS can be effectively normalized, in logarithmic space, by replacing terminals $a \in \mathbb{Z}$ by words over the alphabet $\{-1, 0, 1\}$ and then putting the resulting grammar into Chomsky normal form. In addition, non-productive nonterminals, and production rules in which they occur, can be removed. So in order to simplify our proofs, we consider w.l.o.g. only GVAS of this simpler form.

Assumption. We restrict our attention to GVAS $G = (V, A, R, S, c_{init})$ in Chomsky normal form and where $A = \{-1, 0, 1\}$ and every $X \in V$ is productive.

The rest of the paper is devoted to the proof of Theorem 3.3. Before delving into its technical details, we give a high-level description the proof. In the next section, we characterize unboundedness in terms of certificates, which are complete parse trees whose nodes are labeled by natural numbers (or $-\infty$). These certificates contain a growing pattern that can be pumped to produce infinitely many reachable (1-dimensional) vectors, thereby witnessing unboundedness. We then prove that certificates need not be too large. To do so, we first show in Section 5 how to bound the size of growing patterns. Then, we bound the height and labels of "minimal" certificates in Section 6. Both bounds are singly-exponential in the size of the GVAS. Thus, the existence of a certificate can be checked by an alternating Turing machine running in polynomial space. This entails the desired ExpTime upper-bound stated in Theorem 3.3.

4 Certificates of Unboundedness

Following our previous work on the GVAS coverability problem [10], we annotate parse trees in a way that is consistent with the VAS semantics. A *flow tree* for a GVAS $G = (V, A, R, S, c_{init})$ is a complete[2] parse tree (T, sym) for G equipped with two functions $in, out : T \to \mathbb{N} \cup \{-\infty\}$, assigning an *input* and an *output* value to each node, with $in(\varepsilon) = c_{init}$, and satisfying, for every node $t \in T$, the following *flow conditions*:

1. If t is internal with children $t0, \ldots, tk$, then $in(t0) \leq in(t)$, $out(t) \leq out(tk)$, and $in(t(j+1)) \leq out(tj)$ for every $j = 0, \ldots, k-1$.
2. If t is a leaf, then $out(t) \leq in(t) + a$ if $sym(t) = a \in A$, and $out(t) \leq in(t)$ if $sym(t) = \varepsilon$.

We shortly write $t : c\#d$ to mean that $(in(t), sym(t), out(t)) = (c, \#, d)$. The *size* of a flow tree is the size of its underlying parse tree. Figure 3 (left) shows a flow tree for the GVAS of Example 3.4, with start symbol X_1 and initial (1-dimensional) vector $c_{init} = 5$.

[2] Compared to [10] where flow trees are built on arbitrary parse trees, the flow trees that we consider here are always built on complete parse trees.

Remark 4.1. The flow conditions enforce the VAS semantics along a depth-first pre-order traversal of the complete parse tree. But, as in [10], we only require inequalities instead of equalities. This corresponds to a *lossy* VAS semantics, where the counter can be non-deterministically decreased [2]. The use of inequalities in our flow conditions simplifies the presentation and allows for certificates of unboundedness with smaller input/output values. Note that equalities would be required to get certificates of reachability, but the latter problem is out of the scope of this paper.

Lemma 4.2. *For all d with $c_{init} \xrightarrow{S} d$, there exists a flow tree with $out(\varepsilon) = d$.*

Our main ingredient to prove Theorem 3.3 is a small model property. First, we show in this section that unboundedness can always be witnessed by a flow tree of a particular form, called a certificate (see Definition 4.5 and Figure 3). Then, we will provide in Theorem 6.10 exponential bounds on the height and input/output values of "minimal" certificates. This will entail the desired ExpTime upper-bound for the prefix-closed GVAS boundedness problem.

We start by bounding the size of flow trees that do not contain an iterable pattern, i.e., a nonterminal that repeats, below it, with a larger or equal input value. Formally, a flow tree (T, sym, in, out) is called *good* if it contains a node t and a proper ancestor $s \prec t$ such that $sym(s) = sym(t)$ and $in(s) \leq in(t)$. It is called *bad* otherwise. We bound the size of bad flow trees by (a) translating them into bad nested sequences, and (b) using a bound given in [8] on the length of bad nested sequences. Let us first recall some notions and results from [8]. Our presentation is deliberately simplified and limited to our setting.

Let $(S, \preceq, \|\cdot\|)$ be the normed quasi-ordered set defined by $S \stackrel{\text{def}}{=} V \times \mathbb{N}$, $(X, m) \preceq (Y, n) \stackrel{\text{def}}{\Leftrightarrow} X = Y \wedge m \leq n$, and $\|(X, m)\| = m$. A *nested sequence* is a finite sequence $(s_1, h_1), \ldots, (s_\ell, h_\ell)$ of elements in $S \times \mathbb{N}$ satisfying $h_1 = 0$ and $h_{j+1} \in h_j + \{-1, 0, 1\}$ for every index $j < \ell$ of the sequence. A nested sequence $(s_1, h_1), \ldots, (s_\ell, h_\ell)$ is called *good* if there exists $i < j$ such that $s_i \preceq s_j$ and $h_i \leq h_{i+1}, \ldots, h_j$. A *bad* nested sequence is one that is not good. A nested sequence $(s_1, h_1), \ldots, (s_\ell, h_\ell)$ is called *n-controlled*, where $n \in \mathbb{N}$, if $\|s_j\| < n + j$ for every index j of the sequence.

Theorem 4.3 ([8, TheoremVI.1]). *Let $n \in \mathbb{N}$ with $n \geq 2$. Every n-controlled bad nested sequence has length at most $F_{\omega.|V|}(n)$.*

The function $F_{\omega.|V|} : \mathbb{N} \to \mathbb{N}$ used in the theorem is part of the *fast-growing hierarchy*. Its precise definition (see, e.g., [8]) is not important for the rest of the paper. The following lemma provides a bound on the size of bad flow trees. Notice that this lemma applies to arbitrary GVAS (not necessarily prefix-closed).

Lemma 4.4. *Every bad flow tree has at most $F_{\omega.|V|}(c_{init} + 2)$ nodes.*

A good flow tree contains an iterable pattern that can be "pumped". However, the existence of such a pattern does not guarantee unboundedness. For that, we need stronger requirements on the input and output values, as defined below.

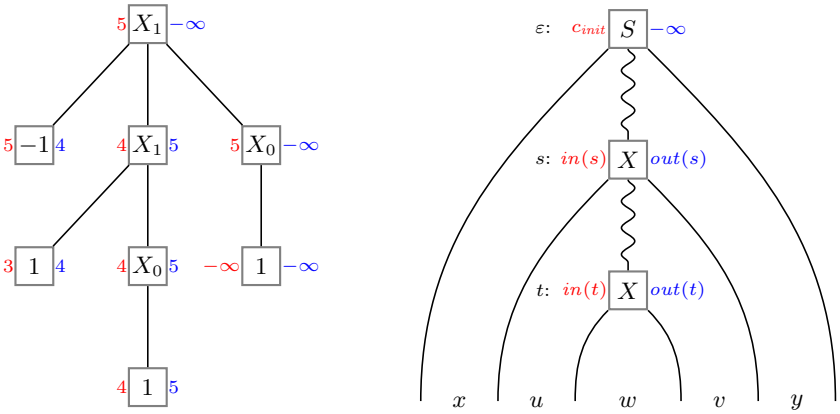


Fig. 3. Left: a flow tree for the GVAS of Example 3.4 with $c_{init} = 5$. Input and output values are indicated in red and blue, respectively. **Right:** A certificate with $sym(t) = sym(s) = X$ and yield $xuwvy \in A^*$. It must hold that either $in(s) < in(t)$ or $in(s) = in(t)$ and $out(t) < out(s)$.

Definition 4.5 (Certificates). *A* certificate *for a given GVAS is a flow tree* (T, sym, in, out) *equipped with two nodes* $s \prec t$ *in* T *such that*

$$sym(s) = sym(t) \quad \text{and} \quad in(s) \leq in(t) \quad \text{and} \quad in(s) < in(t) \text{ or } out(t) < out(s)$$

We now present the main result of this section, which shows that unboundedness can always be witnessed by a certificate.

Theorem 4.6. *A prefix-closed GVAS* G *is unbounded if, and only if, there exists a certificate for* G.

5 Growing Patterns

Certificates depicted on Figure 3 (right) introduce words $u \in A^*$ satisfying a sign constraint $\sum u > 0$ or $\sum u = 0$. These words are derivable from words of non-terminal symbols $S_1 \dots S_k$ corresponding to the left children of the nodes between s and t. In order to obtain small certificates, in this section, we provide bounds on the minimal length of words $u' \in A^*$ that can also be derived from $S_1 \dots S_k$ and that satisfy the same sign constraint as u.

Let us first introduce the *displacement* of a GVAS G as the "best shift" achievable by a word in L^G and defined by the following equality[3]:

$$\Delta^G \stackrel{\text{def}}{=} \sup\{\textstyle\sum z \mid z \in L^G\}$$

When the displacement is finite, the following Lemma 5.1 shows that it is achievable by a complete elementary parse tree. We say that a parse tree T is

[3] Notice that Δ^G may be negative.

elementary if for every $s \preceq t$ such that $sym(s) = sym(t)$, we have $s = t$. Notice that the size of an elementary parse tree is bounded by $2^{|V|+1}$.

Lemma 5.1. *Every GVAS G admits a complete elementary parse with a yield w such that $\Delta^G \in \{\sum w, +\infty\}$.*

Given a non-terminal symbol X, we denote by $G[X]$ the context-free grammar obtained from G by replacing the start symbol by X. We are now ready to state the main observation of this section.

Theorem 5.2. *For every sequence $S_1, \ldots, S_k$ of non-terminal symbols of a GVAS G there exists a sequence $T_1, \ldots, T_k$ of complete parse trees T_j for $G_j \stackrel{def}{=} G[S_j]$ with a yield z_j such that $|T_1| + \cdots + |T_k| \leq 3k4^{|V|+1}$, and such that $\sum z_1 \ldots z_k > 0$ if $\Delta^{G_1} + \cdots + \Delta^{G_k} > 0$, and $\sum z_1 \ldots z_k = 0$ if $\Delta^{G_1} + \cdots + \Delta^{G_k} = 0$.*

We first provide bounds on complete parse trees that witness the following properties $\Delta^G = +\infty$ and X is derivable. Formally, a nonterminal X is said to be *derivable* if there exists $w \in (A \cup V)^*$ that contains X and such that $S \stackrel{*}{\Longrightarrow} w$.

Lemma 5.3. *If $\Delta^G = +\infty$, there exists a parse tree for $G[X]$ where X is a non-terminal symbol derivable from the start symbol S with a yield uXv satisfying $u, v \in A^*$, $\sum uv > 0$, and a number of nodes bounded by $4^{|V|+1}$.*

Lemma 5.4. *For every derivable non-terminal symbol X, there exists a parse tree with a yield in A^*XA^* and a number of nodes bounded by $4^{|V|+1}$.*

Proof (of Theorem 5.2). We can assume that $k \geq 1$ since otherwise the proof is trivial. Observe that if $\Delta^{G_1} + \cdots + \Delta^{G_k} < +\infty$ then $\Delta^{G_j} < +\infty$ for every j. It follows from Lemma 5.1 that there exists a complete parse tree T_j for $G[S_j]$ with a yield w_j satisfying $\Delta^{G_j} = \sum w_j$ and a number of nodes bounded by $2^{|V|+1}$. Thus $|T_1| + \cdots + |T_k| \leq k2^{|V|+1}$ and $\sum w_1 \ldots w_k = \Delta^{G_1} + \cdots + \Delta^{G_k}$. So, in this special case the theorem is proved. Now, let us assume that $\Delta^{G_1} + \cdots + \Delta^{G_k} = +\infty$. There exists $p \in \{1, \ldots, k\}$ such that $\Delta^{G_p} = +\infty$. Lemma 5.3 shows that there exists a variable for X derivable from S_p and a parse tree T_+ for $G[X]$ with a yield uXv satisfying $u, v \in A^*$, $\sum uv > 0$, and such that $|T_+| \leq 4^{|V|+1}$. Since S_j is productive, there exists a complete elementary parse tree T_j for $G[S_j]$ with a yield $w_j \in A^*$. For the same reason, there exists a complete elementary parse tree T for $G[X]$ with a yield $w \in A^*$. As X is derivable from S, Lemma 5.4 shows that there exists a parse tree T' for G with a yield labeled by a word in $u'Xv'$ with $u', v' \in A^*$, and a number of nodes bounded by $4^{|V|+1}$. Notice that for any $n \in \mathbb{N}$, we deduce a complete parse tree T_p for $G[S_p]$ with a yield $w_p = u'u^n w v^n v'$ by inserting in T' many (n) copies of T_+ and one copy of T. Observe that $\sum w_1 \ldots w_k \geq -|w_1 \ldots w_{p-1} w_{p+1} \ldots w_k| - |u'wv'| + n \sum uv \geq -k2^{|V|+1} - 4^{|V|+1} + n$. Let us fix n to $2k4^{|V|+1} - 2$. It follows that $\sum w_1 \ldots w_p > 0$. Moreover, we have $|T_p| \leq |T| - 1 + n(|T_+| - 1) + |T'| \leq 2^{|V|+1} + n4^{|V|+1} + 4^{|V|+1} \leq (n+2)4^{|V|+1} \leq 2k4^{|V|+1}$. We derive $|T_1| + \cdots + |T_k| \leq (k-1)2^{|V|+1} + 2k4^{|V|+1} \leq 3k4^{|V|+1}$. We have proved Theorem 5.2. □

6 Small Certificates

We provide in this section exponential bounds on the height and input/output values of minimal certificates in the following sense. Let the *rank* of a flow tree (T, sym, in, out) be the pair

$$\left(|T_{in}| + |T_{out}| \, , \sum_{t \in T_{in}} in(t) + \sum_{t \in T_{out}} out(t)) \right)$$

where $T_{in} = \{t \in T \mid in(t) > -\infty\}$ and $T_{out} = \{t \in T \mid out(t) > -\infty\}$. Notice that $T_{out} \subseteq T_{in}$. We compare ranks using the lexicographic order $\preceq_{\text{lex}}$ over $\mathbb{N}^2$ and let the rank of a certificate $(\mathcal{T}, s, t)$ be the rank of its flow tree $\mathcal{T}$.

Consider a prefix-closed GVAS $G = (V, A, R, S, c_{init})$ that is unbounded. By Theorem 4.6, there exists a certificate for G. Pick a certificate $(\mathcal{T}, s, t)$ among those of least rank. Our goal is to bound the height and input/output values of $\mathcal{T}$. Based on its assumed minimality, we observe a series of facts about our chosen certificate.

First, we observe that some input/output values in $\mathcal{T}$ must be $-\infty$, because higher values would be useless in the sense that they can be set to $-\infty$ without breaking the flow conditions nor the conditions on s and t. This observation is formalized in the two following facts.

Fact 6.1. It holds that $out(p) = -\infty$ for every proper ancestor $p \prec s$. Moreover, $in(p) = out(p) = -\infty$ for every node $p \in T$ such that $s \prec_{\text{lex}} p$ and $p \not\preceq s$.

Fact 6.2. Assume that $in(s) < in(t)$. It holds that $out(p) = -\infty$ for every ancestor $p \preceq t$. Moreover, $in(p) = out(p) = -\infty$ for all $p \in T$ with $t \prec_{\text{lex}} p$.

Next, we observe that the main branch, that contains s and t, must be short.

Fact 6.3. It holds that $|s| \leq |V|$ and $|t| \leq |s| + |V| + 1$.

The next two facts provide *relative* bounds on input and output values for nodes that are not on the main branch.

Fact 6.4. It holds that $in(p) \leq out(p) + 2^{|V|}$ for every node $p \in T$ with $p \not\preceq t$.

Fact 6.5. Let $q \in T$ and let p be the parent of q. If $p = t$ or $p \not\preceq t$, then $out(q) \leq out(p) + 2^{|V|}$. If moreover $sym(p) = sym(q)$, then $out(q) < out(p)$.

The following facts provide *absolute* bounds on the input/output values of nodes s and t. The proofs of the facts below crucially rely on Section 5. Consider the subtrees on the left and on the right of the branch from s to t. The main idea of the proofs is to replace these subtrees by small ones using Theorem 5.2.

Fact 6.6. It holds that $out(t) \leq out(s) \leq 6|V| \cdot 4^{|V|+1}$.

Fact 6.7. It holds that $in(s) \leq in(t) \leq 7|V| \cdot 4^{|V|+1}$.

Now we derive absolute bounds for the input/output values of the remaining nodes on the main branch. These are derived from Facts 6.6 and 6.7, using Facts 6.4 and 6.5 about the way in/output values propagate and the Fact 6.3 that the intermediate path between nodes s and t is short.

Fact 6.8. It holds that $out(p) \leq 4^{2(|V|+1)}$ for every ancestor $p \preceq t$.

Fact 6.9. It holds that $in(p) \leq 4^{2(|V|+1)}$ for every ancestor $\varepsilon \prec p \preceq t$.

We are now ready to derive bounds on the rank of our minimal certificate. Notice that it remains only to bound the depth and the input/output values on branches different from the main branch.

Consider therefore a node q outside the main branch, i.e., $q \not\preceq t$. Let p be the least prefix of q such that $p = t$ or $p \not\preceq t$. We first show that $out(p) \leq 4^{2(|V|+1)}$. If $p = t$ then the claim follows from Fact 6.8. Otherwise, the parent r of p satisfies $r \prec t$. Observe that the other child $\bar{p}$ of r satisfies $\bar{p} \preceq t$. The flow conditions together with the minimality of $(\mathcal{T}, s, t)$ guarantee that

- if $p = r1$ then $out(p) = out(r)$, hence, $out(p) \leq 4^{2(|V|+1)}$ by Fact 6.8, and
- if $p = r0$ then $out(p) = in(r1)$, hence, $out(p) \leq 4^{2(|V|+1)}$ by Fact 6.9.

According to Fact 6.5, the output values on the branch from p down to q may only increase when visiting a new symbol. Moreover, this increase is bounded by $2^{|V|}$. It follows that $out(r) \leq out(p) + |V|2^{|V|}$ for every node r such that $p \prec r \preceq q$. Fact 6.4 entails that $in(r) \leq out(p) + (|V| + 1)2^{|V|}$. We obtain that $\max\{in(r), out(r)\} < 4^{3(|V|+1)}$ for every node r with $p \prec r \preceq q$. Fact 6.5 also forbids the same nonterminal from appearing twice with the same output value, so $|r| \leq |p| + |V| \cdot 4^{3(|V|+1)} + 1$. Observe that $|p| \leq |t|$. We derive from Fact 6.3 that $|r| \leq 4^{4(|V|+1)}$. This concludes the proof of the following theorem.

Theorem 6.10. *A prefix-closed GVAS (V, A, R, S, c_{init}) is unbounded if, and only if, it admits a certificate with height and all input/output values bounded by $c_{init} + 4^{4(|V|+1)}$.*

Proof (of Theorem 3.3). By Theorem 6.10, a certificate for unboundedness is a flow tree of exponential height and with all input and output labels exponentially bounded. An alternating Turing machine can thus guess and verify all branches of such a flow tree, storing intermediate input/output values as well as the remaining length of a branch in polynomial space. The claim then follows from the fact that alternating polynomial space equals exponential time. □

7 Conclusion

We discussed different boundedness problems for pushdown vector addition systems [7,8], which are a known, and very expressive computational model that features nondeterminism, a pushdown stack and several counters. These systems may be equivalently interpreted, in the context of regulated rewriting [3], as vector addition systems with context-free control languages.

We observe that boundedness is reducible to both counter- and stack-boundedness. The stack boundedness problem can be shown to be decidable (with hyper-Ackermannian complexity) by adjusting the algorithm presented in [8].

Here, we single out the special case of the counter-boundedness problem for one-dimensional systems and propose an exponential-time algorithm that solves it. This also improves the best previously known Ackermannian upper bound for boundedness in dimension one.

Currently, the best lower bound for this problem is NP, which can be seen by reduction from the subset sum problem. For dimension two, PSPACE-hardness follows by reduction from the state-reachability of bounded one-counter automata with succinct counter updates [4]. For arbitrary dimensions, TOWER-hardness is known already for the boundedness problem [7,8] but the decidability of counter-boundedness for PVAS remains open.

Acknowledgments. The authors wish to thank M. Praveen for insightful discussions. We also thank the anonymous referees for their useful comments and suggestions.

References

1. Ball, T., Majumdar, R., Millstein, T.D., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: PLDI, pp. 203–213 (2001)
2. Bouajjani, A., Mayr, R.: Model checking lossy vector addition systems. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 323–333. Springer, Heidelberg (1999)
3. Dassow, J., Pun, G., Salomaa, A.: Grammars with controlled derivations. In: Handbook of Formal Languages, pp. 101–154 (1997)
4. Fearnley, J., Jurdziński, M.: Reachability in two-clock timed automata is PSPACE-complete. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 212–223. Springer, Heidelberg (2013)
5. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
6. Kosaraju, S.R.: Decidability of reachability in vector addition systems (preliminary version). In: STOC, pp. 267–281 (1982)
7. Lazic, R.: The reachability problem for vector addition systems with a stack is not elementary. CoRR abs/1310.1767 (2013)
8. Leroux, J., Praveen, M., Sutre, G.: Hyper-ackermannian bounds for pushdown vector addition systems. In: CSL/LICS (2014)
9. Leroux, J., Schmitz, S.: Demystifying reachability in vector addition systems. In: LICS (2015)
10. Leroux, J., Sutre, G., Totzke, P.: On the coverability problem for pushdown vector addition systems in one dimension. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 324–336. Springer, Heidelberg (2015)
11. Lipton, R.J.: The reachability problem requires exponential space. Tech. Rep. 63, Yale University (January 1976)
12. Mayr, E.W.: An algorithm for the general Petri net reachability problem. In: STOC. pp. 238–246 (1981)
13. Rackoff, C.: The covering and boundedness problems for vector addition systems. TCS **6**(2), 223–231 (1978)

Multithreaded-Cartesian Abstract Interpretation of Multithreaded Recursive Programs Is Polynomial

Alexander Malkis(✉)

Technische Universität München, Munich, Germany

Abstract. Undecidability is the scourge of verification for many program classes. We consider the class of shared-memory multithreaded programs in the interleaving semantics such that the number of threads is finite and constant throughout all executions, each thread has an unbounded stack, and the shared memory and the stack-frame memory are finite. Verifying that a given program state does not occur in executions of such a program is undecidable. We show that the complexity of verification drops to polynomial time under multithreaded-Cartesian abstraction. Furthermore, we demonstrate that multithreaded-Cartesian abstract interpretation generates an inductive invariant which is a regular language. Under logarithmic cost measure, both proving non-reachability and creating a finite automaton can be attained in $\mathcal{O}(n \log_2 n)$ time in the number of threads n and in polynomial time in all other quantities.

1 Introduction

Verification of multithreaded programs is hard. In the presence of recursive procedures, the problem of membership in the strongest inductive invariant is undecidable: given a two-threaded program with a stack per thread, one can simulate a Turing tape. However, if the stack depth is the only unbounded quantity, there might be interesting inductive invariants of special forms such that membership in such invariants is decidable. In other words, one might circumvent undecidability by considering specially-formed overapproximations of the set of program states that are reachable from the initial ones.

We now briefly sketch one such interesting form. Let a program state be an $(n+1)$-tuple in which one component contains a valuation of the shared variables and each of the remaining n components contains a valuation of the local variables (including the control flow) of a distinct thread. Let us say that two program states are equivalent if they have the same shared-variables entry. We define a set of states to be of the *multithreaded-Cartesian* form if each equivalence class is an $(n+1)$-dimensional Cartesian product. (A rigorous definition will appear in § 4.) The multithreaded-Cartesian inductive invariants of a program constitute a Moore family.

The author greatly acknowledges useful comments and suggestions from Neil Deaton Jones.

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 114–127, 2015.
DOI: 10.1007/978-3-319-24537-9_11

It is known that in the finite-state case the membership problem for the strongest multithreaded-Cartesian inductive invariant is in PTIME [16]. We extend this result to programs in which each thread has a potentially unbounded stack. Moreover, we show that the strongest multithreaded-Cartesian inductive invariant is a regular language when viewed as a formal language of strings. Computing a corresponding finite automaton as well as solving the membership problem for the strongest multithreaded-Cartesian inductive invariant can be accomplished in time $\mathcal{O}(n \log_2 n)$, where n is the number of threads, and in polynomial time in all the other quantities.

The presentation will proceed as follows.

- After an overview of related work, we rigorously define our program class in § 3 and formulate the problem of determining the strongest multithreaded-Cartesian inductive invariant in the abstract interpretation framework in § 4.
- Next, in § 5, we present a new inference system, which we call TMR, which constructs n automata such that the i^{th} automaton describes an overapproximation of the set of pairs (shared state, stack word of the i^{th} thread) that occur in the computations of the multithreaded program.
- Based on the computation result of TMR, we show how to create an automaton that describes the strongest multithreaded-Cartesian inductive invariant.
- Then, we determine the asymptotic worst-case running times of TMR and the automaton construction under logarithmic cost measure [17].
- In §§ 6–7, we conclude with the proof of correctness of our construction.

We make sure that if some or all of the input quantities (the number of threads, the number of shared states, and the number of different stack frames) are infinite, TMR still leads to a logically valid (but not necessarily executable) description of the multithreaded-Cartesian abstract interpretation. This opens way to using constraint solvers in the infinite case. We will impose finiteness restrictions only when presenting low-level algorithms and computing the running times.

Due to restricted space, most computations and proofs are found in [15].

2 Related Work

There is a large body of work on the analysis of concurrent programs with recursion; we discuss next only the literature which is, subjectively, most related to our work.

The roots of multithreaded-Cartesian abstraction date back to the Owicki-Gries proof method [20], followed by thread-modular reasoning of C. B. Jones [12], and the Flanagan-Qadeer model-checking method for nonrecursive programs [10]. The basic versions of these methods (without auxiliary variables) exhibit the same strength. This strength is precisely characterized by multithreaded-Cartesian abstract interpretation, which was first discovered by R. Cousot [9] and later rediscovered in [14,16].

Flanagan and Qadeer [10] introduce also a method for recursive multithreaded programs, for which they claim an $\mathcal{O}(n^2)$ upper bound on the worst-case running time. Their analysis, which predates TMR, simultaneously computes procedure summaries and distributes the changes of the shared states

between the threads. Where their algorithm is only summarization-based, our TMR is an explicit automaton construction. Our program model is slightly different compared to [10]. First, to simplify our presentation, we remove the concept of a local store: while in practice there may be different kinds of local stores (static-storage function-scope variables in the sense of the C programming language, the registers of a processor, ...), every program can be modified to perform all thread-local computations on the stack. Second, we allow changing the shared state when a stack grows or shrinks to permit richer program models; whenever possible, we also allow infinite-size sets.

An alternative approach to prove polynomial time of multithreaded-Cartesian abstract interpretation could be to apply Horn clauses as done in [19] for some other problems. That way would not reveal regularity or connections to the algorithms of Flanagan and Qadeer; it will also just give a running-time bound for the unit cost measure, whereas we count more precisely in the logarithmic cost measure. We will not discuss it here.

Outside multithreaded-Cartesian abstract interpretation there are many other methods for analyzing concurrent recursive programs.

If the interplay between communication and recursion is restricted, decidable fragments can be identified; we will mention just a few. If only one thread has a stack and the other threads are finite-state, then one can construct the product of the threads and model-check a large class of properties [7,24]. Alternatively, one can allow certain forks and joins but restrict communication only to threads that do not have ongoing procedure calls [5]. In the synchronous execution model, one may restrict the threads to perform all the calls synchronously and also perform all returns synchronously [1]. Alternatively, one may restrict pop operations to be performed only on the first nonempty stack [3].

Without restrictions on the interplay between communication and recursion, one may allow non-termination of the analysis [21] or be satisfied with an approximate analysis, which can be sound [4] or complete [13,22] (for every choice of parameters), but never both. If shared-memory communication is replaced by rendezvous, verification is still undecidable [23].

The summarization idea behind TMR dates back to the works of Büchi [6]. Since then, it has been developed further in numerous variants for computing the exact semantics, e.g., for two-way deterministic pushdown automata with a write-once-read-many store [18], for imperative stack-manipulating programs with a rich set of operations on the stack [2], and in implementations of partial evaluators [11].

3 Programs

Now we introduce notation and our model of recursive multithreaded programs.

Let $\mathbb{N}_0$ (resp. $\mathbb{N}_+$) be the sets of natural numbers with (resp. without) zero. We write X^* (resp. X^+) for the set of finite (resp. finite nonempty) words over an alphabet X, ε for the empty word, and $|w|$ for the length of a word $w \in X^*$.

An *n-threaded recursive program* (from now on, simply a *program*) is a tuple

$$(\mathrm{Glob}, \mathrm{Frame}, \mathrm{init}, (\,\underline{\amalg}_t, \sqcup_t, \,\overline{\amalg}_t)_{t<n})$$

such that the following conditions hold:

- Glob and Frame are arbitrary sets such that, without loss of generality, $(\mathrm{Glob}\times\mathrm{Frame}) \cap \mathrm{Glob} = \emptyset$. (We think of Glob as the set of shared states, e.g., the set of valuations of shared variables. We think of Frame as the set of stack frames, where a stack frame is, e.g., a valuation of procedure-local variables and the control-flow counter. The necessity of the disjointness condition will get clear later on.)
- n is an arbitrary ordinal. (For our convenience, we think of n as both the number of threads and the set of thread identifiers. E.g., we view $(\mathrm{Frame}^+)^n$ as the set of maps $n\to\mathrm{Frame}^+$, and, in the finite case, n as $\{0, 1, \ldots, n-1\}$. Real programs are usually modeled by finite n or $n=\omega$, and we allow arbitrary n.)
- $\mathrm{init} \subseteq \mathrm{Glob}\times(\mathrm{Frame}^+)^n$ is such that $\forall\,(g,l) \in \mathrm{init},\ t{\in}n\colon\ |l_t| = 1$. (By $l_t = l(t)$ we indicate the t^{th} component of $l \in (\mathrm{Frame}^+)^n$. We think of init as of the set of initial states. The depth of the stacks of the threads is 1 in every initial state.)
- For each $t{\in}n$, the transition relation of thread t is given by sets $\underline{\amalg}_t \subseteq (\mathrm{Glob}\times\mathrm{Frame}) \times (\mathrm{Glob}\times\mathrm{Frame}\times\mathrm{Frame})$, $\sqcup_t \subseteq (\mathrm{Glob}\times\mathrm{Frame})^2$, and $\overline{\amalg}_t \subseteq (\mathrm{Glob}\times\mathrm{Frame}\times\mathrm{Frame}) \times (\mathrm{Glob}\times\mathrm{Frame})$. (These are sets of push, internal, and pop transitions of thread t, respectively.)

We denote by $\mathrm{Loc} = \mathrm{Frame}^+$ the set of *local states* of each thread; the elements of $\mathrm{Glob}\times\mathrm{Loc}$ are called *thread states*. The operational semantics of each thread $t{<}n$ is given by the relation $\rightsquigarrow_t \subseteq (\mathrm{Glob}\times\mathrm{Loc})^2$, which is defined by

$$\begin{aligned}(g,w) \rightsquigarrow_t (g',w') \quad &\overset{\text{def}}{\iff}\\ \big((\exists\, a,b,c \in \mathrm{Frame},\ u \in \mathrm{Frame}^*\colon\ & w = au \wedge w' = bcu \wedge ((g,a),(g',b,c)) \in \underline{\amalg}_t)\\ \vee\ (\exists\, a,b \in \mathrm{Frame},\ u \in \mathrm{Frame}^*\colon\ & w = au \wedge w' = bu \wedge ((g,a),(g',b)) \in \sqcup_t)\\ \vee\ (\exists\, a,b,c \in \mathrm{Frame},\ u \in \mathrm{Frame}^*\colon\ & w = abu \wedge w' = cu \wedge ((g,a,b),(g',c)) \in \overline{\amalg}_t)\big)\end{aligned}$$

for $g, g' \in \mathrm{Glob}$ and $w, w' \in \mathrm{Loc}$. Notice that the stacks are always kept nonempty. Let the set of *program states* be

$$\mathrm{State} \;=\; \mathrm{Glob} \times \mathrm{Loc}^n\,.$$

The operational semantics of the whole program is given by the *concrete domain*

$$D \;=\; \mathfrak{P}(\mathrm{State})\,,$$

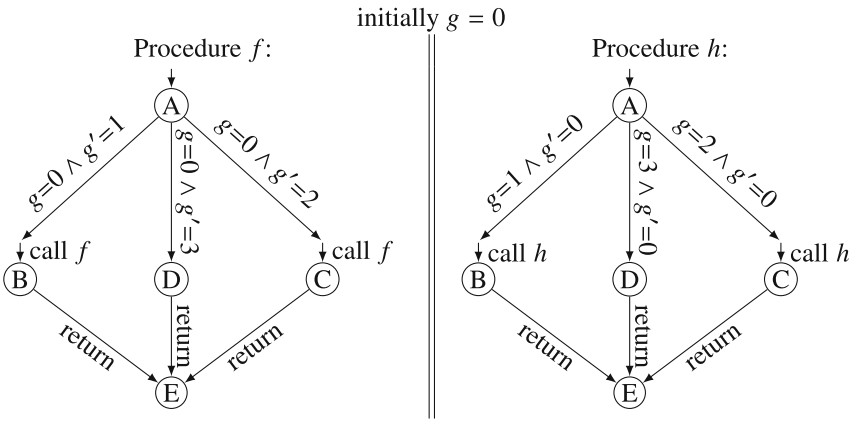
which is the power set of the set of program states, and the successor map

$$\begin{aligned}&\mathrm{post} : D \to D\,,\\ &\qquad Q \mapsto \{(g',l') \mid \exists\, t{\in}n,\ (g,l) \in Q\colon (g,l_t) \rightsquigarrow_t (g',l'_t) \wedge \forall\, s \in n\backslash\{t\}\colon l_s = l'_s\}\,.\end{aligned}$$

Broadly speaking, program analyses compute or approximate the so-called *collecting semantics*, which is the strongest inductive invariant (lfp = least fixpoint)

$$\mathrm{lfp}(\lambda\, S \in D.\ \mathrm{init} \cup \mathrm{post}(S))\,.$$

This set can become rather complex, loosely speaking, due to subtle interplay between concurrency and recursion. A nontrivial example is presented by the following control-flow graph of a two-threaded program over a shared variable g:



Procedures f and h execute in parallel. Roughly speaking, the left thread announces how it builds its stack by changing g from 0 to 1 or 2, and the right thread follows the stack operations of the left thread, confirming that it proceeds by resetting g to 0. Setting g to 3 initiates reduction of the stacks. For simplification, we assume that the thread transitions between each pair of named consecutive control flow locations (A to B, A to C, A to D, B to E, C to E, D to E) are atomic.

We model this program by Glob = {0,1,2,3}, Frame = {A,B,C,D} (E does not occur in computations), n=2, init = $\{(0, (\mathrm{A},\mathrm{A}))\}$, $\sqcup_0 = \{((0,\mathrm{A}), (1,\mathrm{A},\mathrm{B})), ((0,\mathrm{A}), (2,\mathrm{A},\mathrm{C}))\}$, $\sqcup_0 = \{((0,\mathrm{A}), (3,\mathrm{D}))\}$, $\sqcup_0 = \{((g,y,z), (g,z)) \mid g{\in}\mathrm{Glob} \wedge y \in \{\mathrm{B},\mathrm{C},\mathrm{D}\} \wedge z{\in}\mathrm{Frame}\}$, $\sqcup_1 = \{((1,\mathrm{A}), (0,\mathrm{A},\mathrm{B})), ((2,\mathrm{A}), (0,\mathrm{A},\mathrm{C}))\}$, $\sqcup_1 = \{((3,\mathrm{A}), (0,\mathrm{D}))\}$, and $\sqcup_1 = \{((g,y,z), (g,z)) \mid g{\in}\mathrm{Glob} \wedge y \in \{\mathrm{B},\mathrm{C},\mathrm{D}\} \wedge z{\in}\mathrm{Frame}\}$.

One can show that the strongest inductive invariant is

$$\begin{aligned}
&\{0\} \times \left(\begin{array}{l} \{(\mathrm{A}y, \mathrm{A}y), (\mathrm{D}y, \mathrm{D}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, \{(\mathrm{D}y, z) \mid y, z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge\ z \text{ is a suffix of } y\} \\ \cup\, \{(y, \mathrm{D}z) \mid y, z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge\ y \text{ is a suffix of } z\} \\ \cup\, \{(y, z) \mid y, z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge (y \text{ is a suffix of } z\ \vee\ z \text{ is a suffix of } y)\} \end{array}\right) \\
\cup\ &\{1\} \times \{(\mathrm{AB}y, \mathrm{A}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\
\cup\ &\{2\} \times \{(\mathrm{AC}y, \mathrm{A}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\
\cup\ &\{3\} \times \left(\begin{array}{l} \{(\mathrm{D}y, \mathrm{A}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, \{(y, \mathrm{A}z) \mid y, z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge\ y \text{ is a suffix of } z\} \end{array}\right).
\end{aligned}$$

This set, viewed as a formal language over Glob, Frame, and some special symbol separating the stacks, is not context-free.

Notice that $g \in \{0, 3\}$ is a valid postcondition of the considered program. In the next section we will see what multithreaded-Cartesian abstract interpretation is and how it helps proving this postcondition.

4 Multithreaded-Cartesian Abstract Interpretation

Now we are going to describe an approximation operator on the concrete domain of states of a program, essentially recapitulating the key points of [14]. Loosely speaking, the definition of the approximation will not depend on the internal structure of Loc and post.

The *multithreaded-Cartesian approximation* is the map

$$\rho_{\mathrm{mc}} : D \to D, \quad S \mapsto \{(g, l) \in \mathrm{State} \mid \forall\, t{\in}n\, \exists\, \hat{l}{\in}\mathrm{Loc}^n \colon (g, \hat{l}) \in S \,\wedge\, l_t = \hat{l}_t\}\,,$$

which, intuitively, given a set of states, partitions it into blocks according to the shared state, and approximates each block by its Cartesian hull.

One can show that ρ_{mc} is an upper closure operator on $(D, \subseteq)$.

We define the *multithreaded-Cartesian (collecting) semantics* as the least fixpoint

$$\mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S)))\,.$$

For our running example, for any $S \subseteq \mathrm{State}$ we have

$$\rho_{\mathrm{mc}}(S) = \{(g, (l_0, l_1)) \mid (\exists\, \bar{l}_1{\in}\mathrm{Loc} \colon (g, (l_0, \bar{l}_1)) \in S) \wedge (\exists\, \bar{l}_0{\in}\mathrm{Loc} \colon (g, (\bar{l}_0, l_1)) \in S)\}.$$

The multithreaded-Cartesian semantics of our running example is

$$\begin{pmatrix} \{0\} \times (\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \\ \cup\, \{1\} \times \{\mathrm{AB}x \mid x \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, \{2\} \times \{\mathrm{AC}x \mid x \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, \{3\} \times (\{\mathrm{D}x \mid x \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \end{pmatrix} \times (\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+)\,.$$

This set, viewed as a formal language, is regular; a corresponding regular expression is $\big(0(\mathrm{A}|\mathrm{B}|\mathrm{C}|\mathrm{D})(\mathrm{B}|\mathrm{C})^* \mid 1\mathrm{AB}(\mathrm{B}|\mathrm{C})^* \mid 2\mathrm{AC}(\mathrm{B}|\mathrm{C})^* \mid 3(\mathrm{B}|\mathrm{C}|\mathrm{D})(\mathrm{B}|\mathrm{C})^*\big)$ † $(\mathrm{A}|\mathrm{B}|\mathrm{C}|\mathrm{D})(\mathrm{B}|\mathrm{C})^*$, where †$\notin$Frame is a fresh symbol separating the local parts. Notice that the postcondition $g \in \{0, 3\}$ holds also in this abstract semantics.

5 Model-Checking Recursive Multithreaded Programs

Now we develop an efficient algorithm to compute the input program's multithreaded-Cartesian semantics. First we show the inference system TMR, then we show how its output is interpreted as multithreaded-Cartesian semantics, and finally we turn to computational issues.

5.1 Inference System TMR

Given an n-threaded program as described in § 3, our algorithm generates n automata that describe an overapproximation of the set of stack words of the threads that occur in computations.

Fix some "fresh" element $\mathfrak{f} \notin \text{Glob} \,\dot\cup\, (\text{Glob}\times\text{Frame})$. Let

$$V \;=\; \text{Glob}\,\dot\cup\,\{(g',b) \mid \exists\, g{\in}\text{Glob}, a,c \in \text{Frame}, t{\in}n\colon ((g,a),(g',b,c)) \in \uplus_t\}\,\dot\cup\,\{\mathfrak{f}\}\,.$$

We now define binary relations $G_t \subseteq \text{Glob}^2$ and ternary relations $\underset{t}{\rightarrow} \subseteq V\times\text{Frame}\times V$ for all $t{\in}n$ by the following inference system.

$$\text{(TMR INIT)}\ \frac{(g,l) \in \text{init}}{g \xrightarrow[t]{l_t} \mathfrak{f}}\ t{\in}n \qquad \text{(TMR STEP)}\ \frac{((g,a),(g',b)) \in \uplus_t \qquad g \xrightarrow[t]{a} v}{g' \xrightarrow[t]{b} v \qquad (g,g') \in G_t}\ t{\in}n$$

$$\text{(TMR PUSH)}\ \frac{g \xrightarrow[t]{a} v \qquad ((g,a),(g',b,c)) \in \uplus_t}{g' \xrightarrow[t]{b} (g',b) \xrightarrow[t]{c} v \qquad (g,g') \in G_t}$$

$$\text{(TMR POP)}\ \frac{g \xrightarrow[t]{a} v \xrightarrow[t]{b} \bar v \qquad ((g,a,b),(g',c)) \in \uplus_t}{g' \xrightarrow[t]{c} \bar v \qquad (g,g') \in G_t}$$

$$\text{(TMR ENV)}\ \frac{(g,g') \in G_t \qquad g \xrightarrow[s]{a} v}{g' \xrightarrow[s]{a} v}\ t{\neq}s \text{ are in } n$$

TMR INIT gathers stack contents of the initial states. TMR STEP, TMR PUSH, and TMR POP create an automaton describing thread states that occur in computations of the threads in isolation; the stacks are obtained from the upper labels of certain walks. Moreover, the three rules collect information about how the shared state is altered. The rule TMR ENV transfers shared-state changes between the threads.

For our program from page 118, the automata constructed by TMR are in Fig. 1.

5.2 Interpretation of the Output of TMR

Now we define the set of states that the inference system represents.

For that, we extend $\rightarrow$ to words of stack frames in a standard way. For each $t{\in}n$, consider the quaternary relation $\underset{t}{\rightarrow} \subseteq V\times\text{Frame}^*\times\mathbb{N}_0\times V$ (slightly abusing notation, we employ the same symbol as for the ternary relation from § 5.1) defined by the following inference system:

$$\frac{}{v \xrightarrow[t]{\varepsilon} {}^0\, v} \qquad \frac{i{\in}\mathbb{N}_0 \qquad a{\in}\text{Frame} \qquad y{\in}\text{Frame}^* \qquad v \xrightarrow[t]{a} \hat v \xrightarrow[t]{y} {}^i\, \bar v}{v \xrightarrow[t]{ay} {}^{i+1}\, \bar v}$$

For each $t{\in}n$, we define $\underset{t}{\rightarrow}^* \subseteq V\times\text{Frame}^*\times V$ by $\underset{t}{\rightarrow}^* = \bigcup_{i\in\mathbb{N}_0} \underset{t}{\rightarrow}^i$ and $L_{g,t} = \{w \mid g \xrightarrow[t]{w}{}^* \mathfrak{f}\}$ ($g{\in}\text{Glob}$).

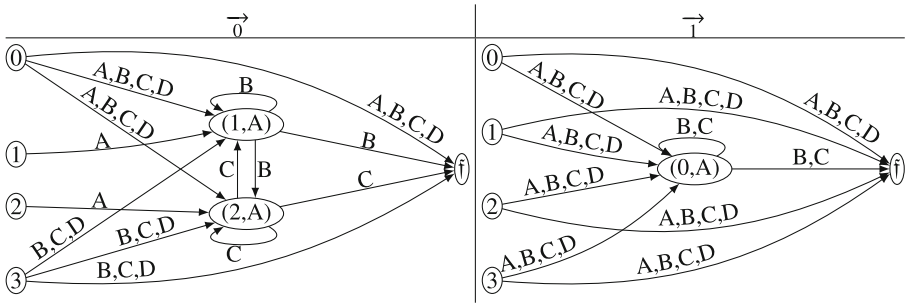


Fig. 1. Automata constructed by TMR for our example. Each arrow on the left carries the lower index 0; each arrow on the right carries the lower index 1.

Informally, a walk $g \xrightarrow[t]{w}{}^{*} \mathfrak{f}$ means that the state (g, w) of thread t occurrs in the approximate semantics, and $g \xrightarrow[t]{w}{}^{*} (g', b)$ means that a procedure call starting with thread state (g', b) can reach (g, w) in thread t in the approximate semantics.

The inference system TMR represents the set

$$\bigcup_{g \in \mathrm{Glob}} \{g\} \times \prod_{t \in n} L_{g,t}\,. \tag{1}$$

5.3 Computing Multithreaded-Cartesian Semantics

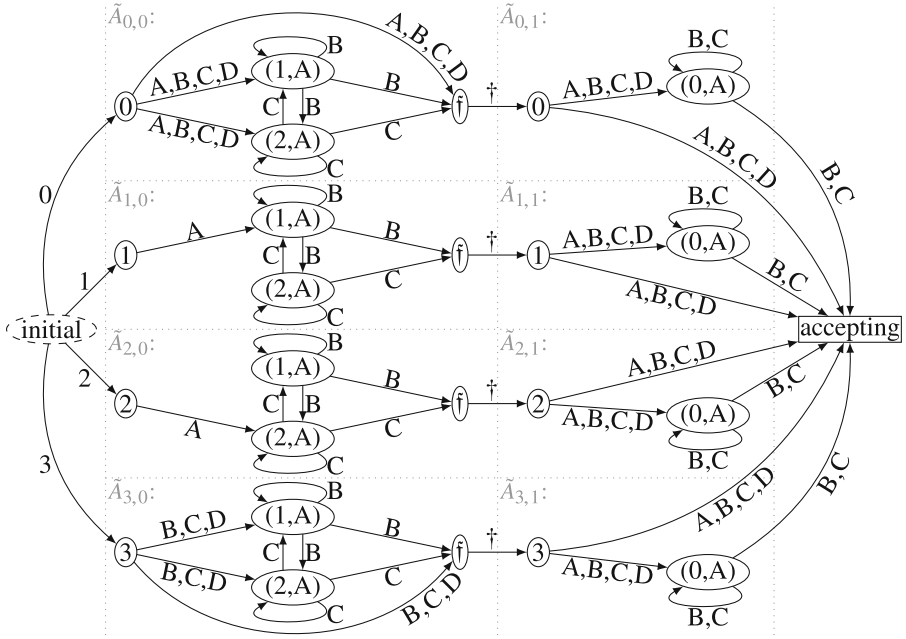
For actual computations, which we discuss now, let us assume finite n, Glob, and Frame till the end of § 5.3.

If we are just interested in checking non-reachability of thread states, executing TMR suffices: if a local state l is not in $L_{g,t}$, then the thread state (g, l) of the t^{th} thread does not occur in computations of the program ($g{\in}$Glob, $l{\in}$Loc, $t{\in}n$). If we are interested in checking non-reachability of single program states, executing TMR also suffices: for $(g, \bar{l}) \in$ State, if $(g, \bar{l}_t) \notin L_{g,t}$ for some $t{\in}n$, then the state (g, l) is unreachable from the initial ones. Executing TMR on a RAM with logarithmic-cost measure can be achieved in $\mathcal{O}\big(n(|\mathrm{init}| + |\mathrm{Glob}|^4|\mathrm{Frame}|^5)(L(|\mathrm{init}|) + L(n) + L(|\mathrm{Glob}|) + L(|\mathrm{Frame}|))\big)$ time, where $L(x)$ is the length of the binary representation of $x{\in}\mathbb{N}_0$. With rigorous definitions of the input the running time is $\mathcal{O}((\text{input length})^2 L(\text{input length}))$.

If we wish to prove more general invariants, we construct a finite automaton for (1) as follows. First, we make the state spaces of the automata accepting $L_{g,t}$ disjoint ($(g,t) \in \mathrm{Glob}{\times}\mathrm{Loc}$), obtaining automata $\tilde{A}_{g,t}$ ($(g,t) \in \mathrm{Glob}{\times}\mathrm{Loc}$). If we wish to obtain a deterministic automaton at the end, we additionally determinize all $\tilde{A}_{g,t}$ ($(g,t) \in \mathrm{Glob}{\times}\mathrm{Loc}$). Second, for each $g{\in}$Glob, chain $\tilde{A}_{g,t}$ for $t{<}n$ to accept exactly the words of the form $w_0 \dagger \ldots \dagger w_{n-1}$ over Frame$\dot\cup\{\dagger\}$ (where $\dagger \notin$Frame is a fresh symbol separating the local parts) such that $(w_t)_{t<n} \in \prod_{t<n} L_{g,t}$. Third, introduce a single initial state that dispatches different g to

$\tilde{A}_{g,0}$ ($g \in \mathrm{Glob}$). Thus, (1) can be viewed as a regular language. The nondeterministic, ε-free automaton can be constructed (including executing TMR) in the same $\mathcal{O}\big(n(|\mathrm{init}| + |\mathrm{Glob}|^4 |\mathrm{Frame}|^5)(L(|\mathrm{init}|) + L(n) + L(|\mathrm{Glob}|) + L(|\mathrm{Frame}|))\big)$ asymptotic time.

For our running example, we transform the left automaton from Fig. 1 into four automata $\tilde{A}_{0,0}$–$\tilde{A}_{3,0}$ accepting $L_{0,0}$–$L_{3,0}$ and the right automaton into four automata $\tilde{A}_{0,1}$–$\tilde{A}_{3,1}$ accepting $L_{0,1}$–$L_{3,1}$. We combine them into a nondeterministic finite automaton for (1) as follows (only the reachable part is shown):



In this graphical representation, the disjoint copies carry the same node labels, and the final states of $\tilde{A}_{0,1}$–$\tilde{A}_{3,1}$ have been merged to a unique accepting state. (Certainly, much more minimization is possible, mimicking sharing in BDDs—which is an interesting topic by itself but not our goal here.)

Theorem 1. *The inference system TMR is equivalent to multithreaded Cartesian abstract interpretation. Formally:*

$$\mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S))) \;=\; \bigcup_{g \in \mathrm{Glob}} \{g\} \times \prod_{t \in n} L_{g,t}\,.$$

Indeed, in our running example, the multithreaded-Cartesian collecting semantics corresponds to the language accepted by the above automaton.

The following §§ 6–7 will be devoted to proving Theorem 1.

6 Model-Checking General Multithreaded Programs

As an intermediate step in proving equivalence of multithreaded Cartesian abstract interpretation and TMR we are going to show another, simpler inference-system for proving properties of multithreaded programs. This inference system (up to names of variables and sets) is due to Flanagan and Qadeer [10]. Its definition does not depend on the internal structure of Loc and $\rightsquigarrow_t$ ($t{\in}n$).

Let us define sets $\widetilde{R}_t \subseteq \mathrm{Glob}{\times}\mathrm{Loc}$ and $\widetilde{G}_t \subseteq \mathrm{Glob}^2$ for all $t{\in}n$ by the following inference system FQ:

$$\text{(FQ INIT)}\ \frac{(g,l) \in \mathrm{init}}{(g,l_t) \in \widetilde{R}_t}\ t{\in}n \qquad \text{(FQ STEP)}\ \frac{(g,l) \in \widetilde{R}_t \qquad (g,l) \rightsquigarrow_t (g',l')}{(g',l') \in \widetilde{R}_t \qquad (g,g') \in \widetilde{G}_t}\ t{\in}n$$

$$\text{(FQ ENV)}\ \frac{(g,g') \in \widetilde{G}_t \qquad (g,l) \in \widetilde{R}_s}{(g',l) \in \widetilde{R}_s}\ s{\neq}t \text{ are in } n$$

For finite-state programs, the families $\widetilde{R}$ and $\widetilde{G}$ can be generated in polynomial time. The algorithm is sound independently of finiteness, e.g., also for recursive programs.

One can show, roughly speaking, that multithreaded-Cartesian abstract interpreration is equivalent to FQ. We will use FQ intermediately, showing FQ $\approx$ TMR.

7 Proof of Theorem 1

We show a semi-formal, high-level proof outline; rigorous details are found in [15].

We start by defining

$$R_t \ = \ \{(g,w) \in \mathrm{Glob}{\times}\mathrm{Loc} \mid g \xrightarrow[t]{w}{}^{*} \mathfrak{f}\} \qquad (t \in n).$$

Informally, the set R_t contains exactly the thread states of the thread t in the invariant denoted by TMR ($t{<}n$).

Now let $G = (G_t)_{t\in n} \in (\mathfrak{P}(\mathrm{Glob}^2))^n$ and $R = (R_t)_{t\in n} \in (\mathfrak{P}(\mathrm{Glob}{\times}\mathrm{Loc}))^n$.

For our running example,

$$R_0 \ = \ \begin{pmatrix} \{0\} \times (\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \cup \{1\} \times \{\mathrm{AB}x \mid x{\in}\{\mathrm{B},\mathrm{C}\}^*\} \\ \cup \{3\} \times (\{\mathrm{D}x \mid x{\in}\{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \quad \cup \{2\} \times \{\mathrm{AC}x \mid x{\in}\{\mathrm{B},\mathrm{C}\}^*\} \end{pmatrix} \ = \ \widetilde{R}_0$$

$$G_0 \ = \ \{(0,1),(0,2),(0,3),(0,0),(3,3)\} \ = \ \widetilde{G}_0$$

$$R_1 \ = \ \{0,1,2,3\} \times (\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \ = \ \widetilde{R}_1$$

$$G_1 \ = \ \{(1,0),(2,0),(3,0),(0,0),(1,1),(2,2),(3,3)\} \ = \ \widetilde{G}_1$$

The equality between the sets generated by TMR and the sets generated by FQ is striking. We will show that it is not by coincidence, essentially proving

$$(\text{result of FQ } =)\ (\widetilde{R},\widetilde{G}) \ = \ (R,G)\ (=\ \text{result of TMR}). \tag{2}$$

This equality will directly imply Thm. 1.

So let $\preceq$ be the componentwise partial order on $(\mathfrak{P}(\mathrm{Glob}\times\mathrm{Loc}))^n \times (\mathfrak{P}(\mathrm{Glob}^2))^n$:

$$(\hat{R},\hat{G}) \preceq (\hat{R}',\hat{G}') \overset{\text{def}}{\iff} \forall\, t{<}n\colon \hat{R}_t \subseteq \hat{R}'_t \wedge \hat{G}_t \subseteq \hat{G}'_t\,.$$

Intuitively, we prove (2) by separating the equality into two componentwise inclusions: soundness (if a safety property holds according to TMR, then the strongest multithreaded-Cartesian invariant implies this property) and completeness (every safety property implied by the strongest multithreaded-Cartesian invariant can be proven by TMR).

The soundness proof will be conceptually short and the completeness proof a bit more intricate, building on ideas from post-saturation of pushdown systems.

7.1 Soundness: Left Componentwise Inclusion in (2)

The crucial step is showing that the result of TMR is closed under FQ:

$$(\text{result of FQ } =)\ (\widetilde{R},\widetilde{G}) \ \preceq\ (R,G)\ (=\ \text{result of TMR})\,.$$

More precisely, the proof goes by applying FQ once to (R,G), thereby obtaining $(\check{R},\check{G})$, and showing $(\check{R},\check{G}) \preceq (R,G)$ componentwise. Internally, it amounts to checking that elements in $(\check{R},\check{G})$ produced by FQ can also be produced by TMR.

7.2 Completeness: Right Componentswise Inclusion in (2)

For each thread t we define its *operational semantics with FQ-context* as the transition relation of thread t in which the thread can additionally change the shared state according to the guarantees defined by FQ:

$$\overset{\widetilde{G}}{\rightsquigarrow}_t \ :=\ \ \rightsquigarrow_t \cup \{((g,w),(g',w)) \mid w \in \mathrm{Loc} \wedge \exists\, s \in n\backslash\{t\}\colon (g,g') \in \widetilde{G}_s\} \qquad (t<n)\,.$$

Let $\overset{\widetilde{G}}{\rightsquigarrow}{}^*_t$, the *bigstep operational semantics with FQ-context*, be the reflexive-transitive closure of $\overset{\widetilde{G}}{\rightsquigarrow}_t$ on the set of thread states $(t<n)$.

Now we examine the system TMR. We view the relation (edge set) $\to$ defined by TMR as an element of $(\mathfrak{P}(V\times\mathrm{Frame}\times V))^n$ (where $v \xrightarrow[t]{a} v'$ means $(v,a,v') \in {\to}(t)$).

One can obtain G and $\to$ inductively by generating iterates $((\xrightarrow[t]{}_i)_{t<n}, (G_{t,i})_{t<n})$ of the derivation operator of TMR for $i{\in}\mathbb{N}_0$ (the right index i meaning the iterate number). More precisely, we start with empty sets $G_{t,0}$ and $\xrightarrow[t]{}_0$ for all $t{<}n$ and obtain $G_{t,i+1}$ and $\xrightarrow[t]{}_{i+1}$ for all $t{<}n$ by applying the rules of TMR exactly once to $G_{t,i}$ and $\xrightarrow[t]{}_i$ for all $t{<}n$. The described sequence of iterates is ascending, and each element derived by TMR has a derivation tree of some finite depth i:

$$\xrightarrow[t]{} \ =\ \bigcup_{i\in\mathbb{N}_0} \xrightarrow[t]{}_i \qquad \text{and} \qquad G_t \ =\ \bigcup_{i\in\mathbb{N}_0} G_{t,i} \qquad\qquad (t<n).$$

Sloppily speaking, the derivation operator of TMR produces graphs on V, and larger iterates contain larger graphs. Given a walk in an edge set $\underset{t}{\rightarrow}_i$, it in general has some "new" edges not present in the prior iterate $i-1$. Different walks connecting the same pair of nodes and carrying the same word label may have a different number of new edges. We let $\mathrm{tr}(t, i, v, \bar{v}, w)$ be the minimal number of new edges in iterate i in walks labeled by w from v to $\bar{v}$ in the edge set $\underset{t}{\rightarrow}_i$.

After these preparations, we create a connection between FQ and TMR. Informally, we show: (i) stack words accepted by the automata created by TMR, together with the corresponding shared state, lie in the sets defined by FQ; (ii) prefixes of such words correspond to ongoing procedure calls as specified by the bigstep operational semantics with FQ-context; (iii) the shared state changes defined by TMR are also defined by FQ.

These claims are proven together by nested induction on the iterate number (outer induction) and the number of new edges $\mathrm{tr}(\dots)$ (inner induction). Formally, we show:

Lemma 2. *For all $i \in \mathbb{N}_0$ and all $j \in \mathbb{N}_0$ we have:*

(i) $\forall\, g{\in}\mathrm{Glob}, t{\in}n, w{\in}\mathrm{Frame}^*\colon\ \mathrm{tr}(t, i, g, \mathfrak{f}, w) = j \ \Rightarrow\ (g, w) \in \widetilde{R}_t$,

(ii) $\forall g,\bar{g}\in\mathrm{Glob}, t{\in}n, b{\in}\mathrm{Frame}, w{\in}\mathrm{Frame}^*\colon$

$\mathrm{tr}(t, i, g, (\bar{g}, b), w) = j \Rightarrow (\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}_t^* (g, w)$,

(iii) $\forall\, t \in n\colon\ G_{t,i} \subseteq \widetilde{G}_t$.

The formal proof proceeds by double induction on (i, j).

Parts (i) and (iii) directly imply that the result of FQ is closed under TMR:

$$(\text{result of FQ } =)\ (\widetilde{R}, \widetilde{G})\ \succeq\ (R, G)\ (=\ \text{result of TMR})\,.$$

7.3 Combining the Left and Right Inclusions

FQ describes the abstract semantics exactly, whence we obtain:

$$\mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S))) \ =\ \bigcup_{g\in\mathrm{Glob}} \{g\} \times \prod_{t\in n} L_{g,t}\,,$$

where "$\subseteq$" follows from § 7.1, and "$\supseteq$" follows from § 7.2.

8 Conclusion

We considered the multithreaded-Cartesian approximation, which is a succinct description of the accuracy of the thread-modular approaches of Owicki and Gries, C. Jones, and Flanagan and Qadeer (without auxiliary variables). We applied it to multithreaded programs with recursion, presenting an algorithm for discovering a representation of the multithreaded-Cartesian collecting semantics. The algorithm creates a finite automaton whose language coincides with the multithreaded-Cartesian collecting semantics. In particular, the involved inductive invariant is shown to be a regular language. The algorithm uses ideas from

a seminal algorithm of Flanagan and Qadeer and works in time $\mathcal{O}(n \log_2 n)$ in the number of threads n and polynomial in other quantities. We remark that, in contrast, the model-checking problem (without abstraction) is known to be undecidable.

While multithreaded programs with recursion occur rarely in practice, the models may contain both concurrency and recursion [8]. For example, in certain cases it is possible to model integer variables as stacks. But even for multithreaded programs whose procedures are nonrecursive, our algorithm TMR offers compact representation of stack contents, which depends only on the number of threads as well as on the sizes of shared states and frames, but not on the stack depth. A useful consequence of equivalence between FQ and TMR is that one may choose inlining procedures or creating an automaton depending on the costs of constructing and running an analysis, well knowing that its precision will not change. This opens way to potential time and space savings without changing the strength of an analysis.

Acknowledgment. Besides Neil Deaton Jones, I acknowledge comments and suggestions from Laurent Mauborgne and Xiuna Zhu. Part of this work was executed while at IMDEA Software, Spain. I also acknowledge the financial support of the projects of the German Federal Ministry for Education and Research, IDs 01IS11035 and 01IS13020. Furthermore, I acknowledge additional typesetting support of Springer-Verlag.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Symposium on Theory of Computing, pp. 202–211. ACM (2004)
2. Andersen, N., Jones, N.D.: Generalizing Cook's transformation to imperative stack programs. In: Karhumäki, J., Rozenberg, G., Maurer, H.A. (eds.) Results and Trends in Theoretical Computer Science. LNCS, vol. 812, pp. 1–18. Springer, Heidelberg (1994)
3. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of Multi-pushdown Automata Is 2ETIME-Complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
4. Bouajjani, A., Esparza, J., Touili, T.: A generic approach to the static analysis of concurrent programs with procedures. International Journal of Foundations of Computer Science **14**(4), 551–582 (2003)
5. Bozzelli, L., La Torre, S., Peron, A.: Verification of well-formed communicating recursive state machines. Theoretical Computer Science **203**(2-3), 382–405 (2008)
6. Büchi, J.R.: Regular canonical systems. Archiv für mathematische Logik und Grundlagenforschung 6, 91–111 (1962/1963)
7. Burkart, O., Steffen, B.: Pushdown processes: parallel composition and model checking. In: Jonsson, B., Parrow, J. (eds.) CONCUR 1994. LNCS, vol. 836, pp. 98–113. Springer, Heidelberg (1994)
8. Chaki, S., Clarke, E., Kidd, N., Reps, T., Touili, T.: Verifying Concurrent Message-Passing C Programs with Recursive Calls. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 334–349. Springer, Heidelberg (2006)

9. Cousot, R.: Fondements des méthodes de preuve d'invariance et de fatalité de programmes paralléles. Ph.D. thesis, Institut national polytechnique de Lorraine, pp. 4–118(4–119), pp. 4–120, 1985. §4.3.2.4.3
10. Flanagan, C., Qadeer, S.: Thread-Modular Model Checking. In: Ball, T., Rajamani, S.K. (eds.) SPIN 2003. LNCS, vol. 2648, pp. 213–224. Springer, Heidelberg (2003)
11. Hansen, T.A., Nikolajsen, T., Träff, J.L., Jones, N.D.: Experiments with implementations of two theoretical constructions. In: Meyer, A.R., Taitslin, M.A. (eds.) Logic at Botik 1989. LNCS, vol. 363, pp. 119–133. Springer, Heidelberg (1989)
12. Jones, C. B.: Tentative steps toward a development method for interfering programs. ACM Trans. Program. Lang. Syst. 5(4), 596–619 (1983)
13. La Torre, S., Napoli, M., Parlato, G.: A Unifying Approach for Multistack Pushdown Automata. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 377–389. Springer, Heidelberg (2014)
14. Malkis, A.: Cartesian abstraction and verification of multithreaded programs. Ph.D. thesis, Albert-Ludwigs-Universität Freiburg (2010)
15. Malkis, A.: Multithreaded-Cartesian abstract interpretation of multithreaded recursive programsis polynomial. Technical report (2015). http://www4.in.tum.de/malkis/Malkis-MultCartAbstIntOfMultRecProgIsPoly_techrep.pdf
16. Malkis, A., Podelski, A., Rybalchenko, A.: Thread-Modular Verification Is Cartesian Abstract Interpretation. In: Barkaoui, K., Cavalcanti, A., Cerone, A. (eds.) ICTAC 2006. LNCS, vol. 4281, pp. 183–197. Springer, Heidelberg (2006)
17. Mehlhorn, K.: Data structures and algorithms 1: sorting and searching. In: EATCS Monographs in Theoretical Computer Science, vol. 1. Springer (1984)
18. Mogensen, T.Æ.: WORM-2DPDAs: An extension to 2DPDAs that can be simulated in linear time. Inf. Process. Lett. **52**(1), 15–22 (1994)
19. Nielson, F., Nielson, R.H., Seidl, H.: Normalizable Horn Clauses, Strongly Recognizable Relations, and Spi. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 20–35. Springer, Heidelberg (2002)
20. Owicki, S.S.: Axiomatic proof techniques for parallel programs. Ph.D. thesis, Cornell University, department of computer science, TR 75–251, July 1975
21. Qadeer, S., Rajamani, S.K., Rehof, J.: Summarizing procedures in concurrent programs. In: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 245–255. ACM, New York (2004)
22. Qadeer, S., Rehof, J.: Context-Bounded Model Checking of Concurrent Software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
23. Ramalingam, G.: Context-sensitive synchronization-sensitive analysis is undecidable. ACM Trans. Program. Lang. Syst. **22**(2), 416–430 (2000)
24. Schwoon, S.: Model-checking pushdown systems. Ph.D. thesis, Technische Universität München, June 2002

Over-Approximating Terms Reachable by Context-Sensitive Rewriting

Nirina Andrianarivelo and Pierre Réty(✉)

LIFO - Université d'Orléans, B.P. 6759, 45067 Orléans Cedex 2, France
{Nirina.Andrianarivelo,Pierre.Rety}@univ-orleans.fr

Abstract. For any left-linear context-sensitive term rewrite system and any regular language of ground terms I, we build a finite tree automaton that recognizes a superset of the descendants of I, i.e. of the terms reachable from I by context-sensitive rewriting.

1 Introduction

There is an increasing need for reliable methods to check security protocols and computer programs (see [4,5] for a survey). Such verification problems can often be encoded with rewrite rules, and reduced to reachability problems [3]. Given a set of rewrite rules R, a set of ground terms I, and a set of undesirable ground terms BAD, it consists in computing the set (denoted $R^*(I)$) of ground terms that are reachable from I by R, and checking that $R^*(I) \cap BAD = \emptyset$.

Example 1. Let $R = \{f(x) \to f(f(x)),\ a \to b,\ c \to a\}$, $I = \{f(a)\}$, $BAD = \{c\}$. Then $R^*(I) = \{f^+(a), f^+(b)\}$, where f^+ denotes several occurrences of f (at least one). Here the elements of BAD are not reachable, i.e. $R^*(I) \cap BAD = \emptyset$.

Several methods have been proposed to compute $R^*(I)$ exactly, or to over-approximate it (see [11] for a survey). However, ordinary term rewriting is not always powerful enough. Indeed, the operational semantics of functional programs can be expressed using Context-Sensitive Term Rewrite Systems (CS-TRS), as described in [9,16]. In this framework, a set of argument numbers $\mu(f)$ is associated to each function symbol f, which indicates the arguments of f allowed to be reduced by rewriting. For instance, consider Example 1 again and let $\mu(f) = \emptyset$. In this case, for each term of the form $t = f(\ldots)$, it is forbidden to rewrite the strict subterms of t. Thus, using context-sensitive rewriting, $R^*_\mu(I) = \{f^+(a)\}$. Now consider that the set of undesirable terms is $BAD' = \{f^+(b)\}$, then $R^*_\mu(I) \cap BAD' = \emptyset$ whereas $R^*(I) \cap BAD' \neq \emptyset$.

In this paper, we compute an over-approximation (say App) of $R^*_\mu(I)$, using a finite tree automaton (i.e. a regular language), assuming that R is left-linear and I is regular. Thus, if $App \cap BAD = \emptyset$, we are sure that $R^*_\mu(I) \cap BAD = \emptyset$. Our work is both an extension of [15], where $R^*_\mu(I)$ is computed in an exact way assuming stronger restrictions (R is linear and right-shallow[1]), and an extension of the completion of tree automata [10], in order to take μ into account and to avoid computing descendants forbidden by μ (as much as possible).

[1] I.e. in the rewrite rule right-hand-sides, every variable occurs at depth at most 1.

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 128–139, 2015.
DOI: 10.1007/978-3-319-24537-9_12

Let us outline the main idea, using Example 1 again. Roughly[2]:

1. Our starting point is composed of the tree automaton $\mathcal{A} = (\Sigma, Q, Q_f, \Delta)$ s.t. $\Sigma = \{f, a, b, c\}$, $Q = \{q_a, q\}$ (states), $Q_f = \{q\}$ (final state), $\Delta = \{a \to q_a,\ f(q_a) \to q\}$ (transitions), which recognizes $I = \{f(a)\}$, and the rewrite system $R = \{f(x) \to f(f(x)),\ a \to b,\ c \to a\}$.
2. **Initialization.** We transform Δ by using the fact that $\mu(f) = \emptyset$. For this, we *mark* positions forbidden by μ, using a *a prime mark (priming)*. A prime means that no rewrite step should be applied at this position. No prime means that a rewrite step (if any) is allowed. Thus we replace the transition $f(q_a) \to q$ by $f(q'_a) \to q$, and add the transition $a \to q'_a$ so that the language recognized by the automaton is unchanged.
   Now $\Delta = \{a \to q_a,\ a \to q'_a,\ f(q'_a) \to q\}$, and we create $Q' = \{q'_a\}$.
3. **Completion.** To get descendants, we compute (so-called) critical pairs between transitions $u \to s \in \Delta$ and rewrite rules of R, only if $s \in Q$. If $s \in Q'$, i.e. s has a prime, no critical pair is computed since no rewrite step is allowed at this position.
   Computing a critical pair between $a \to q_a$ and $a \to b$ generates the transition $b \to q_a$, which is added into Δ.
4. Computing a critical pair between $f(q'_a) \to q$ and $f(x) \to f(f(x))$ generates the transition $f(f(q'_a)) \to q$. However, $f(f(q'_a))$ is not shallow. Then we *normalize*[3] this transition into $f(q'_1) \to q, f(q'_a) \to q'_1$, which are added into Δ. The new state q'_1 has a prime since it occurs at a position forbidden by μ. Now the automaton also recognizes $f(f(a))$, and does not recognize $f(f(b))$, which is not a context-sensitive descendant.
5. There are some more critical pairs, which add some more transitions into Δ. Finally, the completion process stops with $\Delta = \{a \to q_a,\ a \to q'_a,\ f(q'_a) \to q,\ b \to q_a,\ f(q'_1) \to q, f(q'_a) \to q'_1,\ f(q'_1) \to q'_1\}$ Now the current automaton recognizes $\{f^+(a)\}$, i.e. $R^*_\mu(I)$, and does not recognize the elements of $\{f^+(b)\}$, which are not context-sensitive descendants.

The paper is organized as follows. The formal preliminary notions are given in Section 2. Our method for over-approximating context-sensitive descendants is detailed in Section 3, and full examples are given in Section 4. Then Section 5 speaks about related work, and some ideas for further work are discussed in Section 6. All proofs are in [1].

2 Preliminaries

Consider a *finite ranked alphabet* Σ and a set of variables *Var*. Each symbol $f \in \Sigma$ has a unique arity, denoted by $ar(f)$. The notions of *first-order term*, *position* and

[2] Some unnecessary transitions of the current automaton are missing.

[3] It consists in *flattening* the left-hand-side of the transition by using intermediate states.

substitution are defined as usual. Given σ and σ' two substitutions, $\sigma \circ \sigma'$ denotes the substitution such that for any variable x, $\sigma \circ \sigma'(x) = \sigma(\sigma'(x))$. T_Σ denotes the set of ground terms (without variables) over Σ. For a term t, $Var(t)$ is the set of variables of t, $Pos(t)$ is the set of positions of t, and ϵ is the root position. For $p \in Pos(t)$, $t(p)$ is the symbol of $\Sigma \cup Var$ occurring at position p in t, and $t|_p$ is the subterm of t at position p. For $p, p' \in Pos(t)$, $p < p'$ means that p occurs in t strictly above p'. The term t is *linear* if each variable of t occurs only once in t. The term $t[t']_p$ is obtained from t by replacing the subterm at position p by t'.

A *rewrite rule* is an oriented pair of terms, written $l \to r$. We always assume that l is not a variable, and $Var(r) \subseteq Var(l)$. A *rewrite system* R is a finite set of rewrite rules. *lhs* stands for left-hand-side, *rhs* for right-hand-side. The *rewrite relation* $\to_R$ is defined as follows: $t \to_R t'$ if there exist a non-variable position $p \in Pos(t)$, a rule $l \to r \in R$, and a substitution θ s.t. $t|_p = \theta(l)$ and $t' = t[\theta(r)]_p$ (also denoted $t \to_R^p t'$). $\to_R^*$ denotes the reflexive-transitive closure of $\to_R$. t' is a *descendant* of t if $t \to_R^* t'$. If I is a set of ground terms, $R^*(I)$ denotes the set of descendants of elements of I. The rewrite rule $l \to r$ is *left (resp. right) linear* if l (resp. r) is linear. R is *left (resp. right) linear* if all its rewrite rules are left (resp. right) linear. R is *linear* if R is both left and right linear. $l \to r$ is *right-shallow* if r is *shallow*, i.e. every variable of r occurs at depth at most 1.

A *context-sensitive rewrite relation* is a sub-relation of the ordinary rewrite relation in which rewritable positions are indicated by specifying arguments of function symbols. A mapping $\mu : \Sigma \to P(\mathbb{N})$ is said to be a *replacement map* (or Σ-map) if $\mu(f) \subseteq \{1, ..., ar(f)\}$ for all $f \in \Sigma$. A *context-sensitive* term rewriting system (CS-TRS) is a pair $\mathcal{R} = (R, \mu)$ composed of a TRS and a replacement map. The set of μ-replacing positions[4] $Pos^\mu(t)$ ($\subseteq Pos(t)$) is recursively defined: $Pos^\mu(t) = \{\epsilon\}$ if t is a constant or a variable, otherwise $Pos^\mu(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \{i.p \mid i \in \mu(f), p \in Pos^\mu(t_i)\}$. The rewrite relation induced by a CS-TRS $\mathcal{R}$ is defined: $t \hookrightarrow_\mathcal{R} t'$ if $t \to_R^p t'$ for some $p \in Pos^\mu(t)$. The set of descendants of I by context-rewriting according to the CS-TRS $\mathcal{R} = (R, \mu)$ is denoted $R_\mu^*(I)$.

A (bottom-up) finite tree *automaton* is a quadruple $\mathcal{A} = (\Sigma, Q, Q_f, \Delta)$ where Q is the set of states, $Q_f \subseteq Q$ is the set of final states, and Δ is a set of *transitions* of the form $t \to q$ where $t \in T_{\Sigma \cup Q}$ and $q \in Q$. A transition is *normalized* if it is of the form $f(q_1, \ldots, q_n) \to q$ where $f \in \Sigma$ and $q_1, \ldots, q_n, q \in Q$, or of the form $q_1 \to q$ (*empty transition*, also called *epsilon transition*). $\mathcal{A}$ is normalized if all transitions in Δ are normalized. Sets of *states* will be denoted by letters Q, S, D, and states by q, s, d.

The rewrite relation induced by Δ is denoted by $\to_\Delta$ or $\to_\mathcal{A}$. A ground term t is *recognized* by $\mathcal{A}$ into q if $t \to_\Delta^* q$. Let $L(\mathcal{A}, q) = \{t \in T_\Sigma \mid t \to_\Delta^* q\}$. The language recognized by $\mathcal{A}$ is $L(\mathcal{A}) = \cup_{q \in Q_f} L(\mathcal{A}, q)$. A set I of ground terms is regular if there exists a finite automaton $\mathcal{A}$ s.t. $I = L(\mathcal{A})$. A *Q-substitution* σ is a substitution s.t. $\forall x \in Dom(\sigma)$, $\sigma(x) \in Q$.

[4] Also called positions allowed by μ.

3 Computing Context-Sensitive Descendants

3.1 Closure Under Context-Sensitive Rewriting

The main idea is: given a context-sensitive rewrite system (R, μ), we consider a set of states Q to recognize subterms at positions allowed by μ (i.e. rewritable positions), and another set Q' for those forbidden by μ. To compute context-sensitive descendants, rewrite steps will be applied to (sub)-terms recognized into states of Q, and not on those recognized into states of Q'.

Definition 1. A *context-sensitive automaton* $\mathcal{A} = (\Sigma, Q \cup Q', Q_f, \Delta, rm')$ is composed of a tree automaton and a mapping rm' such that $Q \cap Q' = \emptyset$, $Q_f \subseteq Q$, and $rm' : Q' \to Q$ is an injective mapping. rm' stands for 'remove primes'.

We will often use $q, q_1, q_2, \ldots$ for elements of Q, and $q', q'_1, q'_2, \ldots$ for elements Q', and we will write $rm'(q') = q$ and $rm'(q'_i) = q_i$.

Definition 2. We extend rm' to terms, so that $rm' : T_{\Sigma \cup Q \cup Q'} \to T_{\Sigma \cup Q \cup Q'}$, by:

- $rm'(q) = q$ if $q \in Q$,
- and $rm'(f(t'_1, \ldots, t'_n)) = f(t_1, \ldots, t_n)$ such that $\forall i \begin{vmatrix} t_i = rm'(t'_i) \text{ if } i \in \mu(f) \\ t_i = t'_i \text{ otherwise} \end{vmatrix}$

Note that rm' does not remove all primes. Actually, rm' removes primes (if any) from states occurring at positions allowed by μ, so that rewrite steps are computed. For example, if $\mu(f) = \{1\}$, then $rm'(f(q'_1, q'_2)) = f(q_1, q'_2)$.

For computing context-sensitive descendants, a context-sensitive automaton should satisfy a compatibility property (with μ).

Definition 3. Let $\mathcal{A} = (\Sigma, Q \cup Q', Q_f, \Delta, rm')$ be a context-sensitive automaton. $\mathcal{A}$ is μ-compatible if $\forall (t \to s) \in \Delta$, $(rm'(t) \to rm'(s)) \in \Delta$.

Example 2. let $Q = \{q_a, q_f\}$, $Q' = \{q'_a\}$, $Q_f = \{q_f\}$, $\Delta = \{a \to q'_a,\ f(q'_a) \to q_f\}$, and assume that $rm'(q'_a) = q_a$ and $\mu(f) = \{1\}$. This automaton is not μ-compatible because $a \to q_a$ and $f(q_a) \to q_f$ are missing in Δ.

Lemma 1. *If $\mathcal{A}$ is μ-compatible,*
$\forall t \in T_{\Sigma \cup Q \cup Q'}, \forall s \in Q \cup Q',\ (t \to^*_\Delta s \implies rm'(t) \to^*_\Delta rm'(s))$

The notion of critical pair is at the heart of the technique. A critical pair is a way to detect a possible rewrite step issued from a term $t \in L(\mathcal{A}, q)$, by a rewrite rule $l \to r$. To check that this rewrite step is allowed by μ, we suppose that $q \in Q$, i.e. $q \notin Q'$. A convergent critical pair means that the rewrite step is already handled i.e. if $t \to_{l \to r} s$ then $s \in L(\mathcal{A}, q)$. Consequently, the language of a normalized automaton having only convergent critical pairs is closed under rewriting.

Definition 4. Let $l \to r$ be a rewrite rule and σ be a $(Q \cup Q')$-substitution such that $\sigma l \to^*_\Delta q$ and $q \in Q$. Then $(rm'(\sigma r), q)$ is called *critical-pair* (CP for short). The critical pair is said *convergent* if $rm'(\sigma r) \to^*_\Delta q$.

Example 3. Consider Example 2 again, and the rewrite rule $f(x) \to g(x)$ with $\mu(g) = \{1\}$. Then $(g(q_a), q_f)$ is a critical pair, which is not convergent. Note that $L(\mathcal{A})$ is not closed by context-sensitive rewriting since $f(a) \in L(\mathcal{A})$ whereas $g(a) \notin L(\mathcal{A})$.

The use of rm' in Definition 4 is crucial if a position forbidden by μ becomes allowed after a rewrite step. For instance, consider the rewrite system $\{h(x) \to i(x), c \to d\}$ with $\mu(h) = \emptyset$ and $\mu(i) = \{1\}$. Then $h(c) \to i(c) \to i(d)$ whereas $\neg(h(c) \to h(d))$. So, within $h(c)$, c should be recognized into a state of Q' (say q'_c), whereas within $i(c)$, c should be recognized into a state of Q (say q_c). The migration of q'_c into q_c is achieved thanks to rm'.

To get closure under context-sensitive rewriting, the automaton should be μ-compatible to take μ into account, and normalized. Indeed, if it is not normalized, we may have for example $h(\sigma l) \to^*_\Delta q$ whereas $\neg(\exists q_1 \in Q, \sigma l \to^*_\Delta q_1)$, i.e. there is no critical pair to take the rewrite step by $l \to r$ into account.

Theorem 1. *Let (R, μ) be a left-linear context-sensitive rewrite system, and $\mathcal{A}$ be a μ-compatible normalized automaton.*
If all critical pairs are convergent, then $L(\mathcal{A})$ is closed by context-sensitive rewriting, i.e. $(t \in L(\mathcal{A}) \wedge t \hookrightarrow^*_{(R,\mu)} t') \Longrightarrow t' \in L(\mathcal{A})$.

Example 4. Consider Example 2 again, and the rewrite rule $a \to b$. All critical pairs are convergent since there are no critical pairs. However $f(a) \in L(\mathcal{A})$ and $f(a) \hookrightarrow_{(R,\mu)} f(b)$, whereas $f(b) \notin L(\mathcal{A})$. This comes from the fact that $\mathcal{A}$ is not μ-compatible.
Now, if Δ is replaced by $\Delta' = \{a \to q'_a, a \to q_a, b \to q_a, f(q_a) \to q_f\}$, the automaton is μ-compatible. Considering the rewrite rule $a \to b$, there is one critical pair : (b, q_a), which is convergent. Thus $f(a) \in L(\mathcal{A})$, $f(a) \hookrightarrow_{(R,\mu)} f(b)$, and $f(b) \in L(\mathcal{A})$.

3.2 Normalization

Consider a non-convergent critical pair (t, q). If we add the transition $t \to q$ into Δ, the critical pair becomes convergent. Unfortunately, the transition $t \to q$ is not necessarily normalized.

Example 5. Consider $R = \{f(x) \to g(h(x)), a \to b\}$, $\mu(f) = \{1\}$, $\mu(g) = \emptyset$, $\mu(h) = \{1\}$, and an automaton defined by $Q = \{q_a, q_f\}$, $Q_f = \{q_f\}$, and $\Delta = \{a \to q_a, f(q_a) \to q_f\}$. Note that $L(\mathcal{A}) = \{f(a)\}$. From the transition $f(q_a) \to q_f$ and the rewrite rule $f(x) \to g(h(x))$, we get the critical pair $(g(h(q_a)), q_f)$. The corresponding transition $g(h(q_a)) \to q_f$ is not normalized.

To get closure under rewriting, all transitions should be normalized. We give an algorithm to transform a pair (t, s) into normalized transitions. Note that if t is a state, the algorithm will return empty transitions (which are normalized).

Input : a pair (t, s) s.t $t \in T_{\Sigma \cup Q \cup Q'}$ and $s \in Q \cup Q'$
Output : a set of normalized transitions

function $\mathsf{Norm}_{\mathcal{A}}(t, s)$

1. If the transition $t \to s$ is normalized, return $\{t \to s\} \cup \{rm'(t) \to rm'(s)\}$
2. else let $t = f(t_1, \ldots, t_n)$, and $J = \{j \in \{1, \ldots, n\} \mid t_j \notin Q \cup Q'\}$
   2.1. for each $i \in \{1, \ldots, n\}$, let s_i be a state defined by
      2.1.1. if $t_i \in Q \cup Q'$ then $s_i = t_i$
      2.1.2. else
         i) if $s \in Q$ and $i \in \mu(f)$
         ii) then either choose $s_i \in Q$, or s_i is a new state and add s_i to Q
         iii) else either choose $s_i \in Q'$, or s_i (and q_i) are new states s.t. $rm'(s_i) = q_i$ and add s_i to Q' (and q_i to Q)
   2.2. return
   $\{f(s_1, \ldots, s_n) \to s\} \cup \{rm'(f(s_1, \ldots, s_n)) \to rm'(s))\} \cup \{\cup_{j \in J} \mathsf{Norm}_{\mathcal{A}}(t_j, s_j)\}$

In the previous algorithm, whenever a transition is generated, a transition obtained by applying rm' on both sides is also generated. This is for preserving the μ-compatibility of the automaton. On the other hand, the non-determinism of the algorithm (Items ii and iii) is "don't care", i.e. only one choice has to be achieved. For any choice, the normalization algorithm terminates. However, introducing new states may create new critical pairs, whose normalization may also create new states and new critical pairs, and the global completion process may not terminate. This is why choosing s_i among the existing states is sometimes necessary to make completion terminate. But it may lead to a strict over-approximation of the descendants.

Example 6. Consider Example 5 again with the critical pair $(g(h(q_a)), q_f)$. Recall that $\mu(g) = \emptyset$, $\mu(h) = \{1\}$.
Running $\mathsf{Norm}_{\mathcal{A}}(g(h(q_a)), q_f)$ goes through the case iii), and two new states q'_1, q_1 are created s.t. $rm'(q'_1) = q_1$. Moreover q'_1 is added to Q' whereas q_1 is added to Q. Then $\mathsf{Norm}_{\mathcal{A}}(g(h(q_a)), q_f)$ returns (note that $rm'(g(q'_1)) = g(q'_1)$):

$$\{g(q'_1) \to q_f\} \cup \{g(q'_1) \to q_f\} \cup \mathsf{Norm}_{\mathcal{A}}(h(q_a), q'_1)$$

Since the transition $h(q_a) \to q'_1$ is already normalized, $\mathsf{Norm}_{\mathcal{A}}(h(q_a), q'_1)$ returns:

$$\{h(q_a) \to q'_1\} \cup \{h(q_a) \to q_1\}$$

Finally we get the set of transitions $\{g(q'_1) \to q_f, h(q_a) \to q'_1, h(q_a) \to q_1\}$.

Lemma 2. *$t \to^*_{\mathsf{Norm}_{\mathcal{A}}(t,s)} s$, i.e. the pair (t, s) is convergent.*

3.3 Initialization

As in [15], we first introduce non-final states and transitions to recognize the ground subterms of the rewrite rule right-hand-sides. For a term t, let $PosG(t) = \{p \in Pos(t) \mid p \neq \epsilon \wedge Var(t|_p) = \emptyset\}$. Let $PosGout(t)$ be the outermost elements of $PosG(t)$, i.e. $PosGout(t) = \{p \in PosG(t) \mid \neg(\exists p' \in PosG(t), p' < p)\}$.

Given (R, μ), we introduce the set of states $Q_R = \{q_{r,p} \mid l \to r \in R \wedge p \in PosG(r)\}$ and $Q'_R = \{q'_{r,p} \mid l \to r \in R \wedge p \in PosG(r)\}$ s.t. $rm'(q'_{r,p}) = q_{r,p}$, and the transitions $\Delta_R = \cup_{l \to r \in R} \cup_{p \in PosG(r)} \{r(p)(q'_{r,p.1}, \ldots, q'_{r,p.n}) \to q'_{r,p}, rm'(r(p)(q'_{r,p.1}, \ldots, q'_{r,p.n})) \to q_{r,p}\}$. Note that the transitions with rm' are for ensuring μ-compatibility.

From a normalized automaton $\mathcal{A}_0 = (\Sigma, Q_0, Q_f, \Delta_0)$ and a left-linear context-sensitive rewrite system (R, μ), a $\mu-$compatible normalized context-sensitive automaton $\mathcal{A} = (\Sigma, Q \cup Q', Q_f, \Delta, rm')$ that recognizes the same language as $\mathcal{A}_0$, is built as follows:

function $\mathsf{Init}_{(R,\mu)}(\mathcal{A}_0)$

1. for each $q \in Q_0$, a new state q' (also denoted $add'(q)$) is created, and let $rm'(q') = q$
2. extend add' to trees of $T_{\Sigma \cup Q_0}$: $add'(f(t_1, \ldots, t_n)) = f(add'(t_1), \ldots, add'(t_n))$
3. let $Q = Q_0 \cup Q_R$ and $Q' = \{add'(q) \mid q \in Q_0\} \cup Q'_R$
4. let $\Delta = \cup_{(t \to q) \in \Delta_0}(\{add'(t) \to add'(q)\} \cup \{rm'(add'(t)) \to q\}) \cup \Delta_R$
5. return $\mathcal{A} = (\Sigma, Q \cup Q', Q_f, \Delta, rm')$

In Step 4, $\{rm'(add'(t)) \to q\}$ is for ensuring μ-compatibility (note that $q = rm'(add'(q))$).

Example 7. Let $R = \{f(x) \to g(x)\}$ and $\mu(f) = \emptyset$, $\mu(h) = \{1\}$. Note that $Q_R = Q'_R = \Delta_R = \emptyset$.
Let $\mathcal{A}_0$ s.t. $Q_0 = \{q_a, q_f\}$, $Q_f = \{q_f\}$, $\Delta_0 = \{a \to q_a, f(q_a) \to q_f, h(q_a) \to q_f\}$. The language recognized by $\mathcal{A}_0$ is $L(\mathcal{A}_0) = \{f(a), h(a)\}$.
Then $\mathsf{Init}_{(R,\mu)}(\mathcal{A}_0)$ returns the automaton $\mathcal{A}$ s.t. $Q = \{q_a, q_f\}$, $Q' = \{q'_a, q'_f\}$, $rm'(q'_a) = q_a$, $rm'(q'_f) = q_f$, and $\Delta = \{a \to q'_a, a \to q_a, f(q'_a) \to q'_f, f(q'_a) \to q_f, h(q'_a) \to q'_f, h(q_a) \to q_f\}$. Note that $L(\mathcal{A}) = \{f(a), h(a)\} = L(\mathcal{A}_0)$.

Lemma 3. *$\mathcal{A}$ is μ-compatible and $L(\mathcal{A}_0) \subseteq L(\mathcal{A})$.*

Lemma 4. *$\forall t \in T_\Sigma, \forall s \in (Q \cup Q') \backslash (Q_R \cup Q'_R)$, $(t \to^*_\Delta s \implies t \to^*_{\Delta_0} rm'(s))$. Consequently $L(\mathcal{A}) \subseteq L(\mathcal{A}_0)$.*

3.4 Simplification

Roughly, the simplification step consists in replacing each outermost ground subterm of a given rewrite rule right-hand-side r, by its corresponding state in Q_R or Q'_R. Actually, a simplification step simplifies a critical pair.

function $\mathsf{Simplify}(rm'(\sigma r), q)$

1. let us write $PosGout(r) = \{p_1, \ldots, p_n\}$
2. then return $(rm'(\sigma(r)[q'_{r,p_1}]_{p_1} \cdots [q'_{r,p_n}]_{p_n}),\ q)$

Example 8. $R = \{h(x) \to r = f(x, g(a))\}$, $\mu(f) = \{1, 2\}$, $\mu(g) = \{1\}$. The initialization gives $\Delta_R = \{a \to q'_{r,2.1}, a \to q_{r,2.1}, g(q'_{r,2.1}) \to q'_{r,2}, g(q_{r,2.1}) \to q_{r,2}\}$. Let $\sigma = (x/q_1)$. So $\mathsf{Simplify}(\sigma(r), q) = \mathsf{Simplify}(f(q_1, g(a)), q)$ returns the pair $(f(q_1, q_{r,2}), q)$, and one has $g(a) \to^*_{\Delta_R} q_{r,2}$. Note that the corresponding transition $f(q_1, q_{r,2}) \to q$ is normalized.

More generally, if r is shallow and let $(t, q) = \mathsf{Simplify}(rm'(\sigma r), q)$, then the transition $t \to q$ is normalized. On the other hand, if $PosGout(r) = \emptyset$, then $\mathsf{Simplify}(rm'(\sigma r), q) = (rm'(\sigma r), q)$.

3.5 Reduction

Le (t, s) be a pair, whose corresponding transition $t \to s$ is not normalized, and suppose that t is reducible into u by non-empty transitions of the automaton. Since the size of u is less than the size of t, it is easier to normalize the pair (u, s) instead of (t, s). The replacement of (t, s) by (u, s) is called *reduction*.
function $\mathsf{Reduce}_{\mathcal{A}}(t, s)$

1. if $t \to s$ is not normalized
   and there exists a non-empty transition $(t_1 \to s_1) \in \Delta$ s.t. $t \to_{[p,\, t_1 \to s_1]} u$
   and $[(p \in Pos^{\mu}(t) \wedge s_1 \in Q) \vee (p \notin Pos^{\mu}(t) \wedge s_1 \in Q')]$
2. then return $\mathsf{Reduce}_{\mathcal{A}}(u, s)$
3. else return (t, s)

In Step 1, if there exist several transitions like $t_1 \to s_1$ that allow to reduce t, then one of them is chosen arbitrarily.

Example 9. $\Delta = \{s(q_1) \to q_2,\ g(q_2, q_3) \to q_4\}$, $\mu(f) = \mu(s) = \{1\}$, $\mu(g) = \{1, 2\}$. Then $\mathsf{Reduce}_{\mathcal{A}}(f(g(s(q_1), q_3)), q) = (f(q_4), q)$.

3.6 Completion

The main algorithm of our method is presented here.

Input: a normalized automaton $\mathcal{A}_0 = (\Sigma, Q_0, Q_f, \Delta_0)$ and a left-linear context-sensitive rewrite system (R, μ).
Output: a context-sensitive automaton $\mathcal{A}$ such that $R^*_{\mu}(L(\mathcal{A}_0)) \subseteq L(\mathcal{A})$.

The main two steps of the algorithm are:

1. $\mathcal{A} = \mathsf{Init}_{(R,\mu)}(\mathcal{A}_0)$. Let us write $\mathcal{A} = (\Sigma, Q \cup Q', Q_f, \Delta, rm')$.
2. while there exists a non-convergent critical pair (cpl, cpr) in $\mathcal{A}$ do
   2.1. $\Delta = \Delta \cup \mathsf{Norm}_{\mathcal{A}}(\mathsf{Reduce}_{\mathcal{A}}(\mathsf{Simplify}(cpl, cpr)))$

Theorem 2. *Let (R, μ) be a left-linear context-sensitive rewrite system, and $\mathcal{A}_0$ be a normalized automaton. When the algorithm stops, $L(\mathcal{A})$ is closed by context-sensitive rewriting and $R^*_{\mu}(L(\mathcal{A}_0)) \subseteq L(\mathcal{A})$.*

Note that it is always possible to make completion terminate, for example by fixing a bound for the number of states. And if this bound is reached, $\mathsf{Norm}_{\mathcal{A}}$ should re-use existing states instead of creating new ones.

However, if the rewrite system is right-shallow, the transition obtained after applying $\mathsf{Simplify}$ is already normalized (see Section 3.4). Then $\mathsf{Reduce}_{\mathcal{A}}$ and $\mathsf{Norm}_{\mathcal{A}}$ do nothing, and no new states are introduced. Therefore, the completion algorithm will stop and generate an automaton similar to that of [15]. Consequently, using the result of [15] we get:

Corollary 1. *If the context-sensitive rewrite system is linear and right-shallow, then the completion algorithm stops and generates the context-sensitive descendants in an exact way.*

4 Examples

The following example shows the role of the states with primes, and of the simplification step.

Example 10. $R = \{h(x) \to r = f(x, h(b))\}$, $\mu(h) = \mu(f) = \mu(s) = \emptyset$.
Let $\mathcal{A}_0$ be the automaton defined by $Q_0 = \{q_a, q_f\}$, $Q_f = \{q_f\}$, and $\Delta_0 = \{a \to q_a,\ s(q_a) \to q_a,\ h(q_a) \to q_f\}$. Note that $L(\mathcal{A}_0) = \{h(s^n(a)) \mid n \in \mathbb{N}\}$ and $R^*_\mu(L(\mathcal{A}_0)) = \{h(s^n(a)),\ f(s^n(a), h(b)) \mid n \in \mathbb{N}\}$ where s^n denotes n occurrences of s.

The initialization step gives:
$Q_R = \{q_{r,2}, q_{r,2.1}\}$, $Q'_R = \{q'_{r,2}, q'_{r,2.1}\}$, $rm'(q'_{r,2.1}) = q_{r,2.1}$, $rm'(q'_{r,2}) = q_{r,2}$,
$\Delta_R = \{b \to q'_{r,2.1},\ b \to q_{r,2.1},\ h(q'_{r,2.1}) \to q'_{r,2},\ h(q'_{r,2.1}) \to q_{r,2}\}$,
$Q' = \{q'_a, q'_f\}$, $rm'(q'_a) = q_a$, $rm'(q'_f) = q_f$,
$\Delta = \{a \to q'_a,\ a \to q_a,\ s(q'_a) \to q'_a,\ s(q'_a) \to q_a,\ h(q'_a) \to q'_f,\ h(q'_a) \to q_f\} \cup \Delta_R$.
With $h(q'_a) \to q_f$ and the rewrite rule, we get the critical pair $(f(q'_a, h(b)), q_f)$. Then Simplify $(f(q'_a, h(b)), q_f) = (f(q'_a, q'_{r,2}), q_f)$, and the normalization will add the transition $f(q'_a, q'_{r,2}) \to q_f$ to Δ.
$h(q'_{r,2.1}) \to q_{r,2}$ and the rewrite rule generate the critical pair $(f(q'_{r,2.1}, h(b)), q_{r,2})$. Then Simplify$(f(q'_{r,2.1}, h(b)), q_{r,2}) = (f(q'_{r,2.1}, q'_{r,2}), q_{r,2})$, and the normalization will add the transition $f(q'_{r,2.1}, q'_{r,2}) \to q_{r,2}$ to Δ.

There is no other critical pair. The process stops and the automaton generates $\{h(s^n(a)),\ f(s^n(a), h(b)) \mid n \in \mathbb{N}\} = R^*_\mu(L(\mathcal{A}_0))$. Note that $f(a, f(b, h(b))) \in R^*(L(\mathcal{A}_0))$ whereas $f(a, f(b, h(b))) \notin R^*_\mu(L(\mathcal{A}_0))$, i.e. $R^*_\mu(L(\mathcal{A}_0)) \neq R^*(L(\mathcal{A}_0))$, and notice that the automaton generates only the elements of $R^*_\mu(L(\mathcal{A}_0))$.

In this example, R is right-shallow, and our completion computes an automaton similar to that of [15][5].

The following rewrite system is not right-shallow, and shows the role of the reduction step.

Example 11. $R = \{f(x) \to s(f(x))\}$, $\mu(f) = \mu(s) = \{1\}$. Let $\mathcal{A}_0$ be the automaton defined by $Q_0 = \{q_a, q_f\}$, $Q_f = \{q_f\}$, and $\Delta_0 = \{a \to q_a,\ f(q_a) \to q_f\}$. Note that $L(\mathcal{A}_0) = \{f(a)\}$ and $R^*_\mu(L(\mathcal{A}_0)) = \{s^n(f(a)) \mid n \in \mathbb{N}\}$ where s^n denotes n occurrences of s.

The initialization step gives $Q_R = Q'_R = \Delta_R = \emptyset$, $Q' = \{q'_a, q'_f\}$, $rm'(q'_a) = q_a$, $rm'(q'_f) = q_f$, $\Delta = \{a \to q'_a,\ a \to q_a,\ f(q'_a) \to q'_f,\ f(q_a) \to q_f\}$.
With $f(q_a) \to_\Delta q_f$ and $f(q_a) \to_R s(f(q_a))$, we get the critical pair $(s(f(q_a)), q_f)$. However $s(f(q_a)) \to_\Delta s(q_f)$, i.e. Reduce$_\mathcal{A}(s(f(q_a)), q_f) = (s(q_f), q_f)$, and the normalized transition $s(q_f) \to q_f$ is added to Δ. No more critical pairs are

[5] In [15], a tilde is used instead of a prime, but tilde over a state means that a rewrite step is allowed, whereas in our approach a prime means that rewriting is forbidden.

detected, and the algorithm stops. Now the automaton generates $\{s^n(f(a)) \mid n \in \mathbb{N}\} = R^*_\mu(L(\mathcal{A}_0))$.

If $\mathsf{Reduce}_\mathcal{A}$ were not applied, then the critical pair $(s(f(q_a)), q_f)$ would be normalized into the transitions $s(q_1) \to q_f$, $f(q_a) \to q_1$. But there would be one more critical pair due to $f(q_a) \to q_1$, which would add some more transitions, and so on. In this case, if the normalization process always introduces new states, the completion process would not terminate.

The following rewrite system is not right-shallow, and shows what happens when a subterm forbidden by μ becomes allowed by μ after applying a rewrite step.

Example 12.
Let $R = \{f(x,y) \to h(s(x), s(y))\}$, with $\mu(f) = \emptyset$, $\mu(h) = \{1\}$, $\mu(s) = \{1\}$. Let $\mathcal{A}_0$ defined by $Q_0 = \{q\}$, $Q_f = \{q\}$, and $\Delta_0 = \{a \to q, f(q,q) \to q\}$. Thus $L(\mathcal{A}_0) = \{a, f(a,a), f(f(a,a),a), f(a,f(a,a)), f(f(a,a),f(a,a)) \ldots\}$. $R^*_\mu(L(\mathcal{A}_0))$ is obtained from the terms of $L(\mathcal{A}_0)$, by replacing some occurrences of f by the pattern $h(s(),s())$ along the left branch, starting from the root. For example $h(s(a),s(a))$, $h(s(f(a,a)),s(a))$, $h(s(h(s(a),s(a))),s(a))$, $h(s(f(a,a)),s(f(a,a)))$ are in $R^*_\mu(L(\mathcal{A}_0))$, whereas $h(s(a),h(s(a),s(a)))$ is not in $R^*_\mu(L(\mathcal{A}_0))$.
The initialization step gives $Q_R = Q'_R = \Delta_R = \emptyset$, and

$$Q' = \{q'\},\ rm'(q') = q, \text{ and}$$
$$\Delta = \{a \to q',\ a \to q,\ f(q',q') \to q',\ f(q',q') \to q\}.$$

Using $f(q',q') \to q$ and $f(x,y) \to h(s(x),s(y))$, we get the critical pair $(rm'(h(s(q'),s(q'))),q) = (h(s(q),s(q')),q)$. This critical pair is not convergent, and cannot be simplified nor reduced. It is not normalized. Then $\mathsf{Norm}_\mathcal{A}$ creates the transitions $h(q_1,q'_1) \to q$, $s(q) \to q_1$, $s(q') \to q'_1$, and add them to Δ.

No more critical pair is detected, then the algorithm stops with $\Delta =$

$$\{a \to q',\ a \to q,\ f(q',q') \to q',\ f(q',q') \to q,\ h(q_1,q'_1) \to q,\ s(q) \to q_1,\ s(q') \to q'_1\}$$

Now, one can see that $L(\mathcal{A}) = R^*_\mu(L(\mathcal{A}_0))$.

In the previous examples $L(\mathcal{A}) = R^*_\mu(L(\mathcal{A}_0))$. However, one may have $L(\mathcal{A}) \supset R^*_\mu(L(\mathcal{A}_0))$, i.e. a strict over-approximation.

Example 13. Let $I = \{f(a,b)\}$ and $R = \{f(x,y) \to f(s(x),p(y)),\ b \to c\}$, with $\mu(f) = \mu(s) = \mu(p) = \emptyset$. Then $R^*_\mu(I) = \{f(s^n(a),p^n(b)) \mid n \in \mathbb{N}\}$ is not a regular tree language and then it cannot be expressed by a tree automaton. So completion will necessarily lead to a strict over-approximation, by losing the link between the number of s and the number of p. Nevertheless, the elements of $\{f(s^n(a),p^n(c))\}$, which are in $R^*(I)$ but not in $R^*_\mu(I)$, will not be generated.

5 Related Work

To the best of our knowledge, the only method to express descendants by regular languages in the framework of context-sensitive rewriting, is that of

Sakai et al. [15]. This method returns a tree automaton that recognizes the set of descendants in an exact way (it is not an over-approximation), assuming that the rewrite system is linear and right-shallow. This is why this method cannot deal with Examples 11 and 12, whose rewrite systems are not right-shallow.

Genet et al. compute an over-approximation of the descendants without strategy [10], or according to the innermost strategy [13]. They do not consider context-sensitive rewriting.

Some results of Falke et al. [6,8] also deal with context-sensitive rewriting, but they do not study reachability problems. They study termination problems and propose a method for proving inductive theorems. The termination problems are based on dependency pairs, and the inductive theorem prover is based on the inference system of Reddy [7].

6 Further Work

With our method, and more generally with every method based on the completion of tree automata, the quality of the approximation highly depends on the way the completion is achieved. When normalizing critical pairs, existing states may be used instead of introducing new ones. This helps to make completion terminate. However, the choice of the states to be re-used is crucial for the quality of the approximation. Some heuristics have been developed for ordinary rewriting. Recently, an heuristic using a set of equations E has been presented [12], and an upper-bound for the approximation is given, which allows to estimate the quality of the approximation. We intend to extend these heuristics to context-sensitive completion, so that they could be used within our method.

Another interesting problem to study is: does our method take the map μ into account in a perfect way? In other words, may our method generate descendants that are not context-sensitive descendants? If the re-use of existing states in the normalization process is allowed, we get an over-approximation, and wrong context-sensitive descendants (that are ordinary descendants) may be generated. What about if the re-use of existing states is forbidden?

When considering ordinary rewriting, the set of descendants of a set of terms I is not a regular tree language, even if I is, except if strong restrictions are assumed over the rewrite system. It is the same when considering context-sensitive rewriting. This is why we cannot compute the descendants in an exact way, except for some particular cases. The use of tree languages more expressive than the regular ones, could lead to more precise computations. It has already been studied for ordinary rewriting [2,14], but not for context-sensitive rewriting.

References

1. Andrianarivelo, N., Réty, P.: Over-Approximating Terms Reachable by Context-Sensitive Rewriting (full version). Technical Report RR-2015-02, LIFO, Université d'Orléans (2015)

2. Boichut, Y., Chabin, J., Réty, P.: Over-approximating descendants by synchronized tree languages. In: Proceedings of the International Conference RTA. LIPIcs, vol. 21, pp. 128–142 (2013)
3. Boichut, Y., Genet, T., Jensen, T., Le Roux, L.: Rewriting approximations for fast prototyping of static analyzers. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 48–62. Springer, Heidelberg (2007)
4. Cortier, V., Delaune, S., Lafourcade, P.: A Survey of Algebraic Properties Used in Cryptographic Protocols. Journal of Computer Security **14**(1), 1–43 (2006)
5. D'Silva, V., Kroening, D., Weissenbacher, G.: A Survey of Automated Techniques for Formal Software Verification. IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems **27**(7), 1165–1178 (2008)
6. Falke, S., Kapur, D.: Dependency Pairs for rewriting with non-free constructors. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 426–442. Springer, Heidelberg (2007)
7. Falke, S., Kapur, D.: Inductive decidability using implicit induction. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 45–59. Springer, Heidelberg (2006)
8. Falke, S., Kapur, D.: Termination of context-sensitive rewriting with built-in numbers and collection data structures. In: Escobar, S. (ed.) WFLP 2009. LNCS, vol. 5979, pp. 44–61. Springer, Heidelberg (2010)
9. Futatsugi, K., Goguen, J.A., Jouannaud, J.-P., Meseguer, J.: Principles of OBJ2. In: Proceedings of the International Symposium POPL, pp. 52–66 (1985)
10. Genet, T.: Decidable approximations of sets of descendants and sets of normal forms. In: Nipkow, T. (ed.) RTA 1998. LNCS, vol. 1379, pp. 151–165. Springer, Heidelberg (1998)
11. Genet, T.: Reachability Analysis of Rewriting for Software Verification. Université de Rennes 1 (2009), Habilitation document. http://www.irisa.fr/celtique/genet/publications.html
12. Genet, T., Rusu, V.: Equational Approximations for Tree Automata Completion. Journal of Symbolic Computation **45**(5), 574–597 (2010)
13. Genet, T., Salmon, Y.: Reachability analysis of innermost rewriting. In: Proceedings of the International Conference RTA. LIPIcs, vol. 36, pp. 177–193 (2015)
14. Kochems, J., Ong, C.-H.L.: Improved functional flow and reachability analyses using indexed linear tree grammars. In: Proceedings of the International Conference RTA. LIPIcs, vol. 10, pp. 187–202 (2011)
15. Kojima, Y., Sakai, M.: Innermost reachability and context sensitive reachability properties are decidable for linear right-shallow term rewriting systems. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 187–201. Springer, Heidelberg (2008)
16. Lucas, S.: Context-sensitive computations in functional and functional logic programs. Journal of Functional and Logic Programming **1998**(1), January 1998

Reducing Bounded Realizability Analysis to Reachability Checking

Masaya Shimakawa[1(✉)], Shigeki Hagihara[1], and Naoki Yonezaki[2]

[1] Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Tokyo, Japan
masaya@fmx.cs.titech.ac.jp
[2] The Open University of Japan, Chiba, Japan

Abstract. Realizability verification of reactive system specifications can detect dangerous situations that can arise, which were not expected while drawing the specifications. However, such verification typically involves complex, intricate analyses. The complexity of the realizability problem is 2EXPTIME-complete. To avoid this difficulty, Schewe et al. introduced the notion of bounded realizability. While realizability is the property that a model of a reactive system exists that satisfies a given specification, bounded realizability requires the existence of a model of size k that satisfies the given specification. They presented a method based on satisfiability modulo theories (SMT) for bounded realizability checking. Here, we present a more efficient method for checking bounded realizability. Our method reduces bounded realizability checking to satisfiability (SAT)-based reachability checking and is faster because in many cases, the result is obtained by reachability checking of small steps. We show the complexity of a bounded realizability problem for linear temporal logic (LTL) specifications is NEXPTIME-complete, in which the upper bound is derived from our SAT-encoding technique. We also report experimental results that show the effectiveness of our method.

1 Introduction

Many safety-critical systems are considered reactive systems, and they interact with their environment. Such systems should be designed to respond appropriately to any request from the environment at any time. Verifying this property during the specification phase is important for reducing their development costs. This property of specifications is known as realizability[19][1], or the property that a system model exists that satisfies a given specification in all cases. Realizability verification can detect dangerous cases, and a system model can be synthesized if the specification is realizable[19]. However, such verification typically involves complex, intricate analyses. Therefore, it can only be applied at a limited scale.

To avoid this difficulty, Schewe et al. introduced the bounded property of realizability in [21][13], and presented a method based on satisfiability modulo theories (SMT) in [13] [1]. They restricted the size of witnesses (models of correct systems) or

[1] The method is also for distributed systems.

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 140–152, 2015.
DOI: 10.1007/978-3-319-24537-9_13

counterexamples (models of oppositional environments) to a given k. The advantage of bounded (un-)realizability checking is the ability to check the existence of a small witness (counterexample) efficiently. In our experience, many practical unrealizable specifications have a small counterexample. This approach can detect it at low cost.

This paper presents a more efficient method for bounded realizability, and considers the complexity of the bounded realizability problem based on this method.

Our method constructs a universal co-Büchi tree automaton that accepts models of a system (environment) that satisfy (do not satisfy) the specifications in all cases. Then, we check bounded nonemptiness for the automaton. We reduce the bounded nonemptiness checking to satisfiability (SAT)-based reachability checking. In many cases, the result is obtained via the reachability checking of small steps. Therefore, our method can check at lower cost.

Based on our method, we show that the bounded realizability problem for specifications described in linear temporal logic (LTL) is NEXPTIME-complete. The upper bound of the complexity is derived from our SAT-encoding technique. The lower bound is proved by a reduction from an EXP-square tiling problem, which is NEXPTIME-complete, to a bounded realizability problem. In the reduction, we use only G (Globally) and X (neXt) operators. Therefore, it is shown that the bounded realizability problem remains NEXPTIME-complete even when the specification language is the fragment LTL(G, X), where only G and X operators are allowed.

2 Realizability and Bounded Realizability

2.1 Reactive Systems and Environments

A reactive system is a system that responds to requests from an environment in a timely fashion.

Definition 1. *A reactive system is a reaction function* $f : (2^X)^* \to 2^Y$*, where* X *is a set of events caused by the environment,* Y *is a set of events caused by the system. An environment is a reaction function* $g : (2^Y)^+ \to 2^X$*.*

We refer to events caused by the environment as 'input events,' and those caused by the system as 'output events.' The reaction function f(resp., g) relates sequences of sets of previously occurring input events (resp., output events) to a set of current output events (resp., input events).

2.2 Reactive System Specifications

The timing of input and output events is an essential element of reactive systems. LTL is a suitable language for describing the timing of events. In this paper, we use LTL to describe the specifications of reactive systems. In LTL, in addition to the operators $\wedge, \vee, \oplus, \rightarrow, \neg$, $\top$ and $\bot$, we can use the temporal operators $\mathbf{X}$,

G, **F** and **U**. We treat input events and output events as atomic propositions. Behavior σ is an infinite sequence of sets of events. $\sigma \models_{bhv} \varphi$ represents that σ satisfies φ, which is defined as usual.

2.3 Realizability

It is important for reactive system specifications to satisfy realizability. Realizability requires the existence of a reactive system such that for any input events with any timing, the system produces output events such that the specification holds.

Definition 2. *A specification Spec is realizable if there exists a reactive system* f *such that* $f \models_{sys}$ *Spec, where* $f \models_{sys}$ *Spec is defined as* $\forall a_0, a_1, a_2, \ldots \in 2^X.((f(\epsilon) \cup a_0)(f(a_0) \cup a_1)(f(a_0a_1) \cup a_2) \ldots \models_{bhv}$ *Spec).*

We also define unrealizability in the same manner.

Definition 3. *A specification Spec is unrealizable if there exists an environment* g *such that* $g \models_{env} \neg$*Spec, where* $g \models_{env} \neg$*Spec is defined as* $\forall b_0, b_1, \ldots \in 2^Y.(b_0 \cup g(b_0))(b_1 \cup g(b_0b_1)) \ldots \models_{bhv} \neg$*Spec).*

Note that a specification *Spec* is not realizable if and only if *Spec* is unrealizable, when the specification is described in LTL (or ω-regular expression, finite automata for ω-word).

2.4 Bounded Realizability

In bounded (un-)realizability, we restrict the size of the witness or counterexample to some k, that is, we consider only witnesses or counterexamples that are represented by a transition machine of size k.
Transition machine is a tuple $T = (\Sigma, D, S, s_I, \eta, o)$ where Σ is an alphabet, D is a finite set of directions, S is a finite set of states, s_I is an initial state, $\eta : S \times D \to S$ is a transition function, and $o : S \to \Sigma$ is a labeling function. We define $\zeta : D^* \to S$ as follows:$\zeta(\epsilon) = s_I$, $\zeta(x \cdot c) = \eta(\zeta(x), c)$. The size $|T|$ of T is defined as $|T| = |S|$. A transition machine $T = (2^Y, 2^X, S, s_I, \eta, o)$ represents a reactive system f_T such that for all $\bar{a} \in (2^X)^*$, $f_T(\bar{a}) = o(\zeta(\bar{a}))$. In addition, a transition machine $T = (2^X, 2^Y, S, s_I, \eta, o)$ represents an environment g_T such that for all $\bar{b} \in (2^Y)^+$, $g_T(\bar{b}) = o(\zeta(\bar{b}))$.

Definition 4. *Let Spec be a specification;* $k \in \mathbb{N}$. *Spec is* k*-realizable if there exists a transition machine* T *such that* $f_T \models_{sys}$ *Spec and* $|T| = k$. *Spec is* k*-unrealizable if there exists* T *such that* $g_T \models_{env} \neg$*Spec and* $|T| = k$.

Note that in a bounded setting, the fact that a specification *Spec* is not k-realizable does not imply that *Spec* is k-unrealizable. By definition, it is obvious that if *Spec* is k-realizable, *Spec* is also realizable. Moreover, in the case that *Spec* is described in LTL (or ω-regular expression, finite automata for ω-word), if *Spec* is realizable, *Spec* is k-realizable for some k.

2.5 Procedure for Bounded Realizability Checking

This subsection outlines a procedure for checking bounded realizability using ω-automata, which is presented in [13]. Bounded unrealizability also can be checked in the same way.

1. We obtain a universal co-Büchi word automaton $\mathcal{A}_{bhv}$ such that $L(\mathcal{A}_{bhv}) = \{\sigma \mid \sigma \models_{bhv} \mathit{Spec}\}$.
2. From $\mathcal{A}_{bhv}$, we construct a universal co-Büchi tree automaton $\mathcal{A}_{sys}$ such that $L(\mathcal{A}_{sys}) = \{T \mid \ f_T \models_{sys} \mathit{Spec}\}$.
3. We check whether there exists a transition system T of size k such that $T \in L(\mathcal{A}_{sys})$ (k-*nonempty*). If it is k-nonempty, we conclude that *Spec* is k-realizable. Otherwise, we conclude that *Spec* is not k-realizable.

In [13], an SMT-based method of bounded nonemptiness checking for a universal co-Büchi tree automaton (UCT) is given.[2] Here, we reduce bounded nonemptiness checking for UCT to a reachability problem, and solve the problem using a SAT solver. Our method is more efficient, as described in Sect. 5.

3 Reachability-Based Bounded Nonemptiness Checking for UCT

In this section, we present a reachability-based method of bounded nonemptiness checking for UCT, which is based on that of bounded universality checking for NBA in [24][25].

3.1 Universal co-Büchi Tree Automata(UCT)

We define the syntax and semantics for UCT.

Syntax : The UCT is a tuple $\mathcal{A} = (\Sigma, D, Q, q_I, \delta, F)$ where Σ is an alphabet, D is a finite set of directions, Q is a finite set of locations, q_I is an initial location, $\delta \subseteq Q \times \Sigma \times D \times Q$ is a transition relation, and $F \subseteq Q$ is a set of accepting location.

Semantics for Transition Machines : Using run graph, we define the semantics of UCT for transition machines. For a UCT $\mathcal{A} = (\Sigma, D, Q, q_I, \delta, F)$ and a transition machine $T = (\Sigma, D, S, s_i, \eta, o)$, the run graph $G = (V, v_I, E, C)$ is defined as follows: $V := Q \times S$ (the set of nodes), $v_I := (q_I, s_I)$ (the initial node), $E := \{((q, s), (q', s')) \mid (q, o(s), c, q') \in \delta, s' = \eta(s, c)\}$ (the set of edges), and $C := \{(q, s) \mid q \in F, s \in S\}$ (the set of accepting nodes). We say $\mathcal{A}$ accepts T, if for all paths from the initial node in the run graph for $\mathcal{A}$ and T, the number of occurrences of accepting nodes in the path is finite.

[2] The method is also for distributed systems.

3.2 Characterization for Transition Machines of Size k Accepted by UCT

Here, we characterize transition machines of size k that are accepted by UCT.

The number of accepting nodes in a run graph is bounded. From this, the following result is derived. [3]

Lemma 1. *If for all paths $\tilde{v}$ in a run graph G from a node v, the number of occurrences of accepting nodes in $\tilde{v}$ is finite, then for all paths $\tilde{v}$ from v in G, accepting nodes occur at most $|C|$ times in $\tilde{v}$.*

The property that "for all paths $\tilde{v}$ from v, the number of occurrences of accepting nodes is at most j"(denoted by $AtMost(v, j)$) is characterized as follows: For $v \in V \setminus C$ (for $v \in C$), $AtMost(v, j)$ holds if and only if for all successors $v' \in vE$, $AtMost(v', j)$ holds ($AtMost(v', j-1)$ holds). In addition, for all $v \in C$, $AtMost(v, 0)$ does not hold. Based on this idea, the following result is derived:

Theorem 1 ([25]). *Let $G = (V, v_I, E, C)$ be a run graph and $d \in \mathbb{N}$. For all paths $\tilde{v}$ from v_I in G, accepting nodes occur at most d times if and only if there exist a sequence $V_0, V_1, \ldots, V_d$ of sets of nodes such that the following are true:*
1. The following condition (denoted by $I(V_0)$) holds:
$v \in V_0 \Longleftrightarrow$ (*if* $v \in V \setminus C$ *then* $\forall v' \in vE.\ v' \in V_0$ *else* $\bot$)
2. For all $0 \leq j < d$, the following condition (denoted by $T(V_j, V_{j+1})$) holds:
$v \in V_{j+1} \Longleftrightarrow$ (*if* $v \in V \setminus C$ *then* $\forall v' \in vE.\ v' \in V_{j+1}$ *else* $\forall v' \in vE.\ v' \in V_j$)
3. $v_I \in V_d$ holds (denoted by $F(V_d)$).

From the above theorem, the following is derived.

Theorem 2. *Let $\mathcal{A} = (\Sigma, D, Q, q_I, \delta, F)$ be a UCT and $k \in \mathbb{N}$. For all $d \in \mathbb{N}$, (2) implies (1), and for $d \geq k \cdot |F|$, (1) implies (2), where (1) and (2) are as follows:*
(1) There exists a transition machine of size k that is accepted by $\mathcal{A}$.
(2) There exists a transition machine T of size k such that for some sequence $V_0, V_1, \ldots, V_d$ of sets of nodes of the run graph G for $\mathcal{A}$ and T, $I(V_0) \wedge \bigwedge_{0 \leq j < d} T(V_j, V_{j+1}) \wedge F(V_d)$ holds.

3.3 SAT Encoding

We present a reduction to a SAT problem based on Theorem 2. That is, for a UCT $\mathcal{A}$ and k, we give a construction of a Boolean formula $|[acc(\mathcal{A}, k, d)]|$ such that condition (2) of Theorem 2 holds iff $|[acc(\mathcal{A}, k, d)]|$ is satisfiable.

Variables: We use the following variables (assuming $\Sigma = 2^Y$), to represent a transition machine of size k (where $S = \{1, 2, \ldots k\}$), and $V_0, V_1, \ldots, V_d$. (a)y_i for $y \in Y$, $1 \leq i \leq k$, which indicate whether $y \in o(i)$ holds, (b) $tr_{(i,c,i')}$ for

[3] If the number of occurrences of accepting nodes in a path exceeds $|C|$, then there exists an accepting node v_c that occurs at least twice, which implies the existence of a path on which the accepting node v_c occurs infinitely often.

Table 1. The definition of $|[I(\mathcal{A},k)]|_0$, $|[T(\mathcal{A},k)]|_{j,j+1}$ and $|[F(\mathcal{A},k)]|_d$

$\|[I(\mathcal{A},k)]\|_0$	$\bigwedge_{q\in Q\setminus F,1\le i\le k}\left(v^0_{(q,i)} \leftrightarrow \bigwedge_{(q,b,c,q')\in\delta,1\le i'\le k}((\|[b]\|_i \wedge tr_{(i,c,i')}) \rightarrow v^0_{(q',i')})\right) \wedge \bigwedge_{q\in F,1\le i\le k}\left(\neg v^0_{(q,i)}\right)$
$\|[T(\mathcal{A},k)]\|_{j,j+1}$	$\bigwedge_{q\in Q\setminus F,1\le i\le k}\left(v^{j+1}_{(q,i)} \leftrightarrow \bigwedge_{(q,b,c,q')\in\delta,1\le i'\le k}((\|[b]\|_i \wedge tr_{(i,c,i')}) \rightarrow v^{j+1}_{(q',i')})\right) \wedge \bigwedge_{q\in F,1\le i\le k}\left(v^{j+1}_{(q,i)} \leftrightarrow \bigwedge_{(q,b,c,q')\in\delta,1\le i'\le k}((\|[b]\|_i \wedge tr_{(i,c,i')}) \rightarrow v^{j}_{(q',i')})\right)$
$\|[F(\mathcal{A},k)]\|_d$	$v^d_{(q_I,0)}$

$1 \le i \le k$, $c \in D$, $1 \le i' \le k$, which indicate whether $\eta(i,c) = i'$ holds, (c)$v^j_{(q,i)}$ for $q \in Q, 1 \le i \le k, 0 \le j \le d$, which indicate whether $(q,i) \in V_j$ holds.

Constraints: To represent "be a transition machine correctly," we prepare the formula $|[det(k)]| := \bigwedge_{1\le i\le k,c\in D}\bigvee_{1\le i'\le k} tr_{(i,c,i')} \wedge \bigwedge_{1\le i\le k,c\in D}\bigwedge_{1\le i'\le k,i''\ne i'}(tr_{(i,c,i')} \rightarrow \neg tr_{(i,c,i'')})$. We define the formulas $|[I(\mathcal{A},k)]|_0$, $|[T(\mathcal{A},k)]|_{j,j+1}$ and $|[F(\mathcal{A},k)]|_d$, which indicate that $I(V_0)$, $T(V_i,V_{i+1})$ and $F(V_d)$ hold, respectively. The definitions are in Table 1, where $|[b]|_i$ is the formula $\bigwedge_{y\in b} y_i \wedge \bigwedge_{y\notin b}\neg y_i$, indicating that the label of state i is b.

We define the formula $|[acc(\mathcal{A},k,d)]|$ by $|[acc(\mathcal{A},k,d)]| := |[det(k)]| \wedge |[I(\mathcal{A},k)]|_0 \wedge \bigwedge_{0\le j<d}|[T(\mathcal{A},k)]|_{j,j+1} \wedge |[F(\mathcal{A},k)]|_d$.

Theorem 3. *Let $\mathcal{A} = (2^Y, D, Q, q_I, \delta, F)$ and $k \in \mathbb{N}$. For all $d \in \mathbb{N}$, (2) implies (1), and for $d \ge k \cdot |F|$, (1) implies (2), where (1) and (2) are as follows:*
(1) There exists a transition machine of size k that is accepted by $\mathcal{A}$.
(2) $|[acc(\mathcal{A},k,d)]|$ is satisfiable.

Theorem 4. *Let $\mathcal{A} = (2^Y, D, Q, q_I, \delta, F)$ and $k, d \in \mathbb{N}$. The size of $|[acc(\mathcal{A},k,d)]|$ is $\mathcal{O}(k^3 \cdot D + k^2 \cdot d \cdot |Y| \cdot |\delta|)$. Checking that $\mathcal{A}$ is k-nonempty can be reduced to the SAT problem for a formula of size $\mathcal{O}(k^3 \cdot D + k^3 \cdot |F| \cdot |Y| \cdot |\delta|)$.*

3.4 Incremental Checking

If $|[acc(\mathcal{A},k,d)]|$ is satisfiable, even for small d, then there exists a transition machine of size k that is accepted by $\mathcal{A}$. The smaller the value of d is, the smaller the size of $|[acc(\mathcal{A},k,d)]|$ and the lower the checking cost. Therefore, it is effective to check whether $|[acc(\mathcal{A},k,d)]|$ is satisfiable incrementally for $d = 0, 1, \ldots, d_{max}(= k \cdot |F|)$. An incremental approach reduces the cost of finding a witness. Furthermore, using the induction technique for incremental SAT-based unbounded reachability checking in [22][3], at the stage at which d is less than $k \cdot |F|$, we judge that $\mathcal{A}$ is not k-nonempty. Because we characterize transition machines of size k that are accepted by UCT, using reachability, it is straightforward to apply the induction technique described in [22][3] to our method. These are advantages of our approach based on reachability checking.

4 Complexity of Bounded Realizability Problem

In this section, we show that the bounded realizability problem for a specification described in LTL and a non-negative integer k encoded in binary (i.e., we

consider the size of k to be $\lfloor \log k \rfloor + 1$) is NEXPTIME-complete. That is, we show (1) the bounded realizability problem is in NEXPTIME (the class of problems solvable in $O(2^{p(n)})$ time by a non-deterministic Turing machine, where $p(n)$ is a polynomial function of the input size n) and (2) the bounded realizability problem is NEXPTIME-hard. Moreover, we show that the bounded realizability problem remains NEXPTIME-complete even when the specification language is the fragment LTL(G, X) where only G and X operators are allowed.

4.1 Upper Bound

Theorem 5. *The bounded realizability problem for a specification described in LTL and a non-negative integer k encoded in binary is in NEXPTIME.*

Proof. We prove that the procedure in Sect. 2.5 is accomplished in $O(2^{p(n)})$ time using a non-deterministic Turing machine. $\mathcal{A}_{bhv}$ and $\mathcal{A}_{sys}$ can be constructed within $O(2^{|Spec|})$ time, even by a deterministic Turing machine, and the size of $\mathcal{A}_{bhv}$ and $\mathcal{A}_{sys}$ is also $O(2^{|Spec|})$[27][15] , where $|Spec|$ is the length of *Spec*. From Theorem 4, the bounded nonemptiness problem for a UCT and a non-negative integer k encoded in unary is in NP. Then, Step 3 is accomplished within $O(p(|\mathcal{A}_{sys}| + k))$ time by a non-deterministic Turing machine. Therefore, we can solve the bounded realizability problem in $O(2^{p(|Spec|+(\lfloor \log k \rfloor + 1))})$ time using a non-deterministic Turing machine.

4.2 Lower Bound

Here, we show that the bounded realizability problem is NEXPTIME-hard, by providing polynomial time reduction from the EXP-square tiling problem (see, e.g., [7][5]) to the bounded realizability problem. The tiling problem is NEXPTIME-complete.

Definition 5. *The EXP-square tiling problem is as follows: For a given $(T, H, V, t_{init}, t_{final}, m)$, where T is a finite set of tile types, $H, V \subseteq T \times T$ are horizontal and vertical adjacency constraints, $t_{init} \in T$ is the initial tile type, $t_{final} \in T$ is the final tile type, and m is a natural number encoded in unary, determine whether there exists an assignment function $f : [0, (2^m - 1)] \times [0, (2^m - 1)] \to T$ such that the following conditions are satisfied:*
1. $f(0, 0) = t_{init}$
2. $f((2^m - 1), (2^m - 1)) = t_{final}$
3. for any $0 \leq j \leq 2^m - 1$, $0 \leq i < 2^m - 1$, $(f(i, j), f(i + 1, j)) \in H$ holds.
4. for any $0 \leq i \leq 2^m - 1$, $0 \leq j < 2^m - 1$, $(f(i, j), f(i, j + 1)) \in V$ holds.

As shown in Fig. 1, the tiling grid has $2^m \times 2^m$ points. Intuitively, this leads to the following question: "For a given tiling grid, can a tile be assigned to each point (i, j) for which $0 \leq i < 2^m$ and $0 \leq j < 2^m$, satisfying conditions 1–4?" Condition 1 is the condition for the initial tile, and states that the tile of type t_{init} is assigned to the leftmost, uppermost point. Condition 2 is the analogous condition for the final tile. Condition 3 is the condition for horizontal lines,

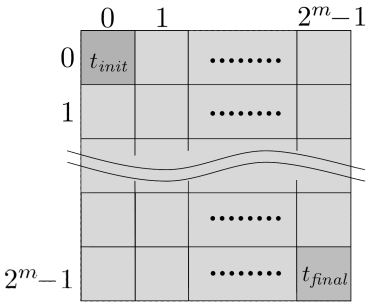


Fig. 1. The EXP-square tiling problem

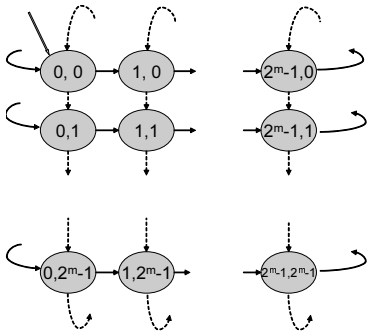


Fig. 2. The representation of a tiling by a transition machine

and states that horizontally neighboring tiles satisfy the horizontal adjacency constraint H. Condition 4 is the analogous condition for vertical lines.

Theorem 6. *The bounded realizability problem for a specification written in LTL and a non-negative integer k encoded in binary is NEXPTIME-hard.*

Proof. We provide a polynomial time reduction from the EXP-square tiling problem to the bounded realizability problem. That is, for the EXP-square tiling problem $(T, H, V, t_{init}, t_{final}, m)$, we construct a formula φ_{tiling} and a non-negative integer k_{tiling} such that φ_{tiling} is k_{tiling}-realizable if and only if the answer to the tiling problem $(T, H, V, t_{final}, t_{final}, m)$ is affirmative.

In our reduction, a tiling assignment is represented by a transition machine that has $2^m \times 2^m$ states, illustrated in Fig. 2. A state of the transition machine corresponds to a grid point. The transition machine has horizontal and vertical directions, and the horizontal (vertical) successor of a state corresponds to the point to the right of it (beneath it). The horizontal (vertical) successor of a rightmost (lowermost) state is the leftmost state of the same row (column). Moreover, the tile type of each point of the grid is represented by a label of each state.

We introduce the following input event. dir_x: If this event is caused, the direction is horizontal. Otherwise, the direction is vertical. We also introduce the following output event. (a) $x_0, \ldots, x_{m-1}$: These are used to identify a column, by keeping track of the number of horizontal transitions. (b) $y_0, \ldots, y_{m-1}$: These are used to identify a row, by keeping track of the number of vertical transitions. (c) $tile_t$ for $t \in T$: "the tile of type t is placed on a point" is related to "the event $tile_t$ is caused on the state s corresponding to the point $(tile_t \in o(s))$."

The formula φ_{tiling} is the conjunction of the formulas (A)–(H) in Table 2. Here, we use the following abbreviations: $(\overline{x} = 2^m - 1) \equiv \bigwedge_{0 \leq i < m} x_i, (\overline{y} = 2^m - 1) \equiv \bigwedge_{0 \leq i < m} y_i$. Formula (C) represents the statement "the horizontal successor of the state corresponding to a point (x, y) is the state corresponding to the point $(x+1 \ mod \ 2^m), y)$." Formula (D) is the vertical version of the above. Formula (F) represents the statement "the final tile of type t_{final} is placed on

Table 2. The definition of formulas (A)–(H)

(A)Single assignment of tile types:	$\mathbf{G}\left((\bigvee_{t\in T} tile_t) \wedge \bigwedge_{t\in T}\left(tile_t \rightarrow \bigwedge_{t'\neq t} \neg tile_{t'}\right)\right)$
(B)The constraint for the initial state:	$\bigwedge_{0\leq i<m} \neg x_i \ \wedge \ \bigwedge_{0\leq i<m} \neg y_i$
(C)The constraint for horizontal transitions:	$\mathbf{G}\left(dir_x \rightarrow \bigwedge_{0\leq i<m}\left((x_i \oplus \bigwedge_{0\leq j<i} x_j) \leftrightarrow \mathbf{X}x_i\right)\right) \wedge \mathbf{G}\left(dir_x \rightarrow \bigwedge_{0\leq i<m}(y_i \leftrightarrow \mathbf{X}y_i)\right)$
(D)The constraint for vertical transitions:	$\mathbf{G}\left(\neg dir_x \rightarrow \bigwedge_{0\leq i<m}\left((y_i \oplus \bigwedge_{0\leq j<i} y_j) \leftrightarrow \mathbf{X}y_i\right)\right) \wedge \mathbf{G}\left(\neg dir_x \rightarrow \bigwedge_{0\leq i<m}(x_i \leftrightarrow \mathbf{X}x_i)\right)$
(E)The constraint for condition 1:	$tile_{t_{init}}$
(F)The constraint for condition 2:	$\mathbf{G}((\overline{x} = 2^m - 1 \ \wedge \ \overline{y} = 2^m - 1) \rightarrow tile_{t_{final}})$
(G)The constraint for condition 3:	$\mathbf{G}\left((\overline{x} \neq 2^m - 1 \wedge dir_x) \rightarrow \bigvee_{(t,t')\in H}(tile_t \wedge \mathbf{X}tile_{t'})\right)$
(H)The constraint for condition 4:	$\mathbf{G}\left((\overline{y} \neq 2^m - 1 \wedge \neg dir_x) \rightarrow \bigvee_{(t,t')\in V}(tile_t \wedge \mathbf{X}tile_{t'})\right)$

the point $(2^m-1, 2^m-1)$." Formula (G) represents the statement "if the current point is not in the (2^m-1)-th column (i.e., a point exists to the right of it), the tile at the current point and the tile at the point to the right of it, which corresponds to the horizontal successor, satisfy horizontal adjacency constraints." Formula (H) is the vertical version of that of condition 3.

We define $k_{tiling} = 2^m \times 2^m$ (encoded as $2m+1$ bits). Because the bound $2^m \times 2^m$, formula (C) and (D), states of the transition machine and points of the tiling grid are in one-to-one correspondence.

Observe that in the reduction to EXP-square tiling problem, only G and X operators are used. From this, the following is derived. Therefore, even for LTL(G,X) specifications, the bounded realizability problem is NEXPTIME-complete.

Theorem 7. *Even when a specification is written in LTL(G,X), the bounded realizability problem for the specification and a non-negative integer k encoded in binary is NEXPTIME-hard.*

4.3 Discussion

We discuss the complexity of the bounded realizability problem in relation to that of the satisfiability problem, the strong satisfiability problem[17], and the unbounded realizability problem. Strong satisfiability is the property that no path-like counterexample exists in the specification. The satisfiability problem for a specification in LTL is PSPACE-complete[26], the strong satisfiability problem is EXPSPACE-complete[23], and the realizability problem is 2EXPTIME-complete[20]. Because PSPACE $\subseteq$ NEXPTIME $\subseteq$ EXPSPACE $\subseteq$ 2EXPTIME holds, the bounded realizability problem is more difficult than, or of equal difficulty to, the satisfiable problem, and is easier than, or of equal difficulty to, the strong satisfiability problem and the realizability problem.

Table 3. The checking time (in seconds) of bounded and unbounded (un-)realizablity

	bounded (un-)realizability					unbounded	
				SMT			
	k	judgement	our method	yices	MathSAT	Acacia+	Unbeast
ele_2^{real}	6	no	121.89	T/O	384.63	0.88	0.06
	7	yes	230.87	T/O	1007.38		
lb_2^{real}	5	no	0.16	0.59	0.88	0.08	0.43
	6	yes	0.13	10.14	2.62		
lb_3^{real}	5	no	3.62	906.08	116.16	0.14	0.46
	6	yes	0.33	T/O	649.77		
arb_2^{real}	3	no	0.86	3.79	8.37	1.37	T/O
	4	no	11.39	59.70	107.71		
	5	no	252.71	T/O	719.82		
sa_5^{real}	4	no	0.81	1.94	1017.53	0.17	1.27
	5	yes	12.3	646.10	T/O		
sa_6^{real}	5	no	44.97	66.73	T/O	0.55	3.59
	6	yes	627,2	T/O	T/O		
ele_2^{unreal}	1	yes	0.12	0.40	0.42	9.59	0.47
ele_3^{unreal}	1	yes	0.85	T/O	T/O	T/O	T/O
lb_5^{unreal}	1	no	1.34	5.72	5.15	6.25	T/O
	2	yes	1.91	187.77	46.13		
lb_6^{unreal}	1	no	20.41	92.90	43.77	102.07	T/O
	2	yes	24.55	T/O	T/O		
arb_4^{unreal}	1	no	19.76	T/O	T/O	1165.77	T/O
	2	yes	2.06	T/O	T/O		
arb_5^{unreal}	1	no	819.58	T/O	T/O	T/O	T/O
	2	yes	25.62	T/O	T/O		

5 Experiments

We implemented our method, and compared the execution time with the SMT-based method in [13] and unbounded realizability checkers.[4] The implementation of our method is as follows: The construction of the UCT in Steps 1 and 2 of the procedure in Sect. 2.5 is based on [16]. The k-nonemptiness checking of Step 3 is accomplished incrementally for $d = 0, 1, \ldots$, as explained in Sect. 3.4. We use MiniSat 2.2[10] as the SAT solver. In the implementation of the SMT-based method in [13], we check k-nonemptiness using Yices 2.3.1[9] or MathSAT 5[8] as the SMT solver. As an unbounded realizability checker, we use Unbeast 0.6b[11] and Acacia+ 2.3[6].

The upper columns in Table 3 show the total checking time of (un-)bounded realizability for realizable specifications ele_n^{real}, lb_n^{real}, arb_n^{real} and sa_n^{real}. ele_n^{real} is the n-floor elevator specification[2] with the fairness assumption. lb_n^{real} is a realizable version of the n-process load balancer specification in [11][5]. arb_n^{real} is the n-process arbiter specification in [18]. sa_n^{real} is the n-process simple arbiter specification which equals the UCT specification in [13]. The notation "T/O" corresponds to a calculation that required more than 1200s. The lower columns in Table 3 show the total checking time of (un-)bounded unrealizability for unrealizable specifications ele_n^{unreal}, lb_n^{unreal} and arb_n^{unreal}. ele_n^{unreal} is the n-floor elevator

[4] The environment was OS:Ubuntu 12.04, CPU:Core2duo 3.0GHz, 6 GB memory.

[5] Here, we use the specification $(6) \wedge (7) \wedge (10) \rightarrow (1) \wedge (2) \wedge (5) \wedge (8) \wedge (9)$ in [11].

specification[2] without the fairness assumption. lb_n^{unreal} is an unrealizable version of n-process load balancer specification in [11][6]. arb_n^{unreal} is an unrealizable version of arb_n^{real}.

Our Method vs. Unbounded Realizability Checker: The results indicate that in the cases that there is no small witness or counterexample, the unbounded realizability checkers can detect a witness or counterexample faster. This is because the bounded realizability problem is NEXPTIME for k in binary, i.e., NP for k in unary. In comparison, when there is a small witness or counterexample, our method can detect it faster. In our experience, many practical unrealizable specifications have a small counterexample. Our method shines at detecting it.
Our Method vs. SMT-based Method: In many cases, our method is faster than SMT-based methods because we reduce bounded nonemptiness checking for UCT to d-step reachability checking, and the result is obtained by reachability checking of small steps. In the verification of ele_n^{unreal} on $k = 1$, lb_n^{unreal} on $k = 2$ and arb_n^{unreal} on $k = 2$, our method can detect a counterexample at stage on $d = 0$, $d = 1$ and $d = 1$, respectively. In the SMT-based method, the number of occurrences of an accepting node is treated as integer. The value of d in our method is considered a bound of it. Because in many cases the result is obtained when the bound is small, our method can check the existence of a witness or counterexample more quickly.

6 Related Works

Checking Realizability and Synthesizing Reactive Systems: Generally, when checking the realizability of a specification written in LTL, specifications are converted into deterministic ω-automata. In a naive procedure for checking realizability, Safra's construction is used for the determinization of ω-automata, although Safra's construction is complex and difficult to implement. Therefore, procedures in which Safra's construction is not used are proposed in [15] and [12]. The procedure in [12] is a refinement of the procedure in [15]. In the procedure in [12], the acceptance condition of automata is bounded, as with our method. That is, the condition that the number of occurrences of accepting states is "at most d," instead of "finite" is used as an acceptance condition of the automata. An incremental approach is also taken. A point of difference with our method is that the number of witnesses or counterexamples is not bounded in the approach. Several tools have been proposed for checking the realizability of specifications written in LTL, including Lily[14], Acacia+[6], Unbeast[11]. These tools are based on the above method.

SAT-Based or SMT-Based Verification Methods: SAT techniques are used widely in formal verification. The most famous method is bounded model checking (BMC) (see [4]). In BMC, we check whether there exists counter-traces of a size k. Bounded realizability checking[21], which is the focus of this work, is a representative example the field of verification for reactive system specification.

[6] Here, we use the specification $(6) \wedge (7) \rightarrow (1) \wedge (2) \wedge (4) \wedge (5)$ in [11].

The bounded checking methods for necessary (of sufficient) conditions of realizability were also studied. In [24][25], a method that detects a counterexample represented by a loop of size k was presented.

7 Conclusion

This paper presents an efficient method for bounded realizability checking. We showed that the complexity of the bounded realizability problem is NEXPTIME-complete. In our method, we reduce bounded nonemptiness checking for UCT to SAT-based reachability checking. In many cases in which there exists a small witness or counterexample, the result is obtained by reachability checking of small steps. We implemented our method and demonstrated that it can check the existence of small witnesses or counterexamples efficiently. We believe that our method is of practical use in the requirement analysis phase when developing safety critical systems.

Acknowledgments. This work was supported by JSPS KAKENHI Grant Number 15K15969.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable specifications of reactive systems. In: Ausiello, G., Dezani-Ciancaglini, M., Rocca, S.R.D. (eds.) Automata, Languages and Programming. LNCS, vol. 372, pp. 1–17. Springer, Heidelberg (1989)
2. Aoshima, T., Yonezaki, N.: Verification of reactive system specifications with outer event conditional formula. In: Proc. International Symposium on Principles of Software Evolution, pp. 189–193 (2000)
3. Armoni, R., Fix, L., Fraer, R., Huddleston, S., Piterman, N., Vardi, M.Y.: SAT-based induction for temporal safety properties. Electr. Notes Theor. Comput. Sci. **119**(2), 3–16 (2005)
4. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
5. Boas, P.V.E.: The convenience of tilings. In: Complexity, Logic, and Recursion Theory. Lecture Notes in Pure and Applied Mathematics, vol. 187, pp. 331–363 (1997)
6. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.-F.: Acacia+, a tool for LTL synthesis. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 652–657. Springer, Heidelberg (2012)
7. Chlebus, B.S.: From domino tilings to a new model of computation. In: Skowron, A. (ed.) Computation Theory. LNCS, vol. 208, pp. 24–33. Springer, Heidelberg (1985)
8. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013)

9. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Heidelberg (2014)
10. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
11. Ehlers, R.: Unbeast: symbolic bounded synthesis. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 272–275. Springer, Heidelberg (2011)
12. Filiot, E., Jin, N., Raskin, J.-F.: An antichain algorithm for LTL realizability. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 263–277. Springer, Heidelberg (2009)
13. Finkbeiner, B., Schewe, S.: SMT-based synthesis of distributed systems. In: Proc. Second Workshop on Automated Formal Methods, pp. 69–76 (2007)
14. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: Proc. FMCAD, pp. 117–124 (2006)
15. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: Proc. FOCS, pp. 531–542 (2005)
16. Mochizuki, S., Shimakawa, M., Hagihara, S., Yonezaki, N.: Fast translation from LTL to Büchi automata via non-transition-based automata. In: Merz, S., Pang, J. (eds.) ICFEM 2014. LNCS, vol. 8829, pp. 364–379. Springer, Heidelberg (2014)
17. Mori, R., Yonezaki, N.: Several realizability concepts in reactive objects. In: Proc. Information Modeling and Knowledge Bases IV: Concepts, Methods and Systems, pp. 407–424 (1993)
18. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2006)
19. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. POPL, pp. 179–190 (1989)
20. Rosner, R.: Modular Synthesis of Reactive Systmes. Ph.D. thesis, Weizmann Institute of Science (1992)
21. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 474–488. Springer, Heidelberg (2007)
22. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: Johnson, S.D., Hunt Jr, W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 108–125. Springer, Heidelberg (2000)
23. Shimakawa, M., Hagihara, S., Yonezaki, N.: Complexity of strong satisfiability problems for reactive system specifications. IEICE Trans. Inf. & Syst. **E96–D**(10), 2187–2193 (2013)
24. Shimakawa, M., Hagihara, S., Yonezaki, N.: SAT–based bounded strong satisfiability checking of reactive system specifications. In: Mustofa, K., Neuhold, E.J., Tjoa, A.M., Weippl, E., You, I. (eds.) ICT-EurAsia 2013. LNCS, vol. 7804, pp. 60–70. Springer, Heidelberg (2013)
25. Shimakawa, M., Hagihara, S., Yonezaki, N.: Bounded strong satisfiability checking of reactive system specifications. IEICE Trans. Inf. & Syst. **E97–D**(7), 1746–1755 (2014)
26. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. J. ACM **32**(3), 733–749 (1985)
27. Tauriainen, H.: On translating linear temporal logic into alternating and nondeterministic automata. Research Report A83, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland (2003)

Rearranging Two Dimensional Arrays by Prefix Reversals

Akihiro Yamamura(✉)

Department of Computer Science and Engineering, Akita University, 1-1, Tegata-Gakuenmachi, Akita, Japan
yamamura@ie.akita-u.ac.jp

Abstract. Generalizing the pancake sorting problem, we consider a reachability problem which asks whether an arbitrary two dimensional array can be obtained from an initial array by prefix reversals. In the case of the pancake sorting problem, sorting is always possible, whereas, it is not clear whether a rearrangement of two dimensional arrays is always possible. We shall prove any array is reachable from the initial array by prefix reversals unless the numbers of both rows and columns are divisible by 4. Using group theory, we also give a necessary and sufficient condition that an array is reachable from the initial array in such a case. We also give upper bounds on the number of prefix reversals to rearrange.

Keywords: Pancake sorting problem · Prefix reversals · Rearrangement

1 Introduction

The problem of rearranging by prefix reversals, as known as the *pancake sorting problem*, is to sort randomly piled pancakes of different size. A prefix reversal is an operation of reversing elements in the list including the beginning element. The minimum number of prefix reversals to sort a given list of integers into an ascending order was asked by Gates and Papadimitriou and an upper bound $\frac{5}{3}n$ for sorting a list of length n was given in [6]. They conjectured that $\frac{19}{16}n$ is required, however, it is disproved in [7]. A better bound $\frac{18}{11}n$ was given in [4]. On the other hand, the pancake sorting problem is shown to be NP-hard in [3]. A similar problem of sorting a permutation by reversals, which is related to genomes evolving by inversions, is studied by many authors (e.g. [1,2]). Another variant called the *burnt pancake problem* is studied many authors (e.g. [5,8]).

In this paper we introduce a two dimensional variant of the unburnt pancake sorting problem. We consider a two dimensional array instead of a list. Given an $n \times m$ array filled with integers of different size, we ask whether every permutation of the integers on the array is reachable from an initial array by prefix reversals. A prefix reversal in this setting consists in inserting a spatula vertically or horizontally into the array and rotating a lefthand part or an upper part 180 degree. As we will see, a rearrangement is not always reachable from the initial array. We shall show the reachability depends on the numbers of rows and columns of arrays and on the permutations operated on entries of arrays.

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 153–165, 2015.
DOI: 10.1007/978-3-319-24537-9_14

2 Rearrangement of Two Dimensional Arrays

2.1 Prefix Reversals

We formulate a rearrangement of a two dimensional array by prefix reversals. Suppose A is an $n \times m$ array. Then A comprises of $n \times m$ cells. We denote the entry in the (i, j) position of A by a_{ij} (see Fig 1). We employ the standard matrix representation (a_{ij}) to denote an array A. The *standard array* $E_{n\times m}$ of size $n \times m$ is defined to be (e_{ij}) where e_{ij} is the integer $(i-1) \times m + j$ (see Fig. 2). We denote $A = \frac{A_1}{A_2}$ if A comprises of an upper block A_1 and a lower block A_2, or $A = A_3|A_4$ if A comprises of a left block A_3 and a right block A_4. For an $n \times m$ array $A = (a_{ij})$, the *reversal* of A is the $n \times m$ array (b_{ij}) such that $b_{ij} = a_{n-i+1,m-j+1}$ for every (i, j). We denote it by $R(A)$ (see Fig. 3).

Suppose A is an $n \times m$ array. The transformation $\frac{A_1}{A_2} \Rightarrow \frac{R(A_1)}{A_2}$ is called a *horizontal prefix reversal* and denoted by $hr(1, i)$ if A_1 has i rows. The transformation $A_3|A_4 \Rightarrow R(A_3)|A_4$ is called a *vertical prefix reversal* and denoted by $vr(1, j)$ if A_3 has j columns. The *horizontal suffix reversal* $\frac{A_1}{A_2} \Rightarrow \frac{A_1}{R(A_2)}$ and the *vertical suffix reversal* $A_3|A_4 \Rightarrow A_3|R(A_4)$ are obtained by compositions of prefix reversals $hr(1, n) \circ hr(1, n-i) \circ hr(1, n)$ and $vr(1, m) \circ vr(1, m-j) \circ vr(1, m)$, respectively. We denote them by $hr(i+1, m)$ and $vr(j+1, n)$.

a_{11}	a_{12}	$\cdots$	a_{1m-1}	a_{1m}
a_{21}	a_{22}	$\cdots$	a_{2m-1}	a_{2m}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{n1}	a_{n2}	$\cdots$	a_{nm-1}	a_{nm}

Fig. 1. $n \times m$ array A

1	2	$\cdots$	m
$m+1$	$m+2$	$\cdots$	$2m$
$2m+1$	$2m+2$	$\cdots$	$3m$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(n-1)m+1$	$\cdots$	$\cdots$	nm

Fig. 2. $n\times m$ standard array $E_{n\times m}$

a_{nm}	a_{nm-1}	$\cdots$	a_{n2}	a_{n1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_{2m}	a_{2m-1}	$\cdots$	a_{22}	a_{21}
a_{1m}	a_{1m-1}	$\cdots$	a_{12}	a_{11}

Fig. 3. R(A)

Let us see a concrete example. Suppose A is a 2×3 array $\begin{bmatrix} 3\ 6\ 5 \\ 2\ 4\ 1 \end{bmatrix}$. We can obtain $E_{2\times3}(= \begin{bmatrix} 1\ 2\ 3 \\ 4\ 5\ 6 \end{bmatrix})$ from A by operating compositions of prefix reversals $vr(1, 1) \circ hr(1, 2) \circ hr(1, 1) \circ hr(1, 2) \circ vr(1, 2) \circ hr(1, 1)$;

$$\begin{bmatrix} 3\ 6\ 5 \\ 2\ 4\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 5\ 6\ 3 \\ 2\ 4\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 4\ 2\ 3 \\ 6\ 5\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1\ 5\ 6 \\ 3\ 2\ 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 6\ 5\ 1 \\ 3\ 2\ 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 4\ 2\ 3 \\ 1\ 5\ 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 1\ 2\ 3 \\ 4\ 5\ 6 \end{bmatrix}$$

We should note that $hr(1, 2)$ and $vr(1, 3)$ are the same operation. There are numerous ways to reach $E_{2\times3}$ from A.

2.2 Reachability Problem

We denote the symmetric group of n elements by S_n as usual. It is known that $|S_n| = n!$ and every element of S_n is a product of transpositions, that is, 2-cycles. A permutation σ in S_n is called *even* if σ can be written as a product of an even number of transpositions, otherwise σ is called *odd*. The alternating group of degree n, denoted by A_n, is the subgroup of S_n consisting of all even permutations. It is known that A_n is a proper normal subgroup of S_n for every $n > 1$ and has the order $\frac{n!}{2}$ (see [9]).

Let A be an $n \times m$ array in which an integer in $\{1, 2, 3, \ldots, nm\}$ is placed on each position such that no two distinct positions possess the same number. Suppose that σ is a permutation on the set $\{1, 2, 3, \ldots, nm\}$. The $n \times m$ array obtained from A by operating σ on each number placed on A is called a *rearrangement* of A by σ, and denoted by $\sigma(A)$. The *reachability problem of two dimensional arrays* is to ask whether a given $n \times m$ array $\sigma(E_{n\times m})$, where σ is a permutation of the set $\{1, 2, 3, \ldots, nm\}$, can be rearranged to the standard array $E_{n\times m}$ by vertical and horizontal prefix reversals. If it is reachable, we also ask the minimum number of prefix reversals for all rearrangements. Note that the pancake sorting problem is a reachability problem for $1 \times m$ arrays and always possible to rearrange. Unlike the pancake sorting problem, it is not trivial to decide whether an arbitrary array is reachable from the standard array by prefix reversals.

Let us recall the example above. Let A be the array $\begin{bmatrix} 3 & 6 & 5 \\ 2 & 4 & 1 \end{bmatrix}$. Then we have $A = \begin{bmatrix} \sigma(1) & \sigma(2) & \sigma(3) \\ \sigma(4) & \sigma(5) & \sigma(6) \end{bmatrix} = \sigma(E_{2\times 3})$, where σ is the permutation (1 3 5 4 2 6). As we saw above, the standard array $E_{2\times 3}$ is reachable from A by 6 prefix reversals. In fact we have the following theorem.

Theorem 1. *Let n and m be positive integers.*
(1) Suppose either $n \not\equiv 0 \pmod 4$ or $m \not\equiv 0 \pmod 4$ holds. Given σ in S_{nm}, the standard array $E_{n\times m}$ is reachable from the array $\sigma(E_{n\times m})$ by prefix reversals.
(2) Suppose $n \equiv m \equiv 0 \pmod 4$. Given σ in S_{nm}, the standard array $E_{n\times m}$ is reachable from the array $\sigma(E_{n\times m})$ by prefix reversals if and only if σ is even.

We shall give a sketch of the proof of Theorem 1 in the subsequent sections. We classify the proof into four cases according to the parameter n: (Case 1) $n = 2$ (or $m = 2$), (Case 2) $n \equiv 1, 3 \pmod 4$, $n > 2$ and $m > 2$, (Case 3) $n \equiv 2 \pmod 4$, $n > 2$ and $m > 2$, (Case 4) $n \equiv 0 \pmod 4$ and $m \equiv 0 \pmod 4$. We shall show an arbitrary array is reachable from the standard array unless the number of rows is divisible by 4.

Note that we use horizontal and vertical prefix reversals and so row operations and column operations are symmetrical in the sense that if a rearrangement with respect to rows can be realized then a similar rearrangement with respect to columns can be realized and vice versa provided that the conditions on n and m are same. Thus, we omit the row-column dual cases in this paper for the sake of simplicity.

We discuss the first three cases in Section 3 and the fourth case in Section 4. We also give upper bounds on necessary prefix reversals to rearrange when it is reachable in the last section.

2.3 Elementary Operations

We introduce elementary operations on arrays used frequently in the subsequent sections. We suppose A is an $n \times m$ array throughout this section. It is easy to see that if the composition $hr(1,i) \circ hr(1,1) \circ hr(1,i)$ is operated to A, we obtain an array that coincides with A except for the ith row is reversed. Such an operation is called a *row reversal* and denoted by $rr(i)$. For example, we see the composition $hr(1,2) \circ hr(1,1) \circ hr(1,2)$ reverses the second row of the array $E_{4,4}$ as follows.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \rightarrow \begin{bmatrix} 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & 6 & 7 & 8 \\ 4 & 3 & 2 & 1 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 8 & 7 & 6 & 5 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Similarly, the composition $vr(1,j) \circ vr(1,1) \circ vr(1,j)$ reverses the jth column of A and called the *column reversal* and denoted by $cr(j)$.

Now we operate the composition $hr(n,n) \circ vr(1,1) \circ hr(n,n) \circ hr(1,1) \circ vr(1,1) \circ hr(1,1)$ to A. The resulting array is obtained from A by exchanging the four corners diagonally. For example, we see the composition $hr(4,4) \circ vr(1,1) \circ hr(4,4) \circ hr(1,1) \circ vr(1,1) \circ hr(1,1)$ exchanges the four corners of $E_{4,4}$ diagonally, that is, 1 and 16 are exchanged and 4 and 13 are exchanged, respectively. We call this operation a *corner exchanging* and denote it by ce.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 3 & 2 & 1 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \rightarrow \begin{bmatrix} 13 & 3 & 2 & 1 \\ 9 & 6 & 7 & 8 \\ 5 & 10 & 11 & 12 \\ 4 & 14 & 15 & 16 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 13 \\ 9 & 6 & 7 & 8 \\ 5 & 10 & 11 & 12 \\ 4 & 14 & 15 & 16 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1 & 2 & 3 & 13 \\ 9 & 6 & 7 & 8 \\ 5 & 10 & 11 & 12 \\ 16 & 15 & 14 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 1 & 15 & 14 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$$

We consider certain even permutations on the entries of A. We consider entries on the (i,k), (i,l), (j,k), and (j,l) positions of A ($i \neq j$ and $k \neq l$). These four positions are the intersections of the ith and jth rows and kth and lth columns. The permutations $(a_{i,k}, a_{j,k})(a_{i,l}, a_{j,l})$, $(a_{i,k}, a_{i,l})(a_{j,k}, a_{j,l})$ and $(a_{i,k}, a_{j,l})(a_{i,l}, a_{j,k})$ of entries of A are called a *twin transposition along column*, a *twin transposition along row* and a *twin transposition along diagonal*, respectively. These permutations play an important role in the proof of Theorem 1.

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{i,k} & \cdots & a_{i,l} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{j,k} & \cdots & a_{j,l} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{j,k} & \cdots & a_{j,l} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{i,k} & \cdots & a_{i,l} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \text{ (Twin transposition along column)}$$

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{i,k} & \cdots & a_{i,l} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{j,k} & \cdots & a_{j,l} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{i,l} & \cdots & a_{i,k} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{j,l} & \cdots & a_{j,k} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$ (Twin transposition along row)

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{i,k} & \cdots & a_{i,l} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{j,k} & \cdots & a_{j,l} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{j,l} & \cdots & a_{j,k} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{i,l} & \cdots & a_{i,k} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$ (Twin transposition along diagonal)

Note that the twin transposition $(a_{i,k}, a_{j,k})(a_{i,l}, a_{j,l})$ along column is realized by $hr(j,n) \circ hr(1,i) \circ vr(m-k+1,m) \circ vr(1,m-l+1) \circ hr(1,1) \circ ce \circ hr(1,1) \circ vr(m-k+1,m) \circ vr(1,m-l+1) \circ hr(j,n) \circ hr(1,i)$. It is denoted by $tt((i,k) \leftrightarrow (j,k),(i,l) \leftrightarrow (j,l))$.

Let us see how the transposition (7 17)(9 19) is realized in a 5×5 array. First we reverse the first two rows and the last two rows by $hr(4,5) \circ hr(1,2)$. Second, we reverse the first two columns and the last two columns by $vr(4,5) \circ vr(1,2)$. Then we reverse the first row by $hr(1,1)$. Now we operate a corner exchanging ce. After the corner exchanging, we operate the same operations above in the reverse order, that is, $hr(4,5) \circ hr(1,2) \circ vr(4,5) \circ vr(1,2) \circ hr(1,1)$. By these operations, the twin transposition along column (7 17)(9 19) is realized.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & \boxed{7} & 8 & ⑨ & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & \boxed{17} & 18 & ⑲ & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & ⑨ & 8 & \boxed{7} & 6 \\ 5 & 4 & 3 & 2 & 1 \\ 11 & 12 & 13 & 14 & 15 \\ 25 & 24 & 23 & 22 & 21 \\ 20 & ⑲ & 18 & \boxed{17} & 16 \end{bmatrix} \rightarrow \begin{bmatrix} ⑲ & 20 & 8 & 16 & \boxed{17} \\ 24 & 25 & 3 & 21 & 22 \\ 12 & 11 & 13 & 15 & 14 \\ 4 & 5 & 23 & 1 & 2 \\ ⑨ & 10 & 18 & 6 & \boxed{7} \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} \boxed{17} & 16 & 8 & 20 & ⑲ \\ 24 & 25 & 3 & 21 & 22 \\ 12 & 11 & 13 & 15 & 14 \\ 4 & 5 & 23 & 1 & 2 \\ ⑨ & 10 & 18 & 6 & \boxed{7} \end{bmatrix} \rightarrow \begin{bmatrix} \boxed{7} & 16 & 8 & 20 & ⑨ \\ 24 & 25 & 3 & 21 & 22 \\ 12 & 11 & 13 & 15 & 14 \\ 4 & 5 & 23 & 1 & 2 \\ ⑲ & 10 & 18 & 6 & \boxed{17} \end{bmatrix} \rightarrow \begin{bmatrix} ⑨ & 20 & 8 & 16 & \boxed{7} \\ 24 & 25 & 3 & 21 & 22 \\ 12 & 11 & 13 & 15 & 14 \\ 4 & 5 & 23 & 1 & 2 \\ ⑲ & 10 & 18 & 6 & \boxed{17} \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 10 & ⑲ & 8 & \boxed{17} & 6 \\ 5 & 4 & 3 & 2 & 1 \\ 11 & 12 & 13 & 14 & 15 \\ 25 & 24 & 23 & 22 & 21 \\ 20 & ⑨ & 18 & \boxed{7} & 16 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & \boxed{17} & 8 & ⑲ & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & \boxed{7} & 18 & ⑨ & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix}$$

Similarly, the twin transposition along row $(a_{i,k}, a_{i,l})(a_{j,k}, a_{j,l})$ is realized by $hr(j,n) \circ hr(1,i) \circ vr(m-k+1,m) \circ vr(1,m-l+1) \circ vr(1,1) \circ ce \circ vr(1,1) \circ vr(m-k+1,m) \circ vr(1,m-l+1) \circ hr(j,n) \circ hr(1,i)$, and the twin transposition along diagonal $(a_{i,k}, a_{j,l})(a_{i,l}, a_{j,k})$ is realized by $hr(j,n) \circ hr(1,i) \circ vr(m-k+1,m) \circ vr(1,m-l+1) \circ ce \circ vr(m-k+1,m) \circ vr(1,m-l+1) \circ hr(j,n) \circ hr(1,i)$. The twin transpositions

along row and diagonal are denoted by $tt((i,k) \leftrightarrow (i,l),(j,k) \leftrightarrow (j,l))$ and $tt((i,k) \leftrightarrow (j,l),(i,l) \leftrightarrow (j,k))$, respectively.

3 Reachable Arrays

If $n = 1$ or $m = 1$, the reachability problem of $n \times m$ arrays is equivalent to the pancake sorting problem and so it is always possible. In this case, rearrangement requires at most $\frac{18}{11}n$ horizontal prefix reversals for an $n \times 1$ array and at most $\frac{18}{11}m$ vertical prefix reversals for a $1 \times m$ array, respectively, by [4]. Thus, we may assume $n > 1$ and $m > 1$. In Section 3.1, we examine the case of $n = 2$. The case of $m = 2$ can be shown similarly. Then we consider the cases $n \equiv 1, 2, 3 \pmod 4$ assuming $n > 2$ and $m > 2$ after Section 3.1.

We recall that row operations and column operations are symmetrical in the sense that if a rearrangement with respect to rows can be realized then a similar rearrangement with respect to columns can be realized and vice versa provided that the conditions on n and m are same. Therefore the cases that $m \equiv 1, 2, 3 \pmod 4$ can be shown similarly and we will not discuss these cases.

3.1 $n = 2$

We shall show how to rearrange a $2 \times m$ array A ($m \geq 2$) to the standard array $E_{2\times m}$ by prefix reversals. First, we show how to exchange two entries on the $(1,p)$ and $(1,q)$ positions ($p < q$). We operate $hr(2,2) \circ vr(1,1) \circ vr(q,m) \circ vr(1,p)$ to A. Then two entries on the $(1,p)$ and $(1,q)$ positions are moved to the $(1,1)$ and $(2,1)$ positions. Then we operate $vr(1,1)$. After that we operate $vr(1,p) \circ vr(q,m) \circ vr(1,1) \circ hr(2,2)$. Then we realize the transposition of two entries on the $(1,p)$ and $(1,q)$ positions.

Let us see a concrete example, where we exchange 2 and 4.

$$\begin{bmatrix} 1 & \textcircled{2} & 3 & \boxed{4} & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix} \to \begin{bmatrix} 8 & 7 & 3 & 12 & 11 & 10 \\ \textcircled{2} & 1 & 9 & 6 & 5 & \boxed{4} \end{bmatrix} \to \begin{bmatrix} \textcircled{2} & 7 & 3 & 12 & 11 & 10 \\ 8 & 1 & 9 & 6 & 5 & \boxed{4} \end{bmatrix} \to$$

$$\begin{bmatrix} \textcircled{2} & 7 & 3 & 12 & 11 & 10 \\ \boxed{4} & 5 & 6 & 9 & 1 & 8 \end{bmatrix} \to \begin{bmatrix} \boxed{4} & 7 & 3 & 12 & 11 & 10 \\ \textcircled{2} & 5 & 6 & 9 & 1 & 8 \end{bmatrix} \to \begin{bmatrix} \boxed{4} & 7 & 3 & 12 & 11 & 10 \\ 8 & 1 & 9 & 6 & 5 & \textcircled{2} \end{bmatrix} \to$$

$$\begin{bmatrix} 8 & 7 & 3 & 12 & 11 & 10 \\ \boxed{4} & 1 & 9 & 6 & 5 & \textcircled{2} \end{bmatrix} \to \begin{bmatrix} 1 & \boxed{4} & 3 & \textcircled{2} & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix}$$

It is easy to see that we can similarly exchange when two entries are on different rows. Therefore we can realize any transposition by prefix reversals. Since every permutation is a composition of transpositions, we can realize any permutation by prefix reversals. Consequently, every array is reachable from any other in this case.

We remark that we do not use any twin transpositions in this case although we will see we use considerable number of twin transpositions in the other cases.

3.2 $n \equiv 1, 3 \pmod 4$

We have $n = 2k+1$ for some positive integer k. We shall show any transposition is realized by prefix reversals.

Horizontal Transposition of $(k+1, 1)$ and $(k+1, 2)$ Entries. Let us call a permutation exchanging two elements on a certain row (on a certain column) a *horizontal transposition* (*vertical transposition*, respectively).

We shall show how to realize a horizontal transposition of entries on the $(k+1, 1)$ and $(k+1, 2)$ positions in a $(2k+1) \times m$ array A. There are $2k$ rows except for the $k+1$ row in A. First, we operate k twin transpositions along diagonal $tt((i, 1) \leftrightarrow (n-i+1, 2), (i, 2) \leftrightarrow (n-i+1, 1))$ for every $i = 1, 2, 3, \ldots, k$. Second, we operate $vr(1, 2)$. The resulting array is obtained from A by transposing the entries of $(k+1, 1)$ and $(k+1, 2)$ positions.

Let us see a concrete example. We realize a transposition (7 8) in a $5 \times m$ array below. So $k = 2$ in this case. First, we operate 2 twin transpositions along diagonal; $tt((1, 1) \leftrightarrow (5, 2), (1, 2) \leftrightarrow (5, 1))$ and $tt((2, 1) \leftrightarrow (4, 2), (2, 2) \leftrightarrow (4, 1))$. Second, we operate $vr(1, 2)$ and the desired transposition (7 8) is realized. Colored prints below indicate the positions where the twin transpositions are carried out.

$$\begin{bmatrix} 1 & 2 & 3 & \cdots \\ 4 & 5 & 6 & \cdots \\ ⑦ & ⑧ & 9 & \cdots \\ 10 & 11 & 12 & \cdots \\ 13 & 14 & 15 & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} 14 & 13 & 3 & \cdots \\ 4 & 5 & 6 & \cdots \\ ⑦ & ⑧ & 9 & \cdots \\ 10 & 11 & 12 & \cdots \\ 2 & 1 & 15 & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} 14 & 13 & 3 & \cdots \\ 11 & 10 & 6 & \cdots \\ ⑦ & ⑧ & 9 & \cdots \\ 5 & 4 & 12 & \cdots \\ 2 & 1 & 15 & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & \cdots \\ 4 & 5 & 6 & \cdots \\ ⑧ & ⑦ & 9 & \cdots \\ 10 & 11 & 12 & \cdots \\ 13 & 14 & 15 & \cdots \end{bmatrix}$$

Transposition of Entries on Arbitrary Positions. We show how to exchange two entries on arbitrary positions (i, p) and (j, q). We move the entries to the $(k+1, 1)$ and $(k+1, 2)$, respectively using twin transpositions. If (i, p) is $(k+1, 1)$ or $(k+1, 2)$ then we do not have to operate any permutation to move. Likewise, if (j, q) is $(k+1, 1)$ or $(k+1, 2)$ then we do not have to operate any permutation to move. So we suppose that both are neither $(k+1, 1)$ nor $(k+1, 2)$.

Suppose that $k+1 \neq i$ and $1 \neq p$. We operate the twin transposition $tt((k+1, 1) \leftrightarrow (i, p), (i, 1) \leftrightarrow (k+1, p))$ along diagonal. Suppose that $k+1 = i$ and $1 \neq p$, that is $(i, p) = (k+1, p)$. We choose any row $1 \leq h \leq n$ such that $h \neq k+1$. Then we operate the twin transposition $tt((k+1, 1) \leftrightarrow (k+1, p), (h, 1) \leftrightarrow (h, p))$ along row. Next suppose that $k+1 \neq i$ and $1 = p$, that is, $(i, p) = (i, 1)$. We choose any column $1 \leq g \leq m$ such that $g \neq 1$ nor $g \neq 2$. Then we operate the twin transposition $tt((k+1, 1) \leftrightarrow (i, 1), (k+1, g) \leftrightarrow (i, g))$ along column. The entry on the (i, p) position is exchanged with the entry on the $(k+1, 1)$ position in any case.

Suppose that $k+1 \neq j$ and $2 \neq q$. We operate the twin transposition $tt((k+1, 2) \leftrightarrow (j, q), (j, 2) \leftrightarrow (k+1, q))$ along diagonal. Suppose that $k+1 = j$ and $2 \neq q$, that is $(j, q) = (k+1, 2)$. We choose any row $1 \leq h \leq n$ such that $h \neq k+1$.

Then we operate the twin transposition $tt((k+1,2) \leftrightarrow (j,q),(h,2) \leftrightarrow (h,q))$ along row. Next suppose that $k+1 \neq j$ and $2 = q$, that is, $(j,q) = (j,2)$. We choose any column $1 \leq g \leq m$ such that $g \neq 1$ nor $g \neq 2$. Then we operate the twin transposition $tt((k+1,2) \leftrightarrow (j,q),(k+1,g) \leftrightarrow (j,g))$ along column. The entry on the (j,q) position is exchanged with the entry on the $(k+1,2)$ position in any case.

After the two twin transpositions above, we exchange the entries on $(k+1,1)$ and $(k+1,2)$ positions by the operation above. Then we operate the same two twin transpositions in the reverse order. Now we realize the transposition of the entries of (i,p) and (j,q) positions. Note that we move several entries other than $(k+1,1)$, $(k+1,2)$, (i,p) and (j,q) positions in the course of operations, however, the other entries stay the same position after all the operations are carried out because we operate the same operations twice in the reverse order.

See how the operations transform the array and exchange $a_{i,p}$ and $a_{j,q}$ when $k+1 \neq i$, $p \neq 1,2$, $k+1 \neq j$ and $q \neq 1,2$. Colored print letters indicate the corresponding twin transpositions. It is routine to see the other cases, e.g. $k+1 = i$ or $p = 1$, also work using twin transpositions along rows and columns.

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i,1} & \cdots & \cdots & a_{i,p} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k+1,1} & a_{k+1,2} & \cdots & a_{k+1,p} & \cdots & a_{k+1,q} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{j,2} & \cdots & \cdots & \cdots & a_{j,q} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k+1,p} & \cdots & \cdots & a_{k+1,1} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i,p} & a_{j,q} & \cdots & a_{i,1} & \cdots & a_{j,2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{k+1,q} & \cdots & \cdots & \cdots & a_{k+1,2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k+1,p} & \cdots & \cdots & a_{k+1,1} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{j,q} & a_{i,p} & \cdots & a_{i,1} & \cdots & a_{j,2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{k+1,q} & \cdots & \cdots & \cdots & a_{k+1,2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i,1} & \cdots & \cdots & a_{j,q} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{k+1,1} & a_{k+1,2} & \cdots & a_{k+1,p} & \cdots & a_{k+1,q} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & a_{j,2} & \cdots & \cdots & \cdots & a_{i,p} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

3.3 $n \equiv 2 \pmod 4$

We have $n = 4k+2$ for some positive integer k. We shall show any transposition is realized by prefix reversals.

Vertical Transposition of $(2k+1,1)$ and $(2k+2,1)$ Entries. We show how to realize a vertical transposition of entries on the $(2k+1,1)$ and $(2k+2,1)$ positions in a $(4k+2,m)$ array. We assume that the size m is larger than 2, otherwise the transposition can be realized by prefix reversals as we saw in Section 3.1.

We operate $vr(1,1)$ to exchange entries on the $(2k+1,1)$ and $(2k+2,1)$ positions, however, this operation also moves the other entries on the first column. We restore the entries on the first column except for $(2k+1,1)$ and $(2k+2,1)$ positions by iterating the following operations for every $1 \leq i \leq k$. The basic idea is to exchange the entries on the $(2i,1)$ and $(4k-2i+3,1)$ positions and $(2i-1,1)$ and $(4k-2i+4,1)$ positions, respectively by moving these four entries in positions for a twin transposition and then exchange couples of these entries, simultaneously.

First, we operate $rr(4k-2i+3) \circ rr(2i)$. Second, we operate $tt((2i,m-1) \leftrightarrow (2i-1,m-1),(2i,m) \leftrightarrow (2i-1,m)) \circ tt((4k-2i+3,m-1) \leftrightarrow (4k-2i+4,m-1),(4k-2i+3,m) \leftrightarrow (4k-2i+4,m))$. These two operations move the entries on the $(2i,1)$ and $(4k-2i+3,1)$ positions and $(2i-1,1)$ and $(4k-2i+4,1)$ positions in positions for a twin transposition. Third, we operate $tt((2i-1,1) \leftrightarrow (4k-2i+4,1),(2i-1,m) \leftrightarrow (4k-2i+4,m))$. This operation realizes the exchange of targeted entries. Fourth, we carry out the same operations in the reverse order, that is, we operate $tt((2i,m-1) \leftrightarrow (2i-1,m-1),(2i,m) \leftrightarrow (2i-1,m)) \circ tt((4k-2i+3,m-1) \leftrightarrow (4k-2i+4,m-1),(4k-2i+3,m) \leftrightarrow (4k-2i+4,m))$. Lastly we operate $rr(4k-2i+3) \circ rr(2i)$. Then the transposition $(a_{2k+1,1}\ a_{2k+2,1})$ is realized. In the iteration process, entries other than the targeted entries are not replaced because we operate the same operation twice.

Let us see an example. We consider 6×4 array (the case of $k=1$) below and aim at transposing 9 and 13, where 9 is on the $(2k+1,1)$ position and 13 is on the $(2k+2,1)$ position. The other elements on the first column are 1, 5, 17 and 21. First, we reverse the first column and then relocate 1, 5, 17 and 21 in the positions where we can use a twin transposition to exchange each other. We reverse the second and the fifth row, and carry out two twin transpositions (3 6)(4 17) and (18 23)(5 24). Then we carry out the twin transposition (1 21)(5 17). After that, we carry out the same operations in the reverse order and we realized the transposition (9 13).

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ ⑨ & 10 & 11 & 12 \\ ⑬ & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \end{bmatrix} \rightarrow \begin{bmatrix} 21 & 2 & 3 & 4 \\ 17 & 6 & 7 & 8 \\ ⑬ & 10 & 11 & 12 \\ ⑨ & 14 & 15 & 16 \\ 5 & 18 & 19 & 20 \\ 1 & 22 & 23 & 24 \end{bmatrix} \rightarrow \begin{bmatrix} 21 & 2 & 3 & 4 \\ 8 & 7 & 6 & 17 \\ ⑬ & 10 & 11 & 12 \\ ⑨ & 14 & 15 & 16 \\ 20 & 19 & 18 & 5 \\ 1 & 22 & 23 & 24 \end{bmatrix} \rightarrow \begin{bmatrix} 21 & 2 & 6 & 17 \\ 8 & 7 & 3 & 4 \\ ⑬ & 10 & 11 & 12 \\ ⑨ & 14 & 15 & 16 \\ 20 & 19 & 23 & 24 \\ 1 & 22 & 18 & 5 \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 1 & 2 & 6 & 5 \\ 8 & 7 & 3 & 4 \\ ⑬ & 10 & 11 & 12 \\ ⑨ & 14 & 15 & 16 \\ 20 & 19 & 23 & 24 \\ 21 & 22 & 18 & 17 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 8 & 7 & 6 & 5 \\ ⑬ & 10 & 11 & 12 \\ ⑨ & 14 & 15 & 16 \\ 20 & 19 & 18 & 17 \\ 21 & 22 & 23 & 24 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ ⑬ & 10 & 11 & 12 \\ ⑨ & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \end{bmatrix}$$

Transposition of Entries of Arbitrary Positions. We can move entries on any positions to the $(2k+1,1)$ and $(2k+2,1)$ positions, respectively, using two

twin transpositions by an argument similar to the one in Section 3.2. Then we carry out the transposition of the entries on the $(2k+1, 1)$ and $(2k+2, 1)$ positions as we described above. After that we operate the same twin transpositions in the reverse order. Then we can exchange the targeted two entries without affecting positions of the other entries and so we can realize an arbitrary transposition.

Remark 1. We exclude the case that $n = 2$ (the case of $k = 0$) from this section and considered it in Section 3.1. Our proof of rearrangement in this section relies on moving two entries at arbitrary positions to the $(2k + 1, 1)$ and $(2k + 2, 1)$ positions using twin transpositions. Since we do not have enough rows to employ twin transpositions in the case of $n = 2$, it is impossible to move two entries to the $(2k + 1, 1)$ and $(2k + 2, 1)$ positions simultaneously by twin transpositions. On the other hand, we can directly replace two entries at arbitrary positions on the $(1, 1)$ and $(2, 1)$ positions using prefix reversals in Section 3.1.

4 Unreachable Arrays

In this section, we discuss unreachability of an $n \times m$ array when $n \equiv 0 \pmod 4$ and $m \equiv 0 \pmod 4$. We shall show that for a given permutation σ the standard array $E_{4k\times 4h}$ is reachable from the array $\sigma(E_{4k\times 4h})$ by prefix reversals if and only if σ is even. We suppose A is an $n \times m$ array, where $n = 4h$ and $m = 4k$ for some positive integers h, k in this section.

4.1 Unreachability of $\sigma(E_{4h\times 4k})$ for Any Odd Permutation σ

We shall show that a permutation realized by prefix reversals is always even. We consider the horizontal prefix reversal $hr(1, p)$ for an arbitrary $1 \le p \le 4k$. The sub-array operated by $hr(1, p)$ is shown below:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,2h} & a_{1,2h+1} & \cdots & a_{1,4h-1} & a_{1,4h} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,2h} & a_{2,2h+1} & \cdots & a_{2,4h-1} & a_{2,4h} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{p-1,1} & a_{p-1,2} & \cdots & a_{p-1,2h} & a_{p-1,2h+1} & \cdots & a_{p-1,4h-1} & a_{p-1,4h} \\ a_{p,1} & a_{p,2} & \cdots & a_{p,2h} & a_{p,2h+1} & \cdots & a_{p,4h-1} & a_{p,4h} \end{bmatrix}$$

After operating $hr(1, p)$ to A, the upper p rows of A turns out to be

$$\begin{bmatrix} a_{p,4h} & a_{p,4h-1} & \cdots & a_{p,2h+1} & a_{p,2h} & \cdots & a_{p,2} & a_{p,1} \\ a_{p-1,4h} & a_{p-1,4h-1} & \cdots & a_{p-1,2h+1} & a_{p-1,2h} & \cdots & a_{p-1,2} & a_{p-1,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{2,4h} & a_{2,4h-1} & \cdots & a_{2,2h+1} & a_{2,2h} & \cdots & a_{2,2} & a_{2,1} \\ a_{1,4h} & a_{1,4h-1} & \cdots & a_{1,2h+1} & a_{1,2h} & \cdots & a_{1,2} & a_{1,1} \end{bmatrix}$$

It is a routine to see that $hr(1,p)$ generates a permutation which is a product of disjoint transpositions $(a_{w,x}, a_{y,z})$ where $1 \leq w, y \leq p$, $1 \leq x \leq 2h$, and $2h+1 \leq z \leq 4h$ satisfying $w+y = p+1$ and $x+z = 4h+1$. Note that there are $p \times 2h$ elements $a_{w,x}$ satisfying $1 \leq w \leq p$ and $1 \leq x \leq 2h$, and also there are $p \times 2h$ elements $a_{y,z}$ satisfying $1 \leq y \leq p$ and $2h+1 \leq z \leq 4h$. Hence, the permutation realized by $hr(1,p)$ is a product of $p \times 2h$ disjoint transpositions and so it is even.

Similarly a permutation realized by any vertical prefix reversal $vr(1,q)$ is even. Therefore every permutation realized by prefix reversals is a product of even permutations, and so, it is contained in the alternating group A_{16ij} which is a proper subgroup of the symmetric group S_{16ij}. Consequently, prefix reversals cannot realize any odd permutation on the entries of a $4h \times 4k$ array.

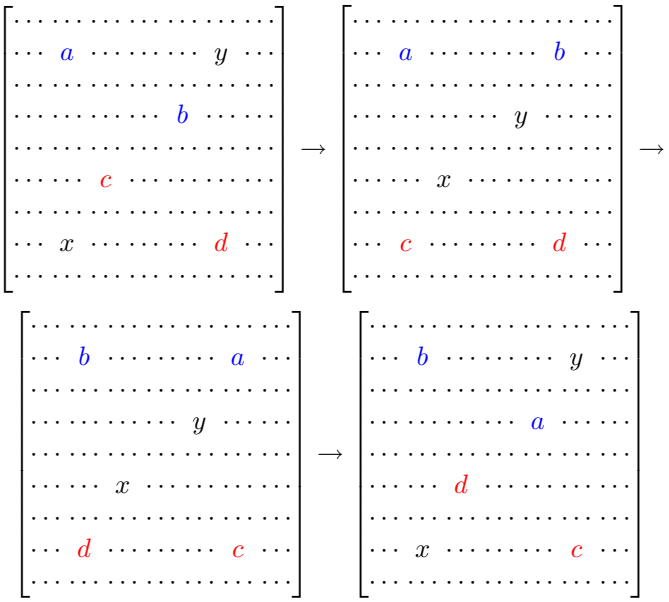
4.2 Reachability of $\sigma(E_{4h\times 4k})$ for Any Even Permutation σ

In Section 4.1, we have shown that prefix reversals can realize only even permutations. We now ask whether any even permutation σ can be realized by prefix reversals. We shall answer this question affirmatively in this section.

Suppose σ is factored as $\tau_1\tau_2\cdots\tau_{2r}$ where τ_i $(1 \leq i \leq 2r)$ is a transposition. If we could prove that $E_{4h\times 4k}$ is reachable from the array $\rho(E_{4h\times 4k})$, where $\rho = \tau_1\tau_2$ and τ_1 and τ_2 are transpositions, by prefix reversals, then the general case can be obtained by induction. Therefore, we consider the case that $\sigma = \tau_1\tau_2$, where τ_1 and τ_2 are transposition. There are two cases to be considered: (1) τ_1 and τ_2 are disjoint and (2) τ_1 and τ_2 are not disjoint.

Suppose τ_1 and τ_2 are not disjoint, that is the case (2). We have $\rho = \tau_1\tau_2 = \tau_1\delta\delta\tau_2$ for any transposition δ because the inverse of a transposition is itself. In particular, we may take any transposition δ that is disjoint both from τ_1 and from τ_2. Then both $\tau_1\delta$ and $\delta\tau_2$ fall in the case (1). Therefore, it suffices to consider only the case (1). Consequently, we shall show that $E_{4h\times 4k}$ is reachable from $\rho(E_{4h\times 4k})$ by prefix reversals if the permutation ρ factors as $\tau_1\tau_2$, where τ_1 and τ_2 are disjoint transpositions.

Suppose that $\rho = \tau_1\tau_2$, $\tau_1 = (a,b)$, $\tau_2 = (c,d)$ and a,b,c,d are distinct from each other (see the array transition below). We suppose that x is placed on the same column as a and the same row as d and that y is placed on the same column as d and on the same row as a. Then we operate two twin transpositions along row, column or diagonal to exchange b and y, and c and x, respectively. Note that since we use twin transpositions, there are other entries replaced. Next we operate twin transpositions $(a\ b)(c\ d)$ to exchange a and b and c and d, respectively. After that we operate the same twin transpositions above in the reverse order. Note that any entries except for a,b,c,d stay the same position because the operations are carried out twice. Therefore, ρ is realized by prefix reversals. It follows that $E_{4h\times 4k}$ is reachable from an array $\sigma(E_{4h\times 4k})$ by prefix reversals if σ is even.



We only considered the case that a, b, c, d are on different rows and columns. Similarly we can show that any composition of disjoint transpositions is realized by twin transpositions as well. It follows that prefix reversals realize all the even permutations. Consequently, the rearranging problem is reachable if and only if σ is even in the case that $n \equiv 0 \pmod 4$ and $m \equiv 0 \pmod 4$.

5 Upper Bound of Prefix Reversals

We give upper bounds on the number of prefix reversals to rearrange.

Theorem 2. *Upper bounds on the number of prefix reversals to rearrange the array $\sigma(E_{n\times m})$ to $E_{n\times m}$ $(n \geq 2)$ is given as follows.*

(1) For $n = 2$ and any m, an upper bound is given by $34m - 17$.

(2) For $n \equiv 1, 3 \pmod 4$, $n > 2$ and $m > 2$, an upper bound is given by $(13n + 100) \times (nm - 1)$.

(3) For $n \equiv 2 \pmod 4$, $n > 2$ and $m > 2$, an upper bound is given by $(38n + 37) \times (nm - 1)$.

(4) For $n \equiv 0 \pmod 4$, $m \equiv 0 \pmod 4$ and an even permutation σ, an upper bound is given by $280 \times (nm - 1)$.

We omit the proof of Theorem 2 due to lack of space. We note that $\frac{18}{11}m$ vertical prefix reversals is required in the case of $n = 1$, that is, for a $1 \times m$ array by the result in [4].

Remark 2. Our estimation is based on our proof of Theorem 1 and the obtained upper bounds are not tight. For example, only 6 prefix reversals are enough to rearrange the example in Section 2.1:

$$\begin{bmatrix}3&6&5\\2&4&1\end{bmatrix} \Rightarrow \begin{bmatrix}5&6&3\\2&4&1\end{bmatrix} \Rightarrow \begin{bmatrix}4&2&3\\6&5&1\end{bmatrix} \Rightarrow \begin{bmatrix}1&5&6\\3&2&4\end{bmatrix} \Rightarrow \begin{bmatrix}6&5&1\\3&2&4\end{bmatrix} \Rightarrow \begin{bmatrix}4&2&3\\1&5&6\end{bmatrix} \Rightarrow \begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$$

While the upper bound we give is $34 \times 3 - 17 = 85$. Therefore, our upper bound seems much larger than a plausible upper bound. As a matter of fact, our estimation of upper bound is very gross. We use numerous twin transpositions to prove reachability because it makes proofs shorter and clearer. However, twin transpositions consume many prefix reversals. We did not employ any elaborate counting argument, while there are several methods to avoid using twin transpositions. Furthermore, we did not classify the parameter m like the parameter n. There is plenty room for reducing the number of prefix reversals if we consider more cases depending on not only n but also m.

Acknowledgments. The author would like to thank anonymous referees whose comments have led to a clearer presentation and improved the paper substantially.

References

1. Berman, P., Karpinski, M.: On some tighter inapproximability results. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 200–209. Springer, Heidelberg (1999)
2. Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-approximation algorithm for sorting by reversals. In: Möhring, R., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 200–210. Springer, Heidelberg (2002)
3. Bulteau, L., Fertin, G., Rusu, I.: Pancake flipping is hard. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 247–258. Springer, Heidelberg (2012)
4. Chitturi, B., Fahle, W., Meng, Z., Morales, L., Shields, C.O., Sudborough, I.H., Voit, W.: An $(18/11)n$ upper bound for rearranging by prefix reversals. Theoretical Computer Science **410**(36), 3372–3390 (2009)
5. Cohen, D.S., Blum, M.: On the problem of rearranging burnt pancakes. Discrete Applied Mathematics **61**(2), 105–120 (1995)
6. Gates, W., Papadimitriou, C.: Bounds for Sorting by Prefix Reversal. Discrete Mathematics **79**, 47–57 (1979)
7. Heydari, M.H., Sudborough, I.H.: On the diameter of the pancake network. J. Algorithms **25**(1), 67–94 (1997)
8. Kaplan, H., Shamir, R., Tarjan, R.E.: Faster and simpler algorithm for sorting signed permutations by reversals. In: ACM-SIAM SODA 1997, pp. 178–187 (1997)
9. Rotman, J.J.: An Introduction to the Theory of Groups, 4th edn. Springer (1994)

The Emptiness Problem for Valence Automata or: Another Decidable Extension of Petri Nets

Georg Zetzsche(✉)

Concurrency Theory Group, Fachbereich Informatik,
Technische Universität Kaiserslautern, Kaiserslautern, Germany
zetzsche@cs.uni-kl.de

Abstract. This work studies which storage mechanisms in automata permit decidability of the reachability problem. The question is formalized using valence automata, an abstract model that generalizes automata with storage. For each of a variety of storage mechanisms, one can choose a (typically infinite) monoid M such that valence automata over M are equivalent to (one-way) automata with this type of storage.

In fact, many interesting storage mechanisms can be realized by monoids defined by finite graphs, called graph monoids. Hence, we study for which graph monoids the emptiness problem for valence automata is decidable. A particular model realized by graph monoids is that of Petri nets with a pushdown stack. For these, decidability is a long-standing open question and we do not answer it here.

However, if one excludes subgraphs corresponding to this model, a characterization can be achieved. This characterization yields a new extension of Petri nets with a decidable reachability problem. Moreover, we provide a description of those storage mechanisms for which decidability remains open. This leads to a natural model that generalizes both pushdown Petri nets and priority multicounter machines.

1 Introduction

For each storage mechanism in one-way automata, it is an important question whether the reachability problem is decidable. It therefore seems prudent to aim for general insights into which properties of storage mechanisms are responsible for decidability or undecidability.

Our approach to obtain such insights is the model of valence automata. These consist of a finite-state control and a (typically infinite) monoid that represents a storage mechanisms. The edge inscriptions consist of an input word and an element of the monoid. Then, a computation is accepting if it arrives in a final state and composing the encountered monoid elements yields the neutral element. This way, by choosing a suitable monoid, one can realize a variety of storage mechanisms as valence automata. Hence, our question becomes: *For which monoids M is the emptiness problem for valence automata over M decidable?*

We address this question for a class of monoids that was introduced in [12] and accommodates a number of storage mechanisms that have been studied in

M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 166–178, 2015.
DOI: 10.1007/978-3-319-24537-9_15

automata theory. Examples include *pushdown stacks*, *partially blind counters* (which behave like Petri net places), and *blind counters* (which may attain negative values; these are in most situations interchangeable with reversal-bounded counters), and combinations thereof. These monoids are defined by graphs and thus called *graph monoids*[1].

A particular type of storage mechanism that can be realized by graph monoids are partially blind counters that can be used simultaneously with a pushdown stack. Automata with such a storage are equivalent to *pushdown Petri nets (PPN)*, i.e. Petri nets where the transitions can also operate on a pushdown stack. This means, a complete characterization of graph monoids with a decidable emptiness problem would entail an answer to the long-standing open question of whether reachability is decidable for this Petri net extension [8].

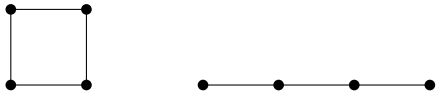
Contribution. While this work does not answer this open question concerning PPN, it does provide a characterization among all graph monoids that avoid this elusive storage type. More precisely, we identify a set of graphs, 'PPN-graphs', each of which corresponds precisely to PPN. Then, among all graphs Γ avoiding PPN-graphs as induced subgraphs, we characterize those for which the graph monoid $\mathbb{M}\Gamma$ results in a decidable emptiness problem. Furthermore, we provide a simple, more mechanical (as opposed to algebraic) description of (i) the storage mechanism emerging as the most general decidable case and (ii) a mechanism subsuming the cases we leave open. The model (i) is a new extension of partially blind counter automata (i.e. Petri nets). While the decidability proof employs a reduction to Reinhardt's priority multicounter machines [8], the model (i) seems to be expressively incomparable to Reinhardt's model. The model (ii) is a class of mechanisms whose simplest instance are the pushdown Petri nets and which also subsumes priority multicounter machines.

Hence, although the results here essentially combine previous work, the perspective of valence automata allows us to identify two natural storage mechanisms that (i) push the frontier of decidable reachability and (ii) let us interpret PPN and priority multicounter machines as special cases of a more powerful model that might enjoy decidability, respectively.

2 Preliminaries

A *monoid* is a set M together with a binary associative operation such that M contains a neutral element. Unless the monoid at hand warrants a different notation, we will denote the neutral element by 1 and the product of $x, y \in M$ by xy. If X is a set of symbols, X^* denoted the set of words over X. An *alphabet* is a finite set of symbols. The empty word is denoted by $\varepsilon \in X^*$.

[1] They are not to be confused with the closely related, but different concept of *trace monoids* [2], i.e. monoids of Mazurkiewicz traces, which some authors also call graph monoids.



(a) C_4 (b) P_4

Fig. 1. Graphs C_4 and P_4.

Valence Automata. As a framework for studying which storage mechanisms permit decidability of the reachability problem, we employ valence automata. They feature a monoid that dictates which computations are valid. Hence, by an appropriate choice of the monoid, valence automata can be instantiated to be equivalent to a concrete automata model with storage. For the purposes of this work, *equivalent* is meant with respect to accepted languages. Therefore, we regard valence automata as language accepting devices.

Let M be a monoid and X an alphabet. A *valence automaton over* M is a tuple $A = (Q, X, M, E, q_0, F)$, in which (i) Q is a finite set of *states*, (ii) E is a finite subset of $Q \times X^* \times M \times Q$, called the set of *edges*, (iii) $q_0 \in Q$ is the *initial state*, and (iv) $F \subseteq Q$ is the set of *final states*. For $q, q' \in Q$, $w, w' \in X^*$, and $m, m' \in M$, we write $(q, w, m) \to_A (q', w', m')$ if there is an edge $(q, v, n, q') \in E$ such that $w' = wv$ and $m' = mn$. The language *accepted* by A is then

$$\mathsf{L}(A) = \{w \in X^* \mid (q_0, \varepsilon, 1) \to_A^* (f, w, 1) \text{ for some } f \in F\}.$$

The class of languages accepted by valence automata over M is denoted by $\mathsf{VA}(M)$. If $\mathcal{M}$ is a class of monoids, we write $\mathsf{VA}(\mathcal{M})$ for $\bigcup_{M \in \mathcal{M}} \mathsf{VA}(M)$.

Graphs. A *graph* is a pair $\Gamma = (V, E)$ where V is a finite set and E is a subset of $\{S \subseteq V \mid 1 \leq |S| \leq 2\}$. The elements of V are called *vertices* and those of E are called *edges*. Vertices $v, w \in V$ are *adjacent* if $\{v, w\} \in E$. If $\{v\} \in E$ for some $v \in V$, then v is called a *looped* vertex, otherwise it is *unlooped*. A *subgraph* of Γ is a graph (V', E') with $V' \subseteq V$ and $E' \subseteq E$. Such a subgraph is called *induced (by V')* if $E' = \{S \in E \mid S \subseteq V'\}$, i.e. E' contains all edges from E incident to vertices in V'. By $\Gamma \setminus \{v\}$, for $v \in V$, we denote the subgraph of Γ induced by $V \setminus \{v\}$. By C_4 (P_4), we denote a graph that is a cycle (path) on four vertices; see figs. 1a and 1b. Moreover, a *clique* is a loop-free graph in which any two distinct vertices are adjacent. Finally, Γ^- denotes the graph obtained from Γ by deleting all loops: We have $\Gamma^- = (V, E^-)$, where $E^- = \{S \in E \mid |S| = 2\}$.

Graph Monoids. Let A be a (not necessarily finite) set of symbols and R be a subset of $A^* \times A^*$. The pair (A, R) is called a *(monoid) presentation*. The smallest congruence of A^* containing R is denoted by $\equiv_R$ and we will write $[w]_R$ for the congruence class of $w \in A^*$. The *monoid presented by* (A, R) is defined as $A^*/\equiv_R$. Note that since we did not impose a finiteness restriction on A, up to isomorphism, every monoid has a presentation. Furthermore, for monoids M_1, M_2 we can find presentations (A_1, R_1) and (A_2, R_2) such that $A_1 \cap A_2 = \emptyset$. We define the *free product* $M_1 * M_2$ to be presented by $(A_1 \cup A_2, R_1 \cup R_2)$.

Note that $M_1 * M_2$ is well-defined up to isomorphism. In analogy to the n-fold direct product, we write $M^{(n)}$ for the n-fold free product of M.

A presentation (A, R) in which A is a finite alphabet is a *Thue system*. To each graph $\Gamma = (V, E)$, we associate the Thue system $T_\Gamma = (X_\Gamma, R_\Gamma)$ over the alphabet $X_\Gamma = \{a_v, \bar{a}_v \mid v \in V\}$. R_Γ is defined as

$$R_\Gamma = \{(a_v\bar{a}_v, \varepsilon) \mid v \in V\} \cup \{(xy, yx) \mid x \in \{a_v, \bar{a}_v\},\ y \in \{a_w, \bar{a}_w\},\ \{v, w\} \in E\}.$$

In particular, we have $(a_v\bar{a}_v, \bar{a}_v a_v) \in R_\Gamma$ whenever $\{v\} \in E$. To simplify notation, the congruence $\equiv_{R_\Gamma}$ is then also denoted by $\equiv_\Gamma$. We are now ready to define graph monoids. To each graph Γ, we associate the monoid

$$\mathbb{M}\Gamma \quad = \quad X_\Gamma^*/\equiv_\Gamma.$$

The monoids of the form $\mathbb{M}\Gamma$ are called *graph monoids*.

Let us briefly discuss how to realize storage mechanisms by graph monoids. First, suppose Γ_0 and Γ_1 are disjoint graphs. If Γ is the union of Γ_0 and Γ_1, then $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_0 * \mathbb{M}\Gamma_1$ by definition. Moreover, if Γ is obtained from Γ_0 and Γ_1 by drawing an edge between each vertex of Γ_0 and each vertex of Γ_1, then $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_0 \times \mathbb{M}\Gamma_1$.

If Γ consists of one vertex v and has no edges, the only rule in the Thue system is $(a_v\bar{a}_v, \varepsilon)$. In this case, $\mathbb{M}\Gamma$ is also denoted as $\mathbb{B}$ and we will refer to it as the *bicyclic monoid*. The generators a_v and $\bar{a}_v$ are then also written a and $\bar{a}$, respectively. It is not hard to see that $\mathbb{B}$ corresponds to a *partially blind counter*, i.e. one that attains only non-negative values and has to be zero at the end of the computation. Moreover, if Γ consists of one looped vertex, then $\mathbb{M}\Gamma$ is isomorphic to $\mathbb{Z}$ and thus realizes a *blind counter*, which can go below zero and also zero-tested in the end.

If one storage mechanism is realized by a monoid M, then the monoid $\mathbb{B} * M$ corresponds to the mechanism that *builds stacks*: A configuration of this new mechanism consists of a sequence $c_0 a c_1 \cdots a c_n$, where $c_0, \ldots, c_n$ are configurations of the mechanism realized by M. We interpret this as a stack with the entries $c_0, \ldots, c_n$. One can open a new stack entry on top (by multiplying $a \in \mathbb{B}$), remove the topmost entry if empty (by multiplying $\bar{a} \in \mathbb{B}$) and operate on the topmost entry using the old mechanism (by multiplying elements from M). In particular, $\mathbb{B} * \mathbb{B}$ describes a pushdown stack with two stack symbols. See [12] for details and more examples.

As a final example, note that if Γ is one edge short of being a clique, then $\mathbb{M}\Gamma \cong \mathbb{B}^{(2)} \times \mathbb{B}^{n-2}$, where n is the number of vertices in Γ. Then, by the observations above, valence automata over $\mathbb{M}\Gamma$ are equivalent to Petri nets with n unbounded places and access to a pushdown stack. Hence, for our purposes, a *pushdown Petri net* is a valence automaton over $\mathbb{B}^{(2)} \times \mathbb{B}^n$ for some $n \in \mathbb{N}$.

3 Results

As a first step, we exhibit graphs Γ for which $\mathsf{VA}(\mathbb{M}\Gamma)$ includes the recursively enumerable languages. Unfortunately, the space restrictions do not permit an inclusion of a proof for the following theorem.

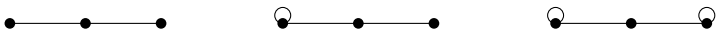
Theorem 1. *Let Γ be a graph such that Γ^- contains C_4 or P_4 as an induced subgraph. Then $\mathsf{VA}(\mathbb{M}\Gamma)$ is the class of recursively enumerable languages. In particular, the emptiness problem is undecidable for valence automata over $\mathbb{M}\Gamma$.*

This unifies and slightly strengthens a few undecidability results concerning valence automata over graph monoids. The case that all vertices are looped was shown by Lohrey and Steinberg [7] (see also the discussion of theorem 4). Another case appeared in [12].

It is not clear whether theorem 1 describes all Γ for which $\mathsf{VA}(\mathbb{M}\Gamma)$ exhausts the recursively enumerable languages. For example, as mentioned above, if Γ is one edge short of being a clique, then valence automata over $\mathbb{M}\Gamma$ are pushdown Petri nets. In particular, the emptiness problem for valence automata is equivalent to the reachability problem of this model, for which decidability is a long-standing open question [8]. In fact, it is already open whether reachability is decidable in the case of $\mathbb{B}^{(2)} \times \mathbb{B}$, although Leroux, Sutre, and Totzke [5] have recently made progress on this case [5]. Therefore, characterizing those Γ with a decidable emptiness problem for valence automata over $\mathbb{M}\Gamma$ would very likely settle these open questions[2].

However, we will show that if we steer clear of pushdown Petri nets, we can achieve a characterization. More precisely, we will present a set of graphs that entail the behavior of pushdown Petri nets. Then, we show that among those graphs that do not contain these as induced subgraphs, the absence of P_4 and C_4 already characterizes decidability.

PPN-graphs. A graph Γ is said to be a *PPN-graph* if it is isomorphic to one of the following three graphs:



We say that the graph Γ is *PPN-free* if it has no PPN-graph as an induced subgraph. Observe that a graph Γ is PPN-free if and only if in the neighborhood of each unlooped vertex, any two vertices are adjacent.

Of course, the abbreviation 'PPN' refers to 'pushdown Petri nets'. This is justified by the following fact.

Proposition 2. *If Γ is a PPN-graph, then $\mathsf{VA}(\mathbb{M}\Gamma) = \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{B})$.*

Transitive forests. In order to exploit the absence of P_4 and C_4 as induced subgraphs, we will employ a characterization of such graphs as transitive forests. The *comparability graph* of a tree t is a simple graph with the same vertices as t, but has an edge between two vertices whenever one is a descendent of the other in t. A simple graph is a *transitive forest* if it is the disjoint union of comparability graphs of trees. For an example of a transitive forest, see fig. 2.

Let DEC denote the smallest isomorphism-closed class of monoids such that

[2] Technically, it is conceivable that there is a decision procedure for each $\mathbb{B}^{(2)} \times \mathbb{B}^n$, but no uniform one that works for all n. However, this seems unlikely.

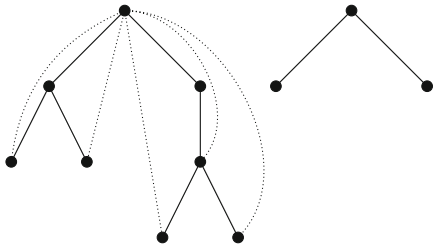


Fig. 2. Example of a transitive forest. The solid edges are part of the trees whose comparability graphs make up the graph.

1. for each $n \geq 0$, we have $\mathbb{B}^n \in \mathsf{DEC}$ and
2. for $M, N \in \mathsf{DEC}$, we also have $M * N \in \mathsf{DEC}$ and $M \times \mathbb{Z} \in \mathsf{DEC}$.

Our main result characterizes those PPN-free Γ for which valence automata over $\mathbb{M}\Gamma$ have a decidable emptiness problem.

Theorem 3. *Let Γ be PPN-free. Then the following conditions are equivalent:*

1. *Emptiness is decidable for valence automata over $\mathbb{M}\Gamma$.*
2. *Γ^- contains neither C_4 nor P_4 as an induced subgraph.*
3. *Γ^- is a transitive forest.*
4. *$\mathbb{M}\Gamma \in \mathsf{DEC}$.*

Note that this generalizes the fact that emptiness is decidable for pushdown automata (i.e. graphs with no edges) and partially blind multicounter automata (i.e. cliques), or equivalently, reachability in Petri nets.

Note that if Γ has a loop on every vertex, then $\mathbb{M}\Gamma$ is a group. Groups that arise in this way are called *graph groups*. In general, if a monoid M is a group, then emptiness for valence automata over M is decidable if and only if the *rational subset membership problem* is decidable for M [4]. The latter problem asks, given a rational set R over M and an element $m \in M$, whether $m \in R$; see [6] for more information. Therefore, 3 extends the following result of Lohrey and Steinberg [7], which characterizes those graph groups for which the rational subset membership problem is decidable.

Theorem 4 (Lohrey and Steinberg [7]). *Let Γ be a graph in which every vertex is looped. Then the rational subset membership problem for the group $\mathbb{M}\Gamma$ is decidable if and only if Γ^- is a transitive forest.*

Lohrey and Steinberg show decidability by essentially proving that $\mathsf{VA}(\mathbb{M}\Gamma)$ is semilinear in their case. Here, we extend this argument by showing that in the equivalent cases of theorem 3, the Parikh images of $\mathsf{VA}(\mathbb{M}\Gamma)$ are those of languages accepted by priority multicounter machines. The latter were introduced and shown to have a decidable reachability problem by Reinhardt [8].

Intuition for Decidable Cases. In order to provide an intuition for those storage mechanisms (not containing a pushdown Petri net) with a decidable emptiness problem, we present an equally expressive class of monoids for which the corresponding storage mechanisms are easier to grasp. Let $\mathsf{SC}^{\pm}$ be the smallest isomorphism-closed class of monoids with

1. for each $n \in \mathbb{N}$, we have $\mathbb{B}^n \in \mathsf{SC}^{\pm}$,
2. for each $M \in \mathsf{SC}^{\pm}$, we also have $\mathbb{B} * M \in \mathsf{SC}^{\pm}$ and $M \times \mathbb{Z} \in \mathsf{SC}^{\pm}$.

Thus, $\mathsf{SC}^{\pm}$ realizes those storage mechanisms that can be constructed from a finite set of *partially blind counters* ($\mathbb{B}^n$) by *building stacks* ($M \mapsto \mathbb{B} * M$) and *adding blind counters* ($M \mapsto M \times \mathbb{Z}$). Then, in fact, the monoids in $\mathsf{SC}^{\pm}$ produce the same languages as those in DEC.

Proposition 5. $\mathsf{VA}(\mathsf{DEC}) = \mathsf{VA}(\mathsf{SC}^{\pm})$.

While our decidability proof for $\mathsf{SC}^{\pm}$ is a reduction to priority multicounter machines, it should be stressed that the mechanisms realized by $\mathsf{SC}^{\pm}$ are quite different from priority counters and very likely not subsumed by them in terms of accepted languages. For example, $\mathsf{SC}^{\pm}$ contains pushdown stacks ($\mathbb{B} * \mathbb{B}$)—if the priority multicounter machines could accept all context-free languages (or even just the semi-Dyck language over two pairs of parentheses), this would easily imply decidability for pushdown Petri nets. Indeed, $\mathsf{SC}^{\pm}$ can even realize stacks where each entry consists of n partially blind counters (since $\mathbb{B} * (\mathbb{B}^n) \in \mathsf{SC}^{\pm}$). On the other hand, priority multicounter machines do not seem to be subsumed by $\mathsf{SC}^{\pm}$ either: After building stacks once, $\mathsf{SC}^{\pm}$ only allows adding *blind* counters (and building stacks again). It therefore seems unlikely that a mechanism in $\mathsf{SC}^{\pm}$ can accept the languages even of a priority 2-counter machine.

Intuition for Open Cases. We also want to provide an intuition for the remaining storage mechanisms, i.e. those defined by monoids $\mathbb{M}\Gamma$ about which theorems 1 and 3 make no statement. To this end, we describe a class of monoids that are expressively equivalent to these remaining cases. The remaining cases are given by those graphs Γ where Γ^- does not contain C_4 or P_4, but Γ contains a PPN-graph. Let REM denote the class of monoids $\mathbb{M}\Gamma$, where Γ is such a graph. Let SC^{+} be the smallest isomorphism-closed class of monoids with

1. $\mathbb{B}^{(2)} \times \mathbb{B} \in \mathsf{SC}^{+}$ and
2. for each $M \in \mathsf{SC}^{+}$, we also have $\mathbb{B} * M \in \mathsf{SC}^{+}$ and $M \times \mathbb{B} \in \mathsf{SC}^{+}$.

This means, SC^{+} realizes those storage mechanisms that are obtained from *a pushdown stack, together with one partially blind counter* ($\mathbb{B}^{(2)} \times \mathbb{B}$) by the transformations of *building stacks* ($M \mapsto \mathbb{B} * M$) and *adding partially blind counters* ($M \mapsto M \times \mathbb{B}$).

Proposition 6. $\mathsf{VA}(\mathsf{REM}) = \mathsf{VA}(\mathsf{SC}^{+})$.

Of course, SC^{+} generalizes pushdown Petri nets, which correspond to monoids $\mathbb{B}^{(2)} \times \mathbb{B}^n$ for $n \in \mathbb{N}$. Moreover, SC^{+} also subsumes priority multicounter machines

(see p. 175): Every time we build stacks, we can use the new pop operation to realize a zero test on all the counters we have added so far. Let $M_0 = \mathbf{1}$ and $M_{k+1} = \mathbb{B} * (M_k \times \mathbb{B})$. Then, priority k-counter machines correspond to valence automata over M_k where the stack heights never exceed 1.

4 Proof of the Characterization

First, we mention existing results that are ingredients to our proofs. A *class of languages* is a collection of languages that contains at least one non-empty language. If $\mathcal{C}$ is a language class such that for languages $L \subseteq X^*$ in $\mathcal{C}$, homomorphisms $h\colon X^* \to Y^*$, $g\colon Z^* \to X^*$, and regular sets $R \subseteq X^*$, we have $h(L) \in \mathcal{C}$, $g^{-1}(L) \in \mathcal{C}$, and $L \cap R \in \mathcal{C}$, we call $\mathcal{C}$ a *full trio*.

Let $\mathcal{C}$ be a class of languages. A $\mathcal{C}$*-grammar* is a quadruple $G = (N, T, P, S)$ where N and T are disjoint alphabets and $S \in N$. P is a finite set of pairs (A, M) with $A \in N$ and $M \subseteq (N \cup T)^*$, $M \in \mathcal{C}$. We write $x \Rightarrow_G y$ if $x = uAv$ and $y = uwv$ for some $u, v, w \in (N \cup T)^*$ and $(A, M) \in P$ with $w \in M$. The *language generated by* G is $\mathsf{L}(G) = \{w \in T^* \mid S \Rightarrow_G^* w\}$. The class of all languages that are generated by $\mathcal{C}$-grammars is denoted $\mathsf{Alg}(\mathcal{C})$.

The well-known theorem of Chomsky and Schützenberger [1], expressed in terms of valence automata, states that $\mathsf{VA}(\mathbb{Z} * \mathbb{Z})$ is the class of context-free languages. This formulation, along with a new proof, is due to Kambites [3]. Let Reg and CF denote the class of regular and context-free languages, respectively. Then we have $\mathsf{Reg} = \mathsf{VA}(\mathbf{1})$ and $\mathsf{CF} = \mathsf{Alg}(\mathsf{Reg})$. Here, $\mathbf{1}$ denotes the trivial monoid $\{1\}$. Since furthermore valence automata over $\mathbb{B} * \mathbb{B}$ are equivalent to pushdown automata, we have in summary:

$$\mathsf{CF} = \mathsf{VA}(\mathbb{B} * \mathbb{B}) = \mathsf{Alg}(\mathsf{VA}(\mathbf{1})) = \mathsf{Alg}(\mathsf{CF}) = \mathsf{VA}(\mathbb{Z} * \mathbb{Z}).$$

In order to work with general free products, we use the following result, which expresses the languages in $\mathsf{VA}(M_0 * M_1)$ in terms of $\mathsf{VA}(M_0)$ and $\mathsf{VA}(M_1)$.

Proposition 7 ([12]). *Let M_0 and M_1 be monoids. Then $\mathsf{VA}(M_0 * M_1)$ is included in $\mathsf{Alg}(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1))$.*

As a partial converse to theorem 7, we have the following. For a monoid M, we define $R_1(M) = \{x \in M \mid \exists y \in M\colon xy = 1\}$. Observe that the set $R_1(M)$ can be thought of as the storage contents that can be reset.

Proposition 8 ([11]). *For every monoid M, $\mathsf{VA}(\mathbb{B} * \mathbb{B} * M) = \mathsf{Alg}(\mathsf{VA}(M))$. Moreover, if $R_1(M) \neq \{1\}$, then $\mathsf{VA}(\mathbb{B} * M) = \mathsf{Alg}(\mathsf{VA}(M))$.*

Since valence automata over $\mathbb{B} * \mathbb{B}$ are essentially pushdown automata and since $\mathsf{Alg}(\mathsf{VA}(\mathbf{1})) = \mathsf{Alg}(\mathsf{Reg})$, the equality $\mathsf{VA}(\mathbb{B} * \mathbb{B} * M) = \mathsf{Alg}(\mathsf{VA}(M))$ generalizes the equivalence between pushdown automata and context-free grammars.

We will also use the fact that if $\mathsf{VA}(M_i) \subseteq \mathsf{VA}(N_i)$ for $i \in \{0, 1\}$, then $\mathsf{VA}(M_0 \times M_1) \subseteq \mathsf{VA}(N_0 \times N_1)$. This can be deduced from a characterization of $\mathsf{VA}(M_0 \times M_1)$ in terms of $\mathsf{VA}(M_0)$ and $\mathsf{VA}(M_1)$ by Kambites [3].

Proof (theorem 2). By definition, we have $\mathbb{M}\Gamma \cong \mathbb{B} \times (M_0 * M_1)$, where $M_i \cong \mathbb{B}$ or $M_i \cong \mathbb{Z}$ for $i \in \{0,1\}$. We show that $\mathsf{VA}(M_0 * M_1) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$ in any case. This suffices, since it clearly implies $\mathsf{VA}(\mathbb{M}\Gamma) = \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{B})$. If $M_0 \cong M_1 \cong \mathbb{B}$, the equality $\mathsf{VA}(M_0 * M_1) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$ is trivial, so we may assume $M_0 \cong \mathbb{Z}$.

If $M_1 \cong \mathbb{Z}$, then $M_0 * M_1 \cong \mathbb{Z} * \mathbb{Z}$, meaning that $\mathsf{VA}(M_0 * M_1)$ is the class of context-free languages and thus $\mathsf{VA}(M_0 * M_1) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$.

If $M_1 \cong \mathbb{B}$, then $\mathsf{VA}(\mathbb{Z} * \mathbb{B}) = \mathsf{Alg}(\mathsf{VA}(\mathbb{Z}))$ by theorem 8. Since $\mathsf{VA}(\mathbb{Z})$ is included in the context-free languages, we have $\mathsf{Alg}(\mathsf{VA}(\mathbb{Z})) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$. □

We shall now prove theorem 3. Note that the implication "1 ⇒ 2" immediately follows from theorem 1. The implication "2 ⇒ 3" is an old graph-theoretic result of Wolk.

Theorem 9 (Wolk) [10]). *A simple graph Γ is a transitive forest if and only if Γ does not contain C_4 or P_4 as an induced subgraph.*

The implication "3 ⇒ 4" is a simple combinatorial observation. An analogous fact is part of Lohrey and Steinberg's proof of theorem 4.

Lemma 10. *If Γ is PPN-free and Γ^- is a transitive forest, then $\mathbb{M}\Gamma \in \mathsf{DEC}$.*

Proof. Let $\Gamma = (V, E)$. We proceed by induction on $|V|$. Observe that by theorem 9, every induced subgraph of a transitive forest is again a transitive forest. Since furthermore every induced proper subgraph Δ of Γ is again PPN-free, our induction hypothesis implies $\mathbb{M}\Delta \in \mathsf{DEC}$ for such graphs. If Γ is empty, then $\mathbb{M}\Gamma \cong \mathbf{1} \cong \mathbb{B}^0 \in \mathsf{DEC}$. Hence, we assume that Γ is non-empty. If Γ is not connected, then Γ is the disjoint union of graphs Γ_1, Γ_2, for which $\mathbb{M}\Gamma_1, \mathbb{M}\Gamma_2 \in \mathsf{DEC}$ by induction. Hence, $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_1 * \mathbb{M}\Gamma_2 \in \mathsf{DEC}$.

Suppose Γ is connected. Since Γ^- is a transitive forest, there is a vertex $v \in V$ that is adjacent to every vertex in $V \setminus \{v\}$. We distinguish two cases.

- If v is a looped vertex, then $\mathbb{M}\Gamma \cong \mathbb{Z} \times \mathbb{M}(\Gamma \setminus v)$, and $\mathbb{M}(\Gamma \setminus v) \in \mathsf{DEC}$ by induction.
- If v is an unlooped vertex, then Γ being PPN-free means that in $\Gamma \setminus v$, any two distinct vertices are adjacent. Hence, $\mathbb{M}\Gamma \cong \mathbb{B}^m \times \mathbb{Z}^n$, where m and n are the number of unlooped and looped vertices in Γ, respectively. Therefore, $\mathbb{M}\Gamma \in \mathsf{DEC}$. □

In light of theorems 1, 9 and 10, it remains to be shown that emptiness is decidable for valence automata over monoids in DEC. This will involve two facts (theorem 11 and theorem 12) about the languages arising from monoids in DEC.

The following generalization of Parikh's theorem by van Leeuwen will allow us to exploit our description of free products by algebraic extensions. If X is an alphabet, $X^{\oplus}$ denotes the set of maps $\alpha\colon X \to \mathbb{N}$. The elements of $X^{\oplus}$ are called *multisets*. The *Parikh map* is the map $\Psi\colon X^* \to X^{\oplus}$ where $\Psi(w)(x)$ is the number of occurrences of x in w. By $\mathcal{P}(S)$, we denote the power set of the set S. A *substitution* is a map $\sigma\colon X \to \mathcal{P}(Y^*)$ and given $L \subseteq X^*$, we write $\sigma(L)$ for the set of all words $v_1 \cdots v_n$, where $v_i \in \sigma(x_i)$, $1 \leq i \leq n$, for $x_1 \cdots x_n \in L$ and

$x_1, \ldots, x_n \in X$. If $\sigma(x)$ belongs to $\mathcal{C}$ for each $x \in X$, then σ is a *$\mathcal{C}$-substitution*. The class $\mathcal{C}$ is said to be *substitution closed* if $\sigma(L) \in \mathcal{C}$ for every member L of $\mathcal{C}$ and every $\mathcal{C}$-substitution σ.

Theorem 11 (van Leeuwen [9]). *For each substitution closed full trio $\mathcal{C}$, we have $\Psi(\mathsf{Alg}(\mathcal{C})) = \Psi(\mathcal{C})$.*

For $\alpha, \beta \in X^{\oplus}$, let $\alpha + \beta \in X^{\oplus}$ be defined by $(\alpha + \beta)(x) = \alpha(x) + \beta(x)$. With this operation, $X^{\oplus}$ is a monoid. For a subset $S \subseteq X^{\oplus}$, we write $S^{\oplus}$ for the smallest submonoid of $X^{\oplus}$ containing S. A subset of the form $\mu + F^{\oplus}$ for $\mu \in X^{\oplus}$ and a finite $F \subseteq X^{\oplus}$ is called *linear*. A finite union of linear sets is called *semilinear*. By $\mathsf{SLI}(\mathcal{C})$ we denote the class of languages $h(L \cap \Psi^{-1}(S))$, where $h\colon X^* \to Y^*$ is a morphism, L belongs to $\mathcal{C}$, and $S \subseteq X^{\oplus}$ is semilinear.

Proposition 12 ([11]). *For each monoid M, $\mathsf{SLI}(\mathsf{VA}(M)) = \bigcup_{i \geq 0} \mathsf{VA}(M \times \mathbb{Z}^n)$.*

We will prove decidability for DEC by reducing the problem to the reachability problem of priority multicounter machines, whose decidability has been established by Reinhardt [8]. Priority multicounter machines are an extension of Petri nets with one inhibitor arc. Intuitively, a priority multicounter machine is a partially blind multicounter machine with the additional capability of restricted zero tests: The counters are numbered from 1 to k and for each $\ell \in \{1, \ldots, k\}$, there is a zero test instruction that checks whether counters 1 through ℓ are zero. Let us define priority multicounter machines formally.

A *priority k-counter machine* is a tuple $A = (Q, X, E, q_0, F)$, where (i) X is an alphabet, (ii) Q is a finite set of states, (iii) E is a finite subset of $Q \times X^* \times \{0, \ldots, k\} \times \mathbb{Z}^k \times Q$, and its elements are called *edges* or *transitions*, (iv) $q_0 \in Q$ is the *initial state*, and (v) $F \subseteq Q$ is the set of *final states*. For triples (q, u, μ) and (q', u', μ') with $q, q' \in Q$ and $\mu, \mu' \in \mathbb{N}^k$, with $\mu = (m_1, \ldots, m_k)$, we write $(q, u, \mu) \to_A (q', u', \mu')$ if for some $(q, x, \ell, \nu, q') \in E$, we have $u' = ux$, $\mu' = \mu + \nu$, and $m_i = 0$ for $1 \leq i \leq \ell$. The *language accepted by A* is defined as

$$\mathsf{L}(A) = \{w \in X^* \mid (q_0, \varepsilon, 0) \to_A^* (f, w, 0) \text{ for some } f \in F\}.$$

A *priority multicounter machine* is a priority k-counter machine for some $k \in \mathbb{N}$. The class of languages accepted by priority multicounter machines is denoted by Prio. Reinhardt has shown that the reachability problem for priority multicounter machines is decidable [8], which can be reformulated as follows.

Theorem 13 (Reinhardt [8]). *Emptiness is decidable for priority multicounter machines.*

The idea of the proof of "4 ⇒ 1" is, given a valence automaton over some $M \in \mathsf{DEC}$, to construct a Parikh-equivalent priority multicounter machine. This construction makes use of the following simple fact. A full trio $\mathcal{C}$ is said to be *Presburger closed* if $\mathsf{SLI}(\mathcal{C}) \subseteq \mathcal{C}$.

Lemma 14. Prio *is a Presburger closed full trio and closed under substitutions.*

Proof. The fact that Prio is a full trio can be shown by standard automata constructions. Given a priority multicounter machine A and a semilinear set $S \subseteq X^{\oplus}$, we add $|X|$ counters to A that ensure that the input is contained in $\mathsf{L}(A) \cap \Psi^{-1}(S)$. This proves that Prio is Presburger closed.

Suppose $\sigma\colon X \to \mathcal{P}(Y^*)$ is a Prio-substitution. Furthermore, let A be a priority k-counter machine and let $\sigma(x)$ be given by a priority ℓ-counter machine for each $x \in X$. We construct a priority $(\ell+k)$-counter machine B from A by adding ℓ counters. B simulates A on counters $\ell+1, \ldots, \ell+k$. Whenever A reads x, B uses the first ℓ counters to simulate the priority ℓ-counter machine for $\sigma(x)$. Using the zero test on the first ℓ counters, it makes sure that the machine for $\sigma(x)$ indeed ends up in a final configuration. Then clearly $\mathsf{L}(B) = \sigma(\mathsf{L}(A))$. □

In order to show that every $L \in \mathsf{VA}(M)$ for $M \in \mathsf{DEC}$ has a Parikh equivalent in Prio, we use theorem 12 and theorem 7. By induction with respect to the definition of DEC, it suffices to prove that

$$\Psi(\mathsf{VA}(M)), \Psi(\mathsf{VA}(N)) \subseteq \Psi(\mathsf{Prio}) \quad \text{implies} \quad \begin{array}{l} \Psi(\mathsf{VA}(M \times \mathbb{Z})) \subseteq \Psi(\mathsf{Prio}) \text{ and} \\ \Psi(\mathsf{VA}(M * N)) \subseteq \Psi(\mathsf{Prio}). \end{array}$$

According to theorem 12 and theorem 7, this boils down to showing that we have $\Psi(\mathsf{SLI}(\mathsf{Prio})) \subseteq \Psi(\mathsf{Prio})$ and $\Psi(\mathsf{Alg}(\mathsf{Prio})) \subseteq \Psi(\mathsf{Prio})$. The former is a consequence of theorem 14 and the latter is an instance of theorem 11.

Lemma 15. *We have the effective inclusion* $\Psi(\mathsf{VA}(\mathsf{DEC})) \subseteq \Psi(\mathsf{Prio})$. *More precisely, given* $M \in \mathsf{DEC}$ *and* $L \in \mathsf{VA}(M)$, *one can construct an* $L' \in \mathsf{Prio}$ *with* $\Psi(L') = \Psi(L)$.

Proof. We proceed by induction with respect to the definition of DEC. In the case $M = \mathbb{B}^n$, we have $\mathsf{VA}(M) \subseteq \mathsf{Prio}$, because priority multicounter machines generalize partially blind multicounter machines.

Suppose $M = N \times \mathbb{Z}$ and $\Psi(\mathsf{VA}(N)) \subseteq \Psi(\mathsf{Prio})$ and let $L \in \mathsf{VA}(M)$. By theorem 12, we have $L = h(K \cap \Psi^{-1}(S))$ for some semilinear set S, a morphism h, and $K \in \mathsf{VA}(N)$. Hence, there is a $\bar{K} \in \mathsf{Prio}$ with $\Psi(\bar{K}) = \Psi(K)$. With this, we have $\Psi(L) = \Psi(h(\bar{K} \cap \Psi^{-1}(S)))$ and since Prio is Presburger closed, we have $h(\bar{K} \cap \Psi^{-1}(S)) \in \mathsf{Prio}$ and thus $\Psi(L) \in \Psi(\mathsf{Prio})$.

Suppose $M = M_0 * M_1$ and $\Psi(\mathsf{VA}(M_i)) \subseteq \Psi(\mathsf{Prio})$ for $i \in \{0,1\}$. Let L be a member of $\mathsf{VA}(M)$. According to theorem 7, this means L belongs to $\mathsf{Alg}(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1))$. Since $\Psi(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)) \subseteq \Psi(\mathsf{Prio})$, we can construct a Prio-grammar G with $\Psi(\mathsf{L}(G)) = \Psi(L)$. By theorem 11 and theorem 14, this implies $\Psi(L) \in \Psi(\mathsf{Prio})$. □

The following theorem is a direct consequence of theorem 15 and theorem 13: Given a valence automaton over M with $M \in \mathsf{DEC}$, we construct a priority multicounter machine accepting a Parikh-equivalent language. The latter can then be checked for emptiness.

Lemma 16. *For each* $M \in \mathsf{DEC}$, *the emptiness problem for valence automata over* M *is decidable.*

This completes the proof of theorem 3. Let us now prove the expressive equivalences, theorems 5 and 6.

Proof (theorem 5). Since $\mathsf{SC}^{\pm} \subseteq \mathsf{DEC}$, the inclusion "$\supseteq$" is immediate. We show by induction with respect to the definition of DEC that for each $M \in \mathsf{DEC}$, there is an $M' \in \mathsf{SC}^{\pm}$ with $\mathsf{VA}(M) \subseteq \mathsf{VA}(M')$. This is trivial if $M = \mathbb{B}^n$, so suppose $\mathsf{VA}(M) \subseteq \mathsf{VA}(M')$ and $\mathsf{VA}(N) \subseteq \mathsf{VA}(N')$ for $M, N \in \mathsf{DEC}$ and $M', N' \in \mathsf{SC}^{\pm}$. Observe that by induction on the definition of $\mathsf{SC}^{\pm}$, one can show that there is a common $P \in \mathsf{SC}^{\pm}$ with $\mathsf{VA}(M') \subseteq \mathsf{VA}(P)$ and $\mathsf{VA}(N') \subseteq \mathsf{VA}(P)$. Of course, we may assume that $R_1(P) \neq \{1\}$. Then we have

$$\begin{aligned}\mathsf{VA}(M * N) &\subseteq \mathsf{Alg}(\mathsf{VA}(M) \cup \mathsf{VA}(N)) \subseteq \mathsf{Alg}(\mathsf{VA}(M') \cup \mathsf{VA}(N')) \\ &\subseteq \mathsf{Alg}(\mathsf{VA}(P)) = \mathsf{VA}(\mathbb{B} * P),\end{aligned}$$

in which the first inclusion is due to theorem 7 and the equality in the end is provided by theorem 8. Since $\mathbb{B} * P \in \mathsf{SC}^{\pm}$, this completes the proof for $M * N$. Moreover, $\mathsf{VA}(M) \subseteq \mathsf{VA}(M')$ implies $\mathsf{VA}(M \times \mathbb{Z}) \subseteq \mathsf{VA}(M' \times \mathbb{Z})$ and we have $M' \times \mathbb{Z} \in \mathsf{SC}^{\pm}$. □

Proof (theorem 6). By induction, it is easy to see that each $M \in \mathsf{SC}^{+}$ is isomorphic to some $\mathbb{M}\Gamma$ where Γ contains a PPN-graph and Γ^{-} is a transitive forest. By theorem 9, this means Γ^{-} contains neither C_4 nor P_4. This proves the inclusion "$\supseteq$".

Because of theorem 9, for the inclusion "$\subseteq$", it suffices to show that if Γ^{-} is a transitive forest, then there is some $M \in \mathsf{SC}^{+}$ with $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(M)$. We prove this by induction on the number of vertices in $\Gamma = (V, E)$. As in the proof of theorem 10, we may assume that for every induced proper subgraph Δ of Γ, we find an $M \in \mathsf{SC}^{+}$ with $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(M)$. If Γ is empty, then $\mathbb{M}\Gamma \cong \mathbf{1}$ and $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{B})$. Hence, we may assume that Γ is non-empty.

If Γ is not connected, then $\Gamma = \Gamma_1 \uplus \Gamma_2$ with graphs Γ_1, Γ_2 such that there are $M_1, M_2 \in \mathsf{SC}^{+}$ with $\mathsf{VA}(\mathbb{M}\Gamma_i) \subseteq \mathsf{VA}(M_i)$ for $i \in \{1, 2\}$. By induction with respect to the definition of SC^{+}, one can show that there is a common $N \in \mathsf{SC}^{+}$ with $\mathsf{VA}(M_i) \subseteq \mathsf{VA}(N)$ for $i \in \{1, 2\}$. Since then $R_1(N) \neq \{1\}$, we have

$$\begin{aligned}\mathsf{VA}(\mathbb{M}\Gamma) &= \mathsf{VA}(\mathbb{M}\Gamma_1 * \mathbb{M}\Gamma_2) \subseteq \mathsf{Alg}(\mathsf{VA}(\mathbb{M}\Gamma_1) \cup \mathsf{VA}(\mathbb{M}\Gamma_2)) \\ &\subseteq \mathsf{Alg}(\mathsf{VA}(M_1) \cup \mathsf{VA}(M_2)) \subseteq \mathsf{Alg}(\mathsf{VA}(N)) = \mathsf{VA}(\mathbb{B} * N)\end{aligned}$$

and $\mathbb{B} * N \in \mathsf{SC}^{+}$ as in the proof of theorem 5.

Suppose Γ is connected. Since Γ^{-} is a transitive forest, there is a vertex $v \in V$ that is adjacent to every vertex in $V \setminus \{v\}$. By induction, there is an $M \in \mathsf{SC}^{+}$ with $\mathsf{VA}(\mathbb{M}(\Gamma \setminus v)) \subseteq \mathsf{VA}(M)$. Depending on whether v is looped or not, we have $\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus v) \times \mathbb{Z}$ or $\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus v) \times \mathbb{B}$. Since $\mathsf{VA}(\mathbb{Z}) \subseteq \mathsf{VA}(\mathbb{B} \times \mathbb{B})$ (one blind counter can easily be simulated by two partially blind counters), this yields $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(\mathbb{M}(\Gamma \setminus v) \times \mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}(M \times \mathbb{B} \times \mathbb{B})$ and the fact that $M \times \mathbb{B} \times \mathbb{B} \in \mathsf{SC}^{+}$ completes the proof. □

Conclusion. Of course, an intriguing open question is whether the storage mechanisms corresponding to SC^+ have a decidable reachability problem. First, since their simplest instance are pushdown Petri nets, this extends the open question concerning the latter's reachability. Second, they subsume the priority multi-counter machines of Reinhardt. This makes them a candidate for being a quite powerful model for which reachability might be decidable.

Observe that if these storage mechanisms turn out to exhibit decidability, this would mean that Lohrey and Steinberg's characterization (theorem 4) remains true for all graph monoids. This can be interpreted as evidence for decidability.

Acknowledgments. The author is grateful to the anonymous referees, whose helpful comments improved the presentation of this work.

References

1. Berstel, J.: Transductions and Context-Free Languages. Teubner, Stuttgart (1979)
2. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific, Singapore (1995)
3. Kambites, M.: Formal Languages and Groups as Memory. Commun. Algebra **37**, 193–208 (2009)
4. Kambites, M., Silva, P.V., Steinberg, B.: On the rational subset problem for groups. J. Algebra **309**, 622–639 (2007)
5. Leroux, J., Sutre, G., Totzke, P.: On the coverability problem for pushdown vector addition systems in one dimension. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 324–336. Springer, Heidelberg (2015)
6. Lohrey, M.: The rational subset membership problem for groups: A survey (to appear)
7. Lohrey, M., Steinberg, B.: The submonoid and rational subset membership problems for graph groups. J. Algebra **320**(2), 728–755 (2008)
8. Reinhardt, K.: Reachability in petri nets with inhibitor arcs. In: Proc. of RP 2008 (2008)
9. van Leeuwen, J.: A generalisation of Parikh's theorem in formal language theory. In: Loeckx, J. (ed.) Automata, Languages and Programming. LNCS, vol. 14, pp. 17–26. Springer, Heidelberg (1974)
10. Wolk, E.S.: A Note on 'The Comparability Graph of a Tree'. P. Am. Math. Soc. **16**(1), 17–20 (1965)
11. Zetzsche, G.: Computing downward closures for stacked counter automata. In: Proc. of STACS 2015 (2015)
12. Zetzsche, G.: Silent transitions in automata with storage. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 434–445. Springer, Heidelberg (2013)

Author Index