Margaret Archibald
Vasco Brattka
Valentin Goranko
Benedikt Löwe (Eds.)

LNAI 5489

Infinity in Logic and Computation

International Conference, ILC 2007
Cape Town, South Africa, November 2007
Revised Selected Papers



Springer

Lecture Notes in Artificial Intelligence 5489

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

FoLLI Publications on Logic, Language and Information

Editors-in-Chief

Luigia Carlucci Aiello, *University of Rome "La Sapienza", Italy*
Michael Moortgat, *University of Utrecht, The Netherlands*
Maarten de Rijke, *University of Amsterdam, The Netherlands*

Editorial Board

Carlos Areces, *INRIA Lorraine, France*
Nicholas Asher, *University of Texas at Austin, TX, USA*
Johan van Benthem, *University of Amsterdam, The Netherlands*
Raffaella Bernardi, *Free University of Bozen-Bolzano, Italy*
Antal van den Bosch, *Tilburg University, The Netherlands*
Paul Buitelaar, *DFKI, Saarbrücken, Germany*
Diego Calvanese, *Free University of Bozen-Bolzano, Italy*
Ann Copestake, *University of Cambridge, United Kingdom*
Robert Dale, *Macquarie University, Sydney, Australia*
Luis Fariñas, *IRIT, Toulouse, France*
Claire Gardent, *INRIA Lorraine, France*
Rajeev Goré, *Australian National University, Canberra, Australia*
Reiner Hähnle, *Chalmers University of Technology, Göteborg, Sweden*
Wilfrid Hodges, *Queen Mary, University of London, United Kingdom*
Carsten Lutz, *Dresden University of Technology, Germany*
Christopher Manning, *Stanford University, CA, USA*
Valeria de Paiva, *Palo Alto Research Center, CA, USA*
Martha Palmer, *University of Pennsylvania, PA, USA*
Alberto Policriti, *University of Udine, Italy*
James Rogers, *Earlham College, Richmond, IN, USA*
Francesca Rossi, *University of Padua, Italy*
Yde Venema, *University of Amsterdam, The Netherlands*
Bonnie Webber, *University of Edinburgh, Scotland, United Kingdom*
Ian H. Witten, *University of Waikato, New Zealand*

Margaret Archibald Vasco Brattka
Valentin Goranko Benedikt Löwe (Eds.)

Infinity in Logic and Computation

International Conference, ILC 2007
Cape Town, South Africa, November 3-5, 2007
Revised Selected Papers

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Margaret Archibald
Vasco Brattka
University of Cape Town
Department of Mathematics and Applied Mathematics
Rondebosch 7701, South Africa
E-mail: {margaret.archibald, vasco.brattka}@uct.ac.za

Valentin Goranko
Technical University of Denmark, Informatics and Mathematical Modeling
Richard Petersens Plads, 2800 Kongens Lyngby, Denmark
E-mail: vfgo@imm.dtu.dk

Benedikt Löwe
Universiteit van Amsterdam, Institute for Logic, Language and Computation
Postbus 94242, 1090 GE Amsterdam, The Netherlands
E-mail: bloewe@science.uva.nl

Library of Congress Control Number: 2009936649

CR Subject Classification (1998): I.2, F.1, F.2.1-2, F.4.1, G.1.0

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-642-03091-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-03091-8 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2009
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12700868 06/3180 5 4 3 2 1 0

Preface

The conference on Infinity in Logic and Computation (ILC) took place at the University of Cape Town (UCT) in South Africa during November 3–5, 2007. It was the first conference in South Africa to focus on infinity in automata theory, logic, computability and verification. One purpose of this conference was to catalyze new interactions among local and international researchers and to expose postgraduate students to recent research trends in these fields. The topics of the conference included automata on infinite objects; combinatorics, cryptography and complexity; computability and complexity on the real numbers; infinite games and their connections to logic; logic, computability, and complexity in finitely presentable infinite structures; randomness and computability; transfinite computation; and verification of infinite state systems.



Fig. 1. The University of Cape Town

The organizers of the conference used the opportunity of this conference to invite some of the speakers to offer a short course during the Summer School on Logic and Computation that followed the conference during November 6–9, 2007. This summer school was attended by researchers, but in particular it was a unique opportunity for local graduate students to become familiar with current research directions in logic and computation.



Fig. 2. ILC 2007 Conference Photo

The following courses were offered at the summer school:

Computability on the Reals. Vasco Brattka
The Logic of the Smooth Topos. Willem L. Fouché
Finitely Presentable Infinite Structures. Valentin Goranko
Strategies and Algorithms for Infinite Games. Erich Grädel
Infinite Time Computation (CANCELLED). Joel David Hamkins
Determinacy Axioms and Their Logical Strength. Benedikt Löwe
Effective Randomness. Joseph S. Miller

The conference and summer school were closely related to research activities of the Laboratory of Foundational Aspects of Computer Science (FACS Lab), a research unit at the Department of Mathematics and Applied Mathematics. Both the conference and summer school were co-hosted by the Meraka Institute, the African Advanced Institute for Information and Communication Technology (Fig. 3), and sponsored by Dimension Data (Fig. 4). Due to the generous support of the co-host and the sponsor we were able to attract as invited speakers a number of leading international experts from the USA, Germany, South Africa and Switzerland. The invited speakers of ILC 2007 were Willem L. Fouché (Pretoria, South Africa), Erich Grädel (Aachen, Germany), Thomas A. Henzinger (Lausanne, Switzerland), Peter Hertling (Munich, Germany), Joseph S. Miller (Storrs

CT, USA), and Helmut Prodinger (Stellenbosch, South Africa). Joel Hamkins (New York NY, USA) was scheduled to talk, but had to cancel due to visa issues.



The major objective of the Meraka Institute is to facilitate national economic and social development in South Africa through human capital development and needs-based research and innovation, leading to products and services based on information and communication technology.

Fig. 3. Meraka Institute — African Advanced Institute for Information and Communication Technology



Founded in 1983 and headquartered in South Africa, Dimension Data is a specialist IT services and solution provider that helps clients plan, build, support and manage their IT infrastructures.

Fig. 4. Dimension Data

The conference originally received 27 submissions (from Algeria, China, France, Germany, Japan, The Netherlands, Norway, Russia, South Africa and the UK), which were thoroughly reviewed by the Programme Committee. The authors of accepted presentations were invited after the conference to submit their paper versions for this proceedings volume. We received 13 submissions for the volume; 20 external reviewers were involved in the peer-review process; in the end, the nine papers that are printed in this volume were accepted. Two of them, the paper by Chatterjee and Henzinger and the paper by Louard and Prodinger, correspond to invited talks given at ILC 2007; the other seven correspond to contributed papers (two of which were not presented at the conference due to travel constraints). We would like to thank the Programme Committee and the additional 20 external reviewers for their work that was instrumental in producing this volume. The members of the Programme Committee were: Ahmed Bouajjani (Paris, France), Vasco Brattka (Cape Town, South Africa), Valentin Goranko (Johannesburg, South Africa), Ker-I Ko (Stony Brook NY, USA), Orna Kupferman (Jerusalem, Israel), Benedikt Löwe (Amsterdam, The Netherlands; *Chair*), Elvira Mayordomo (Zaragoza, Spain), Dieter Spreen (Siegen, Germany), Wolfgang Thomas (Aachen, Germany), Yde Venema (Amsterdam, The Netherlands), Klaus Weihrauch (Hagen, Germany), Philip Welch (Bristol, UK), Mariko Yasugi (Kyoto, Japan), and Jeffery Zucker (Hamilton, Canada).

With profound sadness, we report that one of the authors of a contributed paper, Nadia Busi, passed away between the time of submission for ILC 2007 and the conference. Her coauthor, Claudia Zandron, was not able to come to Cape Town and present the paper. Their paper published in this volume is a

tribute to Nadia, and should be seen as unfinished research, a project started before Nadia's untimely death. We also publish an obituary in this volume.



Fig. 5. Fog and clouds enshroud the cable car station at Table Mountain

Last, but not least, the editors would like to thank their co-organizer Hans-Peter Künzi, and the indispensable student assistants Christian Geist, Mashudu Makananise, and Thyla J. van der Merwe. We should also like to thank the Editors-in-Chief of the FoLLI subseries of LNAI for supporting the production of the volume at an early stage.

Conference Schedule

On Sunday, the partipants of ILC 2007 went on a memorable excursion. The participants from the northern hemisphere, intent on fleeing the cold November weather were looking forward to the summer of South Africa, only to be greeted by ice-cold wind and fog (Fig. 5–7).

Saturday, 3 November 2007

08:00 Registration (Hoerikwaggo Building; Upper Campus)
09:20 Opening
09:30 Thomas A. Henzinger (Invited Speaker), *Three Sources of Infinity in Computation: Nontermination, Real Time and Probabilistic Choice*
10:30 Coffee Break
11:00 Michael Ummels, *The Complexity of Finding Nash Equilibria in Infinite Multiplayer Games*

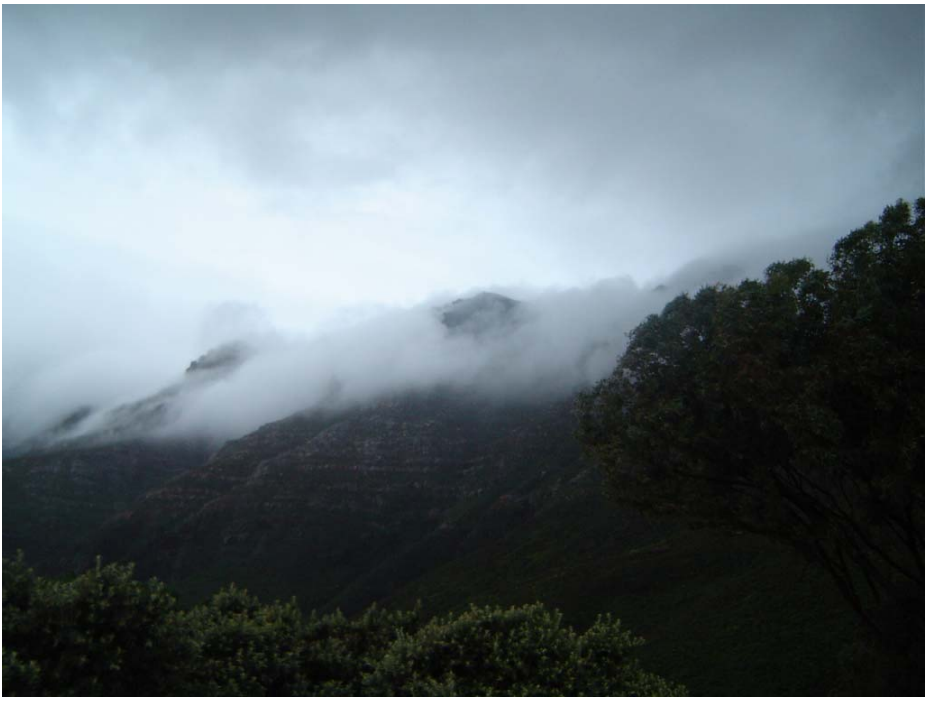


Fig. 6. More clouds

11:20 Jacques Duparc and Alessandro Facchini, *Complexity of Δ^1_2 Sets and μ-Calculus: When Infinite Games Make Modal Logic and Descriptive Set Theory Meet*
11:40 Douadi Mihoubi, *On the Sets of Infinite Words Recognized by Deterministic One Turn Pushdown Automata*
12:00 Valentin Goranko and Wilmari Bekker, *Infinite-State Verification of the Basic Tense Logic on Rational Kripke Models*
12:20 Govert van Drimmelen, *Tableaux and Automata for Parikh's Game Logic*
12:40 Lunch Break
14:30 Erich Grädel (Invited Speaker), *Banach-Mazur Games on Graphs*
15:30 Coffee Break
15:50 Mark Hogarth, *Non-Turing Computers Are the New Non-Euclidean Geometries*
16:10 Feng Liu, Zhoujun Li and Ti Zhou, *Boundary on Agent Number Needed in Security Protocol Analysis Based on Horn Logic*

Sunday, 4 November 2007

09:30 Peter Hertling (Invited Speaker), *Topological Complexity and Degrees of Discontinuity*
10:30 Coffee Break



Fig. 7. A windy view on Hout Bay from the Chapman's Peak Drive

11:00 Timothy McNicholl, *Computational Issues in the Theory of Bounded Analytic Functions*
11:20 Klaus Weihrauch, *Computable Elementary Topology*
11:40 Vasco Brattka and Mashudu Makananise, *Limit Computable Functions and Subsets on Metric Spaces*
12:00 Eyvind Martol Briseid, *Effective Rates of Convergence for Picard Iteration Sequences*
12:20 Philipp Gerhardy, *Proof Mining in Ergodic Theory and Topological Dynamics*
12:40 Lunch Break
14:30 Willem L. Fouché (Invited Speaker), *Ramsey Theory and the Symmetries of Countably Categorical Structure*
15:30 Coffee Break
15:50 Joseph S. Miller (Invited Speaker), *Extracting Information Is Hard*
16:50 Excursion
19:00 Conference Dinner

Monday, 5 November 2007
09:30 Helmut Prodinger (Invited Speaker), *Infinity in Combinatorics: Asymptotic Enumeration*
10:30 Coffee Break
11:00 Margaret Archibald and Arnold Knopfmacher, *The Average Position of the First Maximum in a Sample of Geometric Random Variables*
11:20 George Davie, *Complexity Withholding Strings*
11:40 Petrus H. Potgieter and Elemér E. Rosinger, *Three Perspectives on Right-Sizing the Infinite*

12:00 Willem L. Fouché, *Fractal Geometry of an Algorithmically Random Brownian Motion*
12:20 Lunch Break
14:30 Yoshiki Tsujii, Takakazu Mori, Mariko Yasugi and Hideki Tsuiki, *Fractals Defined by Infinite Contractions and Mutual-Recursive Sets*
14:50 Hideki Tsuiki and Shuji Yamada, *On Finite-Time Computable Functions*
15:10 Boris Melnikov and Elena Melnikova, *Some More on the Billiard Languages and Corresponding Forbidden Languages*
15:30 Alain Finkel, Étienne Lozes and Arnaud Sangnier, *Towards Model-Checking Pointer Systems*
15:50 Adrian R. D. Mathias, *Easy Games with Hard Strategies*
16:10 Closing

For more information about the conference, please check the website at

`http://www.mth.uct.ac.za/FACS-Lab/ILC07/`.

March 2009

Margaret Archibald
Vasco Brattka
Valentin Goranko
Benedikt Löwe

Table of Contents

Nadia Busi (1968–2007) 1
Claudio Zandron

Symbolic Model Checking of Tense Logics on Rational Kripke Models 2
Wilmari Bekker and Valentin Goranko

Genetic Systems without Inhibition Rules 21
Nadia Busi and Claudio Zandron

Probabilistic Systems with LimSup and LimInf Objectives 32
Krishnendu Chatterjee and Thomas A. Henzinger

A Playful Glance at Hierarchical Questions for Two-Way Alternating Automata 46
Jacques Duparc and Alessandro Facchini

Towards Model-Checking Programs with Lists 56
Alain Finkel, Étienne Lozes, and Arnaud Sangnier

Representations of Numbers as $\sum_{k=-n}^{n} \varepsilon_k k$: A Saddle Point Approach 87
Guy Louchard and Helmut Prodinger

Sets of Infinite Words Recognized by Deterministic One-Turn Pushdown Automata 97
Douadi Mihoubi

Fine-Continuous Functions and Fractals Defined by Infinite Systems of Contractions 109
Yoshiki Tsujii, Takakazu Mori, Mariko Yasugi, and Hideki Tsuiki

Is **P** = **PSPACE** for Infinite Time Turing Machines? 126
Joost Winter

Author Index 139

Nadia Busi (1968–2007)

Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione, Università di Milano-Bicocca, viale Sarca 336, I-20126, Milano, Italy
zandron@disco.unimib.it



Nadia Busi obtained a Master in Computer Science in 1993, at the University of Bologna. In 1997 she obtained a PhD in Theoretical Computer Science at the University of Siena; her thesis entitled "Petri Nets with Inhibitor and Read Arcs: Semantics, Analysis and Application to Process Calculi" won the annual prize of the Italian Chapter of EATCS for the best Italian PhD thesis.

The scientific activity of Nadia was very broad, and covered different aspects of theoretical computer science. Her first interests concerned the investigation of expressiveness problems in concurrency theory. In collaboration with Roberto Gorrieri and Gianluigi Zavattaro, she studied primitive nets, a subclass of Petri nets with inhibitor arcs, investigating various decidable properties of such nets. She also studied different properties of various process calculi, by mapping them on primitive nets.

Other aspects of her scientific activity on this subject, concerned the study of the relative expressive power of recursion, replication and iteration in basic process calculi. She also studied some aspects of Petri nets with read and inhibitor arcs as well as service-oriented computing, security and stochastic Petri nets.

I had the privilege to meet and collaborate with Nadia a few years ago, when she started the investigation of bio–inspired computational models. In particular, she studied Cardelli's brane calculi and Paun's membrane systems, producing a number of results concerning the relations between these formalisms, defining their computational properties, and considering their application in the framework of systems biology. We developed together a new class of computational models inspired from the functioning of genetic gates, called genetic systems, and we started the investigation of the properties of such models.

It has been shocking to hear that on September 5, 2007, Nadia passed away suddenly, after a brief illness; she was only 39 years old. She was able to involve various colleagues in her work, sharing her enthusiasm, knowledge, and her ideas. The scientific community lost a prolific and enthusiastic member but, most of all, Nadia leaves a deep hole in many people who met her, who lost a sincere friend.

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, p. 1, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Symbolic Model Checking of Tense Logics on Rational Kripke Models

Wilmari Bekker[1,2] and Valentin Goranko[2,⋆]

[1] Department of Mathematics, University of Johannesburg, PO Box 524, Auckland Park, 2006, South Africa
bekkerw@gmail.com
[2] School of Mathematics, University of the Witwatersrand, Private Bag 3, Wits 2050, South Africa
goranko@maths.wits.ac.za

Abstract. We introduce the class of *rational Kripke models* and study symbolic model checking of the basic tense logic $\mathbf{K}_t$ and some extensions of it on that class. Rational Kripke models are based on (generally infinite) *rational graphs*, with vertices labeled by the words in some regular language and transitions recognized by asynchronous two-head finite automata, also known as *rational transducers*. Every atomic proposition in a rational Kripke model is evaluated in a rational set of states. We show that every formula of $\mathbf{K}_t$ has an effectively computable rational extension in every rational Kripke model, and therefore local model checking and global model checking of $\mathbf{K}_t$ in rational Kripke models are decidable. These results are lifted to a number of extensions of $\mathbf{K}_t$. We study and partly determine the complexity of the model checking procedures.

1 Introduction

Verification of models with infinite state spaces using algorithmic symbolic model checking techniques has been an increasingly active area of research over recent years. One very successful approach to infinite state verification is based on the representation of sets of states and transitions by means of automata. It is the basis of various automata-based techniques for model checking, e.g., of linear and branching-time temporal logics on finite transition systems [23,17], regular model checking [7], pushdown systems [8,24,11], automatic structures [14,6] etc.

⋆ This research has been supported by the National Research Foundation of South Africa through a research grant and a student bursary. Part of the work of the second author was done during his visit to the Isaac Newton Institute, Cambridge, as a participant to the 'Logic and Algorithms' programme in 2006. We wish to thank Arnaud Carayol, Balder ten Cate, Carlos Areces, Christophe Morvan, and Stéphane Demri, for various useful comments and suggestions. We are also grateful to the anonymous referee for his/her careful reading of the submitted version and many remarks and corrections which have helped us improve the content and presentation of the paper. (*Received by the editors.* 10 February 2008. *Revised.* 18 September 2008; 15 October 2008. *Accepted.* 17 October 2008.)

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, pp. 2–20, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In most of the studied cases of infinite-state symbolic model checking (except for automatic structures), the logical languages are sufficiently expressive for various reachability properties, but the classes of models are relatively restricted.

In this paper we study a large and natural class of *rational Kripke models*, on which global model checking of the basic tense[1] logic $\mathbf{K}_t$ (with forward and backward one-step modalities) and of some extensions thereof, are decidable. The language of $\mathbf{K}_t$ is sufficient for expressing *local properties*, i.e., those referring to a bounded width neighborhood of predecessors or successors of the current state. In particular, pre-conditions and post-conditions are local, but not reachability properties. Kesten et al. [15] have formulated the following minimal requirements for an *assertional language $\mathcal{L}$ to be adequate for symbolic model checking*:

1. The property to be verified and the initial conditions (i.e., the set of initial states) should be expressible in $\mathcal{L}$.
2. $\mathcal{L}$ should be effectively closed under the boolean operations, and should possess an algorithm for deciding equivalence of two assertions.
3. There should exist an algorithm for constructing the predicate transformer pred, where $\mathrm{pred}(\phi)$ is an assertion characterizing the set of states that have a successor state satisfying ϕ.

Assuming that the property to be verified is expressible in $\mathbf{K}_t$, the first condition above is satisfied in our case. Regarding the set of initial states, it is usually assumed a singleton, but certainly an effective set, and it can be represented by a special modal constant S. The second condition is clearly satisfied, assuming the equivalence is with respect to the model on which the verification is being done. As for the third condition, $\mathrm{pred}(\phi) = \langle R \rangle \phi$. Thus, the basic modal logic $\mathbf{K}$ is *the minimal natural logical language satisfying these requirements*, and hence it suffices for specification of *pre-conditions* over regular sets of states. The tense extension $\mathbf{K}_t$ enables specification of post-conditions, as well, thus being the basic adequate logic for specifying *local properties* of transition systems and warranting the potential utility of the work done in the present paper. In particular, potential areas of applications of model checking of the basic tense logic to verification of infinite state systems are *bounded model checking* [2], applied to infinite state systems, and (when extended with reachability) *regular model checking* [7] – a framework for algorithmic verification of generally infinite state systems which essentially involves computing reachability sets in regular Kripke models.

The paper is organized as follows: in Section 2 we introduce $\mathbf{K}_t$ and rational transducers. Section 3 introduces and discusses rational Kripke models, and in Section 4 we introduce synchronized products of transducers and automata. We use them in Section 5 to show decidability of global and local symbolic model checking of $\mathbf{K}_t$ in rational Kripke models and in Section 6 we discuss its complexity. The model checking results are strengthened in Section 7 to hybrid and other extensions of $\mathbf{H}_t(U)$, for which some model checking tasks remain decidable.

[1] We use the term 'tense' rather than 'temporal' to emphasize that the accessibility relation is not assumed transitive, as in a usual flow of time.

2 Preliminaries

2.1 The Basic Tense Logic $\mathbf{K}_t$

We consider transition systems with one transition relation R. The *basic tense logic* $\mathbf{K}_t$ for such transition systems extends the classical propositional logic with two unary modalities: one associated with R and the other with its inverse R^{-1}, respectively denoted by $[R]$ and $[R^{-1}]$. The generalization of what follows to the case of languages and models for transition systems with many relations is straightforward. Note that the relation R is not assumed transitive, and therefore the language of $\mathbf{K}_t$ cannot express R-reachability properties.

2.2 Rational Transducers and Rational Relations

Rational transducers, studied by Eilenberg [9], Elgot and Mezei [10], Nivat, Berstel [1], etc., are *asynchronous automata* on pairs of words. Intuitively, these are finite automata with two autonomous heads that read the input pair of words asynchronously, i.e. each of them can read arbitrarily farther ahead of the other. The transitions are determined by a finite set of pairs of (possibly empty) words; alternatively, a transition can be labeled either by a pair of letters (when both heads make a move on their respective words) or by $\langle a, \epsilon\rangle$ or $\langle \epsilon, a\rangle$, where a is a letter, and ϵ is the empty word (when one of the heads reads on, while the other is waiting). The formal definition follows.

Definition 1. *A **(rational) transducer** is a tuple $\mathcal{T} = \langle Q, \Sigma, \Gamma, q_i, F, \rho\rangle$ where Σ and Γ are the input and output alphabets respectively, Q a set of states, $q_i \in Q$ a unique starting state, $F \subseteq Q$ a set of accepting states and $\rho \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q$ is the transition relation, consisting of* finitely many *tuples, each containing the current state, the pair of letters (or ε) triggering the transition, and the new state. Alternatively, one can take $\rho \subseteq Q \times \Sigma^* \times \Gamma^* \times Q$.*

The language recognized by the transducer $\mathcal{T}$ is the set of all pairs of words for which it has a reading that ends in an accepting state. Thus, the transducer $\mathcal{T}$ recognizes a binary relation $R \subseteq \Sigma^* \times \Gamma^*$.

This is the 'static' definition of rational transducers; they can also be defined 'dynamically', as reading an input word, and transforming it into an output word, according to the transition relation which is now regarded as a mapping from words to sets of words (because it can be non-deterministic).

Example 1. For $\mathcal{T} = \langle Q, \Sigma, \Gamma, q_i, F, \rho\rangle$ let: $Q = \{q_1, q_2\}$; $\Sigma = \{0, 1\} = \Gamma$; $q_i = q_1$; $F = \{q_2\}$; $\rho = \{(q_1, 0, 0, q_1), (q_1, 1, 1, q_1), (q_1, \epsilon, 0, q_2), (q_1, \epsilon, 1, q_2)\}$

Notice that in the representation of $\mathcal{T}$ there is only one edge between two states but that an edge may have more than one label.

A relation $R \subseteq \Sigma^* \times \Gamma^*$ is **rational** if it is recognizable by a rational transducer. Equivalently (see [1]), given finite alphabets Σ, Γ, a (binary) **rational relation** over (Σ, Γ) is a rational subset of $\Sigma^* \times \Gamma^*$, i.e., a subset generated by a rational

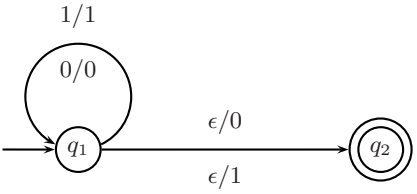


Fig. 1. The transducer $\mathcal{T}$ which recognizes pairs of words of the forms $(u, u0)$ or $(u, u1)$ where $u \in \Sigma^*$

expression (built up using union, concatenation, and iteration) over a finite subset of $\Sigma^* \times \Gamma^*$. Hereafter, we will assume that the input and output alphabets Σ and Γ coincide.

Besides the references above, rational relations have also been studied by Johnson [13], Frougny and Sakarovich [12], and more recently by Morvan [20]. It is important to note that the class of rational relations is closed under *unions*, *compositions*, and *inverses* [1]. On the other hand, the class of rational relations is not closed under intersections, complements, and transitive closure (*ibid*).

3 Rational Kripke Models

3.1 Rational Graphs

Definition 2. *A graph $\mathcal{G} = (S, E)$ is **rational**, if the set of vertices S is a regular language in some finite alphabet Σ and the set of edges E is a rational relation on Σ.*

Example 2. **The infinite grid.** Let $\Sigma = \{0, 1\}$, then the infinite grid with vertices in Σ^* is given by Figure 2 and the edge relation of this graph is recognized by the transducer given in Figure 2.

Example 3. The **complete binary tree** Λ.
Figure 3 contains the complete binary tree with vertices in $\{0, 1\}^*$ and labeled by $\Gamma = \{a, b\}$, as well as the transducer recognizing it, in which the accepting states are labeled respectively by a and b. The pairs of words for which the transducer ends in the accepting state q_4 belong to the left successor relation in the tree (labeled by a), and those for which the transducer ends in the accepting state q_5 belong to the right successor relation in the tree (labeled by b).

An important and extensively studied subclass of rational graphs is the class of *automatic graphs* [14,6]. These are rational graphs whose transition relations are recognized by *synchronized* transducers.

As shown by Blumensath [5], the configuration graph of every Turing machine is an automatic graph. Consequently, important queries, such as reachability, are generally undecidable on automatic graphs, and hence on rational graphs. Furthermore, Morvan showed in [20] that the configuration graphs of Petri nets [21] are rational (in fact, automatic) graphs, too.

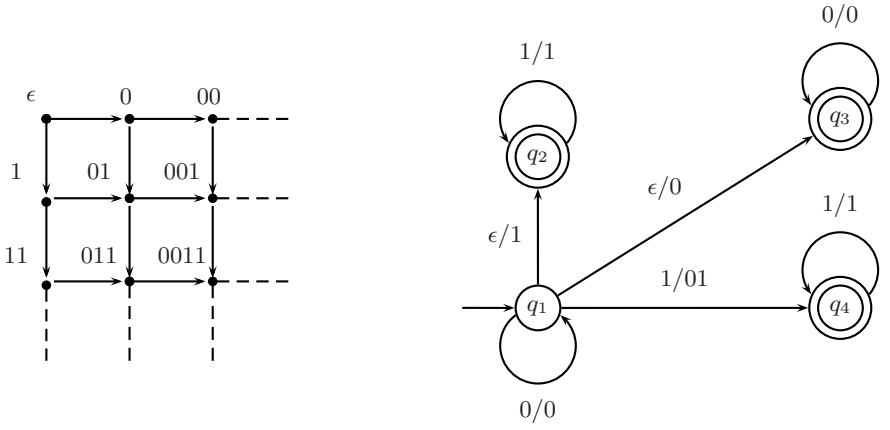


Fig. 2. The infinite grid with set of vertices $S = 0^*1^*$ and a transducer that recognizes the infinite grid

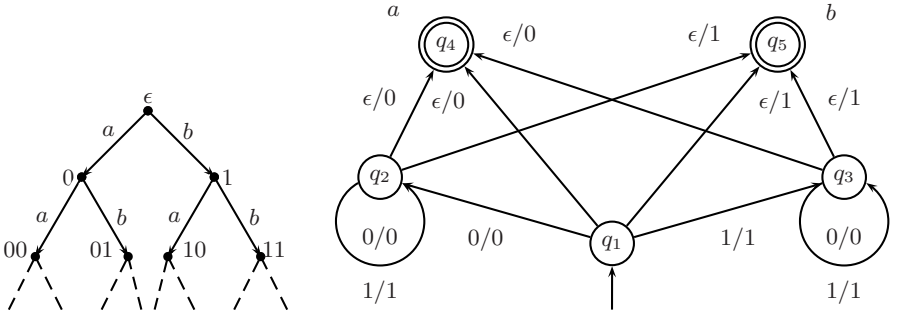


Fig. 3. The complete binary tree Λ and a labeled transducer recognizing it

Moreover, Johnson [13] proved that even very simple first-order definable properties of a rational relation, e.g., reflexivity, transitivity, symmetry, turn out to be undecidable (with an input the transducer recognizing the relation), by reduction from the Post Correspondence Problem (PCP). Independently, Morvan [20] has shown that the query $\exists x Rxx$ on rational frames is undecidable, as well. The reduction of PCP here is straightforward: given a PCP $\{(u_1, v_1), \ldots, (u_n, v_n)\}$, consider a transducer with only one state, which is both initial and accepting, and it allows the transitions $(u_1, v_1), \ldots, (u_n, v_n)$. Then, the PCP has a solution precisely if some pair (w, w) is accepted by the transducer. Inclusion and equality of rational relations are undecidable, too, [1].

Furthermore, in [22] W. Thomas has constructed a single rational graph with undecidable first-order theory, by encoding the halting problem of a universal Turing machine.

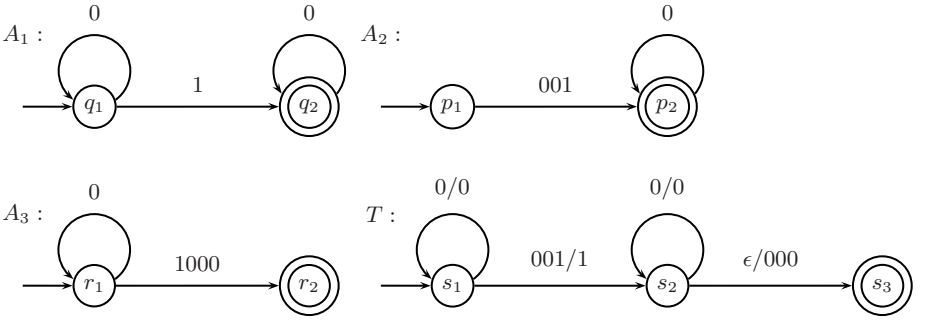


Fig. 4. A finite presentation $\mathcal{M}$: A_1, A_2 and A_3 recognize $S, V(p)$ and $V(q)$ respectively, and T recognizes R

3.2 Rational Kripke Models

Rational graphs can be viewed as Kripke frames, hereafter called *rational Kripke frames.*

Definition 3. *A Kripke model $\mathcal{M} = (\mathcal{F}, V) = (S, R, V)$ is a **rational Kripke model** (RKM) if the frame $\mathcal{F}$ is a rational Kripke frame, and the valuation V assigns a regular language to each propositional variable, i.e., $V(p) \in \mathsf{REG}(\Sigma^*)$ for every $p \in \Phi$. A valuation satisfying this condition is called a rational valuation.*

Example 4. In this example we will present a RKM based on the configuration graph of a Petri net. To make it self-contained, we give the basic relevant definitions here; for more details, cf., e.g., [21]. A Petri net is a tuple (P, T, F, M) where P and T are disjoint finite sets and their elements are called *places* and *transitions* respectively. $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is called a flow function and is such that if $F(x, y) > 0$ then there is an arc from x to y and $F(x, y)$ is the multiplicity of that arc. Each of the places contain a number of *tokens* and a vector of integers $M \in \mathbb{N}^{|P|}$ is called a *configuration* (or, *marking*) of the Petri net if the i^{th} component of M is equal to the number of tokens at the i^{th} place in the Petri net. The *configuration graph* of $\mathcal{N}$ has as vertices all possible configurations of $\mathcal{N}$ and the edges represent the possible transitions between configurations.

Now, let $\mathcal{N} = (P, T, F, M)$ be a Petri net, where $P = \{p_1, p_2\}$, $T = \{t\}$, $F(p_1, t) = 2$, $F(t, p_2) = 3$ and $M = (4, 5)$. Let $\mathcal{M} = (S, R, V)$ where $S = 0^*10^*$, R the transition relation of the configuration graph of $\mathcal{N}$ and V the valuation defined by $V(p) = 0010^*$ and $V(q) = 0^*1000$. Then $\mathcal{M}$ is a RKM and can be presented by the various machines in Figure 4.

4 Synchronized Products of Transducers and Automata

In this section ϵ will denote the empty word, but will also be treated as a special symbol in an extended alphabet.

Definition 4. *Let u be a word in some alphabet Γ and $\gamma \in \Gamma$. The* ***γ-reduction*** *of u, denoted $u|_\gamma$, is the word obtained from u after deleting all occurrences of γ. Likewise, if Y is a language in the alphabet Γ, then the* ***γ-reduction*** *of Y, denoted $Y|_\gamma$, is the language consisting of all γ-reductions of words in Y.*

Lemma 1. *If Y is a regular language over an alphabet Γ then $Y|_\gamma$ is a regular language over the alphabet $\Gamma - \{\gamma\}$.*

Proof. (Sketch) An automaton $\mathcal{A}|_\gamma$ recognizing $Y|_\gamma$, called here the **γ-reduction** of $\mathcal{A}$ can be constructed from an automaton $\mathcal{A}$ recognizing Y as follows:

1. Remove all γ-transitions.
2. Add (q, γ', q'') as a transition in $\mathcal{A}|_\gamma$ whenever (q, γ, q') and (q', γ', q'') are transitions in $\mathcal{A}$ and $\gamma \neq \gamma'$.
3. Finally, define the accepting states of $\mathcal{A}|_\gamma$ as all accepting states of $\mathcal{A}$ plus those states q such that $(q \overset{\gamma^*}{\rightarrow} q')$ in $\mathcal{A}$ and q' is an accepting state in $\mathcal{A}$. ■

Definition 5. *A* ***run*** *of a finite automaton $\mathcal{A} = \langle Q, \Sigma, q^0, F, \delta \rangle$ is a sequence of states and transitions of $\mathcal{A}$: $q_0 \overset{x_1}{\rightarrow} q_1 \overset{x_2}{\rightarrow} q_2 \cdots \overset{x_n}{\rightarrow} q_n$, such that $q_0 = q^0$, $q_j \in Q, x_j \in \Sigma$, and $q_j \in \delta(q_{j-1}, x_j)$ for every $j = 1, 2, \ldots, n$.*

A run is ***accepting*** *if it ends in an accepting state.*

Run and accepting runs of transducers are defined likewise.

Definition 6. *A* ***stuttering run*** *of a finite automaton $\mathcal{A} = \langle Q, \Sigma, q^0, F, \delta \rangle$ is a sequence $q_0 \overset{x_1}{\rightarrow} q_1 \overset{x_2}{\rightarrow} q_2 \cdots \overset{x_n}{\rightarrow} q_n$, such that $q_0 = q^0$, $q_j \in Q$, and either $x_j \in \Sigma$ and $q_j \in \delta(q_{j-1}, x_j)$, or $x_j = \epsilon$ and $q_j = q_{j-1}$ for every $j = 1, 2, \ldots, n$.*

Thus, a stuttering run of an automaton can be obtained by inserting ϵ-transitions from a state to itself into a run of that automaton. If the latter run is accepting, we declare the stuttering run to be an ***accepting stuttering run****.*

A ***stuttering word*** *in an alphabet Σ is any word in $\Sigma \cup \{\epsilon\}$.*

The ***stuttering language*** *of the automaton $\mathcal{A}$ is the set $L^\epsilon(\mathcal{A})$ of all stuttering words whose ϵ-reductions are recognized by $\mathcal{A}$; equivalently, all stuttering words for which there is an accepting stuttering run of the automaton.*

Definition 7. *Let $\mathcal{T} = \langle Q_\mathcal{T}, \Sigma, q^0_\mathcal{T}, F_\mathcal{T}, \rho_\mathcal{T} \rangle$ be a transducer, and let $\mathcal{A}$ be a (non-deterministic) finite automaton given by $\mathcal{A} = \langle Q_\mathcal{A}, \Sigma, q^0_\mathcal{A}, F_\mathcal{A}, \delta_\mathcal{A} \rangle$.*

The ***synchronized product*** *of $\mathcal{T}$ with $\mathcal{A}$ is the finite automaton:*

$$\mathcal{T} \prec \mathcal{A} = \langle Q_\mathcal{T} \times Q_\mathcal{A}, \Sigma, (q^0_\mathcal{T}, q^0_\mathcal{A}), F_\mathcal{T} \times F_\mathcal{A}, \delta_{\mathcal{T} \prec \mathcal{A}} \rangle$$

where $\delta_{\mathcal{T} \prec \mathcal{A}} : (Q_\mathcal{T} \times Q_\mathcal{A}) \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q_\mathcal{T} \times Q_\mathcal{A})$ is such that, for any $p^1_\mathcal{T}, p^2_\mathcal{T} \in Q_\mathcal{T}$ and $p^1_\mathcal{A}, p^2_\mathcal{A} \in Q_\mathcal{T}$ then $(p^2_\mathcal{T}, p^2_\mathcal{A}) \in \delta_{\mathcal{T} \prec \mathcal{A}}((p^1_\mathcal{T}, p^1_\mathcal{A}), x)$ if and only if

1. *either there exists a $y \in \Sigma$ such that $\delta_\mathcal{A}(p^1_\mathcal{A}, y) = p^2_\mathcal{A}$ and $(p^1_\mathcal{T}, x, y, p^2_\mathcal{T}) \in \rho_\mathcal{T}$,*
2. *or $(p^1_\mathcal{T}, x, \epsilon, p^2_\mathcal{T}) \in \rho_\mathcal{T}$ and $p^1_\mathcal{A} = p^2_\mathcal{A}$.*

Note that every run $R_{\mathcal{T}\prec\mathcal{A}} = (p^0_{\mathcal{T}}, p^0_{\mathcal{A}}) \xrightarrow{u_1} (p^1_{\mathcal{T}}, p^1_{\mathcal{A}}) \xrightarrow{u_2} \cdots \xrightarrow{u_n} (p^n_{\mathcal{T}}, p^n_{\mathcal{T}})$ of the automaton $\mathcal{T} \prec \mathcal{A}$ can be obtained from a pair:

a run $R_{\mathcal{T}} = p^0_{\mathcal{T}} \xrightarrow{(u_1/w_1)} p^1_{\mathcal{T}} \xrightarrow{(u_2/w_2)} p^2_{\mathcal{T}} \cdots \xrightarrow{(u_n/w_n)} p^n_{\mathcal{T}}$ in $\mathcal{T}$,
and a stuttering run $R^s_{\mathcal{A}} = p^0_{\mathcal{A}} \xrightarrow{w_1} p^1_{\mathcal{A}} \xrightarrow{w_2} p^2_{\mathcal{A}} \cdots \xrightarrow{w_n} p^n_{\mathcal{A}}$ in $\mathcal{A}$,
by pairing the respective states $p^j_{\mathcal{T}}$ and $p^j_{\mathcal{A}}$ and removing the output symbol w_j for every $j = 1, 2, \ldots, n$.

Let the reduction of $R^s_{\mathcal{A}}$ be the run $R_{\mathcal{A}} = q^0_{\mathcal{A}} \xrightarrow{v_1} q^1_{\mathcal{A}} \xrightarrow{v_2} q^2_{\mathcal{A}} \cdots \xrightarrow{v_m} q^m_{\mathcal{A}}$, with $m \leq n$. Then we say that the run $R_{\mathcal{T}\prec\mathcal{A}}$ is a **synchronization of the runs** $R_{\mathcal{T}}$ and $R_{\mathcal{A}}$.

Note, that the synchronization of accepting runs of $\mathcal{T}$ and $\mathcal{A}$ is an accepting run of $R_{\mathcal{T}\prec\mathcal{A}}$. The following lemma is now immediate:

Lemma 2. *Let $\mathcal{T} = \langle Q_{\mathcal{T}}, \Sigma, q^0_{\mathcal{T}}, F_{\mathcal{T}}, \rho_{\mathcal{T}} \rangle$ be a transducer recognizing the relation $R(\mathcal{T})$ and let $\mathcal{A} = \langle Q_{\mathcal{A}}, \Sigma, q^0_{\mathcal{A}}, F_{\mathcal{A}}, \delta_{\mathcal{A}} \rangle$ be a finite automaton recognizing the language $L(\mathcal{A})$. Then the language recognized by the synchronized product of $\mathcal{T}$ and $\mathcal{A}$ is*

$$L(\mathcal{T} \prec \mathcal{A}) = \{u \mid \exists w \in L^{\epsilon}(\mathcal{A})(uR(\mathcal{T})w).\}$$

5 Model Checking of K_t in Rational Kripke Models

In this section we will establish decidability of the basic model checking problems for formulae of K_t in rational Kripke models.

Lemma 3. *Let Σ be a finite non-empty alphabet, $X \subseteq \Sigma^*$ a regular subset, and let $R \subseteq \Sigma^* \times \Sigma^*$ be a rational relation. Then the sets*

$$\langle R \rangle X = \{u \in \Sigma^* | \exists v \in X(uRv)\}$$

and

$$\left\langle R^{-1} \right\rangle X = \{u \in \Sigma^* | \exists v \in X(vRu)\}$$

are regular subsets of Σ^*.

Proof. This claim essentially follows from results of Nivat (see [1]). However, using Lemmas 1 and 2, we give a constructive proof, which explicitly produces automata that recognize the resulting regular languages. Let $\mathcal{A}$ be a finite automaton recognizing X and $\mathcal{T}$ be a transducer recognizing R. Then, the ϵ-reduction of the synchronized product of $\mathcal{T}$ with $\mathcal{A}$ is an automaton recognizing $\langle R \rangle X$; for $\left\langle R^{-1} \right\rangle X$ we take instead of $\mathcal{T}$ the transducer for R^{-1} obtained from $\mathcal{T}$ by swapping the input and output symbols in the transition relation[2]. ■

Example 5. Consider the automaton $\mathcal{A}$ and transducer $\mathcal{T}$ in Figure 5. The language recognized by $\mathcal{A}$ is $X = 1^*(1 + 0^+)$ and the relation R recognized by $\mathcal{T}$ is $R = \left\{(1^n 0, 10^n 1)^m \left(1^k, 10^k\right) \mid n, m, k \in \mathbb{N}\right\} \cup$

[2] Note that, in general, the resulting automata need not be minimal, because they may have redundant states and transitions.

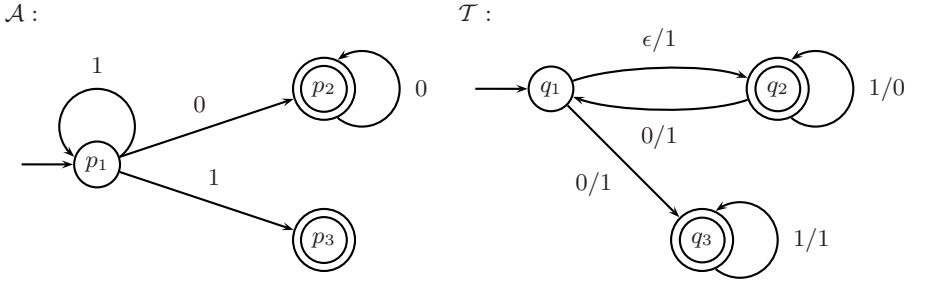


Fig. 5. The automaton $\mathcal{A}$ and the transducer $\mathcal{T}$

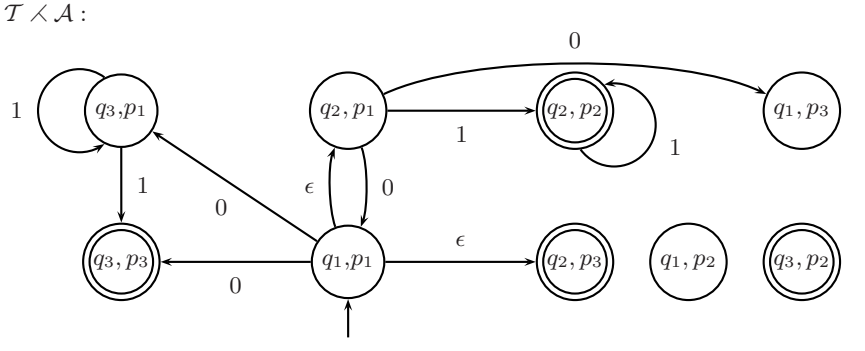


Fig. 6. The synchronized product $\mathcal{T} \asymp \mathcal{A}$ recognizing $\langle R \rangle X$

$\left\{(1^n0, 10^n1)^m \left(01^k, 11^k\right) \mid n, m, k \in \mathbb{N}\right\}$, where X_1X_2 denotes the componentwise concatenation of the relations X_1 and X_2, i.e., $X_1X_2 = \{(u_1u_2, v_1v_2) \mid (u_1, v_1) \in X_1, (u_2, v_2) \in X_2\}$. For instance, if we take $n = 1$, $m = 2$ and $k = 3$ we obtain that $(10, 101)^2(1^3, 10^3) = (1010111, 1011011000) \in R$ (coming from the first set of the union) and $(10, 101)^2(01^3, 11^3) = (10100111, 1011011111) \in R$ (coming from the second set of that union).

Then, the synchronized product $\mathcal{T} \asymp \mathcal{A}$ is the finite automaton given in Figure 6 recognizing $\langle R \rangle X = 0^* + 0^*1^+$. Note that it can be simplified by removing redundant states and edges.

Theorem 1. *For every formula $\varphi \in \mathbf{K}_t$ and rational Kripke model $\mathcal{M} = (\Sigma^*, R, V)$, the set $[\![\varphi]\!]_{\mathcal{M}}$ is a rational language, effectively computable from φ and the rational presentation of $\mathcal{M}$.*

Proof. We prove the claim by induction on φ.

1. If φ is an atomic proposition, the claim follows from the definition of a rational model.
2. The boolean cases follow from the effective closure of regular languages under boolean operations.

3. If $\varphi = \langle R \rangle \psi$ then $[\![\varphi]\!]_{\mathcal{M}} = \langle R \rangle [\![\psi]\!]_{\mathcal{M}}$, which is regular by the inductive hypothesis and Lemma 3. Likewise for the case $\varphi = \langle R^{-1} \rangle \psi$. ■

We now consider the following algorithmic model checking problems, where the Kripke model is supposed to be given by some effective presentation:

1. *Local model checking:* given a Kripke model $\mathcal{M}$, a state s in $\mathcal{M}$, and a formula φ of $\mathbf{K}_t$, determine whether $\mathcal{M}, s \models \varphi$.
2. *Global model checking:* given a Kripke model $\mathcal{M}$ and a formula φ of $\mathbf{K}_t$, determine (effectively) the set $[\![\varphi]\!]_{\mathcal{M}}$ of all states in $\mathcal{M}$ where φ is true.
3. *Checking satisfiability in a model:* given a Kripke model $\mathcal{M}$ and a formula φ of $\mathbf{K}_t$, determine whether $[\![\varphi]\!]_{\mathcal{M}} \neq \varnothing$.

Corollary 1. *Local model checking, global model checking, and checking satisfiability in a model, of formulae in* $\mathbf{K}_t$ *in rational Kripke models are decidable.*

Proof. Decidability of the global model checking follows immediately from Theorem 1. Then, decidability of the local model checking and of checking satisfiability in a rational model follow respectively from the decidability of membership in a regular language, and of non-emptiness of a regular language (cf., e.g., [18]). ■

6 Complexity

We will now attempt to analyze the complexity of global model checking a formula in $\mathbf{K}_t$ on a rational Kripke model. Depending on which of these is fixed, we distinguish two complexity measures (cf., e.g., [16]): **formula (expression) complexity** (when the model is fixed and the formula is feeded as input) and **structure complexity** (when the formula is fixed and the model is feeded as input).

6.1 Normal Forms and Ranks of Formulae

We will first need to define some standard technical notions.

A formula $\varphi \in \mathbf{K}_t$ is in **negation normal form** if every occurrence of the negation immediately precedes a propositional variable. Clearly every formula $\varphi \in \mathbf{K}_t$ is equivalent to a formula $\psi \in \mathbf{K}_t$ in negation normal form, of size linear in the size φ. For the remainder of this section, we will assume that a formula φ we wish to model check is in a negation normal form.

The modal rank of a formula counts the greatest number of nested modalities in the formula, while the alternating box (resp., diamond) rank of a formula counts the greatest number of nested alternations of modalities with an outmost box (resp., diamond) in that formula. Formally:

Definition 8. *The* **modal rank** *for a formula* $\varphi \in \mathbf{K}_t$*, denoted by* $\mathrm{mr}(\varphi)$ *is defined inductively as follows:*

1. *if* p *is an atomic proposition, then* $\mathrm{mr}(p) = 0$ *and* $\mathrm{mr}(\neg p) = 0$*;*

2. $\mathrm{mr}(\phi_1 \vee \psi_2) = \mathrm{mr}(\phi_1 \wedge \psi_2) = \max\{\mathrm{mr}(\psi_1), \mathrm{mr}(\psi_2)\}$;
3. $\mathrm{mr}(\triangle\, \psi) = \mathrm{mr}(\psi) + 1$ *where* $\triangle \in \{[R], \langle R\rangle, [R^{-1}], \langle R^{-1}\rangle\}$.

Definition 9. *The* **alternating box rank** *and* **alternating diamond rank** *of a formula* $\varphi \in \mathbf{K}_t$*, denoted respectively by* $\mathrm{ar}_\Box(\varphi)$ *and* $\mathrm{ar}_\Diamond(\varphi)$*, are defined by simultaneous induction as follows, where* $\triangle \in \{\Box, \Diamond\}$*:*

1. *if* p *is an atomic proposition, then* $\mathrm{ar}_\triangle(p) = 0$ *and* $\mathrm{ar}_\triangle(\neg p) = 0$;
2. $\mathrm{ar}_\triangle(\psi_1 \vee \psi_2) = \mathrm{ar}_\triangle(\psi_1 \wedge \psi_2) = \max\{\mathrm{ar}_\triangle(\psi_1), \mathrm{ar}_\triangle(\psi_2)\}$;
3. $\mathrm{ar}_\Diamond(\langle R\rangle \psi) = \mathrm{ar}_\Box(\psi) + 1$ *and* $\mathrm{ar}_\Box(\langle R\rangle \psi) = \mathrm{ar}_\Box(\psi)$.
   Likewise for $\mathrm{ar}_\Diamond(\langle R^{-1}\rangle \psi)$ *and* $\mathrm{ar}_\Box(\langle R^{-1}\rangle \psi)$.
4. $\mathrm{ar}_\Box([R]\psi) = \mathrm{ar}_\Diamond(\psi) + 1$ *and* $\mathrm{ar}_\Diamond([R]\psi) = \mathrm{ar}_\Diamond(\psi)$.
   Likewise for $\mathrm{ar}_\Diamond([R^{-1}]\psi)$ *and* $\mathrm{ar}_\Box([R^{-1}]\psi)$.

Finally, the **alternation rank** *of* φ*, denoted* $ar(\varphi)$ *is defined to be*

$$ar(\varphi) = \max\{\mathrm{ar}_\Box(\varphi), \mathrm{ar}_\Diamond(\varphi)\}.$$

For instance, $\mathrm{ar}_\Box([R](\langle R\rangle [R]p \vee [R][R^{-1}]\neg q)) = 3$ and $\mathrm{ar}_\Diamond([R](\langle R\rangle [R]p \vee [R][R^{-1}]\neg q)) = 2$, hence $ar([R](\langle R\rangle [R]p \vee [R][R^{-1}]\neg q)) = 3$.

6.2 Formula Complexity

We measure the size of a finite automaton or transducer $\mathcal{M}$ by the number of transition edges in it, denoted $|\mathcal{M}|$.

Proposition 1. *If* $\mathcal{A}$ *is an automaton recognizing the regular language* X *and* $\mathcal{T}$ *a transducer recognizing the rational relation* R*, then the time complexity of computing an automaton recognizing* $\langle R\rangle^m X$ *is in* $O(|\mathcal{T}|^m|\mathcal{A}|)$.

Proof. The size of the synchronized product $\mathcal{T} \curlywedge \mathcal{A}$ of $\mathcal{T}$ and $\mathcal{A}$ is bounded above by $|\mathcal{T}||\mathcal{A}|$ and it can be computed in time $O(|\mathcal{T}||\mathcal{A}|)$. The claim now follows by iterating that procedure m times. ■

However, we are going to show that the time complexity of computing an automaton recognizing $[R]X$ is far worse.

For a regular language X recognized by an automaton $\mathcal{A}$, we define $R_X = \{(u, \epsilon) \,|\, u \in X\}$. A transducer $\mathcal{T}$ recognizing R_X can be constructed from $\mathcal{A}$ by simply replacing every edge (q, x, p) in $\mathcal{A}$ with the edge (q, x, ϵ, p).

Lemma 4. *Let* X *be a regular language. Then the complementation* $\overline{X}$ *of* X *equals* $[R_X]\varnothing$.

Proof. Routine verification. ■

Consequently, computing $[R_X]\varnothing$ cannot be done in less than exponential time in the size of the (non-deterministic) automaton $\mathcal{A}$ for X. This result suggests the following conjecture.

Conjecture 1. The formula complexity of global model checking of a $\mathbf{K}_t$-formula is non-elementary in terms of the alternating box rank of the formula.

6.3 Structure Complexity

Next we analyze the **structure complexity**, i.e. the complexity of global model checking a fixed formula $\varphi \in \mathbf{K}_t$ on an input rational Kripke model. Here the input is assumed to be the transducer and automata presenting the model.

Fix a formula $\varphi \in \mathbf{K}_t$ in negation normal form, then for any input rational Kripke model $\mathcal{M}$ there is a fixed number of operations to perform on the input transducer and automata that can lead to subsequent exponential blowups of the size of the automaton computing $[\![\varphi]\!]_{\mathcal{M}}$. That number is bounded by the modal rank $\mathrm{mr}(\varphi)$ of the formula φ, and therefore the structure complexity is bounded above by an exponential tower of a height not exceeding that modal rank:

$$2^{\cdots^{(\mathrm{mr}(\varphi)\ \text{ times})^{\cdots 2^{|\mathcal{T}||\mathcal{A}|}}}}$$

However, using the alternation rank of φ and Proposition 1 we can do better.

Proposition 2. *The structure complexity of global model checking for a fixed formula $\varphi \in \mathbf{K}_t$ on an input rational Kripke model $\mathcal{M}$, presented by the transducer and automata $\{\mathcal{T}, \mathcal{A}_1, \ldots, \mathcal{A}_n\}$, is bounded above by*

$$2^{\cdots^{(\mathrm{ar}(\varphi)\ \text{ times})^{\cdots 2^{P(|\mathcal{T}|)}}}}$$

where $P(|\mathcal{T}|)$ is a polynomial in $|\mathcal{T}|$ with leading coefficient not greater that $n2^c$ where $c \leq \max\{|\mathcal{A}_i| \mid i = 1, \ldots n\}$ and degree no greater than $\mathrm{mr}(\varphi)$.

Proof. The number of steps in the computation of $[\![\varphi]\!]_{\mathcal{M}}$, following the structure of φ, that produce nested exponential blow-ups can be bounded by the alternation rank, since nesting of any number of diamonds does not cause an exponential blow-up, while nesting of any number of boxes can be reduced by double complementation to nesting of diamonds; e.g., $[R]([R][R]p \vee [R^{-1}]\neg q)$ can be equivalently re-written as $\neg\langle R\rangle(\langle R\rangle\langle R\rangle\neg p \wedge \langle R^{-1}\rangle q)$. The initial synchronized product construction (when a diamond or box is applied to a boolean formula) produces an automaton of size at most $2^c|\mathcal{T}|$, the number of nested product constructions is bounded above by $\mathrm{mr}(\varphi)$, and each of these multiplies the size of the current automaton by $|\mathcal{T}|$. In the worst case, all alternations would take place after all product constructions, hence the upper bound. ■

7 Model Checking Extensions of $\mathbf{K}_t$ on Rational Models

7.1 Model Checking Hybrid Extensions of $\mathbf{K}_t$

A major limitation of the basic modal language is its inability to refer explicitly to states in a Kripke model, although the modal semantics evaluates modal formulae at states. Hybrid logics provide a remedy for that problem. We will

only introduce some basic hybrid logics of interest here; for more details consult, e.g., [3,4].

The *basic hybrid tense logic* $\mathbf{H}_t$ extends the basic tense logic $\mathbf{K}_t$ with a set of new atomic symbols Θ called *nominals* which syntactically form a second type of atomic formulae, which are evaluated in Kripke models in *singleton sets* of states. The unique state in the valuation of a nominal is called its *denotation*. Thus, nominals can be used in $\mathbf{H}_t$ to refer directly to states.

Here is the formal definition of the set of formulae of $\mathbf{H}_t$:

$$\varphi = p \mid i \mid \neg\varphi \mid \varphi \vee \phi \mid \langle R\rangle\, \varphi \mid \left\langle R^{-1}\right\rangle \varphi,$$

where $i \in \Theta$ and $p \in \Phi$.

The basic hybrid logic $\mathbf{H}_t$ can be further extended to $\mathbf{H}_t\,(@)$ by adding the *satisfaction operator* $@$, where the formula $@_i\varphi$ means 'φ *is true at the denotation of* i'. A more expressive extension of $\mathbf{H}_t$ is $\mathbf{H}_t(U)$ involving the *universal modality* with semantics $\mathcal{M}, v \models [U]\varphi$ iff $\mathcal{M}, w \models \varphi$ for every $w \in \mathcal{M}$. The operator $@$ is definable in $\mathbf{H}_t(U)$ by $@_i\varphi := [U](i \rightarrow \varphi)$. Moreover, $\mathbf{H}_t$ can be extended with the more expressive *difference modality* $\langle D\rangle$ (and its dual $[D]$), where $\mathcal{M}, v \models \langle D\rangle\varphi$ iff there exists a $w \neq v$ such that $\mathcal{M}, w \models \varphi$. Note that $[U]$ is definable in $\mathbf{H}_t(D)$ by $[U]\varphi := \varphi \wedge [D]\varphi$.

Yet another extension of $\mathbf{H}_t\,(@)$ is $\mathbf{H}_t\,(@, \downarrow)$ which also involves *state variables* and *binders* that bind these variables to states. Thus, in addition to $\mathbf{H}_t\,(@)$, formulae also include $\downarrow x.\varphi$ for x a state variable. For a formula φ possibly containing free occurrences of a state variable x, and w a state in a given model, let $\varphi\,[x \leftarrow i_w]$ denote the result of substitution of all free occurrences of x by a nominal i_w in φ, where w is the denotation of i_w. Then the semantics of $\downarrow x.\varphi$ is defined by: $\mathcal{M}, w \models \downarrow x.\varphi$ iff $\mathcal{M}, w \models \varphi\,[x \leftarrow i_w]$.

Proposition 3. *For every formula* φ *of the hybrid language* $\mathbf{H}_t(D)$ *(and therefore, of* $\mathbf{H}_t\,(@)$ *and of* $\mathbf{H}_t\,(U)$*) and every rational Kripke model* $\mathcal{M}$*, the set* $[\![\varphi]\!]_{\mathcal{M}}$ *is an effectively computable rational language.*

Proof. The claim follows from Theorem 1 since the valuations of nominals, being singletons, are rational sets, and the difference relation D is a rational relation. The latter can be shown by explicitly constructing a transducer recognizing D in a given rational set, or by noting that it is the complement of the automatic relation of equality, hence it is automatic itself, as the family of automatic relations is closed under complements (cf., e.g., [14] or [6]). ■

Corollary 2. *Global and local model checking, as well as satisfiability checking, of formulae of the hybrid language* $\mathbf{H}_t(D)$ *(and therefore, of* $\mathbf{H}_t\,(@)$ *and* $\mathbf{H}_t\,(U)$*, too) in rational Kripke models are decidable.*

Proposition 4. *Model checking of the* $\mathbf{H}_t\,(@, \downarrow)$*-formula* $\downarrow x.\,\langle R\rangle\, x$ *in* $\mathbf{H}_t\,(@, \downarrow)$ *on a given input rational Kripke model is not decidable.*

Proof. Immediate consequence from Morvan's earlier mentioned reduction [20] of the model checking of $\exists x Rxx$ to the Post Correspondence Problem. ■

Proposition 5. *There is a rational Kripke model on which model checking formulae from the hybrid language is undecidable.*

Proof. (Sketch) The rational graph constructed by Thomas [22] can be used to prove this undecidability, since the first-order properties queried there are also expressible in $\mathbf{H}_t(@, \downarrow)$. ■

7.2 Counting Modalities

We now consider extensions of $\mathbf{K}_t$ with counting (or, graded) modalities:

- $\Diamond^{\geq k}\varphi$ with semantics: 'there exist at least k successors where φ holds';
- $\Diamond^{\leq k}\varphi$ with semantics: 'there exist at most k successors where φ holds';
- $\Diamond^{k}\varphi$ with semantics: 'there exist exactly k successors where φ holds';
- $\Diamond^{\infty}\varphi$ with semantics: 'there exist infinitely many successors where φ holds'.

Clearly, some of these are inter-definable: $\Diamond^{k}\varphi := \Diamond^{\geq k}\varphi \wedge \Diamond^{\leq k}\varphi$, while $\Diamond^{\geq k}\varphi := \neg\Diamond^{\leq k-1}\varphi$ and $\Diamond^{\leq k}\varphi := \neg\Diamond^{\geq k+1}\varphi$.

We denote by $\mathbf{C}_t$ the extension of $\mathbf{K}_t$ with $\Diamond^{\infty}\varphi$ and all counting modalities for all integers $k \geq 0$. Further, we denote by $\mathbf{C}_t^0$ the fragment of $\mathbf{C}_t$ where no occurrence of a counting modality is in the scope of any modal operator.

Proposition 6. *Local model checking of formulae in the language $\mathbf{C}_t^0$ in rational Kripke models is decidable.*

Proof. First we note that each of the following problems: '*Given an automaton A, does its language contain at most / at least / exactly k / finitely / infinitely many words?*' is decidable. Indeed, the case of finite (respectively infinite) language is well-known (cf., e.g., [18], pp. 186–189). A decision procedure[3] for recognizing if the language of a given automaton $\mathcal{A}$ contains at least k words can be constructed recursively on k. When $k = 1$ that boils down to checking non-emptiness of the language (*ibid*). Suppose we have such a procedure P_k for a given k. Then, a procedure for $k + 1$ can be designed as follows: first, test the language $L(\mathcal{A})$ of the given automaton for non-emptiness by looking for any word recognized by it (by searching for a path from the initial state to any accepting state). If such a word w is found, modify the current automaton to exclude (only) w from its language, i.e. construct an automaton for the language $L(\mathcal{A}) \setminus \{w\}$, using the standard automata constructions. Then, apply the procedure P_k to the resulting automaton.

Testing $L(\mathcal{A})$ for having at most k words is reduced to testing for at least $k + 1$ words; likewise, testing for exactly k words is a combination of these.

Now, the claim follows from Theorem 1. Indeed, given a RKM $\mathcal{M}$ and a formula $\varphi \in \mathbf{C}_t^0$, for every subformula $\Diamond^c\psi$ of φ, where $\Diamond^c$ is any of the counting

[3] The procedure designed here is perhaps not the most efficient one. but, it will not make the complexity of the model checking worse, given the high overall complexity of the latter.

modalities listed above, the subformula ψ is in $\mathbf{K}_t$, and therefore an automaton for the regular language $[\![\psi]\!]_{\mathcal{M}}$ is effectively computable, and hence the question whether $\Diamond^c\psi$ is true at the state where the local model checking is performed can be answered effectively. It remains to note that every formula of $\mathbf{C}_t^0$ is a boolean combination of subformulae $\Diamond^c\psi$ where $\psi \in \mathbf{K}_t$. ■

At present, we do not know whether any of the counting modalities preserves regularity in rational models, and respectively whether global model checking in rational models of either of these languages is decidable.

7.3 A Presentation Based Extension

Here we consider a 'presentation-based' extension of the multi-modal version of $\mathbf{K}_t$, where the new modalities are defined in terms of word operations, so they only have meaning in Kripke models where the states are labeled by words (such as the rational Kripke models) hereafter called *Kripke word-models*.

To begin with, for a given alphabet Σ, with every language $X \subseteq \Sigma^*$ we can uniformly associate the following binary relations in Σ^*:

$X? := \{(u,u)\,|u \in X\}$;
$\overrightarrow{X} := \{(uv,v)\,|u \in X, v \in \Sigma^*\}$.

Proposition 7. *For every regular language $X \subseteq \Sigma^*$ the relations $X?$ and $\overrightarrow{X}$ are rational.*

Proof. For each of these, there is a simple uniform construction that produces from the automaton recognizing X a transducer recognizing the respective relation. For instance, the transducer for $\overrightarrow{X}$ is constructed as composition of the transducers (defined just like the composition of finite automata) for the rational relations $\{(u,\varepsilon) \mid u \in X\}$ and $\{(v,v) \mid v \in \Sigma^*\}$. The former is constructed from the automaton $\mathcal{A}$ for X by converting every a-transition in $\mathcal{A}$, for $a \in \Sigma$, to (a,ε)-transition, and the latter is constructed from an automaton recognizing Σ^* by converting every a-transition, for $a \in \Sigma$, to (a,a)-transition. ■

This suggests a natural extension of (multi-modal) $\mathbf{K}_t$ with an infinite family of new modalities associated with relations as above defined over the extensions of formulae. The result is a richer, PDL-like language which extends the star-free fragment of PDL with test and converse by additional program constructions corresponding to the regularity preserving operations defined above. We call that language '*word-based star-free PDL (with test and converse)*', hereafter denoted **WPDL**.

Formally, **WPDL** has two syntactic categories, viz., *programs* PROG and *formulae* FOR, defined over given alphabet Σ, set of atomic propositions AP, and set of atomic programs (relations) REL, by mutual induction as follows:

Formulae FOR:

$$\varphi ::= p \mid \ell_a \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\alpha\rangle\varphi$$

for $p \in \mathsf{AP}$, $a \in \Sigma$, and $\alpha \in \mathsf{PROG}$, where for each $a \in \Sigma$ we have added a special new atomic proposition ℓ_a, used further to translate extended star-free regular expressions to **WPDL**-formulae.

Programs PROG:

$$\alpha ::= \pi \mid \alpha\prime \mid \alpha_1 \cup \alpha_2 \mid \alpha_1 \circ \alpha_2 \mid \varphi? \mid \overrightarrow{\varphi}$$

where $\pi \in \mathsf{REL}$ and $\varphi \in \mathsf{FOR}$.

We note that **WPDL** is not a purely logical language, as it does not have semantics on abstract models but only on word-models (including rational Kripke models), defined as follows. Let $\mathcal{M} = (S, \{R_\pi\}_{\pi \in \mathsf{REL}}, V)$ be a Kripke word-model over an alphabet Σ, with a set of states $S \subseteq \Sigma^*$, a family of basic relations indexed with REL, and a valuation V of the atomic propositions from AP. Then every formula $\varphi \in \mathsf{FOR}$ is associated with the language $[\![\varphi]\!]_{\mathcal{M}} \subseteq \Sigma^*$, defined as before, where $[\![p]\!]_{\mathcal{M}} := V(p)$ for every $p \in \mathsf{AP}$ and $[\![\ell_a]\!] := \{a\} \cap S$ for every $a \in \Sigma$. Respectively, every program α is associated with a binary relation R_α in Σ^*, defined inductively as follows (where $\circ$ is composition of relations):

- $R_{\alpha\prime} := R_\alpha^{-1}$,
- $R_{\alpha_1 \cup \alpha_2} := R_{\alpha_1} \cup R_{\alpha_2}$,
- $R_{\alpha_1 \circ \alpha_2} := R_{\alpha_1} \circ R_{\alpha_2}$,
- $R_{\varphi?} := [\![\varphi]\!]?$,
- $R_{\overrightarrow{\varphi}} := \overrightarrow{[\![\varphi]\!]}$.

Lemma 5. *For every* **WPDL***-formulae φ, ψ and a Kripke word-model $\mathcal{M}$:*

1. $[\![\langle\varphi?\rangle\psi]\!]_{\mathcal{M}} = [\![\varphi]\!]_{\mathcal{M}} \cap [\![\psi]\!]_{\mathcal{M}}$.
2. $[\![\langle\overrightarrow{\varphi}\rangle\psi]\!]_{\mathcal{M}} = [\![\varphi]\!]_{\mathcal{M}}; [\![\psi]\!]_{\mathcal{M}}$ *(where ; denotes concatenation of languages).*

Proof. Routine verification:

1. $[\![\langle\varphi?\rangle\psi]\!]_{\mathcal{M}} = \{w \in \Sigma^* \mid wR_{\varphi?}v \text{ for some } v \in [\![\psi]\!]_{\mathcal{M}}\}$
   $= \{w \in \Sigma^* \mid w = v \text{ for some } v \in [\![\varphi]\!]_{\mathcal{M}} \text{ and } v \in [\![\psi]\!]_{\mathcal{M}}\} = [\![\varphi]\!]_{\mathcal{M}} \cap [\![\psi]\!]_{\mathcal{M}}$.
2. $[\![\langle\overrightarrow{\varphi}\rangle\psi]\!]_{\mathcal{M}} = \{w \in \Sigma^* \mid wR_{\overrightarrow{\varphi}}v \text{ for some } v \in [\![\psi]\!]_{\mathcal{M}}\}$
   $= \{uv \in \Sigma^* \mid u \in [\![\varphi]\!]_{\mathcal{M}}, v \in [\![\psi]\!]_{\mathcal{M}}\} = [\![\varphi]\!]_{\mathcal{M}}; [\![\psi]\!]_{\mathcal{M}}$. ■

Corollary 3. *For every* **WPDL***-formula φ and a rational Kripke model $\mathcal{M}$, the language $[\![\varphi]\!]_{\mathcal{M}}$ is an effectively computable from φ regular language.*

Corollary 4. *Local and global model checking, as well as satisfiability checking, of* **WPDL***-formulae in rational Kripke models is decidable.*

Extended star-free regular expressions over an alphabet Σ are defined as follows:

$$E := a \mid \neg E \mid E_1 \cup E_2 \mid E_1; E_2,$$

where $a \in \Sigma$. Every such expression E defines a regular language $L(E)$, where $\neg, \cup, ;$ denote respectively complementation, union, and concatenation of languages. The question whether two extended star-free regular expressions define the same language has been proved to have a non-elementary complexity in [19].

Every extended star-free regular expression can be linearly translated to an **WPDL**-formula:

- $\tau(a) := \ell_a$,
- $\tau(\neg E) := \neg\tau(E)$,
- $\tau(E_1 \cup E_2) := \tau(E_1) \vee \tau(E_2)$,
- $\tau(E_1; E_2) := \langle\overrightarrow{\tau(E_1)}\rangle\tau(E_2)$.

Lemma 6. *Given an alphabet Σ, consider the rational Kripke model $\mathcal{M}^\Sigma$ with set of states Σ^*, over empty sets of basic relations and atomic propositions. Then, for every extended star-free regular expression E,*

$$L(E) = [\![\tau(E)]\!]_{\mathcal{M}^\Sigma}.$$

Proof. Straightforward induction on E. The only non-obvious case $E = E_1; E_2$ follows from Lemma 5. ■

Consequently, for any extended star-free regular expressions E_1 and E_2, we have that $L(E_1) = L(E_2)$ iff $[\![\tau(E_1)]\!]_{\mathcal{M}^\Sigma} = [\![\tau(E_2)]\!]_{\mathcal{M}^\Sigma}$ iff $\mathcal{M}^\Sigma \models \tau(E_1) \leftrightarrow \tau(E_2)$. Thus, we obtain the following.

Corollary 5. *Global model checking of* **WPDL***-formulae in rational Kripke models has non-elementary formula-complexity.*

Remark: Since the $\overrightarrow{\varphi}$-free fragment of **WPDL** is expressively equivalent to $\mathbf{K}_t$, a translation of bounded exponential blow-up from the family of extended star-free regular expressions to the latter fragment would prove Conjecture 1.

8 Concluding Remarks

We have introduced the class of rational Kripke models and shown that all formulae of the basic tense logic $\mathbf{K}_t$, and various extensions of it, have effectively computable rational extensions in such models, and therefore global model checking and local model checking of such formulae on rational Kripke models are decidable, albeit probably with non-elementary formula complexity.

Since model checking reachability on such models is generally undecidable, an important direction for further research would be to identify natural large subclasses of rational Kripke models on which model checking of K_t extended with the reachability modality $\langle R\rangle^*$ is decidable. Some such cases, defined in terms of the presentation, are known, e.g., rational models with length-preserving or length-monotone transition relation [20]; the problem of finding structurally defined large classes of rational models with decidable reachability is still essentially open.

Other important questions concern deciding bisimulation equivalence between rational Kripke models, as that would allow us to transfer model checking of any property definable in the modal mu-calculus from one to the other. These questions are studied in a follow-up to the present work.

References

1. Berstel, J.: Transductions and Context-Free Languages. Teubner Studienbücher Informatik. B.G. Teubner, Stuttgart (1979)
2. Biere, A., Cimatti, A., Clarke, E., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers 58, 118–149 (2003)
3. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. CUP (2001)
4. Blackburn, P.: Representation, reasoning, and relational structures: a hybrid logic manifesto. Logic Journal of the IGPL 8(3), 339–365 (2000)
5. Blumensath, A.: Automatic structures. Diploma thesis, RWTH Aachen (1999)
6. Blumensath, A., Grädel, E.: Automatic structures. In: Abadi, M. (ed.) Proc. LICS 2000, pp. 51–62 (2000)
7. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 403–418. Springer, Heidelberg (2000)
8. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
9. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, New York (1974)
10. Elgot, C., Mezei, J.: On relations defined by finite automata. IBM J. of Research and Development 9, 47–68 (1965)
11. Esparza, J., Kučera, A., Schwoon, S.: Model-Checking LTL with Regular Valuations for Pushdown Systems. In: Kobayashi, N., Pierce, B.C. (eds.) TACS 2001. LNCS, vol. 2215, pp. 316–339. Springer, Heidelberg (2001)
12. Frougny, C., Sakarovitch, J.: Synchronized rational relations of finite and infinite words. Theor. Comput. Sci. 108(1), 45–82 (1993)
13. Johnson, J.H.: Rational equivalence relations. Theor. Comput. Sci. 47(3), 39–60 (1986)
14. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1994)
15. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich assertional languages. Theor. Comput. Sci. 256(1-2), 93–112 (2001)
16. Kuper, G.M., Vardi, M.Y.: On the complexity of queries in the logical data model. In: Gyssens, M., Van Gucht, D., Paredaens, J. (eds.) ICDT 1988. LNCS, vol. 326, pp. 267–280. Springer, Heidelberg (1988)
17. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. Journal of the ACM 47(2), 312–360 (2000)
18. Martin, J.C.: Introduction to Languages and the Theory of Computation, 3rd edn., pp. 186–189. McGraw-Hill, Inc., New York (2002)
19. Meyer, A.R., Stockmeyer, L.J.: Word problems requiring exponential time: Preliminary report. In: Aho, A.V., et al. (eds.) Proc. STOC 1973, pp. 1–9 (1973)
20. Morvan, C.: On Rational Graphs. In: Tiuryn, J. (ed.) FOSSACS 2000. LNCS, vol. 1784, pp. 252–266. Springer, Heidelberg (2000)
21. Reisig, W.: Petri nets: An Introduction. Springer, New York (1985)

22. Thomas, W.: Constructing infinite graphs with a decidable mso-theory. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 113–124. Springer, Heidelberg (2003)
23. Vardi, M.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency: Structure versus Automata (8th Banff Higher Order Workshop). LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996)
24. Walukiewicz, I.: Model checking CTL properties of pushdown systems. In: Kapoor, S., Prasad, S. (eds.) FST TCS 2000. LNCS, vol. 1974, pp. 127–138. Springer, Heidelberg (2000)

Genetic Systems without Inhibition Rules

Nadia Busi[1] and Claudio Zandron[2,⋆]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura A. Zamboni 7, I-40127 Bologna, Italy
[2] Dipartimento di Informatica, Sistemistica e Comunicazione, Università di Milano-Bicocca, viale Sarca 336, I-20126, Milano, Italy
zandron@disco.unimib.it

In memory of Nadia Busi

Abstract. Genetic Systems are a formalism inspired by genetic regulatory networks, suitable for modeling the interactions between genes and proteins, acting as regulatory products. The evolution is driven by genetic gates: a new object (representing a protein) is produced when all activator objects are available in the system, and no inhibitor object is present. Activators are not consumed by the application of such a rule. Objects disappear because of degradation: each object is equipped with a lifetime, and the object decays when such a lifetime expires.

It is known that such systems are Turing powerful, either when we consider interleaving semantics (a single action is executed in each computational step) as well as if we consider maximal parallel semantics (all the rules that can be applied at a computational step must be applied). In this paper we investigate the power of inhibiting rules.

1 Introduction

Most biological processes are regulated by networks of interactions between regulatory products and genes. To investigate the dynamical properties of these genetic regulatory networks, various formal approaches, ranging from discrete to stochastic and to continuous models, have been proposed (see [9] for a review).

Genetic Systems have been introduced in [6] as a simple discrete formalism for the modeling of genetic networks, and we started an investigation of the ability of such a formalism to act as a computational device. Genetic Systems are based on *genetic gates*, that are rules which model the behaviour of genes, and *objects*, that represent proteins. Proteins both regulate the activity of a gene – by activating or inhibiting transcription – and represent the product of the activity of a gene.

A genetic gate is essentially a *contextual rewriting rule* consisting of three components: the set of activators, the set of inhibitors and the transcription product. A genetic gate is activated if the activator objects are present in the

⋆ (*Received by the editors.* 1 February 2008. *Revised.* 17 September 2008. *Accepted.* 17 September 2008.)

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, pp. 21–31, 2009.
© Springer-Verlag Berlin Heidelberg 2009

system, and all inhibitor objects are absent. The result of the application of a genetic gate rule is the production of a new object (without removing the activator objects from the system).

In biological systems, proteins can disappear in (at least) two ways (see, e.g., [1]): a protein can either decay because its lifetime is elapsed, or because it is neutralized by a repressor protein. To model the decaying process, we equip objects with a lifetime, which is decremented at each computational step. When the lifetime of an object becomes zero, the object disappears. In our model we represent both decaying and persistent objects: while the lifetime of a decaying object is a natural number, persistent objects are equipped with an infinite lifetime. The behaviour of repressor proteins is modeled through *repressor rules*, consisting of two components: the repressor object and the object to be destroyed. When both objects are present in the system, the rule is applied: the object to be destroyed disappears, while the repressor is not removed.

In [6] we investigated the computational power of Genetic Systems with the *maximal parallelism semantics*: all the rules that *can be* applied simultaneously, *must be* applied in the same computational step. In particular, we proposed an encoding of Random Access Machines (RAMs) [16], a well-known, deterministic Turing equivalent formalism.

As the maximal parallelism semantics is a very powerful synchronization mechanism, we also considered the semantics at the opposite side of the spectrum, namely, the *interleaving* (or *sequential*) *semantics*, where a single rule is applied in each computational step. The biological intuition behind this choice is that the cell contains a finite number of RNA polymerases, that are the enzymes that catalyze the transcription of genes: the case of interleaving semantics corresponds to the presence of a single RNA polymerase enzyme.

As shown in [5], the power of Genetic systems is not decreased in this way, as RAMs can still be simulated. Moreover, the encoding turns out to be deterministic, and a RAM terminates if and only if its encoding terminates (i.e. no additional divergent or failed computations are added in the encoding). Hence, both existential termination (i.e., the existence of a terminated, or deadlocked, computation) and universal termination (i.e., all computations terminate) are undecidable in this case.

Such universality results are obtained by using several ingredients: persistent and decaying objects, repressor rules, positive and negative regulation. For this reason, in [7] we investigated what happens when some of these ingredients are removed; we showed that, when we consider interleaving semantics, decaying objects are needed to obtain deterministic encodings of RAMs. Moreover, we showed that universal termination is decidable for Genetic Systems without negative regulation.

We continue here along this line, by showing that negative regulation is not needed when we consider non–deterministic Genetic Systems using maximal parallel semantics: such systems remain universal even without such kind of rules.

The paper is organized as follows. In Section 2 we give basic definitions that will be used throughout the paper. The syntax and the semantics of Genetic

Systems is presented in Section 3. In Section 4 we recall some results concerning weaker Genetic Systems using interleaving semantics, and then we consider Genetic systems with maximal parallel semantics without inhibiting rules: we show that universality can still be obtained even in this case. Section 5 reports some conclusive remarks.

2 Basic Definitions

In this section we provide some basic definitions that will be used throughout the paper. With $\mathbb{N}$ we denote the set of natural numbers, whereas $\mathbb{N}_\infty$ denotes $\mathbb{N} \cup \{\infty\}$. We start with the definition of multisets and multiset operations.

Definition 1. *Given a set S, a* finite multiset *over S is a function $m : S \to \mathbb{N}$ such that the set* $\mathrm{dom}(m) = \{s \in S \mid m(s) \neq 0\}$ *is finite. The* multiplicity *of an element s in m is given by the natural number $m(s)$. The set of all finite multisets over S, denoted by $\mathcal{M}_{\mathrm{fin}}(S)$, is ranged over by m. A multiset m such that* $\mathrm{dom}(m) = \varnothing$ *is called* empty*. The empty multiset is denoted by $\varnothing$. Given the multiset m and m', we write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$ while $\oplus$ denotes their* multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\setminus$ denotes* multiset difference*: $(m \setminus m')(s) =$ if $m(s) \geq m'(s)$ then $m(s) - m'(s)$ else 0.*

The set of parts of a set S is defined as $\mathcal{P}(S) = \{X \mid X \subseteq S\}$. Given a set $X \subseteq S$, with abuse of notation we use X to denote also the multiset

$$m_X(s) = \begin{cases} 1 & \text{if } s \in X \\ 0 & \text{otherwise} \end{cases}$$

We provide some basic definitions on strings, cartesian products and relations.

Definition 2. *A string over S is a finite (possibly empty) sequence of elements in S. Given a string $u = x_1 \dots x_n$, the length of u is the number of occurrences of elements contained in u and is defined as follows: $|u| = n$. The empty string is denoted by λ. With S^* we denote the set of strings over S, and $u, v, w, \dots$ range over S^*. Given $n \geq 0$, with S^n we denote the set of strings of length n over S. Given a string $u = x_1 \dots x_n$, the multiset corresponding to u is defined as follows: for all $s \in S$, $m_u(s) = |\{i \mid x_i = s \wedge 1 \leq i \leq n\}|$. With abuse of notation, we use u to denote also m_u*[1]*.*

Definition 3. *With $S \times T$ we denote the cartesian product of sets S and T, with $\prod_n S$, $n \geq 1$, we denote the cartesian product of n copies of set S and with $\prod_{i=1}^n S_i$ we denote the cartesian product of sets $S_1, \dots, S_n$, i.e., $S_1 \times \dots \times S_n$. The ith projection of $x = (x_1, \dots, x_n) \in \prod_{i=1}^n S_i$ is defined as $\pi_i(x) = x_i$, and lifted to subsets $X \subseteq \prod_{i=1}^n S_i$ as follows: $\pi_i(X) = \{\pi_i(x) \mid x \in X\}$.*

Given a binary relation R over a set S, with R^n we denote the composition of n instances of R, with R^+ we denote the transitive closure of R, and with R^* we denote the reflexive and transitive closure of R.

[1] In some cases we denote a multiset by one of its corresponding strings, because this permits to define functions on multisets in a more insightful way.

3 Genetic Systems

In this section, we present the definition of Genetic Systems and the definitions which we need to describe their functioning. To this aim, given a set X, we define $\mathcal{R}_X = \mathcal{P}(X) \times \mathcal{P}(X) \times (X \times I\!N_\infty)$.

Definition 4. *A* Genetic System *is a tuple* $G = (V, GR, RR, w_0)$ *where*

1. V *is a finite alphabet whose elements are called* objects*;*
2. GR *is a finite multiset*[2] *over* $\mathcal{R}_V$ *of* genetic gates *over* V*; these gates are of the forms* $u_{\text{act}}, \neg u_{\text{inh}} :\rightarrow (b,t)$ *where* $u_{\text{act}} \cap u_{\text{inh}} = \varnothing$*.* $u_{\text{act}} \subseteq V$ *is the* positive regulation (activation)[3]*,* $u_{\text{inh}} \subseteq V$ *the* negative regulation (inhibition)*,* $b \in V$ *the transcription of the gate*[4] *and* $t \in I\!N_\infty$ *the* duration *of object* b*;*
3. RR *is a finite set*[5] *of* repressor rules *of the form* $(\text{rep} : b \rightarrow)$ *where* $\text{rep}, b \in V$ *and* $\text{rep} \neq b$*;*
4. w_0 *is a string over* $V \times I\!N_\infty$*, representing the multiset of objects contained in the system at the beginning of the computation. The objects are of the form* (a,t)*, where* a *is a symbol of the alphabet* V *and* $t > 0$ *represents the decay time of that object.*

We say that a gate is *unary* if $|u_{\text{act}} \oplus u_{\text{inh}}| = 1$. The multiset represented by w_0 constitutes the *initial state* of the system.

A transition between states is governed by an application of the transcription rules (specified by the genetic gates) and of the repressor rules. The gate $u_{\text{act}}, \neg u_{\text{inh}} :\rightarrow (b,t)$ can be activated if the current state of the system contains enough free activators and no free inhibitors. If the gate is activated, the regulation objects (activators) in the set u_{act} are bound to such a gate, and they cannot be used for activating any other gate in the same maximal parallelism evolution step. In other words, the gate $u_{\text{act}}, \neg u_{\text{inh}} :\rightarrow (b,t)$ in a state formed by a multiset of (not yet bound) objects w can be activated if u_{act} is contained in w and no object in u_{inh} appears in w; if the gate performs the transcription, then a new object (b,t) is produced. Note that the objects in u_{act} and u_{inh} are not consumed by the transcription operation, but will be released at the end of the operation and (if they do not disappear because of the decay process) they can be used in the next evolution step. Each object starts with a decay number, which specify the number of steps after which this object disappears. The decay number is decreased after each parallel step; when it reaches the value zero, the object disappears. If the decay number of an object is equal to ∞, then the object is persistent and it never disappears. The repressor rule $(\text{rep} : b \rightarrow)$

[2] Here we use multisets of rules, instead of sets, for compatibility with the definition in [6].

[3] We consider sets of activators, meaning that a genetic gate is never activated by more than one instance of the same protein.

[4] Usually the expression of a genetic gate consists of a single protein.

[5] We use sets of rules, instead of multisets, because each repressor rule denotes a chemical law; hence, a repressor rule can be applied for an unbounded number of times in each computational step.

is activated when both the repressor rep and the object b are present, and the repressor rep destroys the object b.

We adopt the following notation for gates. The activation and inhibition sets are denoted by one of the corresponding strings, i,e, $a, b, \neg c :\rightarrow (c, 5)$ denotes the gate $\{a, b\}, \neg\{c\} :\rightarrow (c, 5)$. If either the activation or the inhibition is empty then we omit the corresponding set, i.e., $a :\rightarrow (b, 3)$ is a shorthand for the gate $\{a\}, \neg\varnothing :\rightarrow (b, 3)$. The *nullary gate* $\varnothing, \neg\varnothing :\rightarrow (b, 2)$ is written as $:\rightarrow (b, 2)$. Once defined Genetic Systems, we are ready to describe their functioning. A transition between two states of the system is governed by an application of the transcription rules (specified by the genetic gates) and of the repressor rules. Different semantics can be adopted, depending on the number of rules that are applied in each computational step, and on the way in which the set of rules to be applied is chosen. In [6] we considered the so-called *maximal parallelism semantics*: all the rules that *can be* applied simultaneously, *must be* applied in the same computational step. In [7] we considered the semantics at the opposite side of the spectrum, i.e., the *interleaving* (or *sequential*) *semantics*. In this case, at each computational step only a single rule is applied. In particular, for deterministic systems, at each computational step there is at most one rule that can be applied. For non–deterministic systems, at each computational step one rule, among all applicable ones, is chosen to be applied.

We give now the definitions for configuration, reaction relation, and heating and decaying function; then, we will give the definition of computational step, considering the two possible types of step, according to the considered semantics for the application of the rules.

A configuration represents the current state of the system, consisting in the (multi)set of objects currently present in the system.

Definition 5. *Let $G = (V, GR, RR, w_0)$ be a Genetic System. A* configuration *of G is a multiset $w \in \mathcal{M}_{\text{fin}}(V \times I\!N_\infty)$. The* initial configuration *of G is the multiset w_0.*

The activation of a genetic gate is formalized by the notion of reaction relation. In order to give a formal definition we need the function $\text{obj} : (V \times I\!N_\infty)^* \rightarrow V^*$, defined as follows. Assume that $(a, t) \in (V \times (I\!N_\infty))$ and $w \subseteq (V \times (I\!N_\infty))^*$. Then, $\text{obj}(\lambda) = \lambda$ and $\text{obj}((a, t)w) = a\ \text{obj}(w)$.

We also need to define a function DecrTime which is used to decrement the decay time of objects, destroying the objects which reached their time limit.

Definition 6. *The function* $\text{DecrTime} : (V \times I\!N_\infty)^* \rightarrow (V \times I\!N_\infty)^*$ *is defined as follows:* $\text{DecrTime}(\lambda) = \lambda$ *and*

$$\text{DecrTime}((a, t)w) = \begin{cases} (a, t-1)\text{DecrTime}(w) & \text{if } t > 1 \\ \text{DecrTime}(w) & \text{if } t = 1 \end{cases}$$

We are now ready to give the notion of *reaction relation*.

Definition 7. *Let $G = (V, GR, RR, w_0)$ be a Genetic System. The* reaction relation $\mapsto$ *over $\mathcal{M}_{\text{fin}}(V \times I\!N_\infty) \times \mathcal{M}_{\text{fin}}(V \times I\!N_\infty)$ is defined as follows:*

$w \mapsto w'$ iff one of the following holds:

- *there exist* $u_{\text{act}}, \neg u_{\text{inh}} :\rightarrow (b,t) \in GR$ *and* $w_{\text{act}} \subseteq w$ *such that*
  - $u_{\text{inh}} \cap \text{dom}(\text{obj}(w)) = \varnothing$
  - $m_{\text{obj}(w_{\text{act}})} = m_{u_{\text{act}}}$[6]
  - $w' = \text{DecrTime}(w) \oplus \{(b,t)\}$
- *there exists* $(\text{rep} : b \rightarrow) \in RR$ *such that*
  - *there exist* $t_{\text{rep}}, t_b \in I\!N_\infty$ *such that* $\{(\text{rep}, t_{\text{rep}}), (b, t_b)\} \subseteq w$
  - $w' = \text{DecrTime}(w) \setminus \{(\text{rep}, t_{\text{rep}}), (b, t_b)\} \oplus \text{DecrTime}((\text{rep}, t_{\text{rep}}))$

Now we are ready to define a computational step of a Genetic system. We first consider an *interleaving computational step* $\rightsquigarrow$:

Definition 8. *Let* $G = (V, GR, RR, w_0)$ *be a Genetic System. The* interleaving computational step $\rightsquigarrow$ *over configurations of* G *is defined as follows:* $w_1 \rightsquigarrow w_2$ *iff one of the following holds:*

- *either* $w_1 \mapsto w_2$*, or*
- $w_1 \not\mapsto$ *and there exists* $(a,t) \in w_1$ *such that* $t \neq \infty$ *and* $w_2 = \text{DecrTime}(w_1)$[7]*.*

We say that a configuration w *is* terminated *if no interleaving step can be performed, i.e.,* $w \not\rightsquigarrow$*. The set of configurations* reachable *from a given configuration* w *is defined as* $Reach(w) = \{w' \mid w \rightsquigarrow^* w'\}$*. The set of* reachable configurations in G *is* $Reach(w_0)$*.*

A *maximal parallelism computational step* $\Rightarrow$ is defined as it follows:

Definition 9. *Let* $G = (V, GR, RR, w_0)$ *be a Genetic System.*

The maximal parallelism computational step $\Rightarrow$ *over (nonpartial) configurations of* G *is defined as follows:* $\gamma_1 \Rightarrow \gamma_2$ *iff one of the following holds:*

- *there exists a partial configuration* γ' *s.t.* $\gamma_1 \mapsto^+ \gamma'$*,* $\gamma' \not\mapsto$ *and* $\gamma_2 = \text{heat\&decay}(\gamma')$*, or*
- $\gamma_1 = (w, R, \varnothing, \varnothing)$*, exists* $(a,t) \in w$ *s.t.* $t \neq \infty$ *and* $\gamma_2 = (\text{DecrTime}(w), R, \varnothing, \varnothing)$*.*

We say that a configuration γ *is* terminated *if no maximal parallelism step can be performed, i.e.,* $\gamma \not\Rightarrow$*.*

Note that a computational step can consist either in the application of a rule, or in the passing of one time unit, in case the system is deadlocked (i.e., no rule can be applied). Consider, e.g., a system with a negative gate $\neg b :\rightarrow (a,3)$ and a positive gate $a :\rightarrow (a,3)$, reaching a configuration containing only the object $(b,2)$. The system cannot evolve until 2 time units have elapsed; then, an object $(a,3)$ is produced and, because of the positive gate, the system will never terminate.

To illustrate the difference between the interleaving and the maximal parallelism semantics, consider a system with gates $\neg d, a :\rightarrow b$ and $\neg b, c :\rightarrow d$ and

[6] We recall that $m_{u_{\text{act}}}$ is the multiset containing exactly one occurrence of each object in the set u_{act}. Hence, the operator $=$ is intended here to be the equality operator on multisets.

[7] With $w_1 \not\mapsto$ we denote the fact that there exist no w' such that $w_1 \mapsto w'$.

with initial state ac. According to the maximal parallelism semantics only the following step can be performed: $ac \Rightarrow acbd \not\Rightarrow$, where $\Rightarrow$ is a maximal parallelism computational step. On the other hand, according to the interleaving semantics also the following sequence of steps can be performed: $ac \rightsquigarrow acb \rightsquigarrow acbb \rightsquigarrow acbbb \rightsquigarrow \ldots$.

4 Computational Power of Genetic Systems

As previously said, in [6] we showed that Genetic Systems with maximal parallelism semantics are Turing powerful, while in [7] we showed that interleaving semantics is enough to get Turing equivalence.

The result presented in the cited papers are proved by using several ingredients: persistent and decaying objects, repressor rules, positive and negative regulation. Are all such ingredients needed in order to obtain such a result? Some partial answers to this question were given in [7], where it is shown that decaying objects are needed to obtain universality. Moreover, it is also shown that universal termination is decidable for Genetic Systems without inhibitors which make use of interleaving semantics. The decidability proof is based on the theory of Well-Structured Transition Systems [10]: the existence of an infinite computation starting from a given state is decidable for finitely branching transition systems, provided that the set of states can be equipped with a well-quasi-ordering, i.e., a quasi-ordering relation which is compatible with the transition relation and such that each infinite sequence of states admits an increasing subsequence.

In particular, it is known that:

Theorem 1. *Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with decidable $\leq$ and computable* Succ. *The existence of an infinite computation starting from a state $s \in S$ is decidable.*

Moreover, it is possible to prove the following theorem (see [7] and [10] for details):

Theorem 2. *Let $G = (V, GR, RR, w_0)$ be a Genetic System. Then $TS(G)$ is a finitely branching, well-structured transition system with decidable $\leq$ and computable* Succ.

As a consequence, universal termination is decidable for Genetic Systems without inhibitors using interleaving semantics, which shows that such systems are not Turing equivalent.

We show now that when in the case of non–deterministic Genetic Systems using maximal parallel semantics universality can be obtained even if we do not use inhibiting rules.

The basic idea follows the one already developed in [6]. The contents of registers is encoded using sets of copies of persistent objects. If register r_i contains value c_i, then c_i copies of object (r_i, ∞) are present in the state of the system.

The fact that the program counter contains the value i (i.e., the next instruction to be executed is the ith) is represented by the presence of object p_i.

We start with the decaying object $(p_1, 1)$. In general, just before starting the execution of the ith instruction, the system contains the object $(p_i, 1)$.

The encoding of a successor instruction $i : \mathrm{Succ}(r_j)$ produces a new instance of a persistent object (r_j, ∞), as well as the new program counter $(p_{i+1}, 1)$, while the encoding of a decrement or jump instruction $i : \mathrm{DecJump}(r_j, s)$ may have two different behaviours. If no object r_j occurs in the system, then the program counter $(p_s, 1)$ is produced; otherwise, if there exists an object r_j in the system, a repressor object $(repr_j, 1)$ for object r_j is produced; the repressor decays just after destroying a copy of (r_j, ∞), and the program counter $(p_{i+1}, 1)$ is produced. If the computation stops (no rule can be applied, and no object with decay time different from ∞ is present), then we consider the output, which is the number of occurrences of object (r_1, ∞).

For the sake of brevity, we consider RAMs that satisfy the following constraint: if the RAM has m instructions, then all the jumps to addresses higher than m are jumps to the address $m+1$. This constraint is not restrictive, as for any RAM not satisfying the constraint above it is possible to construct an equivalent RAM (i.e., a RAM computing the same function) which satisfies it. Given a RAM with m instructions, the first constraint can be easily satisfied by replacing each jump to an address higher than m by a jump to the address $m + 1$.

We stress the fact that the simulation we produce here is non–deterministic, and it introduces some divergent computations. In particular, we consider the use of a "trap" rule of the type loop $:\!\!\rightarrow (\mathrm{loop}, 1)$. Once introduced, the symbol loop activates such a gate in an infinite loop, thus a non–terminating computation, which produces no output.

In order to simulate an increment instruction $(i : \mathrm{Succ}(r_j))$, we consider the following set of rules. The rule $c1$ is used to induce an infinite loop when "wrong" rules are applied.

$$
\begin{array}{ll}
\mathrm{r1}: & p_i :\!\!\rightarrow (p_{i+1}{}', 2) \\
\mathrm{r2}: & p_{i+1}{}' :\!\!\rightarrow (p_{i+1}{}'', 2) \\
\mathrm{r3}: & p_{i+1}{}', p_{i+1}{}'' :\!\!\rightarrow (r_j{}', 2) \\
\mathrm{r4}: & p_{i+1}{}'', r_j{}' :\!\!\rightarrow (p_{i+1}{}''', 1) \\
\mathrm{r5}: & r_j{}' :\!\!\rightarrow (r_j, \infty) \\
\mathrm{r6}: & p_{i+1}{}''' :\!\!\rightarrow (p_{i+1}, 1) \\
\mathrm{c1}: & p_{i+1}{}'' :\!\!\rightarrow (\mathrm{loop}, 1)
\end{array}
$$

Initially, the system contains only $(p_i, 1)$ as nonpersistent object. The only applicable rule is $r1$, which produces $(p_{i+1}{}', 2)$. The object $(p_i, 1)$ disappears, because after applying the rule its decay time reaches the value zero.

Then, using $r2$ we obtain $(p_{i+1}{}'', 2)$, and the decay time of $p_{i+1}{}'$ is decreased, leading to $(p_{i+1}{}', 1)$.

If we use this last object again in rule $r2$, then the object $p_{i+1}{}''$ must be used at the same time in rule $c1$, producing an infinite computation. The only other possibility is to use both objects at the same time in the rule $r3$, thus producing $(r_j{}', 2)$; $p_{i+1}{}'$ disappears because its decay time reaches the value zero, while $(p_{i+1}{}'', 2)$ is replaced by $(p_{i+1}{}'', 1)$.

$r_j{}'$ can now be used in rule $r5$; in this case, $p_{i+1}{}''$ must be used in rule $c1$ producing an infinite loop. If, on the contrary, both $r_j{}'$ and $p_{i+1}{}''$ are used to activate rule $r4$, we produce $(p_{i+1}{}''', 1)$. Because of decay time, we obtain $(r_j{}', 1)$ and, moreover, $p_{i+1}{}''$ disappears.

At this point the only possibility is to apply, at the same time, rules $r5$ and $r6$, using objects $r_j{}'$ and $p_{i+1}{}'''$, respectively, which then disappear. In this way, we produce objects (r_j, ∞) and $(p_{i+1}, 1)$; thus, we correctly simulated an increment instruction and we are ready to simulate instruction $i + 1$.

To simulate a decrement instruction $(i : \mathrm{DecJump}(r_j, s))$ the following set of rules is used. In this case, the rules to check "wrong" computations are rules $c1$, $c2$, and $c3$.

r1 : $p_i :\to (\mathrm{Dec}_{i,j}{}', 2)$
r2 : $\mathrm{Dec}_{i,j}{}' :\to (\mathrm{Dec}_{i,j}{}'', 3)$
r3 : $\mathrm{Dec}_{i,j}{}', \mathrm{Dec}_{i,j}{}'' :\to (\mathrm{Dec}_{i,j}{}''', 2)$
r4 : $\mathrm{Dec}_{i,j}{}'', \mathrm{Dec}_{i,j}{}''' :\to (\mathrm{Dec}_{i,j}{}'''', 1)$
r5 : $\mathrm{Dec}_{i,j}{}'', \mathrm{Dec}_{i,j}{}'''' :\to (I_s{}', 1)$
r6 : $\mathrm{Dec}_{i,j}{}''', r_j :\to (C_{i+1}, 2)$
r7 : $I_s{}' :\to (p^s_{i+1}, 1)$
r8 : $I_s{}', C_{i+1} :\to (\mathrm{Repr}_j, 2)$
r9 : $\mathrm{Repr}_j, C_{i+1} :\to (I_{i+1}{}', 1)$
r10 : $\mathrm{Repr}_j : r_j \to$
r11 : $I_{i+1}{}' :\to (p_{i+1}, 1)$
c1 : $\mathrm{Dec}_{i,j}{}'' :\to (\mathrm{loop}, 1)$
c2 : $\mathrm{Dec}_{i,j}{}'''' :\to (\mathrm{loop}, 1)$
c3 : $C_{i+1} :\to (\mathrm{loop}, 1)$

Again, the system initially contains only $(p_i, 1)$ as nonpersistent object. The only applicable rule is $r1$, which produces $(\mathrm{Dec}_{i,j}{}', 2)$.

We can now apply only rule $r2$; we obtain objects $(\mathrm{Dec}_{i,j}{}', 1)$ and the new created one, $(\mathrm{Dec}_{i,j}{}'', 3)$.

If we apply again rule $r2$ using $\mathrm{Dec}_{i,j}{}'$, then we also have to use rule $c1$, which produces an infinite loop. Instead, if we apply rule $r3$ we obtain objects $(\mathrm{Dec}_{i,j}{}'', 2)$ and $(\mathrm{Dec}_{i,j}{}''', 2)$.

If an object (r_j, ∞) is present, we could apply now rule $r6$; in this case, $\mathrm{Dec}_{i,j}{}''$ must be used in rule $c1$, producing an infinite loop. On the contrary, we can apply rule $r4$, obtaining the objects $(\mathrm{Dec}_{i,j}{}'', 1)$, $(\mathrm{Dec}_{i,j}{}''', 1)$, and $(\mathrm{Dec}_{i,j}{}'''', 1)$.

We can now apply again $r4$; in this case, object $\mathrm{Dec}_{i,j}{}''''$ activates rule $c2$, producing an infinite loop. Another possibility (apart from the activation of $c1$ using $\mathrm{Dec}_{i,j}{}''$) is to apply rule $r5$, producing $(I_s{}', 1)$. In this case, the object $(\mathrm{Dec}_{i,j}{}''', 1)$ has two possibilities to evolve, in this same step: either an object (r_j, ∞) is present, and we have to apply $r6$, or no object (r_j, ∞) is available, and thus $(\mathrm{Dec}_{i,j}{}''', 1)$ disappears.

Let us first consider this last case. The only non–persistent object available at this moment is $(I_s{}', 1)$. The only possible applicable rule is $r7$, which produces the objects $(p_s, 1)$. We have correctly simulated a jump instruction.

If we consider the other possibility, by applying at the same time rules $r4$ and $r5$ we obtain objects $({I_s}', 1)$ and $({C_{i+1}}', 2)$. We could then apply rule $r7$ but, in this case, $({C_{i+1}}', 2)$ must activate rule $c3$ producing an infinite loop. The other possibility is to apply rule $r8$, producing $(\mathrm{Repr}_j, 2)$. The other object still present is $({C_{i+1}}', 1)$.

If we apply rule $r10$, then we also have to apply rule $c3$, which produces an infinite loop. On the contrary, if we apply rule $r9$ we produce the object $({I_{i+1}}', 1)$, and we still have object $(\mathrm{Repr}_j, 1)$.

At this moment the only possibility is to apply at the same time rules $r10$ and $r11$. We remove a single copy of object (r_j, ∞), correctly simulating a decrement instruction, and we obtain the object $(p_{i+1}, 1)$, which can be used to simulate the next instruction.

Thus, as we have just shown, it is possible to simulate RAMs using non–deterministic Genetic Systems, without making use of inhibitors.

5 Conclusion

The present paper contains the results obtained from the investigation of the computational expressiveness of Genetic Systems, a formalism modeling the interactions occurring between genes and regulatory products. In particular, we considered the use of inhibitors, objects which can prevent the application of a genetic rule when present. It is known that Genetic Systems without inhibitors using interleaving semantics are not Turing equivalent. In this paper we showed that non–deterministic Genetic Systems using maximal parallel semantics are universal even if we do not use inhibiting rules.

The paper was planned to be extended, by considering the deterministic version of Genetic Systems using maximal parallel semantics, in order to determine their computational power. Moreover, we were also planning to investigate systems where other ingredients are removed and, in particular, repressor rules. We conjectured that existential termination is decidable in this last case, and this can be proved by providing a mapping of Genetic Systems without repressor rules on safe Contextual Petri Net, which preserves the existence of a deadlock, using a technique similar to the one employed in [7].

Sadly, on September 5, 2007, Nadia passed away unexpectedly, and the development of the paper suddenly stopped. For the final proceedings volume, we decided to leave the paper as it was at the moment of the departure of Nadia, making only some small changes and corrections, but without adding new sections or results.

References

1. Blossey, R., Cardelli, L., Phillips, A.: A Compositional Approach to the Stochastic Dynamics of Gene Networks. In: Priami, C., Cardelli, L., Emmott, S. (eds.) Transactions on Computational Systems Biology IV. LNCS (LNBI), vol. 3939, pp. 99–122. Springer, Heidelberg (2006)

2. Busi, N., Pinna, G.M.: A Causal Semantics for Contextual P/T nets. In: Proc. ICTCS 1995, pp. 311–325. World Scientific, Singapore (1995)
3. Busi, N.: On the computational power of the mate/Bud/Drip brane calculus: Interleaving vs. Maximal parallelism. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 144–158. Springer, Heidelberg (2006)
4. Busi, N., Zandron, C.: Computing with genetic gates, proteins, and membranes. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 250–265. Springer, Heidelberg (2006)
5. Busi, N., Zandron, C.: Computational expressiveness of Genetic Systems (submitted)
6. Busi, N., Zandron, C.: Computing with Genetic Gates. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 105–114. Springer, Heidelberg (2007)
7. Busi, N., Zandron, C.: On the Computational Power of Genetic Gates with Interleaving Semantics: The Power of Inhibition and Degradation. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 173–186. Springer, Heidelberg (2007)
8. Cardelli, L.: Brane Calculi. In: Danos, V., Schachter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 257–278. Springer, Heidelberg (2005)
9. De Jong, H.: Modeling and Simulation of Genetic Regulatory Systems: A Literature Review. Journal of Computatonal Biology 9, 67–103 (2002)
10. Finkel, A., Schnoebelen, P.: Well-Structured Transition Systems Everywhere! Theoretical Computer Science 256, 63–92 (2001)
11. Freund, R.: Asynchronous P Systems and P Systems Working in the Sequential Mode. In: Mauri, G., Păun, G., Jesús Pérez-Jímenez, M., Rozenberg, G., Salomaa, A. (eds.) WMC 2004. LNCS, vol. 3365, pp. 36–62. Springer, Heidelberg (2005)
12. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
13. Montanari, U., Rossi, F.: Contextual Nets. Acta Inform. 32(6), 545–596 (1995)
14. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
15. Reisig, W.: Petri nets: An Introduction. EATCS Monographs in Computer Science. Springer, Heidelberg (1985)
16. Shepherdson, J.C., Sturgis, J.E.: Computability of Recursive Functions. Journal of the ACM 10, 217–255 (1963)

Probabilistic Systems with LimSup and LimInf Objectives

Krishnendu Chatterjee[1] and Thomas A. Henzinger[2,⋆]

[1] Jack Baskin School of Engineering, University of California at Santa Cruz, 1156 High Street, MS: SOE3, Santa Cruz, CA 95064, United States of America
[2] School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne, EPFL Station 14, 1015 Lausanne, Switzerland
{c_krish,tah}@eecs.berkeley.edu

Abstract. We give polynomial-time algorithms for computing the values of Markov decision processes (MDPs) with limsup and liminf objectives. A real-valued reward is assigned to each state, and the value of an infinite path in the MDP is the limsup (resp. liminf) of all rewards along the path. The value of an MDP is the maximal expected value of an infinite path that can be achieved by resolving the decisions of the MDP. Using our result on MDPs, we show that turn-based stochastic games with limsup and liminf objectives can be solved in NP ∩ coNP.

1 Introduction

A *turn-based stochastic game* is played on a finite graph with three types of states: in player-1 states, the first player chooses a successor state from a given set of outgoing edges; in player-2 states, the second player chooses a successor state from a given set of outgoing edges; and probabilistic states, the successor state is chosen according to a given probability distribution. The game results in an infinite path through the graph. Every such path is assigned a real value, and the objective of player 1 is to resolve her choices so as to maximize the expected value of the resulting path, while the objective of player 2 is to minimize the expected value. If the function that assigns values to infinite paths is a Borel function (in the Cantor topology on infinite paths), then the game is determined [12]: the maximal expected value achievable by player 1 is equal to the minimal expected value achievable by player 2, and it is called the *value* of the game.

There are several canonical functions for assigning values to infinite paths. If each state is given a reward, then the *max* (resp. *min*) functions choose the maximum (resp. minimum) of the infinitely many rewards along a path; the

⋆ We thank Hugo Gimbert for explaining his results and pointing out relevant literature on games with limsup and liminf objectives. This research was supported in part by the NSF grants CCR-0132780, CNS-0720884, and CCR-0225610, by the Swiss National Science Foundation, and by the COMBEST project of the European Union. (*Received by the editors.* 9 March 2008. *Revised.* 10 July 2008. *Accepted.* 29 August 2008.)

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, pp. 32–45, 2009.
© Springer-Verlag Berlin Heidelberg 2009

limsup (resp. *liminf*) functions choose the limsup (resp. liminf) of the infinitely many rewards; and the *limavg* function chooses the long-run average of the rewards. For the Borel level-1 functions *max* and *min*, as well as for the Borel level-3 function *limavg*, computing the value of a game is known to be in NP ∩ coNP [10]. However, for the Borel level-2 functions *limsup* and *liminf*, only special cases have been considered so far. If there are no probabilistic states (in this case, the game is called *deterministic*), then the game value can be computed in polynomial time using value-iteration algorithms [1]; likewise, if all states are given reward 0 or 1 (in this case, *limsup* is a Büchi objective, and *liminf* is a coBüchi objective), then the game value can be decided in NP ∩ coNP [3]. In this paper, we show that the values of general turn-based stochastic games with limsup and liminf objectives can be computed in NP ∩ coNP.

It is known that pure memoryless strategies suffice for achieving the value of turn-based stochastic games with limsup and liminf objectives [9]. A strategy is *pure* if the player always chooses a unique successor state (rather than a probability distribution of successor states); a pure strategy is *memoryless* if at every state, the player always chooses the same successor state. Hence a pure memoryless strategy for player 1 is a function from player-1 states to outgoing edges (and similarly for player 2). Since pure memoryless strategies offer polynomial witnesses, our result will follow from polynomial-time algorithms for computing the values of Markov decision processes (MDPs) with limsup and liminf objectives. We provide such algorithms.

An MDP is the special case of a turn-based stochastic game which contains no player-1 (or player-2) states. Using algorithms for solving MDPs with Büchi and coBüchi objectives, we give polynomial-time reductions from MDPs with limsup and liminf objectives to MDPs with max objectives. The solution of MDPs with max objectives is computable by linear programming, and the linear program for MDPs with max objectives is obtained by generalizing the linear program for MDPs with reachability objectives. This will conclude our argument.

Related work. Games with limsup and liminf objectives have been widely studied in game theory; for example, Maitra and Sudderth [11] present several results about games with limsup and liminf objectives. In particular, they show the existence of values in limsup and liminf games that are more general than turn-based stochastic games, such as concurrent games, where the two players repeatedly choose their moves simultaneously and independently, and games with infinite state spaces. Gimbert and Zielonka have studied the strategy complexity of games with limsup and liminf objectives: the sufficiency of pure memoryless strategies for deterministic games was shown in [8], and for turn-based stochastic games, in [9]. Polynomial-time algorithms for MDPs with Büchi and coBüchi objectives were presented in [5], and the solution turn-based stochastic games with Büchi and coBüchi objectives was shown to be in NP ∩ coNP in [3]. For deterministic games with limsup and liminf objectives polynomial-time algorithms have been known, for example, the value-iteration algorithm terminates in polynomial time [1].

2 Definitions

We consider the class of turn-based probabilistic games and some of its subclasses.

Game graphs. A *turn-based probabilistic game graph* ($2\frac{1}{2}$*-player game graph*) $G = ((S, E), (S_1, S_2, S_P), \delta)$ consists of a directed graph (S, E), a partition (S_1, S_2, S_P) of the finite set S of states, and a probabilistic transition function δ: $S_P \to \mathcal{D}(S)$, where $\mathcal{D}(S)$ denotes the set of probability distributions over the state space S. The states in S_1 are the *player*-1 states, where player 1 decides the successor state; the states in S_2 are the *player*-2 states, where player 2 decides the successor state; and the states in S_P are the *probabilistic* states, where the successor state is chosen according to the probabilistic transition function δ. We assume that for $s \in S_P$ and $t \in S$, we have $(s, t) \in E$ iff $\delta(s)(t) > 0$, and we often write $\delta(s, t)$ for $\delta(s)(t)$. For technical convenience we assume that every state in the graph (S, E) has at least one outgoing edge. For a state $s \in S$, we write $E(s)$ to denote the set $\{t \in S \mid (s, t) \in E\}$ of possible successors. The *turn-based deterministic game graphs* (*2-player game graphs*) are the special case of the $2\frac{1}{2}$-player game graphs with $S_P = \emptyset$. The *Markov decision processes* ($1\frac{1}{2}$*-player game graphs*) are the special case of the $2\frac{1}{2}$-player game graphs with $S_1 = \emptyset$ or $S_2 = \emptyset$. We refer to the MDPs with $S_2 = \emptyset$ as *player*-1 MDPs, and to the MDPs with $S_1 = \emptyset$ as *player*-2 MDPs.

Plays and strategies. An infinite path, or a *play*, of the game graph G is an infinite sequence $\omega = \langle s_0, s_1, s_2, \ldots \rangle$ of states such that $(s_k, s_{k+1}) \in E$ for all $k \in \mathbb{N}$. We write Ω for the set of all plays, and for a state $s \in S$, we write $\Omega_s \subseteq \Omega$ for the set of plays that start from the state s. A *strategy* for player 1 is a function σ: $S^* \cdot S_1 \to \mathcal{D}(S)$ that assigns a probability distribution to all finite sequences $\boldsymbol{w} \in S^* \cdot S_1$ of states ending in a player-1 state (the sequence represents a prefix of a play). Player 1 follows the strategy σ if in each player-1 move, given that the current history of the game is $\boldsymbol{w} \in S^* \cdot S_1$, she chooses the next state according to the probability distribution $\sigma(\boldsymbol{w})$. A strategy must prescribe only available moves, i.e., for all $\boldsymbol{w} \in S^*$, $s \in S_1$, and $t \in S$, if $\sigma(\boldsymbol{w} \cdot s)(t) > 0$, then $(s, t) \in E$. The strategies for player 2 are defined analogously. We denote by Σ and Π the set of all strategies for player 1 and player 2, respectively.

Once a starting state $s \in S$ and strategies $\sigma \in \Sigma$ and $\pi \in \Pi$ for the two players are fixed, the outcome of the game is a random walk $\omega_s^{\sigma,\pi}$ for which the probabilities of events are uniquely defined, where an *event* $\mathcal{A} \subseteq \Omega$ is a measurable set of plays. For a state $s \in S$ and an event $\mathcal{A} \subseteq \Omega$, we write $\Pr_s^{\sigma,\pi}(\mathcal{A})$ for the probability that a play belongs to $\mathcal{A}$ if the game starts from the state s and the players follow the strategies σ and π, respectively. For a measurable function $f : \Omega \to \mathbb{R}$ we denote by $\mathbb{E}_s^{\sigma,\pi}[f]$ the *expectation* of the function f under the probability measure $\Pr_s^{\sigma,\pi}(\cdot)$.

Strategies that do not use randomization are called pure. A player-1 strategy σ is *pure* if for all $\boldsymbol{w} \in S^*$ and $s \in S_1$, there is a state $t \in S$ such that $\sigma(\boldsymbol{w} \cdot s)(t) = 1$. A *memoryless* player-1 strategy does not depend on the history of the play but only on the current state; i.e., for all $\boldsymbol{w}, \boldsymbol{w'} \in S^*$ and for all $s \in S_1$ we have

$\sigma(\boldsymbol{w} \cdot s) = \sigma(\boldsymbol{w}' \cdot s)$. A memoryless strategy can be represented as a function $\sigma\colon S_1 \to \mathcal{D}(S)$. A *pure memoryless strategy* is a strategy that is both pure and memoryless. A pure memoryless strategy for player 1 can be represented as a function $\sigma\colon S_1 \to S$. We denote by Σ^{PM} the set of pure memoryless strategies for player 1. The pure memoryless player-2 strategies Π^{PM} are defined analogously.

Given a pure memoryless strategy $\sigma \in \Sigma^{PM}$, let G_σ be the game graph obtained from G under the constraint that player 1 follows the strategy σ. The corresponding definition G_π for a player-2 strategy $\pi \in \Pi^{PM}$ is analogous, and we write $G_{\sigma,\pi}$ for the game graph obtained from G if both players follow the pure memoryless strategies σ and π, respectively. Observe that given a $2\frac{1}{2}$-player game graph G and a pure memoryless player-1 strategy σ, the result G_σ is a player-2 MDP. Similarly, for a player-1 MDP G and a pure memoryless player-1 strategy σ, the result G_σ is a Markov chain. Hence, if G is a $2\frac{1}{2}$-player game graph and the two players follow pure memoryless strategies σ and π, the result $G_{\sigma,\pi}$ is a Markov chain.

Quantitative objectives. A *quantitative* objective is specified as a measurable function $f : \Omega \to \mathbb{R}$. We consider *zero-sum* games, i.e., games that are strictly competitive. In zero-sum games the objectives of the players are functions f and $-f$, respectively. We consider quantitative objectives specified as lim sup and lim inf objectives. These objectives are complete for the second levels of the Borel hierarchy: lim sup objectives are Π_2 complete, and lim inf objectives are Σ_2 complete. The definitions of lim sup and lim inf objectives are as follows.

- *Limsup objectives.* Let $r : S \to \mathbb{R}$ be a real-valued reward function that assigns to every state s the reward $r(s)$. The *limsup* objective lim sup assigns to every play the maximum reward that appears infinitely often in the play. Formally, for a play $\omega = \langle s_1, s_2, s_3, \ldots \rangle$ we have
$$\limsup(r)(\omega) = \limsup\langle r(s_i)\rangle_{i\geq 0}.$$
- *Liminf objectives.* Let $r : S \to \mathbb{R}$ be a real-valued reward function that assigns to every state s the reward $r(s)$. The *liminf* objective lim inf assigns to every play the maximum reward v such that the rewards that appear eventually always in the play is at least v. Formally, for a play $\omega = \langle s_1, s_2, s_3, \ldots \rangle$ we have
$$\liminf(r)(\omega) = \liminf\langle r(s_i)\rangle_{i\geq 0}.$$
The objectives lim sup and lim inf are complementary in the sense that for all plays ω we have $\limsup(r)(\omega) = -\liminf(-r)(\omega)$.

We also define the max objectives, as it will be useful in study of MDPs with lim sup and lim inf objectives. Later we will reduce MDPs with lim sup and lim inf objectives to MDPs with max objectives. For a reward function $r : S \to \mathbb{R}$ the *max* objective max assigns to every play the maximum reward that appears in the play. Observe that since S is finite, the number of different rewards appearing in a play is finite and hence the maximum is defined. Formally, for a play $\omega = \langle s_1, s_2, s_3, \ldots \rangle$ we have
$$\max(r)(\omega) = \max\langle r(s_i)\rangle_{i\geq 0}.$$

Büchi and coBüchi objectives. We define the qualitative variant of lim sup and lim inf objectives, namely, Büchi and coBüchi objectives. The notion of qualitative variants of the objectives will be useful in the algorithmic analysis of $2\frac{1}{2}$-player games with lim sup and lim inf objectives. For a play ω, we define $\mathrm{Inf}(\omega) = \{ s \in S \mid s_k = s \text{ for infinitely many } k \geq 0 \}$ to be the set of states that occur infinitely often in ω.

- *Büchi objectives.* Given a set $B \subseteq S$ of Büchi states, the Büchi objective $\mathrm{Büchi}(B)$ requires that some state in B be visited infinitely often. The set of winning plays is $\mathrm{Büchi}(B) = \{ \omega \in \Omega \mid \mathrm{Inf}(\omega) \cap B \neq \emptyset \}$.
- *co-Büchi objectives.* Given a set $C \subseteq S$ of coBüchi states, the co-Büchi objective $\mathrm{coBüchi}(C)$ requires that only states in C be visited infinitely often. Thus, the set of winning plays is $\mathrm{coBüchi}(C) = \{ \omega \in \Omega \mid \mathrm{Inf}(\omega) \subseteq C \}$. The Büchi and coBüchi objectives are dual in the sense that $\mathrm{Büchi}(B) = \Omega \setminus \mathrm{coBüchi}(S \setminus B)$.

Given a set $B \subseteq S$, consider a boolean reward function r_B such that for all $s \in S$ we have $r_B(s) = 1$ if $s \in B$, and 0 otherwise. Then for all plays ω we have $\omega \in \mathrm{Büchi}(B)$ iff $\limsup(r_B)(\omega) = 1$. Similarly, given a set $C \subseteq S$, consider a boolean reward function r_C such that for all $s \in S$ we have $r_C(s) = 1$ if $s \in C$, and 0 otherwise. Then for all plays ω we have $\omega \in \mathrm{coBüchi}(C)$ iff $\liminf(r_C)(\omega) = 1$.

Values and optimal strategies. Given a game graph G, qualitative objectives $\Phi \subseteq \Omega$ for player 1 and $\Omega \setminus \Phi$ for player 2, and measurable functions f and $-f$ for player 1 and player 2, respectively, we define the *value* functions $\langle\langle 1 \rangle\rangle_{\mathrm{val}}$ and $\langle\langle 2 \rangle\rangle_{\mathrm{val}}$ for the players 1 and 2, respectively, as the following functions from the state space S to the set $\mathbb{R}$ of reals: for all states $s \in S$, let

$$\langle\langle 1 \rangle\rangle^G_{\mathrm{val}}(\Phi)(s) = \sup_{\sigma \in \Sigma} \inf_{\pi \in \Pi} \Pr^{\sigma,\pi}_s(\Phi);$$

$$\langle\langle 1 \rangle\rangle^G_{\mathrm{val}}(f)(s) = \sup_{\sigma \in \Sigma} \inf_{\pi \in \Pi} \mathbb{E}^{\sigma,\pi}_s[f];$$

$$\langle\langle 2 \rangle\rangle^G_{\mathrm{val}}(\Omega \setminus \Phi)(s) = \sup_{\pi \in \Pi} \inf_{\sigma \in \Sigma} \Pr^{\sigma,\pi}_s(\Omega \setminus \Phi);$$

$$\langle\langle 2 \rangle\rangle^G_{\mathrm{val}}(-f)(s) = \sup_{\pi \in \Pi} \inf_{\sigma \in \Sigma} \mathbb{E}^{\sigma,\pi}_s[-f].$$

In other words, the values $\langle\langle 1 \rangle\rangle^G_{\mathrm{val}}(\Phi)(s)$ and $\langle\langle 1 \rangle\rangle^G_{\mathrm{val}}(f)(s)$ give the maximal probability and expectation with which player 1 can achieve her objectives Φ and f from state s, and analogously for player 2. The strategies that achieve the values are called optimal: a strategy σ for player 1 is *optimal* from the state s for the objective Φ if $\langle\langle 1 \rangle\rangle^G_{\mathrm{val}}(\Phi)(s) = \inf_{\pi \in \Pi} \Pr^{\sigma,\pi}_s(\Phi)$; and σ is *optimal* from the state s for f if $\langle\langle 1 \rangle\rangle^G_{\mathrm{val}}(f)(s) = \inf_{\pi \in \Pi} \mathbb{E}^{\sigma,\pi}_s[f]$. The optimal strategies for player 2 are defined analogously. We now state the classical determinacy results for $2\frac{1}{2}$-player games with limsup and liminf objectives.

Theorem 1 (Quantitative determinacy). *For all $2\frac{1}{2}$-player game graphs $G = ((S, E), (S_1, S_2, S_P), \delta)$, the following assertions hold.*

1. *For all reward functions* $r : S \to \mathbb{R}$ *and all states* $s \in S$*, we have*

$$\langle\langle 1\rangle\rangle^G_{\mathrm{val}}(\limsup(r))(s) + \langle\langle 2\rangle\rangle^G_{\mathrm{val}}(\liminf(-r))(s) = 0;$$

$$\langle\langle 1\rangle\rangle^G_{\mathrm{val}}(\liminf(r))(s) + \langle\langle 2\rangle\rangle^G_{\mathrm{val}}(\limsup(-r))(s) = 0.$$

2. *Pure memoryless optimal strategies exist for both players from all states.*

The above results can be derived from the results in [11]; a more direct proof can be obtained as follows: the existence of pure memoryless optimal strategies for MDPs with limsup and liminf objectives can be proved by extending the results known for Büchi and coBüchi objectives. The results (Theorem 3.19) of [7] proved that if for a quantitative objective f and its complement $-f$ pure memoryless optimal strategies exist in MDPs, then pure memoryless optimal strategies also exist in $2\frac{1}{2}$-player games. Hence the pure memoryless determinacy follows for $2\frac{1}{2}$-player games with limsup and liminf objectives.

3 The Complexity of $2\frac{1}{2}$-Player Games with Limsup and LimInf Objectives

In this section we study the complexity of MDPs and $2\frac{1}{2}$-player games with limsup and liminf objectives. We present polynomial time algorithms for MDPs and show that $2\frac{1}{2}$-player games can be decided in NP $\cap$ coNP. In the next subsections we present polynomial time algorithms for MDPs with limsup and liminf objectives by reductions to a simple linear-programming formulation, and then show that $2\frac{1}{2}$-player games can be decided in NP $\cap$ coNP. We first present a remark and then present some basic results on MDPs.

Remark 1. Given a $2\frac{1}{2}$-player game graph G with a reward function $r : S \to \mathbb{R}$ and a real constant c, consider the reward function $(r + c) : S \to \mathbb{R}$ defined as follows: for $s \in S$ we have $(r + c)(s) = r(s) + c$. Then the following assertions hold: for all $s \in S$

$$\langle\langle 1\rangle\rangle^G_{\mathrm{val}}(\limsup(r + c))(s) = \langle\langle 1\rangle\rangle^G_{\mathrm{val}}(\limsup(r))(s) + c;$$

$$\langle\langle 1\rangle\rangle^G_{\mathrm{val}}(\liminf(r + c))(s) = \langle\langle 1\rangle\rangle^G_{\mathrm{val}}(\liminf(r))(s) + c.$$

Hence we can shift a reward function r by a real constant c, and from the value function for the reward function $(r+c)$, we can easily compute the value function for r. Hence without loss of generality for computational purpose we assume that we have reward function with positive rewards, i.e., $r : S \to \mathbb{R}^+$, where $\mathbb{R}^+$ is the set of positive reals.

3.1 Basic Results on MDPs

In this section we recall several basic properties on MDPs. We start with the definition of *end components* in MDPs [5,4] that play a role equivalent to closed recurrent sets in Markov chains.

End components. Given an MDP $G = ((S, E), (S_1, S_P), \delta)$, a set $U \subseteq S$ of states is an *end component* if U is δ-closed (i.e., for all $s \in U \cap S_P$ we have $E(s) \subseteq U$) and the sub-game graph of G restricted to U (denoted $G \restriction U$) is strongly connected. We denote by $\mathcal{E}(G)$ the set of end components of an MDP G. The following lemma states that, given any strategy (memoryless or not), with probability 1 the set of states visited infinitely often along a play is an end component. This lemma allows us to derive conclusions on the (infinite) set of plays in an MDP by analyzing the (finite) set of end components in the MDP.

Lemma 1. *[5,4] Given an MDP G, for all states $s \in S$ and all strategies $\sigma \in \Sigma$, we have* $\Pr_s^\sigma(\{\, \omega \mid \mathrm{Inf}(\omega) \in \mathcal{E}(G) \,\}) = 1$.

For an end component $U \in \mathcal{E}(G)$, consider the memoryless strategy σ_U that at a state s in $U \cap S_1$ plays all edges in $E(s) \cap U$ uniformly at random. Given the strategy σ_U, the end component U is a closed connected recurrent set in the Markov chain obtained by fixing σ_U.

Lemma 2. *Given an MDP G and an end component $U \in \mathcal{E}(G)$, the strategy σ_U ensures that for all states $s \in U$, we have* $\Pr_s^{\sigma_U}(\{\, \omega \mid \mathrm{Inf}(\omega) = U \,\}) = 1$.

Almost-sure winning states. Given an MDP G with a Büchi or a coBüchi objective Φ for player 1, we denote by

$$W_1^G(\Phi) = \{\, s \in S \mid \langle\!\langle 1 \rangle\!\rangle_{\mathrm{val}}(\Phi)(s) = 1 \,\};$$

the sets of states such that the values for player 1 is 1. These sets of states are also referred as the *almost-sure* winning states for the player and an optimal strategy from the almost-sure winning states is referred as an almost-sure winning strategy. The set $W_1^G(\Phi)$, for Büchi or coBüchi objectives Φ, for an MDP G can be computed in $O(n^{\frac{3}{2}})$ time, where n is the size of the MDP G [2].

Attractor of probabilistic states. We define a notion of *attractor* of probabilistic states: given an MDP G and a set $U \subseteq S$ of states, we denote by $\mathrm{Attr}_P(U, G)$ the set of states from where the probabilistic player has a strategy (with proper choice of edges) to force the game to reach U. The set $\mathrm{Attr}_P(U, G)$ is inductively defined as follows:

$$T_0 = U; \quad T_{i+1} = T_i \cup \{\, s \in S_P \mid E(s) \cap T_i \neq \emptyset \,\} \cup \{\, s \in S_1 \mid E(s) \subseteq T_i \,\}$$

and $\mathrm{Attr}_P(U, G) = \bigcup_{i \geq 0} T_i$.

We now present a lemma about MDPs with Büchi and coBüchi objectives and a property of end components and attractors. The first two properties of Lemma 3 follows from Lemma 2. The last property follows from the fact that an end component is δ-closed (i.e., for an end component U, for all $s \in U \cap S_P$ we have $E(s) \subseteq U$).

Lemma 3. *Let G be an MDP. Given $B \subseteq S$ and $C \subseteq S$, the following assertions hold.*

1. *For all* $U \in \mathcal{E}(G)$ *such that* $U \cap B \neq \emptyset$, *we have* $U \subseteq W_1^G(\mathit{Büchi}(B))$.
2. *For all* $U \in \mathcal{E}(G)$ *such that* $U \subseteq C$, *we have* $U \subseteq W_1^G(\mathit{coBüchi}(C))$.
3. *For all* $Y \subseteq S$ *and all end components* $U \in \mathcal{E}(G)$, *if* $X = \mathrm{Attr}_P(Y, G)$, *then either (a)* $U \cap Y \neq \emptyset$ *or (b)* $U \cap X = \emptyset$.

3.2 MDPs with Limsup Objectives

In this subsection we present polynomial time algorithm for MDPs with limsup objectives. For the sake of simplicity we will consider bipartite MDPs.

Bipartite MDPs. An MDP $G = ((S, E), (S_1, S_P), \delta)$ is *bipartite* if $E \subseteq S_1 \times S_P \cup S_P \times S_1$. An MDP G can be converted into a bipartite MDP G' by adding dummy states with an unique successor, and G' is linear in the size of G. In sequel without loss of generality we will consider bipartite MDPs. The key property of bipartite MDPs that will be useful is as follows: for a bipartite MDP $G = ((S, E), (S_1, S_P), \delta)$, for all $U \in \mathcal{E}(G)$ we have $U \cap S_1 \neq \emptyset$.

Informal description of algorithm. We first present an algorithm that takes an MDP G with a positive reward function $r : S \to \mathbb{R}^+$, and computes a set S^* and a function $f^* : S^* \to \mathbb{R}^+$. The output of the algorithm will be useful in reduction of MDPs with limsup objectives to MDPs with max objectives. Let the rewards be $v_0 > v_1 > \cdots > v_k$. The algorithm proceeds in iteration and in iteration i we denote the MDP as G_i and the state space as S^i. At iteration i the algorithm considers the set V_i of reward v_i in the MDP G_i, and computes the set $U_i = W_1^{G_i}(\text{Büchi}(V_i))$, (i.e., the almost-sure winning set in the MDP G_i for Büchi objective with the Büchi set V_i). For all $u \in U_i \cap S_i$ we assign $f^*(u) = v_i$ and add the set $U_i \cap S_1$ to S^*. Then the set $\mathrm{Attr}_P(U_i, G_i)$ is removed from the MDP G_i and we proceed to iteration $i + 1$. In G_i all end components that intersect with reward v_i are contained in U_i (by Lemma 3 part (1)), and all end components in $S^i \setminus U_i$ do not intersect with $\mathrm{Attr}_P(U_i, G_i)$ (by Lemma 3 part(3)). This gives us the following lemma.

Lemma 4. *Let* G *be an MDP with a positive reward function* $r : S \to \mathbb{R}^+$. *Let* f^* *be the output of Algorithm 1.. For all end components* $U \in \mathcal{E}(G)$ *and all states* $u \in U \cap S_1$, *we have* $\max(r(U)) \leq f^*(u)$.

Proof. Let $U^* = \bigcup_{i=0}^{k} U_i$ (as computed in Algorithm 1.). Then it follows from Lemma 3 that for all $A \in \mathcal{E}(G)$ we have $A \cap U^* \neq \emptyset$. Consider $A \in \mathcal{E}(G)$ and let $v_i = \max(r(A))$. Suppose for some $j < i$ we have $A \cap U_j \neq \emptyset$. Then there is a strategy to ensure that U_j is reached with probability 1 from all states in A and then play an almost-sure winning strategy in U_j to ensure $\text{Büchi}(r^{-1}(v_j) \cap S^j)$. Then $A \subseteq U_j$. Hence for all $u \in A \cap S_1$ we have $f^*(u) = v_j \geq v_i$. If for all $j < i$ we have $A \cap U_j = \emptyset$, then we show that $A \subseteq U_i$. The uniform memoryless strategy σ_A (as used in Lemma 2) in G_i is a witness to prove that $A \subseteq U_i$. In this case for all $u \in A \cap S_1$ we have $f^*(u) = v_i = \max(r(A))$. The desired result follows. ■

Algorithm 1. MDPLimSup

Input: MDP $G = ((S, E), (S_1, S_P), \delta)$, a positive reward function $r : S \to \mathbb{R}^+$.
Output: $S^* \subseteq S$ and $f^* : S^* \to \mathbb{R}^+$
1. Let $r(S) = \{ v_0, v_1, \ldots, v_k \}$ with $v_0 > v_1 > \cdots > v_k$;
2. $G_0 := G$; $S^* = \emptyset$;
3. **for** $i := 0$ to k **do** {
   3.1 $U_i := W_1^{G_i}(\text{Büchi}(r^{-1}(v_i) \cap S^i))$;
   3.2 **for** all $u \in U_i \cap S_1$
   $f^*(u) := v_i$;
   3.3 $S^* := S^* \cup (U_i \cap S_1)$;
   3.4 $B_i := \text{Attr}_P(U_i, G_i)$;
   3.5 $G_{i+1} := G_i \setminus B_i$, $S^{i+1} := S^i \setminus B_i$;
}
4. **return** S^* and f^*.

Transformation to MDPs with max **objective.** Given an MDP $G = ((S, E), (S_1, S_P), \delta)$ with a positive reward function $r : S \to \mathbb{R}^+$, and let S^* and f^* be the output of Algorithm 1.. We construct an MDP $\overline{G} = ((\overline{S}, \overline{E}), (\overline{S}_1, \overline{S}_P), \overline{\delta})$ with a reward function $\overline{r}$ as follows:

- $\overline{S} = S \cup \widehat{S}^*$; i.e., the set of states consists of the state space S and a copy $\widehat{S}^*$ of S^*.
- $\overline{E} = E \cup \{ (s, \widehat{s}) \mid s \in S^*, \widehat{s} \in \widehat{S}^* \text{ where } \widehat{s} \text{ is the copy of } s \} \cup \{ (\widehat{s}, \widehat{s}) \mid \widehat{s} \in \widehat{S}^* \}$; i.e., along with edges E, for all states s in S^* there is an edge to its copy $\widehat{s}$ in $\widehat{S}^*$, and all states in $\widehat{S}^*$ are absorbing states.
- $\overline{S}_1 = S_1 \cup \widehat{S}^*$.
- $\overline{\delta} = \delta$.
- $\overline{r}(s) = 0$ for all $s \in S$ and $\overline{r}(s) = f^*(s)$ for $\widehat{s} \in \widehat{S}^*$, where $\widehat{s}$ is the copy of s.

We refer to the above construction as *limsup* conversion. The following lemma proves the relationship between the value function $\langle\langle 1 \rangle\rangle^G_{\text{val}}(\limsup(r))$ and $\langle\langle 1 \rangle\rangle^{\overline{G}}_{\text{val}}(\max(\overline{r}))$.

Lemma 5. *Let G be an MDP with a positive reward function $r : S \to \mathbb{R}^+$. Let $\overline{G}$ and $\overline{r}$ be obtained from G and r by the limsup conversion. For all states $s \in S$, we have*

$$\langle\langle 1 \rangle\rangle^G_{\text{val}}(\limsup(r))(s) = \langle\langle 1 \rangle\rangle^{\overline{G}}_{\text{val}}(\max(\overline{r}))(s).$$

Proof. The result is obtained from the following two case analysis.

1. Let σ be a pure memoryless optimal strategy in G for the objective $\limsup(r)$. Let $\mathcal{C} = \{ C_1, C_2, \ldots, C_m \}$ be the set of closed connected recurrent sets in the Markov chain obtained from G after fixing the strategy σ. Note that since we consider bipartite MDPs, for all $1 \leq i \leq m$, we have

$C_i \cap S_1 \neq \emptyset$. Let $C = \bigcup_{i=1}^{m} C_i$. We define a pure memoryless strategy $\overline{\sigma}$ in $\overline{G}$ as follows

$$\overline{\sigma}(s) = \begin{cases} \sigma(s) & s \in S_1 \setminus C; \\ \widehat{s} & \widehat{s} \in \widehat{S}^* \text{ and } s \in S_1 \cap C. \end{cases}$$

By Lemma 4 it follows that the strategy $\overline{\sigma}$ ensures that for all $C_i \in \mathcal{C}$ and all $s \in C_i$, the maximal reward reached in $\overline{G}$ is at least $\max(r(C_i))$ with probability 1. It follows that for all $s \in S$ we have

$$\langle\!\langle 1 \rangle\!\rangle^{G}_{\mathrm{val}}(\limsup(r))(s) \leq \langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\mathrm{val}}(\max(\overline{r}))(s).$$

2. Let $\overline{\sigma}$ be a pure memoryless optimal strategy for the objective $\max(\overline{r})$ in $\overline{G}$. We fix a strategy σ in G as follows: if at a state $s \in S^*$ the strategy $\overline{\sigma}$ chooses the edge $(s, \widehat{s})$, then in G on reaching s, the strategy σ plays an almost-sure winning strategy for the objective Büchi$(r^{-1}(f^*(s)))$, otherwise σ follows $\overline{\sigma}$. It follows that for all $s \in S$ we have

$$\langle\!\langle 1 \rangle\!\rangle^{G}_{\mathrm{val}}(\limsup(r))(s) \geq \langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\mathrm{val}}(\max(\overline{r}))(s).$$

Thus we have the desired result. ■

Linear programming for the max **objective in $\overline{G}$.** The following linear program characterizes the value function $\langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\mathrm{val}}(\max(\overline{r}))$. For all $s \in \overline{S}$ we have a variable x_s and the objective function is $\min \sum_{s \in \overline{S}} x_s$. The set of linear constraints are as follows:

$$\begin{array}{ll} x_s \geq 0 & \forall s \in \overline{S}; \\ x_s = \overline{r}(s) & \forall s \in \widehat{S}^*; \\ x_s \geq x_t & \forall s \in \overline{S}_1, (s,t) \in \overline{E}; \\ x_s = \sum_{t \in \overline{S}} \overline{\delta}(s)(t) \cdot x_t & \forall s \in \overline{S}_P. \end{array}$$

The correctness proof of the above linear program to characterize the value function $\langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\mathrm{val}}(\max(\overline{r}))$ follows by extending the result for reachability objectives [6]. The key property that can be used to prove the correctness of the above claim is as follows: if a pure memoryless optimal strategy is fixed, then from all states in S, the set $\widehat{S}^*$ of absorbing states is reached with probability 1. The above property can be proved as follows: since r is a positive reward function, it follows that for all $s \in S$ we have $\langle\!\langle 1 \rangle\!\rangle^{G}_{\mathrm{val}}(\limsup(r))(s) > 0$. Moreover, for all states $s \in S$ we have $\langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\mathrm{val}}(\max(\overline{r}))(s) = \langle\!\langle 1 \rangle\!\rangle^{G}_{\mathrm{val}}(\limsup(r))(s) > 0$. Observe that for all $s \in S$ we have $\overline{r}(s) = 0$. Hence if we fix a pure memoryless optimal strategy σ in $\overline{G}$, then in the Markov chain $\overline{G}_\sigma$ there is no closed recurrent set C such that $C \subseteq S$. It follows that for all states $s \in S$, in the Markov $\overline{G}_\sigma$, the set $\widehat{S}^*$ is reached with probability 1. Using the above fact and the correctness of linear-programming for reachability objectives, the correctness proof of the above linear-program for the objective $\max(\overline{r})$ in $\overline{G}$ can be obtained. This shows that the value function $\langle\!\langle 1 \rangle\!\rangle^{G}_{\mathrm{val}}(\limsup(r))$ for MDPs with reward function r can be computed in polynomial time. This gives us the following result.

Theorem 2. *Given an MDP G with a reward function r, the value function $\langle\!\langle 1 \rangle\!\rangle^{G}_{\mathrm{val}}(\limsup(r))$ can be computed in polynomial time.*

Algorithm 2. MDPLimInf

Input: MDP $G = ((S, E), (S_1, S_P), \delta)$, a positive reward function $r : S \to \mathbb{R}^+$.
Output: $S_* \subseteq S$ and $f_* : S_* \to \mathbb{R}^+$.
1. Let $r(S) = \{ v_0, v_1, \ldots, v_k \}$ with $v_0 > v_1 > \cdots > v_k$;
2. $G_0 := G$; $S_* = \emptyset$;
3. **for** $i := 0$ to k **do** {
   3.1 $U_i := W_1^{G_i}(\text{coBüchi}(\bigcup_{j \le i} r^{-1}(v_j) \cap S^i))$;
   3.2 **for** all $u \in U_i \cap S_1$
       $f_*(u) := v_i$;
   3.3 $S_* := S_* \cup (U_i \cap S_1)$;
   3.4 $B_i := \text{Attr}_P(U_i, G_i)$;
   3.5 $G_{i+1} := G_i \setminus B_i$, $S^{i+1} := S^i \setminus B_i$;
}
4. **return** S_* and f_*.

3.3 MDPs with Liminf Objectives

In this subsection we present polynomial time algorithms for MDPs with liminf objectives, and then present the complexity result for $2\frac{1}{2}$-player games with limsup and liminf objectives.

Informal description of algorithm. We first present an algorithm that takes an MDP G with a positive reward function $r : S \to \mathbb{R}^+$, and computes a set S_* and a function $f_* : S_* \to \mathbb{R}^+$. The output of the algorithm will be useful in reduction of MDPs with liminf objectives to MDPs with max objectives. Let the rewards be $v_0 > v_1 > \cdots > v_k$. The algorithm proceeds in iteration and in iteration i we denote the MDP as G_i and the state space as S^i. At iteration i the algorithm considers the set V_i of reward at least v_i in the MDP G_i, and computes the set $U_i = W_1^{G_i}(\text{coBüchi}(V_i))$, (i.e., the almost-sure winning set in the MDP G_i for coBüchi objective with the coBüchi set V_i). For all $u \in U_i \cap S_i$ we assign $f_*(u) = v_i$ and add the set $U_i \cap S_1$ to S_*. Then the set $\text{Attr}_P(U_i, G_i)$ is removed from the MDP G_i and we proceed to iteration $i + 1$. In G_i all end components that contain reward at least v_i are contained in U_i (by Lemma 3 part (2)), and all end components in $S^i \setminus U_i$ do not intersect with $\text{Attr}_P(U_i, G_i)$ (by Lemma 3 part(3)). This gives us the following lemma.

Lemma 6. *Let G be an MDP with a positive reward function $r : S \to \mathbb{R}^+$. Let f_* be the output of Algorithm 2.. For all end components $U \in \mathcal{E}(G)$ and all states $u \in U \cap S_1$, we have* $\min(r(U)) \le f^*(u)$.

Proof. Let $U^* = \bigcup_{i=0}^{k} U_i$ (as computed in Algorithm 2.). Then it follows from Lemma 3 that for all $A \in \mathcal{E}(G)$ we have $A \cap U^* \neq \emptyset$. Consider $A \in \mathcal{E}(G)$ and let $v_i = \min(r(A))$. Suppose for some $j < i$ we have $A \cap U_j \neq \emptyset$. Then there is a strategy to ensure that U_j is reached with probability 1 from all states in A and then play an almost-sure winning strategy in U_j to ensure coBüchi($\bigcup_{l \le j} r^{-1}(v_l) \cap S^j$). Then $A \subseteq U_j$. Hence for all $u \in A \cap S_1$ we have $f_*(u) = v_j \ge v_i$. If for all

$j < i$ we have $A \cap U_j = \emptyset$, then we show that $A \subseteq U_i$. The uniform memoryless strategy σ_A (as used in Lemma 2) in G_i is a witness to prove that $A \subseteq U_i$. In this case for all $u \in A \cap S_1$ we have $f_*(u) = v_i = \min(r(A))$. The desired result follows. ■

Transformation to MDPs with max **objective.** Given an MDP $G = ((S, E), (S_1, S_P), \delta)$ with a positive reward function $r : S \to \mathbb{R}^+$, and let S_* and f_* be the output of Algorithm 2.. We construct an MDP $\overline{G} = ((\overline{S}, \overline{E}), (\overline{S}_1, \overline{S}_P), \overline{\delta})$ with a reward function $\overline{r}$ as follows:

- $\overline{S} = S \cup \widehat{S}_*$; i.e., the set of states consists of the state space S and a copy $\widehat{S}_*$ of S_*.
- $\overline{E} = E \cup \{ (s, \widehat{s}) \mid s \in S_*, \widehat{s} \in \widehat{S}_* \text{ where } \widehat{s} \text{ is the copy of } s \} \cup \{ (\widehat{s}, \widehat{s}) \mid \widehat{s} \in \widehat{S}_* \}$; along with edges E, for all states s in S_* there is an edge to its copy $\widehat{s}$ in $\widehat{S}_*$, and all states in $\widehat{S}_*$ are absorbing states.
- $\overline{S}_1 = S_1 \cup \widehat{S}_*$.
- $\overline{\delta} = \delta$.
- $\overline{r}(s) = 0$ for all $s \in S$ and $\overline{r}(s) = f_*(s)$ for $\widehat{s} \in \widehat{S}_*$, where $\widehat{s}$ is the copy of s.

We refer to the above construction as *liminf* conversion. The following lemma proves the relationship between the value function $\langle\!\langle 1 \rangle\!\rangle^G_{\text{val}}(\liminf(r))$ and $\langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\text{val}}(\max(\overline{r}))$.

Lemma 7. *Let G be an MDP with a positive reward function $r : S \to \mathbb{R}^+$. Let $\overline{G}$ and $\overline{r}$ be obtained from G and r by the liminf conversion. For all states $s \in S$, we have*

$$\langle\!\langle 1 \rangle\!\rangle^G_{\text{val}}(\liminf(r))(s) = \langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\text{val}}(\max(\overline{r}))(s).$$

Proof. The result is obtained from the following two case analysis.

1. Let σ be a pure memoryless optimal strategy in G for the objective $\liminf(r)$. Let $\mathcal{C} = \{ C_1, C_2, \ldots, C_m \}$ be the set of closed connected recurrent sets in the Markov chain obtained from G after fixing the strategy σ. Since G is an bipartite MDP, it follows that for all $1 \leq i \leq m$, we have $C_i \cap S_1 \neq \emptyset$. Let $C = \bigcup_{i=1}^m C_i$. We define a pure memoryless strategy $\overline{\sigma}$ in $\overline{G}$ as follows

$$\overline{\sigma}(s) = \begin{cases} \sigma(s) & s \in S_1 \setminus C; \\ \widehat{s} & \widehat{s} \in \widehat{S}_* \text{ and } s \in S_1 \cap C. \end{cases}$$

   By Lemma 6 it follows that the strategy $\overline{\sigma}$ ensures that for all $C_i \in \mathcal{C}$ and all $s \in C_i$, the maximal reward reached in $\overline{G}$ is at least $\min(r(C_i))$ with probability 1. It follows that for all $s \in S$ we have

$$\langle\!\langle 1 \rangle\!\rangle^G_{\text{val}}(\limsup(r))(s) \leq \langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\text{val}}(\max(\overline{r}))(s).$$

2. Let $\overline{\sigma}$ be a pure memoryless optimal strategy for the objective $\max(\overline{r})$ in $\overline{G}$. We fix a strategy σ in G as follows: if at a state $s \in S_*$ the strategy $\overline{\sigma}$ chooses

the edge $(s, \widehat{s})$, then in G on reaching s, the strategy σ plays an almost-sure winning strategy for the objective coBüchi$(\bigcup_{v_j \geq f_*(s)} r^{-1}(v_j))$, otherwise σ follows $\overline{\sigma}$. It follows that for all $s \in S$ we have

$$\langle\!\langle 1 \rangle\!\rangle^G_{\mathrm{val}}(\liminf(r))(s) \geq \langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\mathrm{val}}(\max(\overline{r}))(s).$$

Thus we have the desired result. ■

Linear programming for the max **objective in $\overline{G}$.** The linear program of subsection 3.2 characterizes the value function $\langle\!\langle 1 \rangle\!\rangle^{\overline{G}}_{\mathrm{val}}(\max(\overline{r}))$. This shows that the value function $\langle\!\langle 1 \rangle\!\rangle^G_{\mathrm{val}}(\liminf(r))$ for MDPs with reward function r can be computed in polynomial time. This gives us the following result.

Theorem 3. *Given an MDP G with a reward function r, the value function $\langle\!\langle 1 \rangle\!\rangle^G_{\mathrm{val}}(\liminf(r))$ can be computed in polynomial time.*

3.4 2½-player Games with Limsup and Liminf Objectives

We now show that 2½-player games with limsup and liminf objectives can be decided in NP ∩ coNP. The pure memoryless optimal strategies (existence follows from Theorem 1) provide the polynomial witnesses and to obtain the desired result we need to present a polynomial time verification procedure. In other words, we need to present polynomial time algorithms for MDPs with limsup and liminf objectives. Since the value functions in MDPs with limsup and liminf objectives can be computed in polynomial time (Theorem 2 and Theorem 3), we obtain the following result about the complexity 2½-player games with limsup and liminf objectives.

Theorem 4. *Given a 2½-player game graph G with a reward function r, a state s and a rational value q, the following assertions hold: (a) whether $\langle\!\langle 1 \rangle\!\rangle^G_{\mathrm{val}}(\limsup(r))(s) \geq q$ can be decided in NP ∩ coNP; and (b) whether $\langle\!\langle 1 \rangle\!\rangle^G_{\mathrm{val}}(\liminf(r))(s) \geq q$ can be decided in NP ∩ coNP.*

References

1. Chatterjee, K., Henzinger, T.A.: Value iteration. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking. LNCS, vol. 5000, pp. 107–138. Springer, Heidelberg (2008)
2. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Simple stochastic parity games. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 100–113. Springer, Heidelberg (2003)
3. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Quantitative stochastic parity games. In: Ian Munro, J. (ed.) SODA 2004, pp. 121–130. SIAM, Philadelphia (2004)
4. Courcoubetis, C., Yannakakis, M.: Markov decision processes and regular events. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 336–349. Springer, Heidelberg (1990)

5. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University (1997)
6. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, Heidelberg (1997)
7. Gimbert, H.: Jeux positionnels. PhD thesis, Université Paris 7 (2006)
8. Gimbert, H., Zielonka, W.: Games where you can play optimally without any memory. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 428–442. Springer, Heidelberg (2005)
9. Gimbert, H., Zielonka, W.: Perfect information stochastic priority games. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 850–861. Springer, Heidelberg (2007)
10. Liggett, T.A., Lippman, S.A.: Stochastic games with perfect information and time average payoff. SIAM Review 11, 604–607 (1969)
11. Maitra, A., Sudderth, W. (eds.): Discrete Gambling and Stochastic Games. Stochastic Modelling and Applied Probability, vol. 32. Springer, Heidelberg (1996)
12. Martin, D.A.: The determinacy of Blackwell games. J. Symb. Log. 63(4), 1565–1581 (1998)

A Playful Glance at Hierarchical Questions for Two-Way Alternating Automata

Jacques Duparc[1] and Alessandro Facchini[1,2,⋆]

[1] Faculté des Hautes Études Commerciales, Institut des Systèmes d'information, Université de Lausanne, 1015 Lausanne, Switzerland
[2] Laboratoire Bordelais de Recherche en Informatique, Université Bordeaux 1, 351 cours de la Libération, 33405 Talence cedex, France
{jacques.duparc,alessandro.facchini}@unil.ch

Abstract. Two-way alternating automata were introduced by Vardi in order to study the satisfiability problem for the modal μ-calculus extended with backwards modalities. In this paper, we present a very simple proof by way of Wadge games of the strictness of the hierarchy of Motowski indices of two-way alternating automata over trees.

1 Introduction

Since the seminal work of Büchi, the idea of translating a logic into appropriate models of finite-state automata on infinite words or infinite trees has become a central paradigm in the theory of verification of concurrent systems. One of the reasons is that this translation reduces the model-checking issue to the non emptyness problem for automata. Two hiearchies are classically used to measure the complexity of recognizable sets of infinite words or trees: the Mostowski index hierarchy, and the Wadge hierarchy. Since it reflects the depth of nesting of positive and negative conditions, the first hierarchy determines the combinatorial complexity of the recognizing automaton and therefore is closely related to the fixpoint alternation hierarchy of the modal μ-calculus, unveiling subtle connections between this logic, (parity) automata and games. The second one on the other hand captures the topological complexity of the languages accepted by such machines. Indeed from a topological point of view, an infinite tree is very similar to an infinite word, which is also very close to a real number. Moreover, evidences indicate that the two hierarchies are closely related. Thus, as is well said in [19], "understanding the structure of these hierarchies helps us to understand the trade-off between expressivness and efficiency in the model-checking method".

For ω-regular sets of infinite words, Wagner was the first to discover remarkable relations between the two hierachies and completely describe them

⋆ Research supported by a grant from the Swiss National Science Foundation, n. 100011-116508: Project "Topological Complexity, Games, Logic and Automata". (*Received by the editors.* 10 October 2008. *Revised.* 2 March 2009. *Accepted.* 17 March 2009.)

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, pp. 46–55, 2009.
© Springer-Verlag Berlin Heidelberg 2009

[22]. The strictness of the Mostowski hierarchy for both deterministic and non-determinsitic automata on infinite trees was proved by Niwinski [18] in 1986. The hierarchy problem for the alternating automata was solved by Bradfield [5] in 1996 (ten years after their introduction by Muller and Schupp). This was when he settled the hierarchy problem for the modal μ-calculus. For what concerns the Wadge hierarchy, Murlak gave in [16] a full description of it for deterministic tree languages. In addition, he proved in [17] that for deterministic languages the Wadge hierarchy is a huge refinement of the weak index hierarchy. It is therefore not unreasonable to conjecture that the same situation holds for ω-regular tree languages too. A first step in this direction was recently given by a nice work of Arnold and Niwinski. In [4] they have shown that, if we restrict our attention to the so-called game languages, the corresponding Wadge hierarchy is strict and its strictness implies the one of the Mostowski hierarchy of alternating tree automata. Unfortunately at the moment there is no description of the Wadge hierarchy for the classe of all recognizable tree languages[1].

Standard programming logics use *forward* modalities, which express weaker preconditions. However, there is a growing interests in enriching standard languages by adding *backward* modalities, expressing stronger postconditions. This kind of modalities motivates the study of procedures for the satisfiability problem of this class of languages. This endeavor is not easy. Mainly because the interaction between these two kinds of modalities can be quite tricky. For instance, even if modal μ-calculus with backwards modalities preserves the tree-model property, it loses the finite model one.

Two-way alternating tree automata were introduced by Vardi in [20] in order to prove the decidability of the satisfiability problem for the extension of the modal μ-calculus with backward modalities. These automata were also used by Grädel and Walukiewicz to solve the same problem but this time for guarded fixed point logic in [11]. As for the standard, or "one-way" case, it is natural to find an answer to the hierarchy problems for the two-way alternating tree automata. It is not difficult to see that Arnold's nice alternative proof of the strictness of Mostowski hierarchy for alternating tree automata can also be applied to the two-way case. Nevertheless, in this paper we would like to give another – very simple – solution to this problem. The idea is – following the approach of Arnold and Niwinski – to use Wadge games and the link between the Wadge hierarchy of game languages and the Mostowski hierarchy of two-way alternating automata. More precisely, we first obtain the hierarchy result by giving a very simple game theoretical proof of the strictness of the Wadge hierarchy of game languages – a result already proved in [4] – and then we prove that the strictness of this hierarchy implies the strictness of the Mostowski hierarchy of two-way alternating automata.

As we said, the main result can be obtained by applying the techniques presented in Arnold's proof of the strictness of the fixpoint hierarchy for the modal μ-calculus. Moreover we essentially make use of another already known result: the strictness of the Wadge hierarchy of game languages [4]. Nevertheless, this

[1] Two first very partial attempts in this direction can be found in [8] and in [7].

work is essentially methodological: it aims at stressing how much the Wadge games may be used in order to obtain easy proofs in hierarchical questions related to parity automata over trees.

2 Preliminaries

2.1 Alternating Tree Automata

Let W be a non empty alphabet. A tree over Σ is a partial function $t : W^* \to \Sigma$ with a prefix closed domain. Such trees may have infinite and finite branches. We call them *conciliatory*. A tree is said to be finitely branching when W is finite and binary if $W = \{1, 2\}$. Moreover, we call a tree *full* if it has only infinite branches. The elements of t are called nodes, and the empty word ε is the root of t. Let T_Σ denote the set of full binary trees over Σ and $T_\Sigma^{\leq\omega}$ denote the set of full finitely branching trees over Σ. Given $v \in \mathsf{dom}(t)$, by $t.v$ we denote the subtree of t rooted in v. In the sequel we only consider full binary trees over Σ. Thus, unless otherwise stated, when we speak of binary (resp. finitely branching) tree, we mean *full* binary (resp. finitely branching) tree.

For every $v \in \{1, 2\}^*$, $v.i$, where $i \in \{1, 2\}$, are the successors of v. If $i = 0$, then $v.i = v$, and for every $i \in \{1, 2\}$ we have $(v.i). - 1 = v$. If $v = \varepsilon$, then $v. - 1$ is undefined. If v is a node in the binary tree t different from the root, then $v. - 1$ denotes the (unique) ancestor of v.

Throughout the paper we will work with alternating automata running on full binary trees where the acceptance condition is given by a parity condition.

More precisely, an alternating parity tree automaton $\mathcal{A}$ is a tuple $\langle \Sigma, Q, \delta, (\iota, \kappa), \Omega \rangle$ where

- Σ is a finite non empty alphabet
- Q is a finite set of states
- δ is the transition function, it associates to every pair $(q, a) \in Q \times \Sigma$ an element of the free distributive lattice generated by $Q \times \{1, 2\}$,
- (ι, κ) is a pairs of natural numbers, called the *Mostowski index* of the automata, given as follows: $\iota \in \{0, 1\}$ and $\iota < \kappa \in \omega$,
- Ω is a mapping from Q into $\{\iota, \ldots, \kappa\}$ such that for every $n \in \{\iota, \ldots, \kappa\}$ there is a $q \in Q$ such that $\Omega(q) = n$.

Without loss of generality, we can suppose that for every $(q, a) \in Q \times \Sigma$, $\delta(q, a)$ is a finite non empty disjunction of finite non empty conjunctions. Thus, δ can be seen as a function from $Q \times \Sigma$ into $\wp(\wp(Q \times \{1, 2\}))$.

Let $t \in T_\Sigma$, and let $\mathcal{A}$ an alternating parity tree automata over the alphabet Σ defined as above. We define the parity game $\mathcal{P}(\mathcal{A}, t)$ as follows:

- the set of player 0's vertices is $V_0 = Q \times \{1, 2\}^*$
- the set of player 1's vertices is $V_1 = \wp(Q \times \{1, 2\}) \times \{1, 2\}^*$
- the initial position of the game is (q_I, ε),
- there is an edge from a node $(q, v) \in V_0$ to a node $(S, w) \in V_1$ iff $S \in \delta(q, t(v))$,

- there is an edge from a node $(S, v) \in V_1$ to a node $(q, v.i) \in V_0$, with $i \in \{1, 2\}$ iff $(q, i) \in S$,
- a play $(q_1, v_1), (S_1, w_1), (q_2, v_2), (S_2, w_2), \ldots$ is winning for player 0 iff the greatest priority occurring infinitely often in the sequence $\Omega(q_1), \Omega(q_2), \ldots$ is even.

We say that the automaton $\mathcal{A}$ accepts the tree t if player 0 has a winning strategy in the parity game $\mathcal{P}(\mathcal{A}, t)$ starting from (q_I, ε). The language $L(\mathcal{A})$ accepted by the alternating parity tree automaton $\mathcal{A}$ over Σ is the subset of T_Σ of binary trees accepted by $\mathcal{A}$. Note that the arena of $\mathcal{P}(\mathcal{A}, t)$ is always a full finitely branching tree.

We consider the following partial order on indices of alternating automata:

$$(\iota, \kappa) \sqsubseteq (\iota', \kappa')$$

iff

$$\text{either } \{\iota, \ldots, \kappa\} \subseteq \{\iota', \ldots, \kappa'\} \text{ or } \{\iota + 2, \ldots, \kappa + 2\} \subseteq \{\iota', \ldots, \kappa'\}$$

The hierarchy induced by the partial order $\sqsubseteq$ on the class of alternating parity tree automata is called the *Mostowski hierarchy of alternating tree automata*.

Given the equivalence between formulae of the modal μ-calculus and alternating parity tree automata, it is not surprising that the Mostowski hierarchy corresponds to the fixpoint alternation hierarchy of the modal μ-calculus. By a result of Bradfield [5,6], we know that both hierarchies are strict.

2.2 Playing the Wadge Games

Consider the space $T_{\mathsf{B}}^{\leq\omega}$, equipped with the standard Cantor topology. Then, if $L, M \subseteq T_{\mathsf{B}}^{\leq\omega}$, we say that L is continuously reducible to M, if there exists a continuous function f such that $L = f^{-1}(M)$. We write $L \leq_w M$ iff L is continuously reducible to M. This is called the *Wadge ordering*. If $L \leq_w M$ and $M \leq_w L$, we write $L \equiv_w M$. If $L \leq_w M$ but not $M \leq_w L$, we write $L <_w M$. Thus, the Wadge hierarchy is the partial order induced by $<_w$ on the equivalence classes given by $\equiv_w$.

Let L and M be two arbitrary sets of full finitely branching trees. The Wadge game $\mathcal{W}(L, M)$ is played by two players, player I and player II. Both players build a tree, say t_{I} and t_{II}. At every round, player I plays first, both players add a level to their corresponding tree: they add children (at least one) to the terminal nodes of their corresponding tree. Player II is allowed to skip her turn, but not forever.

We say that player II wins the game iff $t_{\mathrm{I}} \in L \Leftrightarrow t_{\mathrm{II}} \in M$. This game is designed precisely in order to obtain:

Lemma 1 ([21]). *Let $L, M \subseteq T_\Sigma^{\leq\omega}$. Then $L \leq_w M$ iff Player I has a winning strategy in the game $\mathcal{W}(L, M)$.*

2.3 Game Automata and Game Languages

Consider the alphabet $\Sigma_{(\iota,\kappa)} = \{0,1\} \times \{\iota, \ldots, \kappa\}$ with $\iota \in \{0,1\}$ and $\iota \leq \kappa$. Then, to every full tree t over $\Sigma_{(\iota,\kappa)}$, we associate a parity game $\mathcal{P}(t)$ as follows: a node v in the tree is a position for player 0 iff the first component of the node is 0, and the rank of the node corresponds to its second component.

The set $W_{(\iota,\kappa)}$ corresponds to the class of trees in $T_{\Sigma_{(\iota,\kappa)}}$ for which Player 0 has a winning strategy in the corresponding parity game $\mathcal{P}(t)$. For every index (ι,κ), the set $W_{(\iota,\kappa)}$ is called the *the game language of index* (ι,κ).

The languages $W_{(\iota,\kappa)}$ are indeed ω-regular languages. More precisely, let $W_{(\iota,\kappa)}$ be the alternating tree automata $\mathcal{A}^W_{(\iota,\kappa)} = \langle \Sigma^W, Q^W, \delta^W, (\iota,\kappa), \Omega^W \rangle$ where:

- $\Sigma^W = \Sigma_{(\iota,\kappa)}$
- $Q^W = \{q_\iota, \ldots, q_\kappa\}$
- for every $q \in Q^W$ and every $v \in \mathsf{dom}(t)$

$$\delta(q,v) = \begin{cases} \{\{(q_n,1)\},\{(q_n,2)\}\} & \text{if } t(v) = (0,n), \\ \{\{(q_n,1),(q_n,2)\}\} & \text{if } t(v) = (1,n). \end{cases}$$

- $\Omega^W(q_n) = n$.

Then we have that $L(\mathcal{A}^W_{(\iota,\kappa)}) = W_{(\iota,\kappa)}$.

It is easy to see that every parity game $\mathcal{P}(\mathcal{A},t)$ can be effectively encoded from the root into a full binary tree $t^W \in W_{(\iota,\kappa)}$, with (ι,κ) being the index of the automata $\mathcal{A}$, in such a way that Player 0 has a winning strategy in $\mathcal{P}(\mathcal{A},t)$ iff she has a winning strategy in $\mathcal{P}(t^W)$. Moreover, this "encoding" is continuous. More precisely, we have the following:

Proposition 1. *Let $\mathcal{A}$ be any alternating parity tree automaton of index (ι,κ) over Σ. Then $L(\mathcal{A}) \leq_w W_{(\iota,\kappa)}$.*

Proof. Let $W'_{(\iota,\kappa)}$ be the class of all full finitely branching trees over $\Sigma_{(\iota,\kappa)}$ for which Player 0 has a winning strategy in the corresponding parity game $\mathcal{P}(t)$ Then, the winning strategy for Player II in the Wadge Game $\mathcal{W}(L(\mathcal{A}), W'_{(\iota,\kappa)})$ is given by just playing at every turn the arena of the parity game $\mathcal{P}(\mathcal{A},t)$, where t is the finite tree precisely played by Player I at that moment. By Lemma 1 we have that $L(\mathcal{A}) \leq_w W_{(\iota,\kappa)}$. But clearly $W'_{(\iota,\kappa)} \leq_w W_{(\iota,\kappa)}$. Indeed, the winning strategy for Player II in the Wadge Game $\mathcal{W}(W'_{(\iota,\kappa)}, W_{(\iota,\kappa)})$ is the following. First, given the (finite) finitely branching tree $t : \{\epsilon, 0, 1, \ldots, n\} \to \Sigma$, with $\mathsf{dom}(t) = \{\epsilon, 0, 1, \ldots, n\}$, let $f(t) : \{0,1\}^* \to \Sigma$ be the *binary encoding* of t given by $f(t)(\varepsilon) = f(t)(1^j) = t(\varepsilon)$, with $1 \leq j < n$, $f(t)(1^n) = t(n)$ and $f(t)(1^j 0) = t(j)$, for $0 \leq j < n$. We say that $\epsilon \in \mathsf{dom}(f(t))$ corresponds to $\epsilon \in \mathsf{dom}(t)$, $1^n \in \mathsf{dom}(f(t))$ corresponds to $n \in \mathsf{dom}(t)$ and that for every $1 \leq j < n$, $1^j 0 \in \mathsf{dom}(f(t))$ corresponds to $j \in \mathsf{dom}(t)$. We then define inductively for every finite finitely branching tree t and for every node $v \in \mathsf{dom}(t)$, the binary encoding $f(t)$ of t and which is the (unique) node in $\mathsf{dom}(f(t))$ corresponding to v.

Consider the following strategy for Player II in $\mathcal{W}(W'_{(\iota,\kappa)}, W_{(\iota,\kappa)})$:

1. at the first round, copy Player I's move,
2. for every round $n \geq 2$, for every terminal node v of the tree constructed by Player I after round $n-1$, if t is the tree constructed after Player I's turn at round n, then replace the terminal node corresponding to v with the binary encoding of $t.v$.

By definition of the binary encoding, this is a well-defined *winning* strategy for Player II. By transitivity of $\leq_w$ we therefore obtain that $L(\mathcal{A}) \leq_w W_{(\iota,\kappa)}$. ■

It is worth noticing that the game languages witnesses the strictness of the Mostowski hierarchy of alternating tree automata[2]. This result was recently strengthened by Arnold and Niwinski in [4]. In their paper, the authors show that the class of game languages form a hierarchy with respect to the Wadge reducibility. In this section we give a very short proof of the the same results by exploiting the game-theoretical characterization of Wadge reducibility. But before this, we explicitely establish the link between the Wadge hierarchy and the Mostowski hierarchy.

Lemma 2. *If the Wadge hierarchy of tree game languages is strict, then the Mostowski hierarchy of alternating parity tree automata is strict too.*

Proof. Suppose that the Wadge hierararchy of for tree game languages is strict, but that the Mostowski index hierarchy for alternating parity tree automata collapses. Assume it collapses to the (ι, n) level. Consider the automata recognizing the game tree language $W_{(\iota,n+1)}$. By hypothesis, there is an alternating tree automaton $\mathcal{A}$ of index (ι, n) recognizing the same language. By Proposition 1, we are able to construct a winning strategy for Player I in the Wadge Game $\mathcal{W}(W_{(\iota,n+1)}, W_{(\iota,n)})$. But this contradicts the strictness of the Wadge hierarchy of tree game languages. ■

We now start to give another proof of Arnold and Niwinski's result. First, we have the following lemma:

Lemma 3. *For every n, Player I has a winning strategy in both* $\mathcal{W}(W_{(1,n+1)}, W_{(0,n)})$ *and* $\mathcal{W}(W_{(0,n)}, W_{(1,n+1)})$.

Proof. We only prove that player I has a winning strategy in $\mathcal{W}(W_{(1,n+1)}, W_{(0,n)})$, the other case being identical. We do this by describing the winning strategy for player I in this game. As a first move, player I plays a finite binary tree over $\{(1,1)\}$. Then the strategy goes as follows:

1. if player II skips, then for every terminal node add two children labelled by $(1,1)$
2. otherwise, at every terminal node, add two dual copies of what Player II has already played as successors.

Clearly this is a winning strategy for player I in $\mathcal{W}(W_{(1,n+1)}, W_{(0,n)})$. ■

[2] Cf. [6].

Proposition 2 ([4]). *The Wadge hierarchy for tree game languages is strict.*

Proof. It is trivial to verify that, for every n, player II has a winning strategy in both $\mathcal{W}(W_{(0,n)}, W_{(0,n+1)})$ and $\mathcal{W}(W_{(0,n)}, W_{(1,n+2)})$, and dually in both $\mathcal{W}(W_{(1,n+1)}, W_{(1,n+2)})$ and $\mathcal{W}(W_{(1,n+1)}, W_{(0,n+1)})$. Therefore, if we show that player I has a winning strategy in $\mathcal{W}(W_{(0,n+1)}, W_{(0,n)})$ and $\mathcal{W}(W_{(1,n+2)}, W_{(0,n)})$ (and in the dual case) we are done. We only prove that player I has a winning strategy in $\mathcal{W}(W_{(0,n+1)}, W_{(0,n)})$, the other cases being identical. We do this by describing the winning strategy for player I in this game. As a first move, she plays an finite binary tree over the alphabet $\{(1,0)\}$. Then the strategy goes as follows:

1. if player II skips, then for every terminal node add two nodes labelled by $(1,0)$.
2. otherwise, by Lemma 3, apply the winning strategy for player I in the Wadge game $\mathcal{W}(W_{(1,n+1)}, W_{(0,n)})$.

Clearly this is a winning strategy for player I in $\mathcal{W}(W_{(0,n+1)}, W_{(0,n)})$. ■

By applying Proposition 2 to Lemma 2, we can therefore obtain, as a corollary, an alternative very easy proof of the strictness of the Mostowski hierarchy for alternating tree automata.

Corollary 1. *The Mostowski hierarchy for alternating parity tree automata is strict.*

Mutatis mutandis, the same argument yields the strictness of the Mostowski hierarchy of *weak* alternating parity automata.

2.4 Two-Way Alternating Parity Tree Automata

A two-way alternating parity automaton on binary trees is defined exactly as a (one-way) alternating atomata, except for the transition function and therefore for the parity game defining the acceptance condition. The transition function δ associates to every pair $(q,a) \in Q \times \Sigma$ an element of the free distributive lattice generated by $Q \times \{-1,0,1,2\}$. The associated paritiy game is defined as follows.

Let $t \in T_\Sigma$, and let $\mathcal{A}$ a two-way alternating parity tree automata over the alphabet Σ defined as above. Then the parity game $\mathcal{P}(\mathcal{A}, t)$ is defined by the following conditions:

- the set of player 0 vertices is $V_0 = Q \times \{-1,0,1,2\}^*$
- the set of player 1 vertices is $V_1 = \wp(Q \times \{-1,0,1,2\}) \times \{-1,0,1,2\}^*$
- the initial position of the game is (q_I, ε),
- there is an edge from a node $(q,v) \in V_0$ to a node $(S,w) \in V_1$ iff $S \in \delta(q, t(v))$,
- remember that if $i = 0$, then $v.i = v$, and that for every $i \in \{1,2\}$ we have $(v.i).-1 = v$. If $v = \varepsilon$, then $v.-1$ is undefined. Thus, there is an edge from a node $(S,v) \in V_1$ to a node $(q, v.i) \in V_0$, with $i \in \{-1,0,1,2\}$ iff (q,i) is defined and is a member of S,

– a play $(q_1, v_1), (S_1, w_1), (q_2, v_2), (S_2, w_2), \ldots$ is winning for player 0 iff the greatest priority occurring infinitely often in the sequence $\Omega(q_1), \Omega(q_2), \ldots$ is even.

As for the "one-way" case, we say that the automaton $\mathcal{A}$ accepts the tree t if player 0 has a winning strategy in the parity game $\mathcal{P}(\mathcal{A}, t)$ starting from (q_I, ε). The language $L(\mathcal{A})$ accepted by the to way alternating parity tree automaton $\mathcal{A}$ over Σ is the subset of T_Σ of binary trees accepted by $\mathcal{A}$. Note also that for two-way alternating parity automata, the arena of $\mathcal{P}(\mathcal{A}, t)$ is always a full finitely branching tree.

The hierarchy induced on the class of two-way alternating parity tree automata by the partial order $\sqsubseteq$ on indices of two-way automata is called the *Mostowski hierarchy of two-way alternating parity tree automata*.

This class of automata constitute the natural automata counterpart of the modal μ-calculus *with backward modalities*, that is the logic resulting from the addition of a universal backward modality and an existential backward modality to standard modal μ-calculus [20].

3 The Strictness of the Hierarchy

In order to solve the hierarchy problem for two-way alternating automata, we would like to apply the same argument as the one used in order to establish the strictness of the Mostowski hierarchy for one-way alternating automata over trees. Thence, as before we must establish a link between the Wadge hierarchy of game languages and the Mostowski hierarchy of two-way automata.

Remember that when constructing the parity game associated to the run of a two-way alternating automaton over a binary trees, the two new "backward" symbols -1 and 0 behave in fact as the usual "forward" symbols 1 and 2, that is to say the arena of the parity game a full finitely branching tree. Therefore, exactly as in the one-way case, we obtain the following proposition, whose proof is the same as for Proposition 1:

Proposition 3. *Let $\mathcal{A}$ be any two-way alternating parity tree automaton of index (ι, κ) over Σ. Then $L(\mathcal{A}) \leq_w W_{(\iota,\kappa)}$.*

Everything is now ready to prove the strictness of the Mostowski hierarchy for two-way alternating parity automata over trees. The remaining short step consists in establishing a relation between the Mostowski hierarchy and the Wadge hierarchy for game languages in the same way as it was done in the one-way case.

Lemma 4. *If the Wadge hierarchy of tree game languages is strict, then the Mostowski hierarchy of two-way alternating parity tree automata is strict too.*

Proof. Suppose that the Wadge hierararchy of tree game languages is strict, but that the Mostowski index hierarchy for two-way alternating parity tree automata collapses. Assume it collapses to the (ι, n) level. Consider the (two-way)

automaton recognizing the game tree language $W_{(\iota,n+1)}$. By hypothesis, there is a two-way alternating tree automaton $\mathcal{A}$ of index (ι, n) recognizing the same language. By Proposition 3, we are able to construct a winning strategy for Player I in the Wadge Game $\mathcal{W}(W_{(\iota,n+1)}, W_{(\iota,n)})$. But this contradicts the strictness of the Wadge hierarchy of tree game languages. ■

Thus, by Lemma 4 and Proposition 2, we obtain the hierarchy result for two-way alternating parity automata:

Theorem 1. *The Mostowski hierarchy of two-way alternating parity tree automata is strict*

The correspondence between formulae of the modal μ-calculus with backward modalities and two-way alternating tree automata, also gives the strictness of the fixpoint hierarchy for this extension of the modal μ-calculus.

References

1. Alberucci, L.: Strictness of the Modal μ-Calculus Hierarchy. In: Grädel, E., Thomas, W., Wilke, T. (eds.) Automata, Logics, and Infinite Games. LNCS, vol. 2500, pp. 185–201. Springer, Heidelberg (2002)
2. Arnold, A.: The μ-Calculus Alternation-Depth Hierarchy is Strict on Binary Trees. Theor. Inform. Appl. 33(4/5), 319–340 (1999)
3. Arnold, A., Niwinski, D.: Rudiments of μ-Calculus. Studies in Logic, vol. 146. Elsevier, Amsterdam (2001)
4. Arnold, A., Niwinski, D.: Continuous separation of game languages. Fundamenta Informaticae 81(1-3), 19–28 (2008)
5. Bradfield, J.: The Modal μ-Calculus Alternation Hierarchy is Strict. Theor. Comput. Sci. 195(1), 133–153 (1998)
6. Bradfield, J.: Simplifying the Modal μ-Calculus Alternation Hierarchy. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 39–49. Springer, Heidelberg (1998)
7. Duparc, J., Facchini, A.: The Topological Complexity of Models of the Modal μ-Calculus – On The Alternation Free Fragment and Beyond (submitted)
8. Duparc, J., Murlak, F.: On the Topological Complexity of Weakly Recognizable Tree Languages. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 261–273. Springer, Heidelberg (2007)
9. Duparc, J.: Wadge Hierarchy and Veblen Hierarchy Part 1: Borel Sets of Finite Rank. J. Symb. Log. 66(1), 56–86 (2001)
10. Emerson, E.A., Jutla, C.S.: Tree Automata, μ-Calculus and Determinacy (Extended Abstract). In: Sipser, M. (ed.) Proc. FOCS 1991, pp. 368–377 (1991)
11. Grädel, E., Walukiewicz, I.: Guarded fixed point logics. In: Longo, G. (ed.) Proc. LICS 1999, pp. 45–54 (1999)
12. Henzinger, T.A., Kupferman, O., Qadeer, S.: From prehistoric to postmodern symbolic model checking. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 195–206. Springer, Heidelberg (1998)
13. Kupfermann, O., Safra, S., Vardi, M.: Relating Word and Tree Automata. In: Clarke, E.M. (ed.) Proc. LICS 1996, pp. 322–332 (1996)

14. Mostowski, A.W.: Hierarchies of weak automata and weak monadic formulas. Theor. Comput. Sci. 83, 323–335 (1991)
15. Murlak, F.: On deciding topological classes of deterministic tree languages. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 428–441. Springer, Heidelberg (2005)
16. Murlak, F.: The Wadge hierarchy of deterministic tree languages. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 408–419. Springer, Heidelberg (2006)
17. Murlak, F.: Weak Index vs. Borel Rank. In: Albers, S., Weil, P., Rochange, C. (eds.) Proc. STACS 2008, pp. 573–584 (2008)
18. Niwinski, D.: On fixed point clones. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 464–473. Springer, Heidelberg (1986)
19. Niwinski, D., Walukiewicz, I.: Deciding Nondeterministic Hierarchy of Deterministic Tree Automata. Electr. Notes Theor. Comput. Sci. 123, 195–208 (2005)
20. Vardi, M.Y.: Reasoning about the Past with Two-Way Automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)
21. Wadge W.W.: Reducibility and determinateness on the Baire space. Ph.D. Thesis, Berkeley (1984)
22. Wagner, K.: Eine topologische Charakterisierung einiger Klassen regulärer Folgenmengen. Elektron. Informationsverarbeitung Kybern. 13, 473–487 (1977)
23. Wagner, K.: On ω-regular sets. Inform. and Control 43, 123–177 (1979)
24. Wilke, T.: Alternating Tree Automata, Parity Games, and Modal μ-Calculus. Bul. Belg. Math. Soc. 8(2), 359–391 (2001)

Towards Model-Checking Programs with Lists

Alain Finkel[1], Étienne Lozes[1], and Arnaud Sangnier[1,2,⋆]

[1] Laboratoire Spécification et Vérification, École Normale Supérieure de Cachan & Centre Nationale de la Recheche Scientifique (Unité Mixte de Recherche 8643), 61, avenue du Président Wilson 94235 Cachan Cedex, France
[2] Électricité de France, Recherche et Développement, 6, Quai Watier, BP 49, 78401 Chatou Cedex, France
{finkel,lozes,sangnier}@lsv.ens-cachan.fr

Abstract. We aim at checking safety and temporal properties over models representing the behavior of programs manipulating dynamic singly-linked lists. The properties we consider not only allow to perform a classical shape analysis, but we also want to check quantitative aspect on the manipulated memory heap. We first explain how a translation of programs into counter systems can be used to check safety problems and temporal properties. We then study the decidability of these two problems considering some restricted classes of programs, namely flat programs without destructive update. We obtain the following results: (1) the model-checking problem is decidable if the considered program works over acyclic lists (2) the safety problem is decidable for programs without alias test. We finally explain the limit of our decidability results, showing that relaxing one of the hypothesis leads to undecidability results.

1 Introduction

Context

The model checking techniques of infinite-state systems are now an active research area. These techniques have been successfully applied to a lot of infinite-state transition systems like counter systems, lossy channel systems, pushdown automata, timed automata, hybrid automata, and rewriting systems. For some of these models, there exist today both an impressive set of theoretical results and efficient automatic tools for verifying safety properties.

Among the different models, counter systems enjoy a central position for both theoretical results and maturity of tools like `FAST` [3], `ASPIC` [19], `LASH` [20] and `TREX` [1]. Moreover, counter systems, defined by Presburger relations, allow to model and to verify quantitative properties over various applications such as the TTP protocol [2] and broadcast protocols [16]. Different acceleration methods

⋆ This work has been partially supported by contract 4300038040 between EDF R&D/LSV and by the AVERILES project. (*Received by the editors.* 5 February 2008. *Revised.* 15 October 2008. *Accepted.* 8 February 2009.)

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, pp. 56–86, 2009.
© Springer-Verlag Berlin Heidelberg 2009

for computing the reachability relation of such counter systems have been developed. In particular, an acceleration method (recalled in [17]) has been completely implemented in the verification tool FAST [3].

In this work, we study a peculiar class of infinite state systems, called pointer systems, which allows to model the behavior of programs working over singly-linked lists. For pointer systems, qualitative properties (absence of memory violation like invalid pointer dereferencing, memory leak) are difficult to analyze, and may drastically rely on quantitative properties (lists of equal length, finitely increasing memory allocation,...). Some recent works illustrate this relation between quantitative and qualitative aspects in pointer systems for the purpose of verifying termination [7] or safety [11].

Motivations. Our attempt in this paper is to demonstrate that this relation between quantitative and qualitative properties is central in many aspects of pointer systems. Consider for example the following program that allocates a copy of a list, and then disposes both lists. In this program, the variables x,y,t and u are pointers to list elements.

```
1: void copy-and-delete(List x){   9:  y=x;
2:  List y,t,u;                    10: while (y!=NULL){
3:  y = x; t=NULL; u=t;            11:   y=y->next;
4:  while (y!=NULL){               12:   u=u->next;
5:    y=y->next;                   13:   free(x); x=y;
6:    t=malloc();                  14:   free(t);t=u;
7:    t->next=u;                   15: }
8:    u=t;}
```

Such a program cannot be analyzed correctly by tools based on purely qualitative shape analysis, whereas combining shape analysis and quantitative analysis it can be automatically established that this program is safe reminding that both lists are of equal length after the first loop and that the initial list is acyclic. It is true that this example program seems quite artificial, but it has to be seen as an abstraction of a more complex program in which all the operations which had no effect on the shape of the data structure have been removed. In fact, in this work we consider programs which can only manipulate singly-linked lists and which have to be considered as an abstraction of real programs.

We believe that situations such as the one we present with the above example are not so rare in programs manipulating pointers, and a quantitative shape analysis could be worth to be considered in several practical cases. More generally, this is certainly a promising perspective to consider systems that combine pointers and counters operations and design an analysis that relates both aspects of the computation. Counters could model parameters of list-scanning algorithms (for instance, a procedure that returns the nth element of a list), but also concurrency aspects like semaphores, thread identifier, etc. Quantitative shape analysis could be well suited for model-checking temporal properties relying on the algorithms already proposed for counter systems, such as in [13]. Previous attempts of model-checking temporal properties on pointer systems have been mostly based on either over-approximating, or non terminating

algorithms for which completeness is usually poorly studied. Exact and complete algorithms for rich classes of programs, though less efficient than others in general, can play a crucial role in practice.

In broad lines, we would like to reuse the principles of standard shape analysis to define a quantitative shape analysis, and specifically rely on the tool FAST for the quantitative aspects. FAST implements a loops acceleration that succeeds in computing the reachability set of any flat counter system; the same acceleration can be exploited to model-check CTL* properties on these systems [13]. By flat, we mean that the control flow graph does not contain nested loops. Even if the class of such systems seems to be quite restricted with respect to usual programs, several programs transformations can be tried to flatten the system and cover a much richer class of programs. A flattening procedure is implemented in FAST, and many classes of programs on which this procedure succeeds have been identified [21]. Furthermore many case studies have been verified with this approach.

Our research program, in this paper, was to try to follow the same tracks as for counter systems in the framework of pointer systems. Thus, we and others considered the class of flat pointer systems and tried to define an acceleration for them. Unfortunately, it has been shown that for flat pointer systems, and even for flat pointer systems without destructive update, the problem of reachability of a control state is undecidable. This result is somehow unsettling and we wanted to better understand for which classes of pointer systems some CTL* properties (including reachability and safety) are decidable or not.

Contribution

Our main contribution is to investigate the possibility of a quantitative model-checker for singly-linked lists manipulating programs. We define a CTL* logic which is able to express quantitative properties of the memory managed by pointer systems. For this temporal logic, we show that the model-checking problem reduces to the one for counter systems developed in [13], provided an adequate translation from pointers to counters is given. This result is important for us, since it serves as a foundation for a two-steps analysis of pointer systems that consists in translating them into counter systems and then, with the help of FAST, to verify safety properties. It remains to provide a good translation of pointer systems into counter systems. We obtain three categories of results concerning the translations:

First, we show that, from the experimental point of view, the translation defined in [5] allows us to verify all well-known singly-linked lists case studies, and also some new ones that are not immediately verifiable by the other tools and methodologies. Most of the case studies yield flat counter systems for which we know in advance that FAST will terminate.

Second, we propose a new translation of pointer systems into counter systems and we characterize several classes of pointer systems without destructive update for which our analysis terminates. The main feature of this new translation is that it preserves flatness of the systems (unlike the first translation). Using this new translation, we prove the decidability of the CTL* model-checking for flat

pointer systems without destructive update and with an acyclic initial configuration and the decidability of safety problems for flat pointer systems without destructive update and without alias test. In [11], the authors prove that some safety problems were decidable when considering flat programs without destructive update and with an initial configuration containing at most one cyclic list. We hence extend here these results to the model-checking when the initial configuration is acyclic and we propose a new class of flat programs without destructive update for which the safety problem is decidable.

Third, we explore the limits of our analysis, and of the decidability of CTL* model-checking for classes of flat pointer systems. We show that we cannot extend our analysis of flat pointer systems without destructive update and alias test if cyclic initial configurations are allowed. Conversely, we show that the safety problem becomes undecidable for flat pointer systems that keep their memory configurations acyclic, but can perform destructive update. This last point answers an open problem asked in [11].

Related Work

Many tools and techniques to check safety properties on programs manipulating pointers have been developed recently. `PALE` [26] verifies safety properties on programs annotated with loop invariants. `TVLA` [22] is another tool based on abstract interpretation, where the user has the possibility to refine the abstraction providing adequate predicates. In [27], the framework of predicate abstraction is used to manipulate boolean formulae representing the heap. In [15], the authors present a shape analysis method based on separation logics formulae to analyze programs manipulating singly-linked lists. Their method always terminates but might yield false alarms due to the over-approximation brought by the abstraction. Other methods have been proposed which use already existing model-checking techniques. For instance, in [9], the authors verify safety properties on programs manipulating singly-linked lists using abstract regular model-checking. They have extended their work to programs with more complex data structures [10]. In [8], the authors also propose a translation towards counter systems very similar to the one of [5]. In [24], the authors propose to combine shape analysis and arithmetic analysis using the same kind of techniques. None of these considers temporal model-checking or state a completeness result for these analysis.

Some efforts have already been made to introduce temporal logic for pointer verifications as the Evolution Temporal Logic [29], which used techniques similar to the one presented in `TVLA`, or the Navigation Temporal Logic presented in [14]. Recently, in [12], the authors have introduced a temporal logic based on separation logic. But, to our knowledge, these different logics do not allow to express quantitative properties of the memory heaps.

2 Preliminaries

In this section, we collect some useful notions about counter and pointer systems. We assume a set C of counter variables, and a set V of pointer variables.

2.1 Counter Systems

We recall that *Presburger arithmetic* is the first order theory of the structure $\langle \mathbb{N}, +, = \rangle$. Given a Presburger formula ϕ with free variables belonging to C and $\mathbf{a} \in \mathbb{N}^C$, we write $\mathbf{a} \models \phi$ if ϕ is true for the valuation $\mathbf{a}$. We will denote by $[\![\phi]\!]$ the set described by the formula ϕ. A *Presburger-linear function* f is a partial function which can be represented by a tuple $(A, \mathbf{b}, \phi)$ where A is a square matrix in $\mathbb{N}^{C \times C}$, $\mathbf{b} \in \mathbb{Z}^C$ and ϕ is a Presburger formula such that $f(\mathbf{a}) = A.\mathbf{a} + \mathbf{b}$ for every $\mathbf{a} \models \phi$. We denote by Σ_C the set of such functions.

Definition 1 (Counter system). *A* counter system *is a graph whose edges are labeled with Presburger linear functions, that is a tuple $CS = \langle Q, E \rangle$ where Q is a finite set of control states and $E \subseteq Q \times \Sigma_C \times Q$ is a finite set of transitions.*

To a counter system $CS = \langle Q, E \rangle$, we associate the transition system $TS(CS) = \langle Q \times \mathbb{N}^C, \rightarrow \rangle$ defined by $(q, \mathbf{a}) \rightarrow (q', \mathbf{a}')$ if there is a transition (q, f, q') in E with $f = (A, \mathbf{b}, \phi)$ such that $\mathbf{a} \models \phi$ and $\mathbf{a}' = f(\mathbf{a})$. We see here that when a transition of the counter system is labelled with a Presburger-linear function $(A, \mathbf{b}, \phi)$ the Presburger formula ϕ plays the role of a guard on the transition and the action of the transition is represented by the translation which associates to each $\mathbf{a} \in \mathbb{N}^C$ the vector $A.\mathbf{a} + \mathbf{b}$.

A *simple cycle* in a graph $G = \langle Q, E \rangle$ is a closed path (where the initial and final vertices coincide) with no repeated edge. G is said to be *flat* if every $q \in Q$ belongs to at most one cycle. Let $CS = \langle Q, E \rangle$ be a counter system. We define the monoid of CS, denoted by $\langle CS \rangle$, as the multiplicative monoid generated by the matrices present in the labels of CS. More formally, $\langle CS \rangle = \bigcup_{i \geq 0} \{A_1.A_2.\ldots.A_i \mid \forall j \in \{1, \ldots, i\}$ there exists $(q, (A_j, \mathbf{b}, \phi), q') \in E\}$. A counter system CS is said to have *the finite monoid property* if the multiplicative monoid $\langle CS \rangle$ is finite. Note that since a counter system has a finite number of control states and of transitions, one can decide whether a counter system is flat or not. This also holds for the finite monoid property, in fact using a result of [25], one can prove that the problem of knowing whether the monoid of a counter system is finite is decidable.

For the following theorem, we use the fact that the control states of a counter system $CS = \langle Q, E \rangle$ can be encoded into positive integers (ie $Q \subseteq \mathbb{N}$) and then the set of configurations is represented by $\mathbb{N}^{|C|+1}$.

Theorem 1. *[17] Let CS be a flat counter system $\langle Q, E \rangle$ with the finite monoid property and $TS(CS) = \langle \mathbb{N}^{|C|+1}, \rightarrow \rangle$ its associated transition system. Then the relation $\rightarrow^*$ is effectively Presburger definable.*

Note that as mentioned in [17], this last result extends and completes previous results on acceleration techniques for counter systems presented in [28].

Let us recall the syntax of the temporal logic for counter systems, called FOCTL*(Pr) in [13]:

$$\Phi \ ::= q \mid \psi \mid \exists y.\Phi \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathtt{X}\Phi \mid \Phi \mathtt{U} \Phi \mid \mathtt{A}\Phi$$

where q is a control state, y is a variable of a countable set VAR and ψ is a Presburger formula over $C \cup \mathrm{VAR}$.

We now give the semantics of this temporal logic. We consider a counter system $CS = \langle Q, E\rangle$ and its associated transition system $TS(CS) = \langle Q \times \mathbb{N}^C, \rightarrow\rangle$. Let π be a configuration path in $TS(CS)$. For an integer i, we denote by $\pi(i) \in Q \times \mathbb{N}^C$ the ith configuration of π, $\pi_{\leq i}$ the initial part of π up to position i and $|\pi|$ the length of π. Let ρ be a variable valuation, that is a partial map from VAR to $\mathbb{N}$. For $i \in \mathbb{N}$ and a formula Φ of FOCTL*(Pr), the satisfaction relation $\models$ is inductively defined at position i of a configuration path π as follows:

- $\pi, i \models_\rho q$ iff $\pi(i) = (q, \mathbf{a})$ for some $\mathbf{a} \in \mathbb{N}^C$;
- $\pi, i \models_\rho \psi$ iff $\pi(i) = (q, \mathbf{a})$ with $q \in Q$ and $(\mathbf{a}, \rho) \models \psi$ in Presburger arithmetic;
- $\pi, i \models_\rho \exists y.\Phi$ iff there is $m \in \mathbb{N}$ such that $\pi, i \models_{\rho[y \mapsto m]} \Phi$ where $\rho[y \mapsto m]$ denotes the variable valuation equal to ρ except that the variable y is mapped to the integer value m;
- $\pi, i \models_\rho \neg\Phi$ iff $\pi, i \not\models \Phi$;
- $\pi, i \models_\rho \Phi \wedge \Phi'$ iff $\pi, i \models \Phi$ and $\pi, i \models_\rho \Phi'$;
- $\pi, i \models_\rho \mathtt{X}\Phi$ iff $i < |\pi|$ and $\pi, i+1 \models \Phi$;
- $\pi, i \models_\rho \Phi \mathtt{U} \Phi'$ iff there is a j such that $i \leq j < |\pi|$ and $\pi, j \models \Phi'$, and for all k such that $i \leq k < j$, $\pi, k \models \Phi$;
- $\pi, i \models_\rho \mathtt{A}\Phi$ iff for all configuration path π' such that $\pi_{\leq i} = \pi'_{\leq i}$, we have $\pi', i \models_\rho \Phi$.

We denote by $CS \models \Phi$ the fact that all the configurations paths π in $TS(CS)$ verify $\pi, 0 \models \Phi$. The next result shows that Theorem 1 can be extended to temporal properties, in fact:

Theorem 2. *[13] For a flat counter system CS with the finite monoid property, and a* FOCTL*(Pr) *formula Φ, it is decidable whether $CS \models \Phi$.*

2.2 Pointer Systems

We now define the model of pointer systems that will be the core of our study. We use pointer systems to represent the behavior of programs manipulating singly-linked lists. The main idea of this model consists in representing the memory heap as a graph in which each node has at most one successor. In the sequel, we use the symbol $\perp$ to express that a function is undefined.

Definition 2 (Memory graph). *A* memory graph *is a labeled graph that can be represented by a tuple $MG = (N, \mathrm{succ}, \mathrm{loc})$ such that:*

- *N is a finite set of nodes such that $\{\mathtt{null}, \perp\} \cap N = \varnothing$;*
- succ *is a function from N to $N \cup \{\mathtt{null}, \perp\}$ called the successor function;*
- loc *is a function from V to $N \cup \{\mathtt{null}, \perp\}$ which associates a node with each pointer variable;*
- *for all nodes $n \in N$, there is $v \in V$ and $i \in \mathbb{N}$ such that $n = \mathrm{succ}^i(\mathrm{loc}(v))$.*

Note that the last condition intuitively expresses that the memory graph represents a heap without memory leak, i.e., all nodes are reachable in the graph

from a node pointed to by a variable. We impose that memory graphs do not contain memory leaks because anyway the nodes not reachable from a variable in a memory graph that could remain after executing an instruction would not have any incidence on the behavior of the program. In the sequel, we will see that if an action performed a memory leak, we consider it as a fault. Remark that we could also have a semantic with a garbage collector and in this case we would delete in the graphs the nodes that are not reachable by a pointer variable. We denote by $\mathcal{MG}_V$ ($\mathcal{MG}$ for short) the set of all memory graphs over V. We will say that two memory graphs are equal if there exists an isomorphism between their underlying graphs which respects the positions of the variables. A *cyclic list* in a memory graph is a simple cycle in the underlying graph and a memory graph is said to be *acyclic* if it does not contain any cyclic list.

We define *guarded pointer actions* as pairs denoted (g, a) where guards and actions are defined by the following grammar:

$$\begin{array}{l} g ::= \text{True} \mid \text{Isnull}(x) \mid x = y \mid \neg g \mid g \wedge g \\ a ::= x{:=}e \mid x.\text{succ}{:=}e \mid x{:=}\textbf{malloc} \mid \textbf{free}(x) \mid \textbf{skip} \\ e ::= \text{NULL} \mid x \mid x.\text{succ} \end{array}$$

where x, y are pointer variables belonging to V and x.succ represents the successor node of the cell pointed to by x. We denote by $\mathcal{G}$ the set of pointer guards, $\mathcal{A}$ the set of pointer actions and $\Sigma_P = \mathcal{G} \times \mathcal{A}$. For a memory graph $MG \in \mathcal{MG}$ and a pointer guard $g \in \mathcal{G}$, we denote by $MG \models g$ the fact that MG satisfies g. For a pointer action $a \in \mathcal{A}$, we define the partial function $[\![a]\!]_P : \mathcal{MG} \to \mathcal{MG}$ which associates to a memory graph MG the memory graph $[\![a]\!]_P(MG)$ obtained after executing the action a over MG. The function $[\![a]\!]_P$ is defined partially because there are some situations in which the action a realizes what we call a fault on a memory graph MG and in this case $[\![a]\!]_P(MG)$ is not defined. In our approach, we consider as a fault both memory violation and memory leak. Intuitively, a memory violation occurs when an action tries to move a pointer variable to the successor of the `null` node or to the successor of an undefined node; whereas a memory leak occurs when moving a variable leads to a graph where there exists a node which is not reachable from a node labeled with a variable. We also believe it would not change our results to consider garbage-collected programs (ie programs for which a memory leak is not considered as an error).

Definition 3 (Pointer system). *A* pointer system *is a graph whose edges are labeled with pairs of pointer guards and actions, that is a tuple $PS = \langle Q, E\rangle$ where Q is a finite set of control states and $E \subseteq Q \times \Sigma_P \times Q$.*

We then associate a transition system $TS(PS) = \langle Q \times \mathcal{MG}, \to\rangle$ with a pointer system $PS = \langle Q, E\rangle$. Its transition relation is defined by: $(q, MG) \to (q', MG')$ if there is a transition $q \xrightarrow{(g,a)} q'$ in E such that $MG \models g$ and $MG' = [\![a]\!]_P(MG)$. A configuration of a pointer system $PS = \langle Q, E\rangle$ is a pair (q, MG); for a set of configurations $\mathcal{C}$, we write $\texttt{Reach}(PS, \mathcal{C})$ to denote the set of configurations reachable in $TS(PS)$ from some configuration in $\mathcal{C}$.

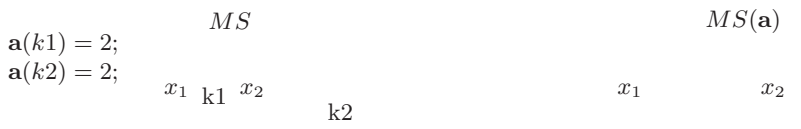


Fig. 1. A memory graph associated with a memory shape and a valuation

2.3 Representing Infinite Sets of Memory Graphs

We introduce now a symbolic representation of memory graphs. Intuitively, a memory shape is a memory graph where the intermediate nodes of an unshared list segments are skipped and replaced by a counter recording the length of this list segment.

Definition 4 (Memory shape). *[5,6] A* memory shape *is a tuple* $MS = (N, \mathrm{succ}, \mathrm{loc}, \ell)$ *such that:*

- $(N, \mathrm{succ}, \mathrm{loc})$ *is a memory graph verifying:*
  - *for all nodes* $n \in N$*, either* $\mathrm{loc}^{-1}(n) \neq \varnothing$*, or* $|\mathrm{succ}^{-1}(\{n\})| \geq 2$*;*
- $l : N \to C$ *is an injective function which associates with each node a counter variable.*

We denote by $\mathcal{MS}$ the set of memory shapes. We will write C_{MS} for the set of counters appearing in a memory shape MS (i.e., the image of the function l). To a pair $(MS, \mathbf{a}) \in \mathcal{MS} \times \mathbb{N}^C$ (such that the values of the counters in C_{MS} are strictly positive), we associate the memory graph $MS(\mathbf{a})$ obtained from the memory graph underlying MS by inserting intermediate nodes on list segments in order to have a list length equal to the value of the counter. As said in [5,6], for a fixed set V there is a finite number of memory shapes and for each memory graph MG there exists a memory shape MS and a valuation $\mathbf{a}$ such that $MG = MS(\mathbf{a})$. An example is given in Figure 1.

To represent infinite sets of memory graphs, we define what we call the *symbolic memory shapes*. A symbolic memory shape is a pair (MS, ψ) where MS is a memory shape and ψ a Presburger formula over C_{MS}. The interpretation of a symbolic memory shape is given by $[\![(MS, \psi)]\!] = \{MS(\mathbf{a}) \mid \mathbf{a} \models \psi\}$.

Definition 5 (Symbolic memory state). *A* symbolic memory state SMS *is a finite set* $\{(MS_1, \psi_1), ..., (MS_r, \psi_r)\}$ *of symbolic memory shapes* (MS_i, ψ_i)*.*

For a symbolic memory state $SMS = \{(MS_1, \psi_1), ..., (MS_r, \psi_r)\}$ the concrete interpretation is given by $[\![SMS]\!] = \bigcup_{i \in \{1,...,r\}} [\![(MS_i, \psi_i)]\!]$. We denote by $\mathcal{SMS}$ the set of symbolic memory states. In [6], the authors have shown that symbolic memory states enjoy good properties, in particular that it is possible to define complement and intersection operators for this representation.

3 Model Checking Issues

In this section, we define the safety and temporal properties we consider in this work. We first formally define the model-checking problems. Then, we recall a

method, presented in [5], to analyze pointer systems translating them into a bisimilar counter system.

3.1 Model-Checking Programs with Pointers

We define now the notions of safety and model-checking when the considered model is a pointer system. A symbolic configuration of a pointer system $PS = \langle Q, E\rangle$ is a finite set of pairs (q, SMS), for $q \in Q$ and SMS a symbolic memory state.

Definition 6 (Safety in pointer systems). *The safety problem for a pointer system is defined by:*

- ***Input:*** *A pointer system* $PS = \langle Q, E\rangle$*, an initial symbolic configuration* $\texttt{INIT}$ *and a "bad" symbolic configuration* $\texttt{BAD}$*;*
- ***Output:*** $\texttt{Reach}(PS, [\![\texttt{INIT}]\!]) \cap [\![\texttt{BAD}]\!] \stackrel{?}{=} \varnothing$.

Note that the problem of deciding whether a given pointer system may reach a given control state, may performs a memory violation, or a memory leak, reduce to this generic safety problem.

We now consider a temporal logic for pointer systems based on the quantitative shape logic of the previous section:

$$\Phi \ ::= \ q \mid SMS \mid \neg\Phi \mid \Phi \wedge \Phi \mid \texttt{X}\Phi \mid \Phi\texttt{U}\Phi \mid \texttt{A}\Phi$$

where q is a control state and SMS is a symbolic memory state. We denote by $\mathrm{CTL}^*_{\mathrm{mem}}$ this logic. We give the semantics of this logic, defined by a relation $\pi, i \models \Phi$ between traces π of a pointer system and a formula Φ of $\mathrm{CTL}^*_{\mathrm{mem}}$. We consider a pointer system $PS = \langle Q, E\rangle$ and its associated transition system $TS(PS) = \langle Q \times \mathcal{MG}, \rightarrow\rangle$. Let π be a configuration path in $TS(PS)$. For an integer i, we denote by $\pi(i) \in Q \times \mathcal{MG}$ the ith configuration of π, $\pi_{\leq i}$ the initial part of π up to position i and $|\pi|$ the length of π. For $i \in \mathbb{N}$ and a formula Φ of $\mathrm{CTL}^*_{\mathrm{mem}}$, the satisfaction relation $\models$ is inductively defined at position i of a configuration path π as follows:

- $\pi, i \models q$ iff $\pi(i) = (q, MG)$ for some $MG \in \mathcal{MG}$;
- $\pi, i \models SMS$ iff $\pi(i) = (q, MG)$ with $q \in Q$ and $MG \in [\![SMS]\!]$;
- $\pi, i \models \neg\Phi$ iff $\pi, i \not\models \Phi$;
- $\pi, i \models \Phi \wedge \Phi'$ iff $\pi, i \models \Phi$ and $\pi, i \models \Phi'$;
- $\pi, i \models \texttt{X}\Phi$ iff $i < |\pi|$ and $\pi, i+1 \models \Phi$;
- $\pi, i \models \Phi\texttt{U}\Phi'$ iff there is a j such that $i \leq j < |\pi|$ and $\pi, j \models \Phi'$, and for all k such that $i \leq k < j$, $\pi, k \models \Phi$;
- $\pi, i \models \texttt{A}\Phi$ iff for all configuration path π' such that $\pi_{\leq i} = \pi'_{\leq i}$, we have $\pi', i \models \Phi$.

We are then interested in solving the model-checking problem of formulae of $\mathrm{CTL}^*_{\mathrm{mem}}$ for pointer systems. We define here this problem.

Definition 7 (Model-checking). *The model-checking problem for pointer systems is defined by:*

- ***Input:*** *A pointer system $PS = \langle Q, E\rangle$, an initial symbolic configuration* `INIT` *and a formula Φ of* $\mathrm{CTL}^{*}_{\mathrm{mem}}$*;*
- ***Output:*** *Do we have $\pi, 0 \models \Phi$ for all traces π of $TS(PS)$ such that $\pi(0) \in [\![\texttt{INIT}]\!]$?*

3.2 From Pointer Systems to Counter Systems

Let $PS = \langle Q_P, E_P\rangle$ be a pointer system. In [5], we give an effective algorithm to build a counter system $CS(PS)$ which is bisimilar to PS. Before to present this translation, let us recall the definition of a bisimulation:

Definition 8 (Bisimulation). *Given two transition systems $TS_1 = (S_1, \rightarrow_1)$ and $TS_2 = (S_2, \rightarrow_2)$, a relation $R \subseteq S_1 \times S_2$ is a bisimulation if and only if, for all $(s_1, s_2) \in R$:*

1. *If there is an $s'_1 \in S_1$ such that $s_1 \rightarrow_1 s'_1$ then there is an $s'_2 \in S_2$ such that $s_2 \rightarrow_2 s'_2$ and $(s'_1, s'_2) \in R$;*
2. *If there is an $s'_2 \in S_2$ such that $s_2 \rightarrow_2 s'_2$ then there is an $s'_1 \in S_1$ such that $s_1 \rightarrow_1 s'_1$ and $(s'_1, s'_2) \in R$.*

The translation presented in [5] used the memory shapes and is based on the following principle: given a memory shape $MS \in \mathcal{MS}$ and a pointer action $a \in \mathcal{A}$, it is possible to define a set $\mathrm{POST}(a, MS)$ of pairs $((A, \mathbf{b}, \phi), MS')$ where $(A, \mathbf{b}, \phi)$ represents a Presburger-linear function and MS' a memory shape such that, for all $((A, \mathbf{b}, \phi), MS') \in \mathrm{POST}(a, MS)$, and for all $\mathbf{a}, \mathbf{a}' \in \mathbb{N}^C$, we have:

$$MS'(\mathbf{a}') = [\![a]\!]_p(MS(\mathbf{a})) \text{ if and only if } \mathbf{a} \models \phi \text{ and } \mathbf{a}' = A.\mathbf{a} + \mathbf{b}$$

The counter system $CS(PS)$ is then equal to $\langle Q_C, E_C\rangle$ where:

- $Q_C = Q_P \times \mathcal{MS}$,
- E_C is the transition relation defined as follows $((q, MS), (A, \mathbf{b}, \phi), (q', MS')) \in E_C$ if and only if there exists a transition $(q, (g, a), q') \in E_P$ such that $MS \models g$ and $((A, \mathbf{b}, \phi), MS') \in \mathrm{POST}(a, MS)$.

In this definition, we say that a memory shape $MS = (N, \mathrm{succ}, \mathrm{loc}, \ell)$ satisfies a guard g if its underlying memory graph $(N, \mathrm{succ}, \mathrm{loc})$ satisfies g. Note that since the sets $\mathcal{MS}$ and Q_P are finite, the counter system $CS(PS)$ can effectively be built. Furthermore, using the property of the function POST, we deduce that the relation:

$$B = \{((q, MG), ((q', MS), \mathbf{a})) \in (Q_P \times \mathcal{MG}) \times (Q_C \times \mathbb{N}^C) \mid q = q' \wedge MG = MS(\mathbf{a})\}$$

is a bisimulation between the transition systems of PS and of $CS(PS)$. We can hence use the counter system $CS(PS)$ to analyze the pointer system PS.

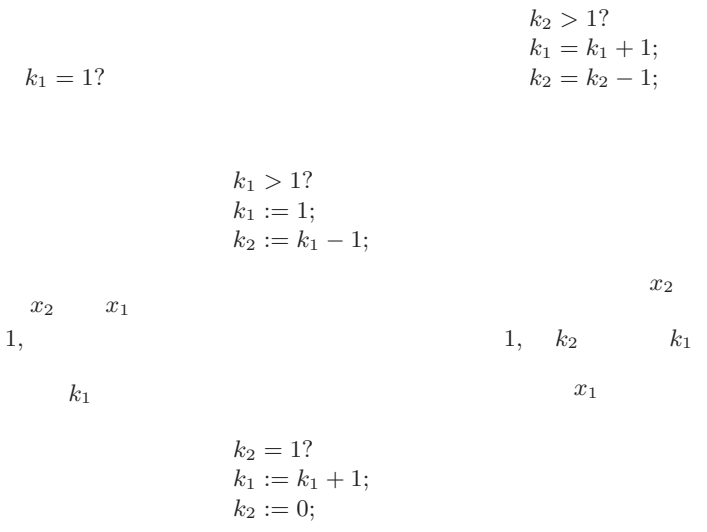


Fig. 2. An example of a counter system $CS(PS)$ obtained from a pointer system PS

Figure 2 gives an example of a connected component of the counter system $CS(PS)$ obtained from the pointer system $PS = \langle\{1\}, \{(1, x_1.\text{succ} = x_1, 1)\}\rangle$ with $V = \{x_1, x_2\}$. We see with this example that this translation does not preserve the flatness of systems. Table 1 gives a list of programs working over acyclic initial configurations (except `parse-cyclic-acyclic`) which we have translated into counter systems and successfully analyzed. Some of these programs are described in Appendix A. Most of them are classical programs, except the program `copy-and-delete` presented in the introduction, the program `split` which divides a single list in two lists and is safe only if the input list has an even length, and the program `parse-cyclic-acyclic` which parses a cyclic and an acyclic list in the same time. We remark that in most of the cases the corresponding counter system is flat (and has always, by definition of the translation, the finite monoid property) which corresponds to the hypothesis of the theorems 1 and 2. When the system is not flat, it can still be analyzed sometimes, as for the `merge` program, using a flattening procedure implemented in the tool `FAST`. But in other cases it might not be fully verified, as for the program `parse-cyclic-acyclic`.

Since the memory shapes appear in the control states of $CS(PS)$, it is possible to translate any temporal logic formula over the symbolic configurations of PS into a temporal logic formula over the configurations of $CS(PS)$. Hence:

Theorem 3. *Let Φ_P be a* $\text{CTL}^*_{\text{mem}}$ *formula. Then there effectively exists a formula Φ_C of* $\text{FOCTL}^*(\text{Pr})$ *such that for all pointer systems PS:*

$$PS \models \Phi_P \text{ if and only if } CS(PS) \models \Phi_C$$

Furthermore, the counter system $CS(PS)$ has the finite monoid property. In fact, in [5], we can see that all the matrices labeling the transitions of $CS(PS)$

Table 1. Examples of programs analyzed by FAST

Program	Is PS flat ?	Is CS(PS) flat ?	Analyzed with FAST ?
reverse	YES	YES	YES
delete	YES	YES	YES
deleteALL	YES	YES	YES
merge	NO	NO	YES
copy-and-delete	YES	YES	YES
split	YES	YES	YES
delete(n)	YES	YES	YES
parse-cyclic-acyclic	YES	NO	NO

are composed of columns in which all the elements are equal to 0 except one which is equal to 1. Using Theorem 2 and the previous result, we hence have the following result.

Corollary 1. *Let PS be a pointer system such that $CS(PS)$ is flat. Then the model-checking is decidable for PS.*

4 Decidability Results for Programs without Destructive Update

After having seen a general method to analyze pointer systems translating them into a counter system, we aim in this section at finding some classes of pointer systems for which the safety and the model-checking problems are decidable. We know that these problems are undecidable for pointer systems in general because it is easy to simulate a Minsky machine with a pointer system. Since there is no obvious notion equivalent to finite monoid property for pointer systems, and since, as we will see, flatness is in general not sufficient to decide reachability properties, we define other restrictions on pointer systems. We will say that a pointer system $PS = \langle Q, E \rangle$ is:

- *without destructive update* if the actions in E are all of the form $x := e$ with $x \in V$;
- *without alias test* if the guards in E do not contain any test like $x = y$ with $x, y \in V$.

In [11], the authors have studied whether flat programs without destructive update, working on a given special shape, could fail to satisfy some assert intructions inserted in the code. This problem reduces to a particular case of safety problem, which is the reachability of a control state. They proved, that this problem is undecidable for a flat pointer system without destructive update, if any initial symbolic configuration is considered, but is decidable for initial symbolic configurations with at most one cyclic list. Hence this result shows that even when we take strong restrictions such as flat pointer systems without destructive update, the problem of reachability of a control state is undecidable.

The work we present here extends and completes the results presented in [11]. In this section we establish the decidability of the safety and model-checking problems for two restricted classes of flat pointer systems without destructive update. It is true that these classes are very restricted, but we should see in the next section that it is hard to obtain decidability results without considering such restrictions.

Theorem 4 (Decidability of safety). *For flat pointer systems without destructive update and without alias test, the safety problem is decidable.*

With this theorem, we hence propose a new class of flat pointer systems without destructive update for which the safety problem is decidable, and for this class there is no need to put restriction on the initial configuration.

With the second theorem, we extend to general temporal properties the results expressed in [11] in the case of programs with an acyclic initial configuration.

Theorem 5 (Decidability of model-checking). *For flat pointer systems without destructive update and with an initial acyclic symbolic configuration, the model-checking is decidable.*

The proofs of these theorems rely both on a translation that maps a pointer system without destructive update to a bisimilar counter system. Moreover, this translation, unlike the translation presented previously, preserves the flatness of the systems, and relies on the notion of a new symbolic representation for memory graphs, called roots memory shapes. We first introduce this notion, then present the translation and sketch the proofs of the two theorems.

4.1 Roots Memory Shape

The basic ingredient of the new translation is the notion of roots memory shapes, that are memory shapes in which all the variables appear only on root nodes or on cyclic lists. They can be formally defined as follows:

Definition 9 (Roots memory shape). *A roots memory shape is a memory shape* $RMS = (N, \mathrm{succ}, \mathrm{loc}, \ell)$ *such that for all* $n \in N$*:*

- *either* $\mathrm{loc}^{-1}(\{n\}) \neq \varnothing$ *and* $\mathrm{succ}^{-1}(\{n\}) = \varnothing$ *(root node);*
- *or* $\mathrm{loc}^{-1}(\{n\}) = \varnothing$ *and* $|\mathrm{succ}^{-1}(\{n\})| \geq 2$ *(shared node);*
- *or* $\mathrm{loc}^{-1}(\{n\}) \neq \varnothing$ *and* $\mathrm{succ}^{-1}(\{n\}) = \{n\}$ *(unshared node on cyclic list).*

Note that we do not really need to impose $\mathrm{loc}^{-1}(\{n\}) = \varnothing$ in the second condition, but we do it to simplify the definition of the translation we give later. We suppose that the set of variables is $V = \{x_1, ..., x_m\}$ and we associate with each pointer variable x_i a counter c_i. We define $C_V = \{c_1, ..., c_m\}$ and suppose that $C_V \subset C$. By giving a value for the counters labeling the nodes, and also a value for the counters c_i associated to pointer variables (which was not the case with memory shapes), we can obtain a memory graph from a roots memory shape. Given a roots memory shape RMS and a valuation $\mathbf{a} \in \mathbb{N}^C$, we define the memory graph $RMS(\mathbf{a})$ as follows (cf. Figure 3 for an example): first, we consider the memory graph $MG = (N, \mathrm{succ}, \mathrm{loc})$ obtained from

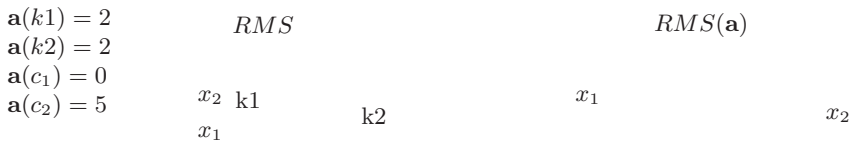


Fig. 3. A memory graph associated with a roots memory shape and a valuation

the interpretation of RMS as a memory shape; then we define $RMS(\mathbf{a})$ to be the memory graph $MG' = (N', \text{succ}', \text{loc}')$ where $N' = N$, $\text{succ}' = \text{succ}$ and $\text{loc}'(x_i) = \text{succ}^{\mathbf{a}(c_i)}(\text{loc}(x_i))$ for all variables x_i (where succ^j represents the jth successor). Note that some valuations may not be admissible for this definition, as $\text{loc}'(x_i)$ may be undefined, or the condition on the absence of memory leaks in the definition of memory graphs may not be satisfied (if all variables located on the same node in RMS all have strictly positive values for their counters). We denote by $\mathcal{RMS}$ the set of roots memory shapes. Since for a finite number of variables the number of memory shapes is finite, we deduce that $\mathcal{RMS}$ is also a finite set. Before to give the definition of the new translation, we define here some useful notions on memory graphs. We consider a memory graph $MG = (N, \text{succ}, \text{loc})$.

A node $n' \in N \cup \{\texttt{null}, \bot\}$ is said to be *reachable* in MG from an other node $n \in N$ if there exists a path in the graph from n to n', i.e., a finite ordered set of nodes $\{n_1, ..., n_r\} \subseteq N \cup \{\texttt{null}, \bot\}$ such that $n_1 = n$, $n_r = n'$ and $\forall i \in \{1, ..., r-1\}, \text{succ}(n_i) = n_{i+1}$. For a node $n \in N$, we denote by $\texttt{List}(MG, n)$ the set $\{n' \in N \cup \{\texttt{null}, \bot\} \mid n' \text{ is reachable from } n \text{ in } MG\}$.

We introduce then the notion of shared nodes. Given two nodes $n, n' \in N$, we define the set $\texttt{Shared}(MG, n, n') = \texttt{List}(MG, n) \cap \texttt{List}(MG, n')$, which represents the nodes that are shared by the list beginning at the node n and the one beginning at the node n'.

We propose also a notation to represent the set of nodes which lay between two nodes. Let $n, n' \in N$, we define $\texttt{Between}(MG, n, n')$ such that:

- if $n' \notin \texttt{List}(MG, n)$, $\texttt{Between}(MG, n, n') = \varnothing$;
- else if $n' = n$, $\texttt{Between}(MG, n, n') = \varnothing$;
- else ($n' \in \texttt{List}(MG, n)$ and $n' \neq n$) $\texttt{Between}(MG, n, n')$ is the set of nodes $\{n_1, ..., n_r\}$ such that $n_1 = \text{succ}(n)$, $n' = \text{succ}(n_r)$, $\forall i \in \{1, ..., r-1\}$, $\text{succ}(n_i) = n_{i+1}$ and $\forall i \in \{1, ..., r\}$, $n_i \notin \{n, n'\}$.

Furthermore, we will say that a variable $v \in V$ is *single* in MG if:

- $\text{loc}^{-1}(\text{loc}(v)) = \{v\}$ (i.e., v is the only variable on the node $\text{loc}(v)$).

We recall that a node $n \in N$ is called a *root node* if $\text{succ}^{-1}(\{n\}) = \varnothing$.

Table 2. Computation of $\mathrm{TEST}_2(g, RMS)$

Hypothesis	$\mathrm{TEST}_2(g, RMS)$
$g := \mathrm{True}$	True
$g := \neg g'$	$\neg\mathrm{TEST}_2(g, RMS)$
$g := g' \wedge g''$	$\mathrm{TEST}_2(g, RMS) \wedge \mathrm{TEST}_2(g', RMS)$
$g := \mathrm{Isnull}(x_i)$ and there is an n such that $n \in \mathtt{List}(RMS, x_j)$ and n is on a cyclic list	False
$g := \mathrm{Isnull}(x_i)$ and $\mathtt{null} \notin \mathtt{List}(RMS, \mathrm{loc}(x_i))$ and there is no n such that $n \in \mathtt{List}(RMS, x_j)$ and n is on a cyclic list	False
$g := \mathrm{Isnull}(x_i)$ and $\mathtt{null} \in \mathtt{List}(RMS, \mathrm{loc}(x_i))$ and there is no n such that $n \in \mathtt{List}(RMS, x_j)$ and n is on a cyclic list	$c_i = \Sigma_{n \in \mathtt{List}(RMS, \mathrm{loc}(x_i)) \cap N} \ell(n)$
$g := x_i = x_j$ and RMS is acyclic and $\mathtt{Shared}(RMS, \mathrm{loc}(x_i), \mathrm{loc}(x_j)) = \varnothing$	False
$g := x_i = x_j$ and RMS is acyclic and $\mathtt{Shared}(RMS, \mathrm{loc}(x_i), \mathrm{loc}(x_j)) \neq \varnothing$	$c_i > \Sigma_{n \in \mathtt{List}(RMS, \mathrm{loc}(x_i)) \setminus \mathtt{Shared}(RMS, \mathrm{loc}(x_i), \mathrm{loc}(x_j))} \ell(n) \wedge$ $c_j > \Sigma_{n' \in \mathtt{List}(RMS, \mathrm{loc}(x_j)) \setminus \mathtt{Shared}(RMS, \mathrm{loc}(x_i), \mathrm{loc}(x_j))} \ell(n') \wedge$ $c_i - \Sigma_{n \in \mathtt{List}(RMS, \mathrm{loc}(x_i)) \setminus \mathtt{Shared}(RMS, \mathrm{loc}(x_i), \mathrm{loc}(x_j))} \ell(n) =$ $c_j - \Sigma_{n' \in \mathtt{List}(RMS, \mathrm{loc}(x_j)) \setminus \mathtt{Shared}(RMS, \mathrm{loc}(x_i), \mathrm{loc}(x_j))} \ell(n')$

4.2 From Pointers to Counters

We present now how to define a counter system that faithfully represents a pointer system without destructive update using a translation which preserves the flatness of the system. In this manner, we define two functions TEST_2 and POST_2, we will then use to build the counter system.

The partial function TEST_2 takes as argument a pointer guard $g \in \mathcal{G}$ and a roots memory shape RMS and returns a Presburger-formula ϕ. This function is defined such that the following property is verified: if $\phi = \mathrm{TEST}_2(g, RMS)$, for all admissible $\mathbf{a} \in \mathbb{N}^C$ (according to RMS), we have:

$$RMS(\mathbf{a}) \models g \text{ if and only if } \mathbf{a} \models \phi$$

Remark that if RMS is a roots memory shape with a cyclic list and if g is a pointer guard of the form $x_i = x_j$ then the arithmetic formula given by the function TEST_2 should use propositions of the form y is divided by x which do not belong to the Presburger arithmetic. In fact, if x_i and x_j are two pointer variables pointing in RMS to the unique node of a cyclic list whose edge is labeled with the counter k, testing if $x_i = x_j$ could be done using the formula $(c_i \geq c_j \Rightarrow k|c_i - c_j) \vee (c_j \geq c_i \Rightarrow k|c_j - c_i)$. To avoid this situation, we restrict the definition domain of TEST_2 such that $\mathrm{dom}(\mathrm{TEST}_2) = \{(g, RMS) \in \mathcal{G} \times \mathcal{RMS} \mid g$ does not use alias test or RMS is acyclic$\}$. The table 2 gives the formal definition of the function TEST_2.

As it has been done for the first introduced translation, we now define the partial function POST_2 which takes as argument a pointer action $a \in \mathcal{A}$ that is not a destructive update and a root memory shape RMS and returns a pair $((A, \mathbf{b}, \phi), RMS')$ such that the following property is satisfied: if $((A, \mathbf{b}, \phi), RMS') = \mathrm{POST}_2(a, RMS)$, for all admissible $\mathbf{a}$ and $\mathbf{a}'$ in $\mathbb{N}^C$ (according to RMS and RMS'), we have:

$$RMS'(\mathbf{a}) = [\![a]\!]_P(RMS(\mathbf{a}')) \text{ if and only if } \mathbf{a} \models \phi \text{ and } \mathbf{a}' = A.\mathbf{a} + \mathbf{b}$$

Note that whereas the function POST was returning a set of pairs $((A, \mathbf{b}, \phi), MS)$ the function POST_2 returns an unique pair. This feature allows us to define a translation which preserves the flatness of systems. The table 3 gives the definition of the function POST_2 in the case of actions of the form $x_i :=$ NULL.

Table 3. Computation of $((A, \mathbf{b}, \phi), RMS') = \mathrm{POST}_2(a, RMS)$ for the action a of the form $x_i :=$ NULL

Hypothesis	$(A, \mathbf{b})$	ϕ	RMS'
x_i is single in RMS and $\mathrm{loc}(x_i) \notin \{\mathtt{null}, \perp\}$	$(\mathrm{Id}, \overrightarrow{0})$	False	RMS
$\mathrm{loc}(x_i) \in \{\mathtt{null}, \perp\}$	$(\mathrm{Id}, \overrightarrow{0})$	True	$\mathrm{loc}'(x_i) = \mathtt{null}$
x_i is not single in RMS and $\mathrm{loc}(x_i)$ belongs to a cyclic list	$c_i := 0$	True	$\mathrm{loc}'(x_i) = \mathtt{null}$
x_i is not single in RMS and $\mathrm{loc}(x_i)$ does not belong to a cyclic list	$c_i := 0$	$\bigvee_{x_j \in \mathrm{loc}^{-1}(\{\mathrm{loc}(x_i)\}) \setminus \{x_i\}} c_j = 0$	$\mathrm{loc}'(x_i) = \mathtt{null}$

The definition for the others actions is given in Appendix B. Note that sometimes the linear function (A, b) is denoted for instance $c_i := c_j$ which means that the function changes only the value of the counter c_i giving it the value of c_j. In these tables, in the column describing RMS', it is sometimes written for instance $\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$, it means that RMS' is obtained by moving in RMS the variable x_i to the node where x_j points on.

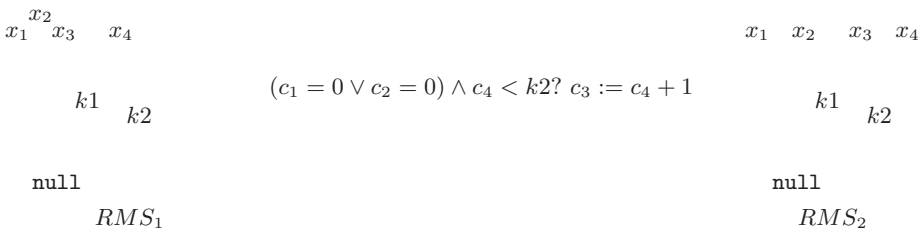


Fig. 4. Effect of the action $x_3 := x_4.\mathrm{succ}$ over RMS_1

Figure 4 presents an example of the results of the computation of POST_2. Intuitively, the conditions ensuring some action will not yield memory fault can be defined with a guard on the counter variables, just as in Figure 4 where the guard $(c_1 = 0 \vee c_2 = 0)$ ensures the absence of memory leak and the guard $(c_4 < k2)$ the absence of memory violation. A pointer action $x_i := x_j$ will correspond to moving x_i to the location of x_j and doing the linear transformation $c_i := c_j$ on counters. In the case of $x_i := x_j.\mathrm{succ}$, it is the same, we move x_i to the location of x_j and we update the counter with the operation $c_i := c_j + 1$. Finally, for the action $x_i := \mathrm{NULL}$, we do the following operations, we move x_i to `null` and we set the counter c_i to 0.

Using TEST_2 and POST_2, we can associate with a pointer system $PS = \langle Q_P, E_P \rangle$ the counter system $CS_2(PS) = \langle Q_C, E_C \rangle$ where:

- $Q_C = Q_P \times \mathcal{RMS}$
- E_C is the transition relation defined by $((q, RMS), (A, b, \phi), (q', RMS')) \in E_C$ if and only if there exists a transition $(q, (g, a), q') \in E_P$ and two Presburger formulae ϕ_1 and ϕ_2 such that $\phi_1 = \mathrm{TEST}_2(g, RMS)$ and $((A, \mathbf{b}, \phi_2), RMS') = \mathrm{POST}_2(a, RMS)$ and $\phi = \phi_1 \wedge \phi_2$.

Note that since E_P and $Q_P \times \mathcal{RMS}$ are finite, we can effectively build the counter system $CS_2(PS)$.

We will show how PS and $CS_2(PS)$ are related. Let us consider the relation R_T between the configurations of the pointer system PS and the ones of its associated counter system $CS_2(PS)$ defined by:

$$R_T = \{ \big((q, MG)\ ,\ ((q, RMS), \mathbf{a})\big) \mid MG = RMS(\mathbf{a})\}.$$

and the relation R_T^{ac} which is the restriction of R_T to acyclic memory graphs:

$$R_T^{ac} = \{ \big((q, MG)\ ,\ ((q, RMS), \mathbf{a})\big) \mid MG = RMS(\mathbf{a}) \text{ and } MG \text{ is acyclic}\}.$$

Proposition 1. *For any pointer system without destructive update PS, $CS_2(PS)$ enjoys the following properties:*

1. *$CS_2(PS)$ has the finite monoid property.*
2. *If PS is flat then $CS_2(PS)$ is flat.*
3. *R_T^{ac} is a bisimulation.*
4. *If PS is without alias test, R_T is a bisimulation.*

Idea of the proof. $CS_2(PS)$ has the finite monoid property because all the matrices given by the function POST_2 are composed of lines in which all the elements are equal to 0 except one which is equal to 1, and the multiplicative monoid of such a set of matrices is finite. The other points of this proposition are direct consequences of the way we build $CS_2(PS)$ and of the properties of the functions TEST_2 and POST_2. ■

Properties 1 and 2 ensure that we will be able to use theorems 1 and 2 (and also the tool `FAST`), and properties 3 and 4 are essential to relate counter properties to pointer properties.

4.3 Translating the Symbolic Configurations

To conclude the proofs of theorems 4 and 5, we have to extend the translation to symbolic configurations and temporal formulae. We shall define $T(\texttt{INIT})$, $T(\texttt{BAD})$ as two symbolic configurations of $CS_2(PS)$ that correspond to `INIT` and `BAD` in PS, and we must moreover define $T(\Phi) \in \mathrm{FOCTL}^*(\mathrm{Pr})$ that corresponds to $\Phi \in \mathrm{CTL}^*_{\mathrm{mem}}$. The key of this translation is to find an effective symbolic representation $T(q, (MS, \psi))$ for the set of (counter systems) configurations that are bisimilar to (pointer systems) configurations in $[\![(q, (MS, \psi))]\!]$. That is, for all roots memory shape, we should represent the set of counters values:

$$T_{RMS}(MS, \psi) \;=\; \{\mathbf{a} \mid \; RMS(\mathbf{a}) \in [\![MS, \psi]\!]\}.$$

This set is not Presburger definable in general, due to cyclic lists, but, as we will see with the next proposition, it is definable in the logic $\mathcal{L}^{\exists}_{|} = \langle \mathbb{N}, +, |, = \rangle^{\exists}$ of the existentially quantified formulae of the Presburger arithmetic with divisibilty. An essential result for our proofs is that the satisfiability problem for this logic is decidable [23].

Proposition 2. *$T_{RMS}(MS, \psi)$ is definable in $\mathcal{L}^{\exists}_{|}$. Moreover, if MS is acyclic, then it is definable in Presburger arithmetic.*

Before to give the proof of this proposition, we introduce some preliminary notions. We will say that a memory shape MS is *compatible* with a roots memory shape RMS (denoted $MS \sqsubset RMS$) if and only if $[\![(MS, \mathrm{True})]\!] \cap [\![(RMS, \mathrm{True})]\!] \neq \varnothing$. Intuitively a memory shape MS is compatible with a roots memory shape RMS if it is possible to obtain a graph isomorphic to RMS from MS moving the pointer variables to a root node or to a node in a cyclic list they are connected to in MS. To be more formal, we introduce the notion of *compatibility function*.

Let $MS = (N, \text{succ}, \text{loc})$ be a memory shape and $RMS = (N', \text{succ}', \text{loc}', \ell')$. We denote by N_R (resp. N'_R) the set of root nodes in MS (resp. in RMS), N_2 (resp. N'_2) the set of nodes with at least two predecessors in MS (resp. in RMS) and N_C (resp. N'_C) the set of nodes belonging to a cyclic list not reachable from a root node in MS (resp. in RMS). We say that a function $g : N' \to N$ is a *compatibility function* between MS and RMS if g is a total injective function such that:

- $g(N'_R) = N_R$, $g(N'_2) = N_2$ and $g(N'_C) \subseteq N_C$;
- for all nodes $n, n' \in N'$, $n' \in \texttt{List}(RMS, n)$ if and only if $g(n') \in \texttt{List}(MS, g(n))$;
- for all nodes $n \in N'$, $\texttt{null} \in \texttt{List}(RMS, n)$ if and only if $\texttt{null} \in \texttt{List}(MS, g(n))$;
- for all nodes $n \in N'$, $\bot \in \texttt{List}(RMS, n)$ if and only if $\bot \in \texttt{List}(MS, g(n))$;
- for all variables $v \in V$, $\text{loc}(v) \in \texttt{List}(MS, g(\text{loc}'(v))$.

We can then deduce the following lemma.

Lemma 1. *Let MS be a memory shape and RMS be a roots memory shape. $MS \sqsubset RMS$ if and only if there exists a compatibility function between MS and RMS.*

We now give the proof of proposition 2.

Proof. In this proof we associate the set $T_{RMS}(MS, \psi)$ with the arithmetic formula that characterizes it. Let (MS, ψ) be a symbolic memory shape and RMS a roots memory shape. We suppose $C_{MS} = \{k_1, ..., k_m\}$, $MS = (N, \text{succ}, \text{loc}, \ell)$ and $RMS = (N', \text{succ}', \text{loc}', \ell')$. We build a logic formula $T_{RMS}(MS, \psi)$ over $C_{RMS} \cup C_V$ as follows:

- If $MS \not\sqsubset RMS$, $T_{RMS}(MS, \psi) = \text{False}$;
- Otherwise Let g be a function of compatibility between MS and RMS;
  1. Rename in ψ and in MS all the counters k_i with $\bar{k}_i$ Such that for all $i \in \{1, ..., m\}$, $\bar{k}_i \notin C$ to obtain a formula $\bar{\psi}$, and we denote by $\bar{l}$ the function which associates to each node $n \in N$ the counter $\bar{k}_i$ such that $\ell(n) = k_i$.
  2. For each node n in N', we define the formula ϕ_n to ensure that the length on the graphs correspond:

$$\phi_n := l'(n) = \bar{l}(g(n)) + \textstyle\sum_{n' \in \texttt{Between}(MS, g(n), g(\text{succ}'(n)))} \bar{l}(n')$$

  3. Let $N_{\text{Cl}} \subseteq N$ (resp. $N'_{\text{Cl}} \subseteq N'$) the set of nodes of MS (resp. of RMS) which belong to a cyclic list. For each variable $x_i \in V$, we define a formula ϕ_i to ensure the pointer variables are located at the same position.
     - If $\text{loc}(x_i) \in \{\texttt{null}, \bot\}$, then

$$\phi_i := c_i = 0$$

- If $\mathrm{loc}(x_i) \notin N_{\mathrm{Cl}}$:

$$\phi_i := \bar{l}(g(\mathrm{loc}'(x_i))) + \textstyle\sum_{n \in \mathtt{Between}(MS, g(\mathrm{loc}'(x_i)), \mathrm{loc}(x_i))} \bar{l}(n) = c_i$$

- Otherwise if $\mathrm{loc}(x_i) \in N_{\mathrm{Cl}}$ and $\mathrm{loc}'(x_i) \in N'_{\mathrm{Cl}}$ (in this case, the cyclic list where x_i points to is necessarily not reachable from a root node, we write $L = \sum_{n \in \mathtt{List}(MS, \mathrm{loc}(x_i))} \bar{l}(n)$ the size of the cyclic list x_i is on then:

$$\phi_i := \textstyle\bigwedge_{\{x_h \mid \mathrm{loc}(x_h) = \mathrm{loc}(x_l)\}} \big(c_i.\mathtt{mod}(L) = c_h.\mathtt{mod}(L)\big) \wedge \bigwedge_{\{x_j \mid \mathrm{loc}(x_j) \in \mathtt{List}(MS, \mathrm{loc}(x_i) \setminus \mathrm{loc}(x_i)\}} \big(c_j.\mathtt{mod}(L) = (c_i + \bar{l}(\mathrm{loc}(x_i)) + \textstyle\sum_{n \in \mathtt{Between}(MS, \mathrm{loc}(x_i), \mathrm{loc}(x_j))} \bar{l}(n)).\mathtt{mod}(L)\big)$$

- Otherwise ($\mathrm{loc}(x_i) \in N_{\mathrm{Cl}}$ and $\mathrm{loc}'(x_i) \notin N'_{\mathrm{Cl}}$), and we denote by $L = \sum_{n \in \mathtt{List}(MS, \mathrm{loc}(x_i))} \bar{l}(n)$, intuitively L encodes the size of the cyclic list pointed to by x_i and $S = \sum_{n \in \mathtt{List}(MS, g(\mathrm{loc}'(x_i))) \setminus N_{\mathrm{Cl}}} \bar{l}(n)$ represents the length of the segment leaving from $g(\mathrm{loc}'(x_i))$ and finishing on the cyclic list, we then have:

$$\phi_i = \big(c_i \geq S \wedge \bar{l}(g(\mathrm{loc}'(x_i))) + \textstyle\sum_{n \in \mathtt{Between}(MS, g(\mathrm{loc}'(x_i)), \mathrm{loc}(x_i))} \bar{l}(n) = S + (c_i - S).\mathtt{mod}(L)\big)$$

4. Finally we obtain:

$$T_{RMS}(MS, \psi) := \exists \bar{k}_1 ... \exists \bar{k}_m . \bar{\psi} \wedge \textstyle\bigwedge_{n \in N'} \phi_n \bigwedge_{x_i \in V} \phi_i$$

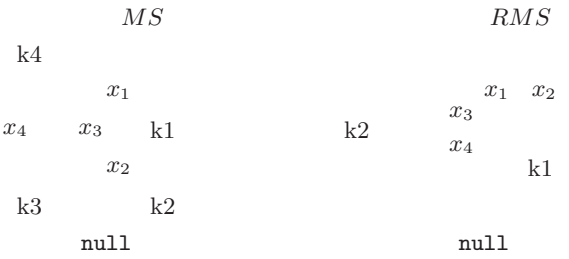
Remark that if MS is acyclic, then by construction $T_{RMS}(MS, \psi)$ is a Presburger formula. And in the other cases, $T_{RMS}(MS, \psi)$ can be rewritten into an equivalent formula of the logic $\mathcal{L}^{\exists}_{|}$. This is due to the fact that $a.\mathrm{mod}[c] = b.\mathrm{mod}[c]$ is equivalent to the formula $c \mid (a - b)$ and that any Presburger formula can be rewritten into an equivalent Presburger formula with only existential quantifiers (by elimination of the quantifiers [18]). ■

When we associate an arithmetic formula φ over the counters $C_{RMS} \cup C_V$ with a roots memory shape RMS, we denote by $[\![(RMS, \varphi)]\!]$ the set of memory graphs, $\{RMS(\boldsymbol{a}) \mid \boldsymbol{a} \models \varphi\}$ We have then the following result by construction of T_{RMS}:

Lemma 2. *Let (MS, ψ) be a symbolic memory shape and RMS a roots memory shape.*

1. *If $MS \sqsubset RMS$, $[\![(RMS, T_{RMS}(MS, \psi))]\!] = [\![(MS, \psi)]\!]$.*
2. *If $MS \not\sqsubset RMS$, $[\![(RMS, T_{RMS}(MS, \psi))]\!] = \varnothing$.*

We will now see how we use this different results to prove the theorems 4 and 5. The main idea consists in reducing the problems of safety and model-checking over pointer systems without destructive update to similar problems over counter systems.



$$T_{RMS}(MS,\psi) := \exists \bar{k}_1.\exists \bar{k}_2.\exists \bar{k}_3.\exists \bar{k}_4.\bar{\psi} \wedge$$
$$k1 = \bar{k}1 + \bar{k}2 \wedge k2 = \bar{k}3 + \bar{k}4 \wedge$$
$$c_1 = 0 \wedge c2 = \bar{k}1 \wedge$$
$$c_4.\texttt{mod}[k2] = (c_3 + \bar{k}3).\texttt{mod}[k2] \wedge$$
$$c_3.\texttt{mod}[k2] = (c_4 + \bar{k}4).\texttt{mod}[k2]$$

Fig. 5. An example of the computation of the formula $T_{RMS}(MS,\psi)$

4.4 Proof of Theorem 4

Let $(q,(MS,\psi))$ be a symbolic configuration of a pointer system. We define:

$$T(q,(MS,\psi)) = \bigcup_{RMS \in \mathcal{RMS}} ((q,RMS), T_{RMS}(MS,\psi))$$

$T(q,(MS,\psi))$ represents a symbolic configuration for $CS_2(PS)$ and furthermore by Lemma 2, we have that $[\![(MS,\psi)]\!] = \bigcup_{RMS \in \mathcal{RMS}} [\![(RMS, T_{RMS}(MS,\psi))]\!]$. The properties of the arithmetic formula $T_{RMS}(MS,\psi)$ and the fact that the relation R_T is a bisimulation between the transition system of PS and the one of $CS_2(PS)$ allows us to state the following lemma.

Lemma 3. *Let PS be a pointer system without destructive update and without alias test, $(q_0,(MS_0,\psi_0))$ an initial symbolic configuration, and $(q_B,(MS_B,\psi_B))$ a bad symbolic configuration for PS. Then:*

$$\texttt{Reach}(PS, [\![(q_0,(MS_0,\psi_0))]\!]) \cap [\![(q_B,(MS_B,\psi_B))]\!] = \varnothing$$

if and only if

$$\texttt{Reach}(CS_2(PS), [\![T(q_0,(MS_0,\psi_0))]\!]) \cap [\![T(q_B,(MS_B,\psi_B))]\!] = \varnothing$$

Using this last lemma and previous results, we can prove Theorem 4.

Proof of Theorem 4. Let PS be a flat pointer system without destructive update and without alias test, $(q_0,(MS_0,\psi_0))$ an initial symbolic configuration, and $(q_B,(MS_B,\psi_B))$ a bad symbolic configuration for PS. By proposition 1, the counter system $CS_2(PS)$ is flat and has the finite monoid property. By Lemma 3, the considered safety problem reduces to the safety problem

for $CS_2(PS)$ with $T(q_0,(MS_0,\psi_0))$ as initial symbolic configuration and $T(q_B,(MS_B,\psi_B))$ as bad symbolic configuration. Furthermore, since for all memory shapes MS, for all Presburger formula ψ over C_{MS} and for all roots memory shape RMS , $T_{RMS}(MS,\psi)$ is a formula of $\mathcal{L}^{\exists}_{|}$, we can deduce from Theorem 1, that this last problem reduces to the satisfiability problem of a formula of $\mathcal{L}^{\exists}_{|}$, which is a decidable problem [23]. ■

4.5 Proof of Theorem 5

We consider then a formula Φ of CTL$^*(MS)$. We define the formula $T(\Phi)$ by induction as follows:

- if $\Phi := (q,(MS,\psi))$ with MS acyclic then:
  $T(\Phi) := \bigvee_{RMS\in\mathcal{RMS}}((q,RMS),T_{RMS}(MS,\psi))$;
- if $\Phi := (q,(MS,\psi))$ with MS not acyclic then:
  $T(\Phi) := \bigvee_{RMS\in\mathcal{RMS}}((q,RMS),\text{False})$;
- if $\Phi := \neg\Phi'$ then $T(\Phi) := \neg T(\Phi')$;
- if $\Phi := \Phi' \wedge \Phi''$ then $T(\Phi) := T(\Phi') \wedge T(\Phi'')$;
- if $\Phi := \mathtt{X}\Phi'$ then $T(\Phi) := \mathtt{X}T(\Phi')$;
- if $\Phi := \Phi'\mathtt{U}\Phi''$ then $T(\Phi) := T(\Phi')\mathtt{U}T(\Phi'')$;
- if $\Phi := \mathtt{A}\Phi'$ then $T(\Phi) := \mathtt{A}T(\Phi')$.

Since, for all acyclic memory shapes MS, for all Presburger formulae ψ over C_{MS} and for all roots memory shape RMS, $T_{RMS}(MS,\psi)$ is a Presburger formula, we deduce that:

Remark 1. For all formulae Φ of CTL$^*(MS)$, $T(\Phi)$ is a formula of CTL$^*(Pr)$.

Moreover, the following lemma holds.

Lemma 4. *Let PS be a pointer system without destructive update, $(q_0,(MS_0,\psi_0))$ an initial acyclic symbolic configuration of PS, and Φ a formula of* CTL$^*(MS)$*. We have that $\pi,0 \models \Phi$ for all configurations paths π of $CS_2(PS)$ such that $\pi(0) \in [\![(q_0,(MS_0,\psi_0))]\!]$, if and only if $\pi',0 \models T(\Phi)$ for all configuration paths π' of the transition system $TS(CS_2(PS))$ such that $\pi'(0) \in [\![T(q_0,(MS_0,\psi_0))]\!]$.*

Proof. This lemma can be proved by induction on the length of the formula Φ using the definition of $T(q_0,(MS_0,\psi_0))$. The first case of the induction is when Φ is of the form $(q,(MS,\psi))$, and it is proved using Lemma 2. The other cases are then proved using that the relation R^{ac}_T is a bisimulation between the transition system of PS and the one of $CS_2(PS)$. ■

This allows us to conclude Theorem 5.
Proof of Theorem 5. Let PS be a flat pointer system without destructive update, $(q_0,(MS_0,\psi_0))$ an initial acyclic symbolic configuration of PS, and Φ a formula of CTL$^*_{\text{mem}}$. By proposition 1, the counter system $CS_2(PS)$ is flat and has the finite monoid property. Besides, by Lemma 4, the considered problem reduces to

the model-checking problem for $CS_2(PS)$ with the initial symbolic configuration $T(q_0, (MS_0, \psi_0))$ and the FOCTL*(Pr) formula $T(\Phi)$. Hence using Theorem 2, we can deduce that the model-checking problem for PS with $(q_0, (MS_0, \psi_0))$ as symbolic initial configuration and Φ as temporal formula is decidable. ■

5 Undecidability Results

In this section, we show that the decidability results we obtained for safety and temporal properties are tight. In particular, these results become false if one relaxes any hypothesis. For instance, Theorem 4 does not hold without the hypothesis of absence of alias test (this is proved in [11]). We list here some new decidability results for some classes of pointer systems very close to the ones we studied in the previous section. All our undecidability results are based on a reduction from satisfiability of Diophantine equations, which is known as undecidable. Diophantine equations are equations of the form $P(\mathbf{k}) = 0$ where P is some polynomial over naturals and $\mathbf{k}$ a vector of positive integer variables. As explained in [11], Diophantine equations can be encoded as a conjunction of arithmetic formulae of the form $k = k' + k''$ or $k = \text{lcm}(k', k'')$ or $k = j$ where k, k', k'' are counter variables and j is a positive integer; the satisfiability problem of such formulae is then undecidable. Below, we use the term of Diophantine equations to describe such conjunctions of arithmetic formulae. Following [11], we now associate with a Diophantine equation $\mathcal{E}$ a pointer system $PS_\mathcal{E}$ for which a certain safety (resp. temporal) property holds if and only if $\mathcal{E}$ has a solution. Our first result shows that theorem 4 does not extend to model-checking:

Theorem 6. *The model-checking problem is undecidable for flat pointer systems without destructive update and without alias test.*

The proof of this result is an adaption of the undecidability proof of [11]. Let us consider $\mathcal{E} = \bigwedge \mathcal{E}_i$ a Diophantine equation. We define the tuple $(MS_\mathcal{E}, PS_\mathcal{E}, \Phi_E)$ such that: $MS_\mathcal{E}$ is a memory shape defined by $MS_\mathcal{E} = MS_1 \uplus .. \uplus MS_n$ (where $\uplus$ represents the disjunctive union of memory shapes and n coincides with the number of conjuncts in $\mathcal{E}$); $\Phi_\mathcal{E} = \bigwedge \Phi_i$ is a temporal property; and $PS_\mathcal{E}$ is a flat pointer system without destructive update and without alias test defined by $PS_\mathcal{E} = PS_1; ..; PS_n$, where ; is the sequential composition and each PS_i is a flat program (without destructive update and without alias test) that either exits correctly (`exit(0)`) and launches the execution of PS_{i+1} or aborts (`exit(1)`):

- if $\mathcal{E}_i$ is $k = k' + k''$, MS_i is the memory shape with three disjoint list segments of length k, k', k'', whose heads are pointed to by some set of fresh variables $\{\mathtt{x}, \mathtt{x}_0\}, \{\mathtt{y}, \mathtt{y}_0\}, \{\mathtt{z}, \mathtt{z}_0\}$ respectively, and whose tails are on `null`. The pointer system PS_i can be described by the program:

  ```
  while (y≠NULL) do x=x.succ; y=y.succ;end while;
  while (z≠NULL) do x=x.succ; z=z.succ;end while;
  if (x=NULL) then exit(0); else exit(1);
  ```

  The temporal property Φ_i expresses that the `exit(0)` is reached (which means without error).

– if $\mathcal{E}_i$ is $k = \text{lcm}(k', k'')$, we define MS_i to be the memory shape with two disjoint cyclic lists pointed to by sets of fresh variables $\{\mathtt{y}, \mathtt{y}_0\}, \{\mathtt{z}, \mathtt{z}_0\}$ respectively, of lengths k', k'', and a single list of length k whose head is pointed by some set of fresh variables $\{\mathtt{x}, \mathtt{x}_0\}$ and ends on `null`. We moreover define PS_i by the program:

```
while (x≠NULL) do x=x.succ;y=y.succ;z=z.succ;end while;
exit(0);
```

The temporal property Φ_i expresses that, each state of the loop verifies $(\mathtt{y}, \mathtt{z}) \neq (\mathtt{y}_0, \mathtt{z}_0)$ until $(\mathtt{x}, \mathtt{y}, \mathtt{z}) = (\mathtt{null}, \mathtt{y}_0, \mathtt{z}_0)$.

– if $\mathcal{E}_i$ is $k = j$, MS_i is the memory shape with one list segment of length k, whose head is pointed on by some set of fresh variables $\{\mathtt{x}, \mathtt{x}_0\}$ and whose tail is on `null`. The pointer system PS_i can be described by a program without loop which performs i times `x:=x.succ` and does `exit(0)` at the end if x points on `null` and `exit(1)` otherwise. The temporal property Φ_i expresses that the `exit(0)` is reached (which means without error).

By construction the Diophantine equation represented by the formula $\mathcal{E}$ has a solution if and only if there exists a configuration path π in $PS_{\mathcal{E}}$ such that $\pi(0) \in [\![(MS_{\mathcal{E}}, \text{True})]\!]$ and $\pi, 0 \models \Phi_{\mathcal{E}}$, which is true if and only if the answer to the model-checking problem on the inputs $PS_{\mathcal{E}}$, $(MS_{\mathcal{E}}, \text{True})$ and $\neg\Phi_{\mathcal{E}}$ is no. We deduce from this the result of Theorem 6. This last proof and the proof of undecidability in [11] build flat pointer systems working over cyclic lists. Hence one may think that the key point of these undecidability results is the use of cyclic lists. We show here that this is not the case. First, we say that a pointer system $PS = \langle Q, E\rangle$ associated with an initial symbolic configuration $(q_0, (MS_0, \psi_0))$ is an *acyclic pointer system* if the memory shape MS_0 is acyclic and all reachable memory graphs MG (i.e., there is a $q \in Q$ such that $(q, MG) \in \texttt{Reach}(PS, [\![(q_0, (MS_0, \psi_0))]\!])$) are acyclic.

Theorem 7. *The safety problem is undecidable for acyclic flat pointer systems.*

Proof. We adapt the previous proof, that works with cyclic lists, to a system that does not work with cyclic lists but can use destructive updates. Note that we only need to adapt the program that tests the equation $k = \text{lcm}(k', k'')$, the rest of the proof being then similar. Consider a list segment l whose head is pointed to by $\{\mathtt{h}, \mathtt{h}'\}$, whose tail is on `null`, and whose last node before `null` is pointed by `t`; we thus define the subprogram: `rotate`(l)=`h=h.succ;h'.succ=NULL; t.succ=h';t=h';h'=h;`

This program moves the first element of the list at the tail of the list. We now consider the memory shape with two such disjoint list segments ℓ_1, ℓ_2 with some pairs of extra variables $\{\mathtt{y}, \mathtt{y}_0\}, \{\mathtt{z}, \mathtt{z}_0\}$ at some point in the middle of l_1 and l_2 respectively, and a disjoint standard list segment l whose head is pointed by $\{\mathtt{x}, \mathtt{x}_0\}$ and whose tail is pointed to `null`. Counters k, k', k'' represent the total length of the lists ℓ, ℓ_1, ℓ_2 respectively. Then the following program exits normally if and only if $k = \text{lcm}(k', k'')$:

```
while (x≠null and not((y=y0) and (z=z0))) do
x=x.succ;y=y.succ;z=z.succ;
```

```
rotate(l1);rotate(l2);end while;
if (x=null and y=y0 and z=z0) then exit(0) else exit(1);
```

which ends the proof. ■

Note that this last result answers a problem that was stated as open in [11].

6 Conclusion

We have proposed a framework for model-checking pointer systems without destructive update. Given any pointer system without destructive update, one may translate it into a bisimilar counter system having the finite monoid property. Then a counter model-checker, `FAST` for instance, may verify it; if the counter system is flat (or flattable [4]), then `FAST` will terminate computing the Presburger representation of the reachability relation.

It was known that safety was undecidable for flat pointer systems without destructive update. We have completed the classification of flat pointer systems without destructive update in showing that the model-checking problem becomes decidable for flat pointer systems without destructive update with an initial acyclic configuration. We prove that if we replace the acyclic hypothesis by the hypothesis of absence of alias test, then safety remains decidable but model-checking becomes undecidable. Moreover, if we remove the hypothesis that the system is without destructive update, for even acyclic flat pointer systems the safety problem is undecidable. The table 4 contains a summary of the main decidability results when considering flat pointer systems

Table 4. Summary of main results

Flat pointer systems	Initial symbolic configuration	Safety problem	Model-checking problem
Without destructive update	Acyclic	Decidable	Decidable
Without destructive update	No Restriction	Undecidable	Undecidable
Without destructive update and without alias test	No Restriction	Decidable	Undecidable
Acyclic	Acyclic	Unecidable	Undecidable

References

1. Annichini, A., Bouajjani, A., Sighireanu, M.: Trex: A tool for reachability analysis of complex systems. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 368–372. Springer, Heidelberg (2001)
2. Bardin, S., Finkel, A., Leroux, J.: FASTer acceleration of counter automata in practice. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 576–590. Springer, Heidelberg (2004)
3. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: Acceleration from theory to practice. Int. J. Softw. Tools Technol. Transf. (to appear, 2008)
4. Bardin, S., Finkel, A., Leroux, J., Schnoebelen, P.: Flat acceleration in symbolic model checking. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 474–488. Springer, Heidelberg (2005)
5. Bardin, S., Finkel, A., Lozes, É., Sangnier, A.: From pointer systems to counter systems using shape analysis. In: Proc. AVIS 2006 (2006)
6. Bardin, S., Finkel, A., Nowak, D.: Toward symbolic verification of programs handling pointers. In: Proc. AVIS 2004 (2004)
7. Berdine, J., Cook, B., Distefano, D., O'Hearn, P.W.: Automatic termination proofs for programs with shape-shifting heaps. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 386–400. Springer, Heidelberg (2006)
8. 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)
9. Bouajjani, A., Habermehl, P., Moro, P., Vojnar, T.: Verifying programs with dynamic 1-selector-linked structures in regular model checking. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 13–29. Springer, Heidelberg (2005)
10. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular tree model checking of complex dynamic data structures. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 52–70. Springer, Heidelberg (2006)
11. Bozga, M., Iosif, R.: On flat programs with lists. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 122–136. Springer, Heidelberg (2007)
12. Brochenin, R., Demri, S., Lozes, É.: Reasoning about sequences of memory states. In: Artemov, S.N., Nerode, A. (eds.) LFCS 2007. LNCS, vol. 4514, pp. 100–114. Springer, Heidelberg (2007)
13. Demri, S., Finkel, A., Goranko, V., van Drimmelen, G.: Towards a model-checker for counter systems. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 493–507. Springer, Heidelberg (2006)
14. Distefano, D., Katoen, J.-P., Rensink, A.: Who is pointing when to whom? In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 250–262. Springer, Heidelberg (2004)
15. Distefano, D., O'Hearn, P.W., Yang, H.: A local shape analysis based on separation logic. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 287–302. Springer, Heidelberg (2006)
16. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: Longo, G. (ed.) LICS 1999, pp. 352–359. IEEE Computer Society Press, Los Alamitos (1999)
17. Finkel, A., Leroux, J.: How to compose Presburger-accelerations: Applications to broadcast protocols. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 145–156. Springer, Heidelberg (2002)

18. Ginsburg, S., Spanier, E.H.: Semigroups, presburger formulas, and languages. Pacific Journal of Mathematics 16(2), 285–296 (1966)
19. Gonnord, L., Halbwachs, N.: Combining widening and acceleration in linear relation analysis. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 144–160. Springer, Heidelberg (2006)
20. Homepage of LASH, `http://www.montefiore.ulg.ac.be/~boigelot/research/lash`
21. Leroux, J., Sutre, G.: Flat counter automata almost everywhere! In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 489–503. Springer, Heidelberg (2005)
22. Lev-Ami, T., Sagiv, M.: TVLA: A system for implementing static analyses. In: Palsberg, J. (ed.) SAS 2000. LNCS, vol. 1824, pp. 280–302. Springer, Heidelberg (2000)
23. Lipshitz, L.: The diophantine problem for addition and divisibility. Transactions of the American Mathematical Society 235, 271–283 (1978)
24. Magill, S., Berdine, J., Clarke, E., Cook, B.: Arithmetic strengthening for shape analysis. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 419–436. Springer, Heidelberg (2007)
25. Mandel, A., Simon, I.: On finite semigroups of matrices. Theor. Comput. Sci. 5(2), 101–111 (1977)
26. Møller, A., Schwartzbach, M.I.: The pointer assertion logic engine. In: PLDI 2001, pp. 221–231. ACM Press, New York (2001)
27. Podelski, A., Wies, T.: Boolean heaps. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 268–283. Springer, Heidelberg (2005)
28. Wolper, P., Boigelot, B.: Verifying systems with infinite but regular state spaces. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 88–97. Springer, Heidelberg (1998)
29. Yahav, E., Reps, T.W., Sagiv, M., Wilhelm, R.: Verifying temporal heap properties specified via evolution logic. In: Degano, P. (ed.) ESOP 2003. LNCS, vol. 2618, pp. 204–222. Springer, Heidelberg (2003)

A Examples of Programs

In this section, we give the description of some of the programs which feature in the table 1.

The program `split`:

```
 1: void split(List x){
 2:  List y,z,t,u;
 3:  u=NULL;
 4:  y=x;
 5:  while(y!=NULL){
 6:   t=y->next;
 7:   z=t->next;
 8:   t->next=u;
 9:   y->next=z;
10:   u=t;
11:   y=z;}
11:  }
```

This program is safe when the acyclic list given in input (pointed to by x) has an even length.

The program `delete(n)`:

```
 1: void delete(List x,int n){
 2:  List y;
 3:  int i=n;
 4:  while(i!=0){
 5:   y=x->next;
 6:   free(x);
 7:   x=y;
 8:   i--;
 9:  }
```

This program is safe when the acyclic list given in input (pointed to by x) has a length greater than the integer n.

The program `parse-cyclic-acyclic`:

```
 1: void parse-cyclic-acyclic(List x,List t){
 2:  List y,u;
 3:  y=x;
 4:  u=t;
 5:  while(y!=NULL){
 6:   y=y->next;
 7:   u=u->next;}
 8:  u->next=NULL;
 9:  }
```

For this last program, we assume that the variable x is pointed to an acyclic list, and the variable t to a cyclic list. One can check that this program yields a

memory leak when the number of elements in the list pointed to by t does not divides the number of elements in the list pointed to by x to which we add 1.

B Description of the Translation $\mathbf{POST_2}$

Table 5. Computation of $((A, \mathbf{b}, \phi), RMS') = \mathrm{POST}_2(a, RMS)$ for the action a of the form $x_i := x_j$

Hypothesis	$(A, \mathbf{b})$	ϕ	RMS'
x_i is single in RMS and $\mathrm{loc}(x_i) \notin \{\mathtt{null}, \bot\}$ and $x_i \neq x_j$	$(\mathrm{Id}, \overrightarrow{0})$	False	RMS
x_i is single in RMS and $\mathrm{loc}(x_i) \in \{\mathtt{null}, \bot\}$ and $x_i \neq x_j$	$c_i := c_j$	True	$\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$
$x_i = x_j$	$(\mathrm{Id}, \overrightarrow{0})$	True	RMS
x_i is not single in RMS and $x_i \neq x_j$ and $\mathrm{loc}(x_i)$ belongs to a cyclic list	$c_i := c_j$	True	$\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$
x_i is not single in RMS and $x_i \neq x_j$ and $\mathrm{loc}(x_i)$ does not belong to a cyclic list	$c_i := c_j$	$\bigvee_{x_l \in \mathrm{loc}^{-1}(\{\mathrm{loc}(x_i)\}) \setminus \{\mathrm{loc}(x_i)\}} c_l = 0$	$\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$

Table 6. Computation of $((A, \mathbf{b}, \phi), RMS') = \mathrm{POST}_2(a, RMS)$ for the action a of the form $x_i := x_j.\mathrm{succ}$ (I)

Hypothesis	$(A, \mathbf{b})$	ϕ	RMS'
$\mathrm{loc}(x_j) \in \{\mathtt{null}, \bot\}$	$(\mathrm{Id}, \overrightarrow{0})$	False	RMS
x_i is single in RMS and $\mathrm{loc}(x_i)$ is not on a cyclic list and $\mathrm{loc}(x_i) \notin \{\mathtt{null}, \bot\}$	$(\mathrm{Id}, \overrightarrow{0})$	False	RMS
x_i is single in RMS and $\mathrm{loc}(x_i)$ is on a cyclic list and $x_i = x_j$	$(\mathrm{Id}, \overrightarrow{0})$	True	RMS
x_i is single in RMS and $\mathrm{loc}(x_i)$ is on a cyclic list and $x_i \neq x_j$	$(\mathrm{Id}, \overrightarrow{0})$	False	RMS
$\mathrm{loc}(x_i) \in \{\mathtt{null}, \bot\}$ and there is an n such that $n \in \mathtt{List}(RMS, x_j)$ and n is on a cyclic list	$c_i := c_j + 1$	True	$\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$
$\mathrm{loc}(x_i) \in \{\mathtt{null}, \bot\}$ and there is no n such that $n \in \mathtt{List}(RMS, x_j)$ and n is on a cyclic list and $\mathrm{loc}(x_j) \notin \{\mathtt{null}, \bot\}$	$c_i := c_j + 1$	$c_j < \Sigma_{n \in \mathtt{List}(RMS, x_j) \cap N} \ell(n)$	$\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$

Table 7. Computation of $((A, \mathbf{b}, \phi), RMS') = \mathrm{POST}_2(a, RMS)$ for the action a of the form $x_i := x_j.\mathrm{succ}$ (II)

Hypothesis	$(A, \mathbf{b})$	ϕ	RMS'
x_i is not single in RMS and x_i does belong to a cyclic list and there is an n such that $n \in \mathtt{List}(RMS, x_j)$ and n is on a cyclic list	$c_i := c_j + 1$	$\bigvee_{x_l \in \mathrm{loc}^{-1}(\{\mathrm{loc}(x_i))\} \setminus \{\mathrm{loc}(x_i)\}\}} c_l = 0$	$\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$
x_i is not single in RMS and x_i does not belong to a cyclic list and there is no n such that $n \in \mathtt{List}(RMS, x_j)$ and n is on a cyclic list and $\mathrm{loc}(x_j) \notin \{\mathtt{null}, \bot\}$	$c_i := c_j + 1$	$c_j < \Sigma_{n \in \mathtt{List}(RMS, x_j) \cap N} \ell(n) \wedge \bigvee_{x_l \in \mathrm{loc}^{-1}(\{\mathrm{loc}(x_i))\}) \setminus \{\mathrm{loc}(x_i)\}} c_l = 0$	$\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$
x_i is not single in RMS and x_i belongs to a cyclic list and there is an n such that $n \in \mathtt{List}(RMS, x_j)$ and n is on a cyclic list	$c_i := c_j + 1$	True	$\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$
x_i is not single in RMS and x_i belongs to a cyclic list and there is no n such that $n \in \mathtt{List}(RMS, x_j)$ and n is on a cyclic list and $\mathrm{loc}(x_j) \notin \{\mathtt{null}, \bot\}$	$c_i := c_j + 1$	$c_j < \Sigma_{n \in \mathtt{List}(RMS, x_j) \cap N} \ell(n)$	$\mathrm{loc}'(x_i) = \mathrm{loc}(x_j)$

Representations of Numbers as $\sum_{k=-n}^{n} \varepsilon_k k$: A Saddle Point Approach

Guy Louchard[1] and Helmut Prodinger[2,⋆]

[1] Université Libre de Bruxelles, Département d'Informatique, CP 212, Boulevard du Triomphe, 1050 Bruxelles, Belgium
louchard@ulb.ac.be
[2] University of Stellenbosch, Mathematics Department, 7602 Stellenbosch, South Africa
hproding@sun.ac.za

Abstract. Using the saddle point method, we obtain from the generating function of the numbers in the title and Cauchy's integral formula asymptotic results of high precision in central and non-central regions.

1 Introduction

We consider the number of representations of m as $\sum_{k=-n}^{n} \varepsilon_k k$, where $\varepsilon_k \in \{0, 1\}$. For $m = 0$, this is sequence A000980 in Sloane's encyclopedia. This problem has a long history: Van Lint [7] found an asymptotic formula for the number of representations of 0, Entringer [2] did the same for a fixed number m. Clark [1] allowed m to be $o(n^{3/2})$.

Here, we extend the range a bit, to $\mathcal{O}(n^{3/2})$. But we improve at the same time the quality of the approximation. We are doing this using the saddle point method. Implicitly, in the earlier writings, it was also used, but $z = 1$ was taken as an approximative saddle point, whereas we employ better approximations, to get more precise results.

We are also able to choose m in a non-central region. As a showcase, we consider $m = n^{7/4}$, but we could deal with other values as well.

Our main findings are the asymptotic formulæ (6) (central range) and (7) (non-central range).

The generating function of the number of representations for fixed n is given by

$$C_n(z) = 2 \prod_{k=1}^{n} (1 + z^k)(1 + z^{-k})$$

and

$$C_n(1) = 2 \cdot 4^n.$$

⋆ (*Received by the editors.* 24 January 2008. *Revised.* 17 July 2008; 6 January 2009. *Accepted.* 11 February 2009.)

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, pp. 87–96, 2009.
Springer-Verlag Berlin Heidelberg 2009

By normalisation, we get the probability generating function of a random variable X_n:

$$F_n(z) = 4^{-n} \prod_{k=1}^{n} (1 + z^k)(1 + z^{-k}),$$

from which we obtain mean $\mathbb{M}$ and variance $\mathbb{V}$:

$$\mathbb{M}(n) = 0, \quad \sigma^2 := \mathbb{V}(n) = \frac{n(n+1)(2n+1)}{12}.$$

By Cauchy's theorem,

$$P_n(j) = \mathbb{P}(X_n = j) = \frac{1}{2\pi i} \int_{\mathcal{C}} F_n(z) \frac{dz}{z^{j+1}},$$

where $\mathcal{C}$ is, say, a circle around the origin.

A referee and the editor have asked us to give more background about the *saddle point method* that we are going to use in this paper. For this, we borrow freely from Odlyzko's excellent treatise [6]:

The saddle point method is based on the freedom to shift contours of integration when estimating integrals of analytic functions (like the integral just mentioned). Let us assume that $f(z)$ is analytic in $|z| < R \leq \infty$. Also, we assume that there is some R_0, such that if $R_0 < r < R$, then $\max_{|z|=r} |f(z)| = f(r)$. This is clearly satisfied for generating functions from combinatorics, which have nonnegative coefficients. Further, it is assumed that $z = r$ is the unique point with $|z| = r$ where the maximum is attained. (If not, there are usually some *periodicities* involved.)

The first step in estimating $[z^n]f(z)$ is to find the saddle point. Under our assumptions, this will be a point $r = r_0$ with $R_0 < r < R$ which minimizes $r^{-n} f(r)$. Cauchy's integral formula is then applied with the contour $|z| = r_0$. Another circle would give the same result, but the absolute values of the function would be much larger, so that there must be (huge) cancellations, and they are not easy to control. The special choice of the path of integration avoids this.

Odlyzko [6] provides the instructive example of estimating $1/n! = [z^n]e^z$ that we reproduce here. We find the minimum of $g(x) = x^{-n}e^x$ on the positive real line by ordinary calculus: $g'(x) = g(x)\big(1 - \frac{n}{x}\big) = 0$, whence $x = n$. So one chooses the contour $z = ne^{i\theta}$, with $-\pi \leq \theta \leq \pi$. Then

$$\frac{1}{n!} = \frac{1}{2\pi i} \int_{|z|=n} \frac{e^z}{z^{n+1}} dz = \frac{1}{2\pi} \int_{-\pi}^{\pi} n^{-n} \exp(ne^{i\theta} - ni\theta) d\theta.$$

Now, $e^{i\theta} = 1 - \theta^2/2 + i\theta + \mathcal{O}(|\theta^3|)$. So, for any symmetric interval, we have

$$\int_{-\theta_0}^{\theta_0} n^{-n} \exp(ne^{i\theta} - ni\theta) d\theta = \int_{-\theta_0}^{\theta_0} n^{-n} \exp(n - n\theta^2/2 + \mathcal{O}(n|\theta^3|)) d\theta.$$

The cancellation of the term $ni\theta$ in $ne^{i\theta} - ni\theta$ is primarily responsible for the success of the saddle point method.

Now, to control the error term $\mathcal{O}(n|\theta^3|)$, one chooses, say, $\theta_0 = n^{-2/5}$. One shows that outside of this range, the integrand is small, and in the approximation

$$\int_{-\theta_0}^{\theta_0} n^{-n} \exp(n - n\theta^2/2) d\theta$$

one can a) pull out the term e^n, and b) in the remaining integral replace the limits of integration by $\pm\infty$. The survey [6] has all the detailed error calculations. But then

$$\int_{-\infty}^{\infty} n^{-n} \exp(-n\theta^2/2) d\theta = \sqrt{\frac{2\pi}{n}},$$

whence we have

$$\frac{1}{n!} \sim n^{-n} e^n \frac{1}{\sqrt{2\pi n}},$$

which is Stirling's approximation in its simplest form. (With more precise expansions, the full form of Stirling's formula can be derived.)

In this instructive example, the saddle point $z = n$ was easy to compute. In more complicated situations, as we will encounter in the rest of this paper, it cannot be exactly computed. But one can work with an *approximate saddle point*, if one controls the errors.

A circle as path of integration can be replaced by a straight line. In fact, only a tiny fraction of the circle is used anyway (think about θ_0, which went to 0), the rest being unimportant. But the small piece of the circle can be replaced by a straight line, if one controls the error.

2 The Gaussian Limit – Approximation near the Mean

We consider values $j = x\sigma$, for $x = \mathcal{O}(1)$ in a neighbourhood of the mean 0.

The Gaussian limit of X_n can be obtained by using the Lindeberg-Lévy conditions, see for instance, Feller [3], but we want more precision.

We know that

$$P_n(j) = \frac{1}{2\pi i} \int_{\Omega} e^{S(z)} dz$$

where

$$S(z) := \ln(F_n(z)) - (j+1) \ln z$$

with

$$\ln(F_n(z)) = \sum_{i=1}^{n} [\ln(1 + z^i) + \ln(1 + z^{-i}) - \ln 4].$$

Since $F_n(z)$ is a (Laurent-)polynomial, the analyticity restriction can be ignored. We will use as Ω a circle around the origin passing (approximately) through the saddle point. We split the exponent of the integrand S as

$$S := S_1 + S_2, \tag{1}$$
$$S_1 := \sum_{i=1}^{n} [\ln(1+z^i) + \ln(1+z^{-i}) - \ln 4],$$
$$S_2 := -(j+1)\ln z.$$

Set

$$S^{(i)} := \frac{d^i S}{dz^i}.$$

As general references for the application of the saddle point method in enumeration we cite [4,6]. To use this method, we must find the solution of

$$S^{(1)}(\tilde{z}) = 0. \tag{2}$$

Set $\tilde{z} := z^* - \varepsilon$, where, here, $z^* = 1$. (This notation always means that z^* is the approximate saddle point and $\tilde{z}$ is the exact saddle point; they differ by a quantity that has to be computed to some degree of accuracy.) This leads, to first order, to

$$-j - 1 + \Big(-\frac{(n+1)^3}{6} + \frac{(n+1)^2}{4} - \frac{n}{12} - \frac{13}{12} - j\Big)\varepsilon = 0. \tag{3}$$

Set $j = x\sigma$ in (3). This shows that, asymptotically, ε is given by a *Puiseux series* of powers of $n^{-1/2}$, starting with $-\sqrt{6}x/n^{3/2}$. To obtain the next terms, we compute the following terms in the expansion of (2). Even powers ε^{2k} lead to a $\mathcal{O}(n^{2k+1}) \cdot \varepsilon^{2k}$ term and odd powers ε^{2k+1} lead to a $\mathcal{O}(n^{2k+3}) \cdot \varepsilon^{2k+1}$ term. Now we expand into powers of $n^{-1/2}$ and equate each coefficient with 0. This leads successively to a full expansion of ε. Note that to obtain a given precision of ε, it is enough to compute a given finite number of terms in the generalization of (3). We obtain (here and in the following, we provide only the first terms, but we computed more with Maple)

$$\varepsilon = -\sqrt{6}xn^{-3/2} + \sqrt{6}(-3/10x^3 + 3/4x^2)n^{-5/2} + (-3x^2 - 6)n^{-3} + \cdots$$

We have, with $\tilde{z} := z^* - \varepsilon$,

$$P_n(j) = \frac{1}{2\pi i}\int_\Omega \exp\Big[S(\tilde{z}) + S^{(2)}(\tilde{z})(z-\tilde{z})^2/2! + \sum_{l=3}^{\infty} S^{(l)}(\tilde{z})(z-\tilde{z})^l/l!\Big]dz$$

(note carefully that the linear term vanishes). Set $z = \tilde{z} + i\tau$. This gives

$$P_n(j) = \frac{1}{2\pi}\exp[S(\tilde{z})]\int_{-\infty}^{\infty} \exp\Big[S^{(2)}(\tilde{z})(i\tau)^2/2! + \sum_{l=3}^{\infty} S^{(l)}(\tilde{z})(i\tau)^l/l!\Big]d\tau. \tag{4}$$

Let us first analyze $S(\tilde{z})$. We obtain

$$S_1(\tilde{z}) = \frac{x^2}{2} + \frac{9x^4}{40n} + \frac{\sqrt{6}x}{n^{3/2}} + \cdots$$
$$S_2(\tilde{z}) = -x^2 - \frac{3x^4}{10n} - \frac{2\sqrt{6}x}{n^{3/2}} + \cdots,$$

and so

$$S(\tilde{z}) = -\frac{x^2}{2} - \frac{3x^4}{40n} - \frac{\sqrt{6}x}{n^{3/2}} + \cdots$$

Also,

$$\begin{aligned}
S^{(2)}(\tilde{z}) &= \frac{n^3}{6} + \Big(-\frac{3}{20}x^2 + \frac{1}{4}\Big)n^2 + \mathcal{O}(n^{3/2}),\\
S^{(3)}(\tilde{z}) &= \mathcal{O}(n^{7/2}),\\
S^{(4)}(\tilde{z}) &= -\frac{1}{20}n^5 + \mathcal{O}(n^4),\\
S^{(l)}(\tilde{z}) &= \mathcal{O}(n^{l+1}), \quad l \geq 5.
\end{aligned}$$

We can now compute (4), for instance by using the classical trick of setting

$$S^{(2)}(\tilde{z})(i\tau)^2/2! + \sum_{l=3}^{\infty} S^{(l)}(\tilde{z})(i\tau)^l/l! = -u^2/2, \tag{5}$$

computing τ as a truncated series in u, starting as

$$\tau = \frac{u}{n^{3/2}}\sqrt{6} + \cdots$$

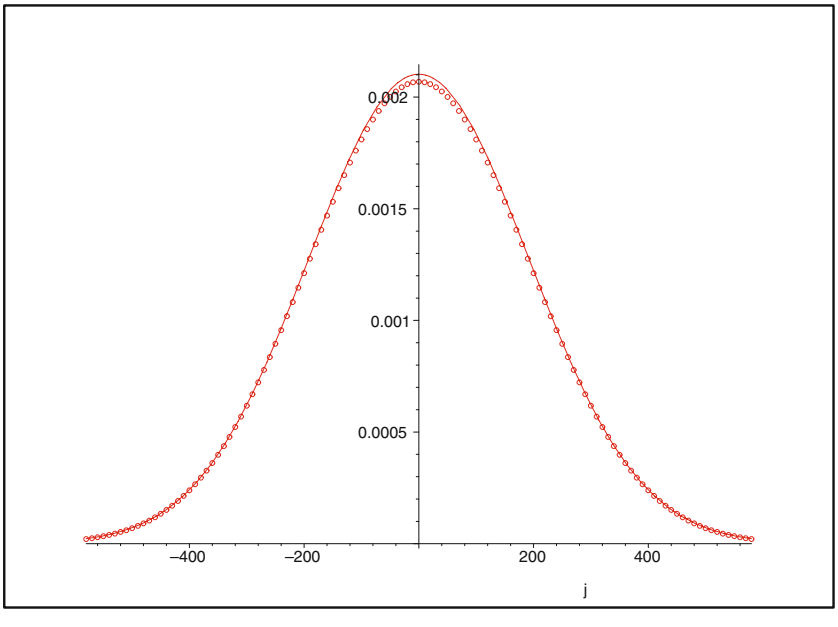


Fig. 1. $P_n(j)$ (circle) and the Gaussian asymptotics (line), $n = 60$

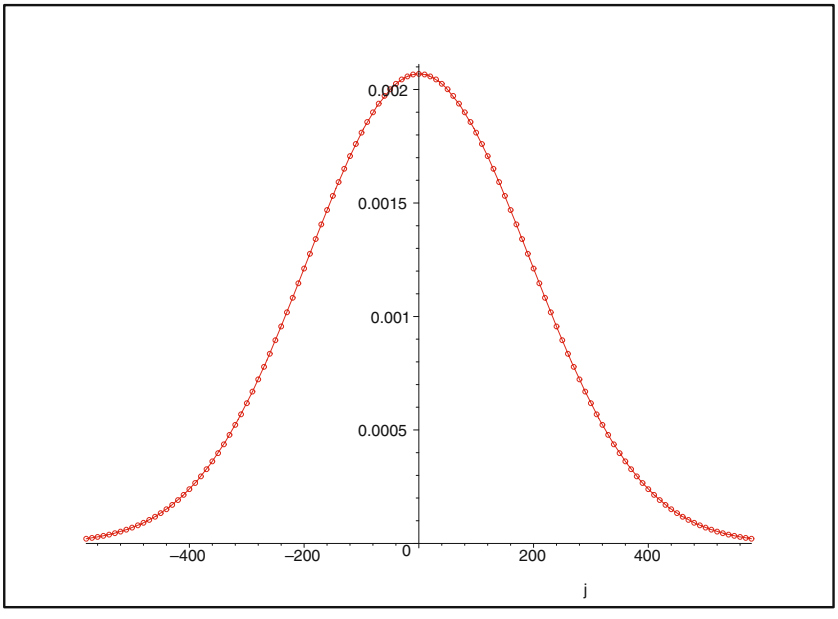


Fig. 2. $P_n(j)$ (circle) and the Gaussian asymptotics with the constant term (line), $n = 60$

setting $d\tau = \frac{d\tau}{du}du$, expanding w.r.t. n and integrating on $[u = -\infty..\infty]$. (This amounts to the *reversion* of a series.) Finally (4) leads to

$$P_n(j) \sim e^{-x^2/2} \cdot \exp\left(\left[-\frac{39}{40} + \frac{9}{20}x^2 - \frac{3}{40}x^4\right]\Big/ n + \mathcal{O}(n^{-3/2})\right)/(2\pi n^3/6)^{1/2}. \quad (6)$$

Note that $S^{(3)}(\tilde{z})$ does not contribute to the $1/n$ correction. Note also that, unlike in the instance of the number of inversions in permutations (see [5]), we have an x^4 term in the first order correction. Let us finally remark that Clark's result is just the main term of (6), with $x = 0$, as he chooses $m = o(\sigma) = o(n^{3/2})$.

To check the effect of the correction, we first give in Figure 1, for $n = 60$, the comparison between $P_n(j)$ and the asymptotics (6), without the $1/n$ term (i.e. the Gaussian contribution). To get a better quality estimation, Figure 2 shows $P_n(j)$ and the asymptotics (6), with the constant term $-39/(40n)$. Figure 3 shows the quotient Q_1 of $P_n(j)$ and the asymptotics (6), without the $1/n$ term. The usual "hat" behaviour is apparent.

Figure 4 shows Q_1 and Q_2: the quotient of $P_n(j)$ and the asymptotics (6), with the constant term $-39/(40n)$.

Figure 5 shows Q_2 and Q_3: the quotient of $P_n(j)$ and the asymptotics (6), with the constant term $-39/(40n)$ and the x^2 term $9x^2/(20n)$.

Finally, Figure 6 shows Q_3 and Q_4: the quotient of $P_n(j)$ and the full asymptotics (6). Q_4 gives indeed a good precision on a larger range.

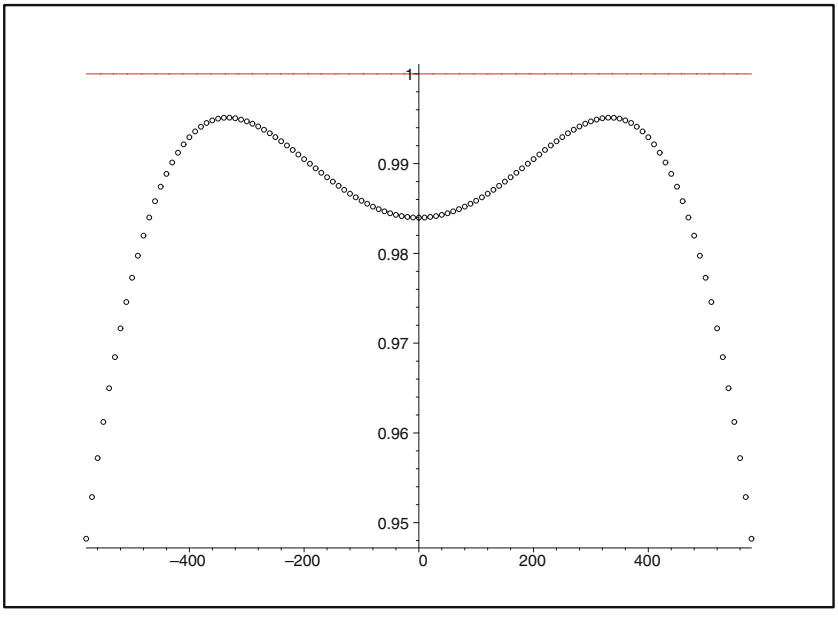


Fig. 3. Q_1

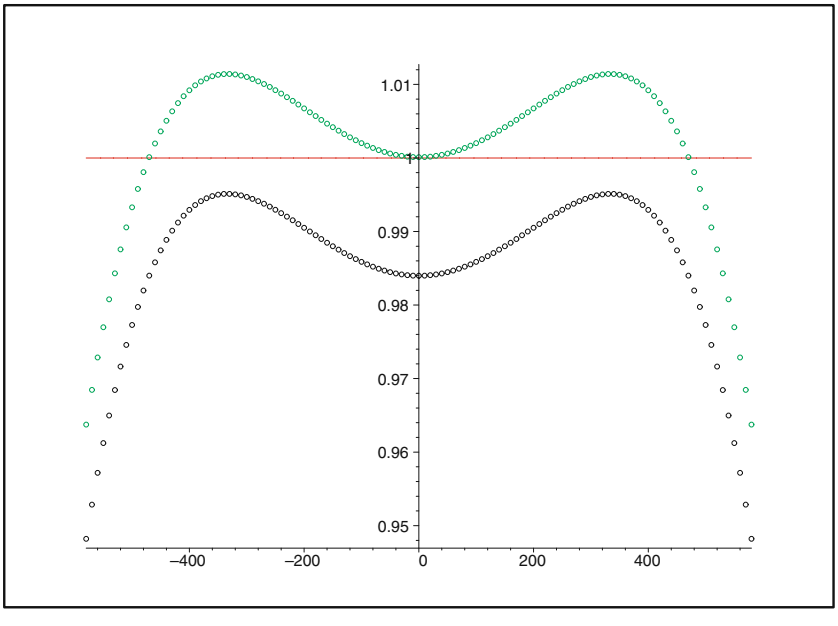


Fig. 4. Q_1 (black) and Q_2 (green)

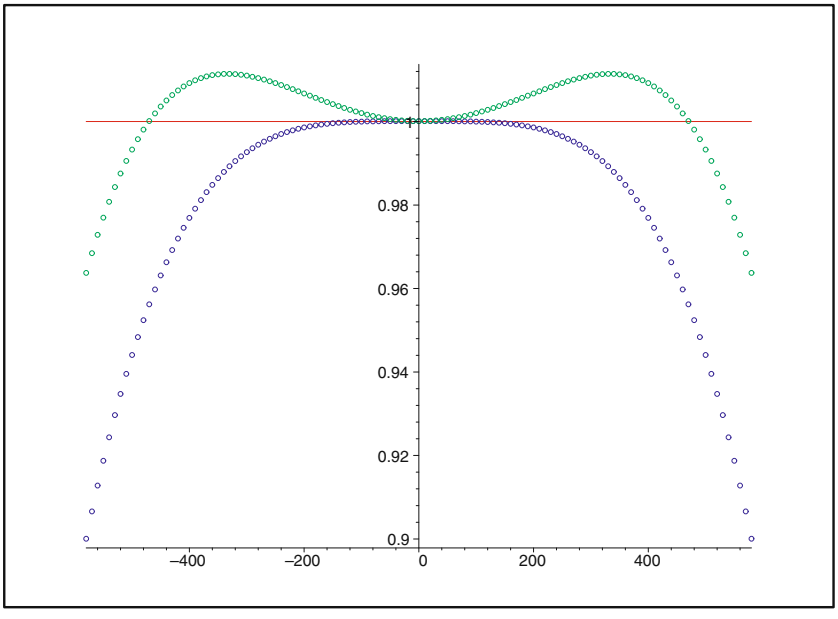


Fig. 5. Q_2 (green) and Q_3 (blue)

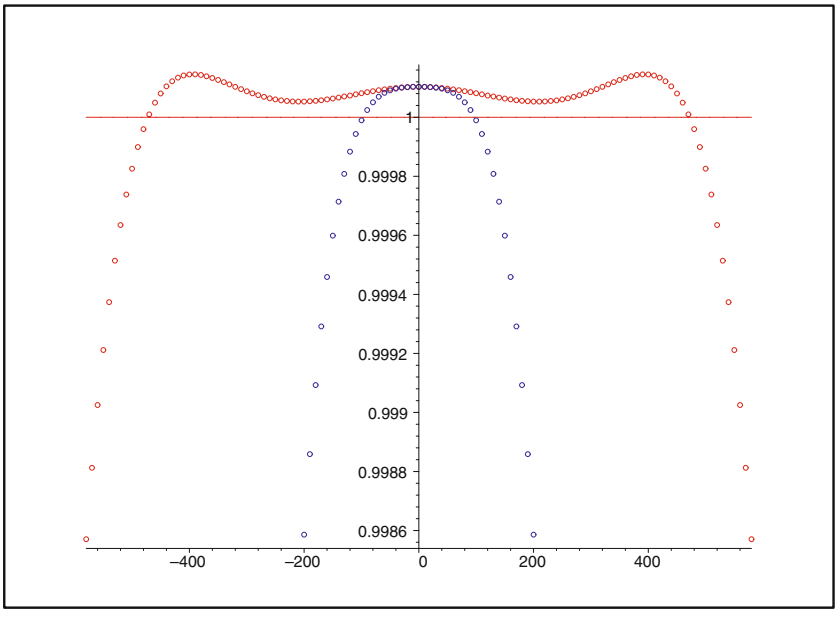


Fig. 6. Q_3 (blue) and Q_4 (red)

For the reader's convenience, here are the quantities that we plot:

$$Q_1 = P_n(j) \Big/ \left[e^{-x^2/2}/(2\pi n^3/6)^{1/2}\right],$$

$$Q_2 = P_n(j) \Big/ \left[e^{-x^2/2} \cdot \exp[-\tfrac{39}{40}n]/(2\pi n^3/6)^{1/2}\right],$$

$$Q_3 = P_n(j) \Big/ \left[e^{-x^2/2} \cdot \exp[(-\tfrac{39}{40} + \tfrac{9}{20}x^2)/n]/(2\pi n^3/6)^{1/2}\right],$$

$$Q_4 = P_n(j) \Big/ \left[e^{-x^2/2} \cdot \exp[(-\tfrac{39}{40} + \tfrac{9}{20}x^2 - \tfrac{3}{40}x^4)/n]/(2\pi n^3/6)^{1/2}\right].$$

3 The Case $j = n^{7/4} - x$

Now we show that our methods are strong enough to deal with probabilities that are far away from the average; viz. $n^{7/4} - x$, for fixed x. Of course, they are very small, but nevertheless we find asymptotic formulæ for them. Later on, $n^{7/4} - x$ will be used as an integer. Equation (3) becomes now

$$-n^{7/4} + x - 1 + \Big(-\frac{(n+1)^3}{6} + \frac{(n+1)^2}{4} - \frac{n}{12} - \frac{13}{12} + x - n^{7/4}\Big)\varepsilon = 0.$$

We have $z^* = 1$ and, as we will see, ε is now given by a *Puiseux series* of powers of $n^{-1/4}$. We obtain

$$\varepsilon = -6n^{-5/4} - \frac{54}{5}n^{-7/4} - \frac{3771}{175}n^{-9/4} + \cdots$$

Again we provide only a few terms, but we mention that more terms than in the previous section are usually necessary. This leads to

$$S_1(\tilde{z}) = 3\sqrt{n} + \frac{81}{10} + \frac{1467}{70\sqrt{n}} + \cdots,$$

$$S_2(\tilde{z}) = -6\sqrt{n} - \frac{54}{5} - \frac{3771}{175\sqrt{n}} + \cdots,$$

and so

$$S(\tilde{z}) = -3\sqrt{n} - \frac{27}{10} - \frac{207}{350\sqrt{n}} + \cdots.$$

Let us mention that x appears only in the $n^{-5/4}$ term and x^2 appears (first) in the n^{-3} term. Also,

$$S^{(2)}(\tilde{z}) = \frac{1}{6}n^3 - \frac{9}{10}n^{5/2} + \frac{607}{700}n^2 + \cdots,$$

$$S^{(3)}(\tilde{z}) - \frac{3}{10}n^{15/4} + \frac{711}{350}n^{13/4} + \cdots,$$

$$S^{(4)}(\tilde{z}) = -\frac{1}{20}n^5 + \cdots,$$

$$S^{(5)}(\tilde{z}) = \mathcal{O}(n^{23/4}).$$

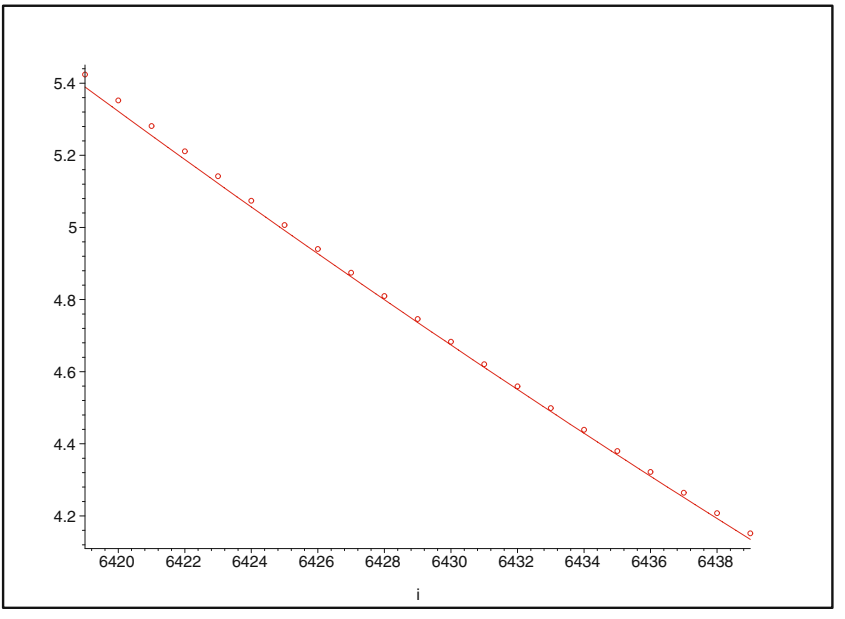


Fig. 7. $P_n(j)$ (circle) and the full asymptotics (7) (line), $n = 150$, scaling$= 10^{21}$

This finally gives

$$P_n(j) \sim e^{-3n^{1/2}-27/10}\Big[1 + 369/175/n^{1/2} + 931359/245000/n + 1256042967/471625000/n^{3/2} + 4104x/175/n^{7/4} - 9521495156603/2145893750000/n^2 + 7561917x/122500/n^{9/4} + (-235974412468129059/341392187500000 + 18x^2)/n^{5/2} + \cdots\Big] \Big/ (2\pi n^3/6)^{1/2}. \quad (7)$$

Figure 7 shows, for $n = 150$, $P_n(j)$ and the full asymptotics (7) up to the $n^{-5/2}$ term, both scaled to 10^{21}. The fit is quite satisfactory.

References

1. Clark, L.: On the representation of m as $\sum_{k=-n}^{n} \epsilon_k k$. Int. J. Math. Math. Sci. 23, 77–80 (2000)
2. Entringer, R.C.: Representation of m as $\sum_{k=-n}^{n} \varepsilon_k k$. Canad. Math. Bull. 11, 289–293 (1968)
3. Feller, W.: Introduction to Probability Theory and its Applications, vol. I. Wiley, Chichester (1968)
4. Flajolet, P., Sedgewick, B.: Analytic combinatorics. Cambridge University Press, Cambridge (2009)
5. Louchard, G., Prodinger, H.: The number of inversions in permutations: A saddle point approach. J. Integer Seq. 6, 03.2.8 (2003)
6. Odlyzko, A.: Graham, R., Götschel, M., Lovász, L. (eds.): Asymptotic Enumeration, pp. 1063–1229. Elsevier Science, Amsterdam (1995)
7. van Lint, J.H.: Representation of 0 as $\sum_{k=-N}^{N} \varepsilon_k k$. Proc. Amer. Math. Soc. 18, 182–184 (1967)

Sets of Infinite Words Recognized by Deterministic One-Turn Pushdown Automata

Douadi Mihoubi*

Laboratoire des Mathématiques Pures et Appliquées, Département de Mathématique, Université de M'sila, Algeria
Mihoubi_douadi@yahoo.fr

Abstract. In this paper we consider deterministic pushdown automata on infinite words with restricted use of the stack. More precisely, this study concerns: (1) Behavior of deterministic one-turn pushdown automata using Büchi and Muller modes of acceptance and (2) Closure properties of these sets by Boolean and limit operators.

1 Introduction

The notion of recognizing infinite words by finite automata was originally due to Büchi [1], in order to prove the decidability of the monadic second order theory of natural numbers. The basic result proved by Büchi is:

> The behavior of a finite automaton is a finite union of sets of the form KR^{ω} where K and R are regular languages of finite words.

However, deterministic Büchi automata are not able to recognize all ω-regular languages. The solution for this question was given by Muller in [10]. The basic result in this perspective was given by McNaughton in [7] who states that any recognizable set of Σ^{ω} can be recognized by an appropriate deterministic Muller automaton. This implies in particular that the class $\mathrm{Rat}\,(\Sigma^{\omega})$ of ω-regular languages is closed under complementation.

A similar theory of sets of infinite words recognized (generated) by pushdown automata (algebraic grammars) was initially studied by Cohen and Gold in [2,3] and Nivat in [11]. Two main characterizations of the family $\mathrm{Alg}\,(\Sigma^{\omega})$ of algebraic languages of infinite words were obtained by the authors:

Theorem 1. *A set X of infinite words is ω-algebraic, i.e., X belongs to* $\mathrm{Alg}\,(\Sigma^{\omega})$ *if and only if it can be represented in the form* $X = \bigcup_{i=1}^{n} A_i B_i^{\omega}$, *where A_i and B_i are algebraic languages of finite words.*

Theorem 2. *The class* $\mathrm{Pda}\,(\Sigma^{\omega})$ *of sets of infinite words recognized by pushdown automata is equal to the class* $\mathrm{Alg}\,(\Sigma^{\omega})$.

* I would like to express my gratitude to the editors. I especially wish to thank the anonymous referee for helpful suggestions and corrections. (*Received by the editors.* 26 January 2008. *Revised.* 15 September 2008. *Accepted.* 26 November 2008.)

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, pp. 97–108, 2009.
© Springer-Verlag Berlin Heidelberg 2009

This paper is the second part of the author's previous work [8] on ω-languages recognized (generated) by one-turn pushdown automata (linear grammars). In [8], sub-classes of the family $\mathrm{Alg}(\Sigma^\omega)$ were studied, viz.

- The class, denoted $\mathrm{Pic}(\Sigma^\omega)$, of ω-languages accepted by a one-turn pushdown automata, and
- The class $\mathrm{Lin}(\Sigma^\omega)$ of ω-languages generated by linear ω-grammars.

It is shown that no equality between these classes exists. An ω-language L in $\mathrm{Pic}(\Sigma^\omega)$ takes the form $L = \bigcup_{i=1}^{n} L_i R_i^\omega$, where for all integers $i = 1, ..., n$; L_i is a linear language and R_i is a regular language of finite words. However, the class $\mathrm{Lin}(\Sigma^\omega)$ is equal to the class $\mathrm{Rat}(\Sigma^\omega)$ of ω-regular languages.

This shows in particular that the family $\mathrm{Pic}(\Sigma^\omega)$ does not respect the definition of ω-Kleene closure, where for any family $\mathcal{F}$ of sets over the alphabet Σ, the ω-Kleene closure of $\mathcal{F}$, denoted ω-KC($\mathcal{F}$), is equal to the set

$$\left\{ L \subseteq \Sigma^\omega \mid L = \bigcup_{i=1}^{n} U_i V_i^\omega \text{ for some } U_i,\, V_i \in \mathcal{F},\, i = 1, ...k;\, k = 1, 2, ... \right\}.$$

This article is about ω-languages recognized by one-turn deterministic pushdown automata with Büchi acceptance conditions for the most part, and with Muller acceptance conditions for the last part.

The paper is organized as follows: Basic notions and preliminaries on infinite words are presented in § 2. After that, § 3 is devoted to give the analysis and synthesis theorems of behaviors of one-turn deterministic pushdown automaton using Büchi acceptance conditions. In § 4, some closure properties of these sets are studied. Finally, § 5 is about ω-languages recognized by one-turn deterministic pushdown automata with Muller acceptance conditions.

2 Preliminaries

Let Σ be a finite set of symbols which we call an alphabet and the elements of Σ letters. A finite word over Σ is a finite sequence $w = (a_1, a_2, ..., a_n)$ of elements of Σ denoted by juxtaposition or concatenation by $w = a_1 a_2 ... a_n$. The integer $n = |w|$ is the length of the word w. The empty sequence () of length 0 is called the empty word and is denoted by ε. The set Σ^* of all words over Σ equipped with the operation of concatenation defined by

$$a_1 a_2 ... a_n \circ b_1 b_2 ... b_m = a_1 a_2 ... a_n b_1 b_2 ... b_m$$

has the structure of a monoid with the empty word ε as a neutral element, called the free monoid on Σ. A language over Σ is any subset of Σ^*, which permits naturally the use of boolean operations over the set of languages $\wp(\Sigma^*)$. The concatenation product of Σ^* may be extended to $\wp(\Sigma^*)$ by defining $AB = \{ab \mid a \in A,\, b \in B\}$ for A, B in $\wp(\Sigma^*)$.

We write $\mathbb{N}^+ = \{1, 2, ...\}$. An infinite word over Σ is an infinite sequence $u : \mathbb{N}^+ \to \Sigma$. We denote by $u[n]$ the finite word $u(1)\,u(2)\,...u(n)$ representing the initial segment of u of length n. We denote by Σ^ω the set of infinite words over the alphabet Σ. We let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ be the set of finite or infinite words on Σ.

A word $\beta \in \Sigma^\infty$ is a left factor of a word $\alpha \in \Sigma^\infty$, denoted by $\beta \leq \alpha$, if there exists $\gamma \in \Sigma^\infty$ such that $\alpha = \beta\gamma$. The relation $\leq$ is a partial order on Σ^∞ called the prefix ordering. We denote by $\mathrm{LF}(\alpha)$ the set of finite left factors of $\alpha \in \Sigma^\infty$. Hence,

$$\mathrm{LF}(\alpha) = \{g \mid g \leq \alpha\} \text{ if } \alpha \in \Sigma^*, \text{ and}$$

$$\mathrm{LF}(\alpha) = \{\alpha[n] \mid n \in \mathbb{N}^*\} \text{ if } \alpha \in \Sigma^\omega.$$

We call any subset L of Σ^* a *language* and any subset L of Σ^ω an *ω-language*. For any $L \subset \Sigma^\infty$, we let $\mathrm{LF}(L) = \{\mathrm{LF}(\alpha) \mid \alpha \in L\}$ denote the set of left factors of words in L.

Given an increasing prefix ordering sequence $u_0 \leq u_1 \leq ... \leq u_n \leq ...$, of elements $u_i \in \Sigma^\infty$, there exists a unique element $\alpha = \lim_{n\to\infty} u_n$ such that: $k = |u_n|$ and $\alpha[k] = u_n$. Then $\alpha = u_n \in \Sigma^*$ if for all $i \geq n$, we have $u_{i+1} = u_i$ or $\alpha \in \Sigma^\omega$. For any language $A \subset \Sigma^*$, we denote the *infinite iteration of A* by

$$A^\omega = \left\{ u \in \Sigma^\omega \mid \text{for all } i \in \mathbb{N}^+, \text{ we have } u = \prod_{i\geq 1} u_i \text{ with } u_i \in A - \{\varepsilon\} \right\}.$$

If $A = \{\varepsilon\}$ then $A^\omega = \{\varepsilon\}$. We write

$$\overrightarrow{A} = \lim A = \{u \in \Sigma^\omega \mid \text{for arbitrarily large } n, \text{ we have } u[n] \in A\}$$

for the closure of A. In others words, $x \in \overrightarrow{A}$ if there is an increasing prefix ordering sequence $(a_n)_{n\in\mathbb{N}}$ of words in A such that: $x = \lim_{n\longrightarrow\infty} a_n$. A set $A \subset \Sigma^*$ is called prefix, if for all words x, y in A, if $x \leq y$ then $x = y$. In other words, the set A satisfies $A(\Sigma^* - \{\varepsilon\}) \cap A = \varnothing$.

Example 3. *For the set $X = ba^*$ over $\Sigma = \{a, b\}$, we have $X^\omega = (ba^*)^\omega$ and $\overrightarrow{X} = ba^\omega$. The set $Y = a^*b$ is a prefix, but the set $X = ba^*$ is not.*

Proposition 4. *Let $A, B \subset \Sigma^*$. Then*

1. $\overrightarrow{A \cup B} = \overrightarrow{A} \cup \overrightarrow{B}$,
2. *if A is prefix then $\overrightarrow{AB} = A\overrightarrow{B}$ with $\overrightarrow{A} = \varnothing$, and*
3. $A^\omega \subset \overrightarrow{A^*}$ *and if A is prefix then* $A^\omega = \overrightarrow{A^*}$.

Example 5. *For the sets X and Y in Example 3, we have $\overrightarrow{Y} = \varnothing$ (because Y is a prefix) and then $\overrightarrow{YX} = Y\overrightarrow{X} = a^*b\overrightarrow{ba^*} = a^*bba^\omega = a^*b^2a^\omega$. The set $Z = (a + b)^*a^\omega$ is not of the form $\overrightarrow{L}$.*

3 Infinite Words and Deterministic One-Turn Pushdown Automata

In this section, we study the behavior of one-turn deterministic pushdown automaton with Büchi acceptance conditions. At first, we give the definition of a one-turn deterministic pushdown automaton with Büchi acceptance conditions, then we establish the theorems of analysis and synthesis for this machine in an ω-computation.

Definition 6. *A one-turn pushdown automaton (1PDA) is the 7-tuple*

$$M = (K, \Pi = \{K_1, K_2\}, \Sigma, \Gamma, \delta, q_0, Z_0)$$

where

1. *K is a finite set of states, $\Pi = \{K_1, K_2\}$ is a partition of K, $q_0 \in K$ is the start state,*
2. *Γ is a finite stack alphabet, $Z_0 \in \Gamma$ is the start symbol,*
3. *Σ is a finite set of input alphabet,*
4. *δ is a transition function defined from $K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to a finite subset of $K \times \Gamma^*$. The function δ takes as its argument a triple $\delta(q, a, Z)$, where*
   (a) q is a state in K,
   (b) a is an input symbol in Σ or $a = \varepsilon$, the empty string,
   (c) Z is a stack symbol.
   The output of δ is a finite set of pairs (p, γ) where
   (a) p is a new state,
   (b) γ is a string of stack symbols that replaces Z at the top of the stack.

The function δ is K_1-increasing and K_2-decreasing:

K_1-increasing: The restriction δ/K_1 is stack increasing, i.e., for all q, $p \in K_1$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, if $(p, \gamma) \in \delta(q, b, Z)$, then $|\gamma| \geq 1$. For all $q \in K_1$, $p \in K_2$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, if $(p, \gamma) \in \delta(q, a, Z)$ then $\gamma = \varepsilon$.

K_2-decreasing: The restriction δ/K_2 is stack decreasing, i.e., for all q, $p \in K_2$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, if $(p, \gamma) \in \delta(q, a, Z)$ then $|\gamma| \leq 1$, where δ/K_i, for $i = 1, 2$, denotes the restriction of δ at states of K_i.

A configuration of a 1PDA is a triple (q, x, γ), where $q \in K$ is the current state, x is the unread portion of the input string, and $\gamma \in \Gamma^*$ is the content of the stack. At first, the automaton is in the initial configuration (q_0, u, Z_0) where u is a word in Σ^∞. For all $a \in \Sigma \cup \{\varepsilon\}$, γ, $\beta \in \Gamma^*$ and $Z \in \Gamma$, if $(p, \beta) \in \delta(q, a, Z)$ then we write $(q, Z\gamma) \overset{a}{\longmapsto} (p, \beta\gamma)$.

The meaning of such a transition is that when the machine is in state q, the top symbol on the store is Z and the input symbol is a, then the next state is p and the word $\beta \in \Gamma^*$ is written on the top of the store instead of Z. We denote by $\overset{*}{\longmapsto}$ the reflexive and transitive closure of the relation $\longmapsto$.

Finally, a 1PDA M is said to be deterministic (1DPDA), if and only if the following conditions are satisfied:

1. $\delta(q, a, Z)$ has at most one member for any q in Q, a in Σ or $a = \varepsilon$, and Z in Γ.
2. If $\delta(q, a, Z)$ is nonempty for some a in Σ, then $\delta(q, \varepsilon, Z)$ must be empty.

Intuitively, the machine is deterministic if there is no choice of moving in any situation. Consequently there is at most one run of the machine for any word in Σ^∞.

Definition 7. *Let M be a 1PDA and let $\sigma = \prod_{i=1}^{\infty} a_i \in \Sigma^\omega$, where $a_i \in \Sigma$ for all $i \geq 1$. An infinite sequence of configurations $r = \{(q_i, \gamma_i)\}_{i \geq 1}$ is called a complete run of M on σ, starting in configuration (q_0, σ, Z_0), iff*

1. *$(q_1, \gamma_1) = (q_0, Z_0)$, and*
2. *for each $i \geq 1$, there exists a $b_i \in \Sigma \cup \{\varepsilon\}$ satisfying $(q_i, \gamma_i) \overset{b_i}{\longmapsto} (q_{i+1}, \gamma_{i+1})$ such that $\prod_{i=1}^{\infty} b_i = \prod_{i=1}^{\infty} a_i$.*

Every such run induces a mapping $r : \mathbb{N}^+ \longrightarrow K$ such that $r(i) = q_i$, where $r(i)$ indicates the state entered in the step i of the run r. We denote by $\mathrm{Inf}(r)$ the set of all states entered infinitely many times in the run r.

Definition 8. *A one-turn deterministic pushdown automaton with Büchi's conditions (1DPDAB) is a couple $M = (M', F)$ where*

1. *M' is a 1DPDA,*
2. *$F \subseteq K$ is a subset of designated final states.*

The infinite word $\sigma \in \Sigma^\omega$ is accepted by M if and only if there exists a unique infinite run r of M on σ starting in configuration (q_0, Z_0) such that $\mathrm{Inf}(r) \cap F \neq \emptyset$.

We denote by $L^\omega(M)$ the set of all infinite words in Σ^ω accepted by M and by $\mathrm{1DPDAB}(\Sigma^\omega)$ the class of ω-languages accepted by a 1DPDAB.

We give now a sequence of lemmas which leads to the theorem of analysis. First, we treat the case when all the subsets F of final states are in K_1.

3.1 The Case When the Subset F of Final States is in K_1

Lemma 9. *Let $M = (M', F)$ be a 1DPDAB. As long as M does not pop out any letter, it behaves like some finite automaton, consequently $L^\omega(M) = \bigcup_{i=1}^{n} K_i R_i^\omega$, where K_i and R_i are regular languages for all $i = 1, ..., n$.*

Proof. Let $M = (M', F)$ be a 1DPDAB, where

$$M' = (K, \Pi = \{K_1, K_2\}, \Sigma, \Gamma, \delta, q_0, Z_0)$$

is a 1DPDA. Since M does not pop out any letter by hypothesis, and the function of transition δ is not defined from K_2 to K_1, then the automaton $M = (M', F)$ is equivalent to the automaton $M_1 = (M_1', F)$, where M_1' is the restriction of M' to the states K_1, i.e., $M_1' = (K_1, \Pi = \{K_1, \emptyset\}, \Sigma, \Gamma, \delta/K_1, q_0, Z_0)$.

Since δ/K_1 is always increasing, the automaton M_1 is equivalent to the automaton $M_2 = (K_1, \Pi = \{K_1, \emptyset\}, \Sigma, \Gamma, \delta_2, q_0, Z_0, F)$, where δ_2 is defined by $(q', z') \in \delta_2(q, b, z)$ if $(q', z'\gamma') \in \delta/K_1(q, b, z)$ for $(q, b, z) \in K_1 \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ (the bottom γ' is not necessary in this case). Finally, the automaton M_2 is identical to a finite state automaton $A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$ with Büchi's conditions where

- $Q_A = \{(q, z)/q \in K_1, z \in \Gamma\}$, $q_{0A} = (q_0, z_0)$.
- δ_A is defined by $\delta_A((q, z), a) = (q_1, z_1)$ if $\delta_2(q, a, z) = (q_1, z_1)$, and
- $F_A = \{(q_{f_i}, z_i) \in F \times \Gamma/\delta_A(q_{f_i}, z_i)\}$.

According to the Büchi-Schutzenberger Theorem, we conclude that

$$L^{\omega}(A) = \bigcup_{i=1}^{n} K_i R_i^{\omega},$$

where $n = |F_A|$ and for all $i = 1, ..., n$, K_i and R_i are regular languages of finite words, with

$$K_i = \left\{ x \in \Sigma^*/((q_0, z_0), x) \xrightarrow{*} ((q_{f_i}, z_i), \varepsilon) \text{ and } (q_{f_i}, z_i) \in F_A \right\}$$

and

$$R_i = \left\{ x \in \Sigma^*/((q_{f_i}, z_i), x) \xrightarrow{*} (q_{f_i}, z_i), \varepsilon \right\}. \qquad \blacksquare$$

Before treating the case where the set of final states is in K_2, we recall from [8] this technical lemma.

Lemma 10. *Let $M = (M_1, F)$ be an 1DPDAB. If M starts to pop out any letter from the stack content, then this stack content will eventually be constant.*

In other words, if $r = (q_i, \gamma_i)_{i \geq 0}$ is an infinite run of M which enters states of K_2, and since the restriction δ/K_2 is decreasing in the domain K_2, then necessarily there is a step t in the run r after which the level of the stack content will be constant.

3.2 The Case When the Subset of Final States is in K_2

Lemma 11. *Let $M = (M', F)$ be a 1DPDAB. If M starts to pop out any letter, then the language it will ultimately recognize is a finite union of languages of the form AB^{ω} where A is a prefix linear language and B is a prefix rational language.*

Proof. Intuitively, if the automaton M starts to pop out any letter, then the language it will ultimately recognize is a finite union of languages of the form AB^{ω} where A is a linear language that leads the machine to some accepting state from which it will behave like some finite automaton looping on this very same accepting state, and B is the induced rational language.

In fact, let L be in 1DPDAB(Σ^ω), then $L = L^\omega(M)$ for some a 1DPDAB $M = (K, \Pi = \{K_1, K_2\}, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. We set $D = \{(q_f, Z) \mid q_f \in F \text{ and } Z \in \Gamma\} = F \times \Gamma$. For all $(q_f, Z) \in D$, we define:

$$A_i = A_{(q_f,Z)} = \{x \in \Sigma^* \mid (q_0, x, Z_0) \stackrel{*}{\longmapsto} (q_f, \varepsilon, Z\gamma) \text{ such that } C_1(q_f, x, Z\gamma)\}$$

where $C_1(q_f, x, Z\gamma)$ denotes that the following condition is satisfied:

$(q_f, Z\gamma)$ is the first configuration entered after reading the word x and satisfying that there is no $\gamma' \in \Gamma^*$ a proper suffix of γ, $q \in K_2$, $x' \in \Sigma^*$ such that

$$(q_f,\, x',\, Z\gamma) \stackrel{*}{\longmapsto} (q, \varepsilon, \gamma')\}. \qquad (C_1)$$

Intuitively, the condition C_1 means that the content of the stack can not strictly decrease after the configuration $(q_f, Z\gamma)$. Such a configuration exists according to Lemma 10. We also define

$$B_i = B_{(q_f,Z)} = \{x \in \Sigma^* \mid (q_f, x, Z\gamma)\} \stackrel{*}{\longmapsto} (q_f, \varepsilon, Z\gamma) \text{ such that } C_2(q_f, x, Z\gamma)\}$$

where $C_2(q_f, x, Z\gamma)$ denotes that the following condition is satisfied:

the pair (q_f, Z) is entered exactly two times in the run of M on x. (C_2)

So, we obtain:

$$L^\omega(M) = \bigcup_{i=1}^{n} A_i B_i^\omega = \bigcup_{(q_f,Z)\in D} A_{(q_f,Z)} B_{(q_f,Z)}^\omega$$

where $n = |D|$ is the size of D, $A_{(q_f,Z)}$ is a prefix linear language, and $B_{(q_f,Z)}$ is a prefix rational language.

In order to prove that for all (q_f,Z), $A_{(q_f,Z)}$ (resp. $B_{(q_f,Z)}$) is linear (resp. rational), since the level of the stack in the automaton M cannot decrease infinitely, it is sufficient to simulate M by two automata (M_1, M_2) with M_1 being a 1DPDA and M_2 being a finite state automaton. Intuitively, the first copy M_1 operates on a finite word and halts at a configuration where the level of the stack can not change to let a hand to the second copy M_2 for scanning the rest of the run. We obtain

$$A_{(q_f,Z)} B_{(q_f,Z)}^\omega = L(M_1)\,(L(M_2))^\omega$$

where $L(M_1)$ is a linear language and $L(M_2)$ is a regular language.

Now, let us suppose that $A_{(q_f,Z)}$ is not a prefix, so, there exists $x, y \in A_{(q_f,Z)}$ such that $x \leq y$ and $x \neq y$. Then $x \in A_{(q_f,Z)}$ if and only if $(q_0, x, Z_0) \stackrel{*}{\longmapsto} (q_f, \varepsilon, Z\gamma)$ such that the condition $C_1(q_f, x, Z\gamma)$ is verified. It is also verified that $y \in A_{(q_f,Z)}$ if and only if $(q_0, y, Z_0) \stackrel{*}{\longmapsto} (q_f, \varepsilon, Z\gamma')$ such that the condition

$C_1(q_f, Z\gamma')$ holds. Since $y = xx'$ with $x' \neq \varepsilon$ and M is deterministic, then necessarily

$$(q_0, y = x.x', Z_0) \overset{*}{\longmapsto} (q_f, \varepsilon.x', Z\gamma) \overset{*}{\longmapsto} (q_f, \varepsilon, Z\gamma')$$

which contradicts condition C_1 (the pair $(q_{f,Z})$ is entered two times in the run of M on y), so $x = y$. Thus, $A_{(q_{f,Z})}$ is a prefix and linear. For the second case, in order to prove that $B_{(q_{f,Z})}$ is rational, it is sufficient to remark that, in the automaton M_2, only the first symbol of the store can change, then we can construct a finite automaton M' from M_2 which accepts the language $B_{(q_{f,Z})}$. The proof that $B_{(q_{f,Z})}$ is a prefix is similar to the one of $A_{(q_{f,Z})}$ changing the condition C_1 to the condition C_2. Finally, it's easy to see that

$$L^{\omega}(M) = \bigcup_{(q_{f,Z}) \in D} A_{(q_{f,Z})} B^{\omega}_{(q_{f,Z})}. \qquad \blacksquare$$

Theorem 12 (Analysis Theorem). *For all ω-languages $L \in \text{1DPDAB}(\Sigma^{\omega})$ there exist four sequences (K_i), (R_i), for $i = 1, 2, ..., n$, and (A_j), (B_j), for $j = 1, 2, ..., m$, where*

1. *for all $i = 1, 2, ..., n$, K_i, R_i are regular languages of finite words,*
2. *for all $j = 1...., m$, A_j is a prefix linear language and B_j is a prefix regular language, and*
3. *$R_i^{\omega} \oplus \bigcup_{j=1}^{m} A_j B_j^{\omega}$.*

Proof. Let $L \in \text{1DPDAB}(\Sigma^{\omega})$, then there exists a 1DPDAB $M = (M', F) = (K, \Pi = \{K_1, K_2\}, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ such that $L = L^{\omega}(M)$. Let $F = F_1 \cup F_2$ with $F_1 \subset K_1$ and $F_2 \subset K_2$ (necessarily $F_1 \cap F_2 = \emptyset$). We consider then the automata M_1, M_2 deduced from M with

$$M_1 = (K, \Pi = \{K_1, \emptyset\}, \Sigma, \Gamma, \delta/K_1, q_0, Z_0, F_1)$$

and $M_2 = (M', F_2)$. By Lemma 9, we can simulate M_1 by a Büchi finite automaton $A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$ and then $L^{\omega}(M_1) = L^{\omega}(A) = \bigcup_{i=1}^{n} K_i R_i^{\omega}$, where K_i, R_i are regular languages of finite words. The behavior of M_2 is, according to Lemma 10, $L^{\omega}(M_2) = \bigcup_{j=1}^{m} A_j B_j^{\omega}$, where A_j is a prefix linear language and B_j is a prefix rational language for all $j = 1, ..., m$. It is clear then that $L^{\omega}(M) = L^{\omega}(M_1) \cup L^{\omega}(M_2)$ with $L^{\omega}(M_1) \cap L^{\omega}(M_2) = \emptyset$. $\blacksquare$

In contrary to no deterministic case, where it was established in [8] that the ω-language L is recognized by a Büchi one-turn pushdown automaton if and only if $L = \bigcup_{i=1}^{n} A_i B_i^{\omega}$, where for all $i = 1, 2, ..., n$, A_i is a linear language and B_i is a rational language, the converse of the theorem [2] is not always true.

Theorem 13 (Synthesis Theorem). *Let $(A_i)_{i=1,...,n}$ be a sequence of prefix linear languages and $(B_i)_{i=1,...,n}$ be a sequence of prefix rational languages, the ω-language $L = \bigcup_{i=1}^{n} A_i B_i^{\omega}$ is not necessarily in* 1DPDAB(Σ^{ω}).

Proof. It is sufficient to consider the language $A = \{a^n b^n c : n \in \mathbb{N}\} \cup \{a^n b^{2n} : n \in \mathbb{N}\}$ which is linear and prefix and the rational language $B = \{a\}$. The language $L = AB^\omega$ satisfies the conditions of Theorem 12. But L is not deterministic. ■

4 Closure Properties

In this section, we study some closure properties of the class 1DPDAB (Σ^ω). It is shown that this class is not closed under Boolean operations and proper morphism.

Proposition 14. *The family* 1DPDAB (Σ^ω) *is strictly included in the family* 1PDAB (Σ^ω).

Proof. Obvious, since any deterministic one-turn PDA is also a one-turn PDA. The ω*-language* $(a+b)^* a^\omega$ *which is in* 1PDAB (Σ^ω) *and not in* 1DPDAB (Σ^ω), *shows that this inclusion is strict.* ■

We introduce the notion of trim one-turn deterministic pushdown automata in order to show that LF (L) is a deterministic linear language if L is in 1DPDAB(Σ^ω).

Let $M = (K, \Pi = \{K_1, K_2\}, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a 1DPDAB. We recall here that a state $q \in K$ is accessible (coaccessible) in M if there exists a run $r : (q_0, x, Z_0) \overset{*}{\longmapsto} (q, \varepsilon, \gamma)$ (a run $r : (q, x, \gamma) \overset{*}{\longmapsto} (t, \varepsilon, \gamma)$ with $t \in F$). An automaton is trim if each state is both accessible and coaccessible. We denote the set of pairs (state, symbol of push) accessible from (q, Z) by $\mathrm{Acc}\,(q, Z) = \{(q', Z') \in K \times \Gamma \mid \exists x \in \Sigma^* : (q, x, Z) \overset{*}{\longmapsto} (q', \varepsilon, Z'\gamma') \text{ with } \gamma' \in \Gamma^*\}$ and those coaccessible from (q, Z) by $\mathrm{Coacc}\,(q, Z) = \{(q', Z') \in K \times \Gamma \mid \exists y \in \Sigma^* : (q', y, Z') \overset{*}{\longmapsto} (q, \varepsilon, Z\gamma) \text{ with } \gamma \in \Gamma^*\}$.

Proposition 15. *To any Büchi one-turn PDA* M, *one may associate a trim Büchi one-turn PDA* M' *such that*

1. *The automata* M *and* M' *recognize the same subset of* Σ^ω, *and*
2. *If* M *is deterministic, so is* M'.

Proof. Let $M = (K, \Pi = \{K_1, K_2\}, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a one-turn PDA with Büchi's condition. It is sufficient then to take the restriction of M to the set of pairs Δ which are accessible and coaccessible where

$$\Delta = \mathrm{Acc}\,(q_0, Z_0) \cap \left\{ \bigcup_{(q_f, Z) \in F \times \Gamma} \mathrm{Coacc}\,(q_f, Z) \right\}. \qquad \blacksquare$$

Proposition 16. *For all* ω*-language* L *in* DBPIC (Σ^ω), *the set* LF (L) *of left factors of* L *is recognized by a deterministic one-turn pushdown automaton.*

Proof. Let $L \in \text{1DPDAB}(\Sigma^\omega)$ then $L = L^\omega(M)$ with

$$M = (K, \Pi = \{K_1, K_2\}, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

is a 1DPDAB. We suppose that M is trim, then it is sufficient to consider the one-turn DPDA $M' = (M_1, K)$ with $M_1 = (K, \Pi = \{K_1, K_2\}, \Sigma, \Gamma, \delta, q_0, Z_0)$, where all the set K is taken for the set of final states, then we have $\text{LF}(L) = L(M')$. ■

Proposition 17. *The family* $\text{1DPDAB}(\Sigma^\omega)$ *is not closed by intersection.*

Proof. It is sufficient to consider the following ω languages L_1, L_2 which are in $\text{1DPDAB}(\Sigma^\omega)$ with $L_1 = \left\{a^i b^i a^j / \ i, j \geq 1\right\} c^\omega$ and $L_2 = \left\{a^i b^j a^j / \ i, j \geq 1\right\} c^\omega$. But, $L = L_1 \cap L_2 = \{a^n b^n c^n; n \geq 1\} c^\omega \notin \text{1DPDAB}(\Sigma^\omega)$, because $L \notin \text{Alg}(\Sigma^\omega)$ and $\text{1DPDAB}(\Sigma^\omega) \subsetneq \text{1PDAB}(\Sigma^\omega) \subsetneq \text{Alg}(\Sigma^\omega)$. ■

Proposition 18. *The family* $\text{1DPDAB}(\Sigma^\omega)$ *is closed under intersection with the family* $\text{Drat}(\Sigma^\omega)$ *of* ω*-rational languages recognized by deterministic finite automata with a Büchi acceptance condition.*

Proposition 19. *The family* $\text{1DPDAB}(\Sigma^\omega)$ *is not closed under morphism.*

Proof. We consider the same examples $L_1 = \left\{a^i b^i a^j / i, j \geq 1\right\} c^\omega$ *and* $L_2 = \left\{a^i b^j a^j / i, j \geq 1\right\} c^\omega$ *cited in Proposition 5. Let* $L = cL_1 \cup dL_2$ *(with* $c, d \notin \{a, b\}$*) which is in* $\text{1DPDAB}(\Sigma^\omega)$*. Let* h *be the morphism from* Σ^ω *to* Σ^ω *defined by:* $h(c) = h(d) = c$ *and for* $x \in \{a, b\} : h(x) = x$*. Then we have,* $h(L) = h(cL_1 \cup dL_2) = h(c)\, h(L_1) \cup h(d)\, h(L_2) = cL_1 \cup cL_2 = c(L_1 \cup L_2)$*. Since* $L' = L_1 \cup L_2$ *is ambiguous then also* cL' *and consequently* cL' *is not deterministic.* ■

Let us now have a look at the subset of Σ^ω which can be recognized by a deterministic one-turn pushdown automaton with a Büchi acceptance condition.

Proposition 20. *Let* M *be a 1DPDAB. Then* $L^\omega(M) = \overrightarrow{L^+(M)}$.

Proof. Let M be a 1DPDAB. If $u \in L^\omega(M)$, then there is a unique infinite run $r = \{(p_i, \gamma_i)\}_{i \geq 0}$ of M on $u = a_0 a_1 a_2 ...$, starting at (q_0, Z_0) such that $\text{Inf}(r) \cap F \neq \emptyset$. Consequently, there exits an infinite subsequence of integers $n_0 < n_1 < n_2 < ...$, satisfying $q_{n_0}, q_{n_1}, q_{n_2}, ... \in F$, then the words $u_k = a_0 ... a_{n_{k-1}}$ for $k \geq 1$ are in $L^+(M)$ and are prefixes of u and $u = \lim_{k \longrightarrow \infty} u_k$. Thus $L^\omega(M) \subset \overrightarrow{L^+(M)}$. Conversely, if $u \in \overrightarrow{L^+(M)}$ then u has infinitely many prefixes in $L^+(M)$. Since M is deterministic, there is a unique run r of M on u such that r meets F infinitely often F i.e., $\text{Inf}(r) \cap F \neq \emptyset$. Thus $u \in L^\omega(M)$. ■

Theorem 21. *A part* U *of* Σ^ω *is recognizable by a one-turn deterministic pushdown automata with Büchi acceptance conditions if and only if* U *is a limit of deterministic linear language* L *of finite words.*

Proof. If $U = L^\omega(M)$ for some 1DPDAB M then $U = \overrightarrow{L^+(M)}$ by Proposition 20. Conversely if $U = \overrightarrow{A}$, where A is a deterministic linear language of finite words, there exists a deterministic one-turn pushdown automaton M' such that $L(M') = A$ (the acceptance mode is by final states). From Proposition 20 again, we have $U = L^\omega(M)$. ■

Proposition 22. *The classes* $1\mathrm{DPDAB}(\Sigma^\omega)$ *and* $\mathrm{Rat}(\Sigma^\omega)$ *are incomparable.*

Proof. It is sufficient to consider the ω*-languages* $L_1 = \{a^n b^n \mid n \geq 1\} a^\omega$ *and* $L_2 = (a+b)^* a^\omega$ *where we have* L_1 *in* $1\mathrm{DPDAB}(\Sigma^\omega)$ *and not in* $\mathrm{Rat}(\Sigma^\omega)$*, however* L_2 *is in* $\mathrm{Rat}(\Sigma^\omega)$ *and not in* $1\mathrm{DPDAB}(\Sigma^\omega)$*.* ■

Proposition 23. *The class* $1\mathrm{DPDAB}(\Sigma^\omega)$ *is not closed under complementation.*

Proof. It is sufficient to consider the ω*-language* $L_2 = (a+b)^* a^\omega$ *which is not in* $1\mathrm{DPDAB}(\Sigma^\omega)$*. On the other hand,* $\overline{L_2} = \{a,b\}^\omega - L_2 = (a^*b)^\omega + (b^+a^*)^\omega \in 1\mathrm{DPDAB}(\Sigma^\omega)$*.* ■

5 One-Turn PDM with Muller Condition

Muller introduced a more powerful condition of acceptance leading to a model of deterministic automata, now referred to as Muller automata. The condition of Muller is just a designated subset T of states, and a run r in an automaton is successful if it starts at the unique initial state and $\mathrm{Inf}(r) \in T$.

In this section, we study the behaviors of one-turn deterministic pushdown automata with Muller acceptance conditions and some closure properties of sets recognized by these automata.

Definition 24. *A one-turn deterministic pushdown automaton with a Muller acceptance condition is a pair* $M = (M_1, T)$ *where* M_1 *is a 1DPDA and* $T \subseteq 2^K$*.*

The infinite word $\sigma \in \Sigma^\omega$ is accepted by M in the sense of Muller if and only if there is a unique infinite run r of M starting at (q_0, Z_0) and $\mathrm{Inf}(r)$ belongs to T. We denote by $1\mathrm{DPDAM}(\Sigma^\omega)$ the family of ω-languages recognized by a deterministic one-turn pushdown automaton with a Muller acceptance condition.

Proposition 25. *For all* ω*-language* $L \in 1\mathrm{DPDAM}(\Sigma^\omega)$ *there exist two sequences of languages* (A_i)*,* (B_i)*,* $i = 1, ..., n$ *where for all* i

1. A_i *is a prefix linear language,*
2. B_i *is a prefix rational language, and*
3. $L = \bigcup_{i=1}^{n} A_i B_i^\omega$*.*

Proof. The proof is similar to the case when the acceptance condition is of Büchi, i.e, to the proof of Theorem 12. ■

Proposition 26. *The family* $1\mathrm{DPDAB}(\Sigma^\omega)$ *is strictly included in the family* $1\mathrm{DPDAM}(\Sigma^\omega)$*.*

Proof. At first, we have $1\mathrm{DPDAB}(\Sigma^\omega) \subseteq 1\mathrm{DPDAM}(\Sigma^\omega)$. In fact, for any one-turn deterministic pushdown automaton $M = (M_1, F)$ with a Büchi acceptance condition we construct an equivalent $M' = (M_1, T)$ with a Muller mode where $T = \{D \subseteq K / D \cap F \neq \emptyset\}$. The ω-language $L = (a+b)^* a^\omega$ which is in $1\mathrm{DPDAM}(\Sigma^\omega)$ and not in $1\mathrm{DPDAB}(\Sigma^\omega)$ shows that the inclusion is strict. ■

Proposition 27. *The family* $1\mathrm{DPDAM}(\Sigma^\omega)$ *is not closed under union, intersection and morphism.*

Proposition 28. *The family* $1\mathrm{DPDAM}(\Sigma^\omega)$ *(with no* ε*-transitions) is closed under complementation.*

Proof. Let $M = (M_1, F)$ be a one-turn deterministic pushdown automaton with a Muller condition and let $L = L^\omega(M)$. We suppose that M has the continuity property, i.e., for all $\sigma \in \Sigma^\omega$ there exists an infinite run of M on σ. With this hypothesis, we have $\overline{L} = \Sigma^\omega - L = L^\omega\left(\overline{M}\right)$ with $\overline{M} = \left(M_1, \overline{F}\right)$, where $\overline{F} = K - F$. We then have $L^\omega(M) \cap L^\omega\left(\overline{M}\right) = \emptyset$ and for all $\sigma \in \Sigma^\omega$, we have $\sigma \in L^\omega(M)$ or $\sigma \in L^\omega\left(\overline{M}\right)$, i.e., $\Sigma^\omega = L^\omega(M) \cup L^\omega\left(\overline{M}\right)$. ■

References

1. Büchi, J.R.: On a decision method in the restricted second-order arithmetic. In: Logic, methodology and philosophy of science. Proceedings of the 1960 International Congress, pp. 1–14. Stanford Univ. Press, CA (1962)
2. Cohen, R., Gold, A.: Theory of ω-languages, I, II. J. Comp. System Sci. 15, 169–208 (1977)
3. Cohen, R., Gold, A.: ω-computation on deterministic pushdown machines. J. Comp. System Sci. 16, 275–300 (1978)
4. Eilenberg, S.: Automata languages and machines, vol. A. Academic press, New York (1977)
5. Linna, M.: On ω-words and ω-computations, Ann. Univ. Turkuensis (1975)
6. Ginsburg, S., Spanier, E.: Finite-turn pushdown automata. SIAM J. Control (4), 429–453 (1966)
7. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Information and Control 9, 521–530 (1966)
8. Mihoubi, D.: Characterization and closure properties of linear ω-languages. Theor. Comput. Sci. 191, 79–95 (1998)
9. Mihoubi, D.: Modes de reconnaissance et equités dans les ω-automates à pic, Thèse de Doctorat, Université Paris Nord (1989)
10. Muller, D.: Infinite sequences and finite machines. In: Ginsburg, S. (ed.) SWCT 1963, pp. 3–16. IEEE, Los Alamitos (1963)
11. Nivat, M.: Sur les ensembles de mots infinis engendrés par une grammaire algèbrique. Inform. Théorique et Appl. 12(3), 259–278 (1978)

Fine-Continuous Functions and Fractals Defined by Infinite Systems of Contractions

Yoshiki Tsujii[1], Takakazu Mori[1], Mariko Yasugi[1], and Hideki Tsuiki[2,⋆]

[1] Faculty of Science, Kyoto Sangyo University, 603-8555, Kyoto, Japan
{tsujiiy,morita,yasugi}@kyoto-su.ac.jp
[2] Graduate School of Human and Environmental Studies, Kyoto University, 606-8501, Kyoto, Japan
tsuiki@i.h.kyoto-u.ac.jp

Abstract. Motivated by our study in [12] of the graph of some Fine-computable (hence Fine-continuous) but not locally uniformly Fine-continuous functions defined according to Brattka's idea in [2], we have developed a general theory of the fractal defined by an infinite system of contractions. In our theory, non-compact invariant sets are admitted. We note also that some of such fractals, including the graph of Brattka's function, are also characterized as graph-directed sets. Furthermore, mutual identity of graph-directed sets and Markov-self-similar sets is established.

1 Introduction

In our study of "Fine-computable" functions in [12], we worked on "Fine-computable" (hence Fine-continuous) but not locally uniformly Fine-continuous functions defined according to Brattka's idea in [2]. In [12], we characterized such a function as the solution of an infinite system of functional equations, and showed that its graph is a fractal defined by an infinite system of contractions.

As we observed these results, we were led to the general theory of the invariant set for an infinite system of contractions in which non-compact invariants are admitted. For some examples which have good symmetric structures, such an invariant set can also be obtained as a member of a pair of graph-directed sets. On the other hand, Tsujii in [13] developed the theory of Markov-self-similar sets. We can in fact show the equivalence of the theory of graph-directed sets and the theory of Markov-self-similar sets.

Fine-topology and Brattka's function as well as the characterization of its graph in terms of an infinite system of contractions are explained in Section 2. In Section 3, we review some basics of self-similar sets and their properties. In Section 4, we study the invariant set which is defined in terms of an infinite system of contractions, and show that some fractal figures such as the graph of

⋆ The four authors have been supported in part by research grants from Kyoto Sangyo University (2007, 070 and 2007, 110) and the Japan Society for the Promotion of Science (18650003 and 18500013), respectively. (*Received by the editors.* 4 February 2008. *Revised.* 19 March 2009. *Accepted.* 6 April 2009.)

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, pp. 109–125, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Brattka's function is obtained in this manner. In Section 5, we present another characterization of the examples in Section 4 in terms of graph-directed sets, that is the figure G and a part of G are defined as an invariant pair for a finite system of contractions. From this characterization, we can derive positivity of the s-dimensional Hausdorff measure $H^s(G)$ for $s > 0$, the dimension of G, from the general theory of graph-directed sets. The equivalence of this theory and the theory of Markov-self-similar sets in [13] is established in Section 6.

Our original purpose was to present a general theory of the invariant set for an infinite system of contractions in order to apply it to some examples of Fine-computable functions. A function of this type is the solution of an infinite system of functional equations, from which some important properties of the function can be deduced [12]. The graph of such a function can be characterized as the invariant set for an infinite system of contractions. In general, the Hausdorff dimension of the invariant set for an infinite system of contractions can be evaluated. For a graph as above, the finiteness of the Hausdorff measure with respect to the dimension can be established.

The examples we have treated possess, however, some symmetric structures. This fact led to us to the idea that they can also be characterized as graph-directed sets. The theory of graph-directed sets has the advantage that it has been well developed [1,3,5,9], and that positivity of the Hausdorff measure of a graph-directed set is a known result. Moreover, this theory concerns only a finite system of contractions.

Let us note here that the graph of each of our examples is not compact, but its closure is obtained from the graph by augmenting a countable set of points.

The theory of Markov-self-similar sets has been proposed and studied in [13] quite independently of the two theories above. It is of interest that the equivalence of the last two theories holds. Since Markov-self-similar sets can be defined in terms of a square matrix, they are relatively easy to deal with.

2 Fine-Topology and Brattka's Function

The Fine-topology on $I = [0, 1)$ is generated by the system of fundamental neighborhoods $\{U_n(x) \,|\, x \in [0, 1), n = 1, 2, 3, \cdots\}$, defined as follows [7] (See also [10,11,12]). For a positive integer n, let

$$I(n, k) = [\frac{k}{2^n}, \frac{k+1}{2^n}), \quad k = 0, 1, \ldots, 2^n - 1,$$
$$U_n(x) = \text{the } I(n, k) \text{ which contains } x.$$

A real function $f(x)$ on $I = [0, 1)$ is called Fine-continuous if, for $x \in [0, 1)$ and an integer k, there is an integer $n(x, k)$ such that $|f(x) - f(y)| < 2^{-k}$ for $y \in U_{n(x,k)}(x)$. A function $f(x)$ is said to be uniformly Fine-continuous on a set U if, for k, there is an integer $n(k)$ such that $|f(y) - f(z)| < 2^{-k}$ if $y, z \in U$ and $z \in U_{n(k)}(y)$. $f(x)$ is called uniformly Fine-continuous if it *is* on the set $I = [0, 1)$. A function $f(x)$ is called locally uniformly Fine-continuous if, for

$x \in [0,1)$, there is an integer $n(x)$ such that $f(x)$ is uniformly Fine-continuous on $U_{n(x)}(x)$.

Example 1 (Brattka, [2]). *Brattka's function is defined as follows. Let $p(x)$ denote the dyadic expansion of x with infinitely many 0's. Define*

$$v(x)=\begin{cases} \sum_{i=0}^{\infty}(\ell_i \bmod 2)2^{-n_i-\sum_{j=0}^{i-1}(n_j+\ell_j)} & (p(x)=0.0^{n_0}1^{\ell_0}0^{n_1}1^{\ell_1}0^{n_2}\cdots) \\ \sum_{i=0}^{m}(\ell_i \bmod 2)2^{-n_i-\sum_{j=0}^{i-1}(n_j+\ell_j)} & (p(x)=0.0^{n_0}1^{\ell_0}0^{n_1}1^{\ell_1}0^{n_2}\cdots 1^{\ell_m}0^{\omega}), \end{cases}$$

where $x \in I$, $n_0 \geq 0$, $n_i > 0$ for $i > 0$ and $\ell_i > 0$ for all $i \geq 0$.

The function $v(x)$ is Fine-continuous (in fact "Fine-computable") but is not locally uniformly Fine-continuous [2]. The graph of $v(x)$ is depicted in Figure 2 (a).

Let us attempt to visualize why $v(x)$ is not locally uniformly Fine-continuous. Let

$$I_k = [1-\frac{1}{2^{k-1}}, 1-\frac{1}{2^k}), \quad k=1,2,\dots\ .$$

Then $\bigcup_{k=1}^{\infty} I_k = I$ and $\{I_k\}$ are mutually disjoint. Define the function $f(x)$ on I by

$$f(x) = (1-(-1)^{k-1})/2, \text{ for } \ x \in I_k, k=1,2,\dots\ .$$

The function $f(x)$ is locally uniformly Fine-continuous since for any $x \in I$ there exists a unique k such that $x \in I_k = U_k(x)$ and $f(x)$ is constant on $U_k(x)$. The function $f(x)$ is not uniformly Fine-continuous since for any integer m there are $x \in I$ and k such that $U_m(x) \supset I_k, I_{k+1}$ and

$$|f(x)-f(y)| = 1 \ \text{ for } \ x \in I_k \ \text{ and } \ y \in I_{k+1}.$$

This means that, for any ε $(0 < \varepsilon < 1)$, there is no m such that $|f(x)-f(y)| < \varepsilon$ for any $z \in I$ and for any $x, y \in U_m(z)$. The graph of $f(x)$ is drawn in Figure 1.

Brattka's function $v(x)$ has miniatures of the function $f(x)$ as above everywhere, and this fact ensures that $v(x)$ is not locally uniformly Fine-continuous. We can see this situation through the equation

$$v(x) = \frac{1+(-1)^i}{2} + \frac{v(V_i^{-1}(x))}{2^i} \quad \text{for } \ x \in I_i \ \ (i=1,2,\dots),$$

where $V_i : [0,1) \to [0,1)$ is defined by $V_i(x) = 1 - \frac{1}{2^{i-1}} + \frac{1}{2^i}x$.

The graph G of $v(x)$ has the following characterization:

$$G = \bigcup_{i=1}^{\infty} S_i(G),$$

where $S_i : \mathbb{R}^2 \to \mathbb{R}^2$ is defined by

$$S_i(x,y) = \Big(1-\frac{1}{2^{i-1}} + \frac{1}{2^i}x, \frac{(-1)^i+1}{2} + \frac{1}{2^i}y\Big), \quad i=1,2,\dots,$$

that is, the graph of the function $v(x)$ is invariant for $\{S_i\}_{i=1}^{\infty}$.

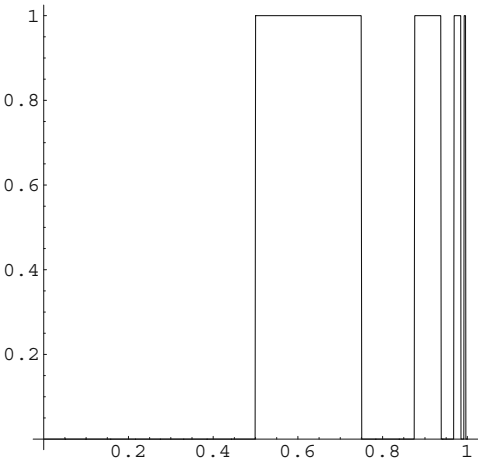
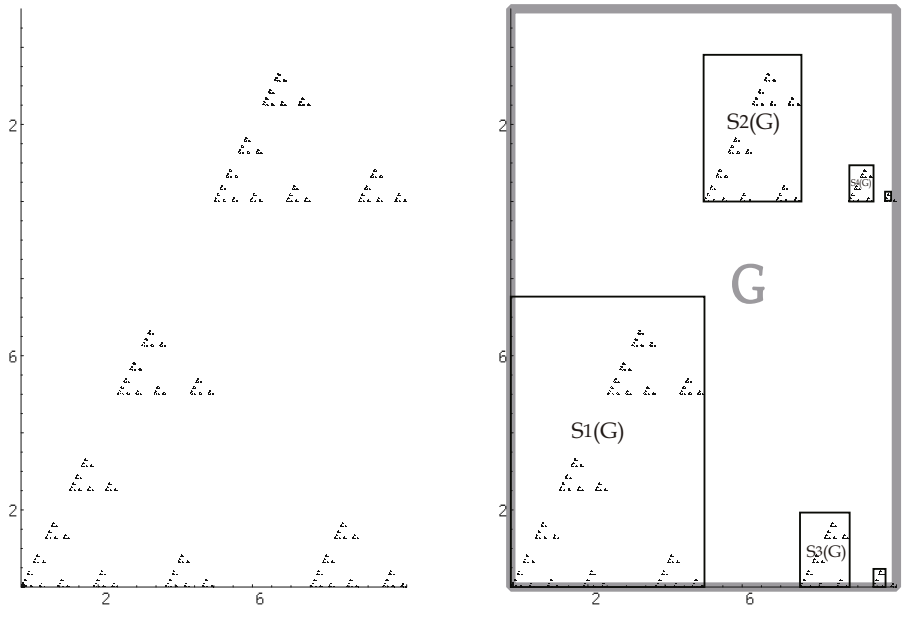


Fig. 1. The function $f(x)$



(a) Brattka's function **(b)** Contractions

Fig. 2. Brattka's function and contractions

This characterization needs an infinite number of contractions, that is, the graph of the function $v(x)$ is not invariant for a finite number of contractions. Notice that, with a finite number of contractions, the corresponding function

becomes locally uniformly Fine-continuous. The relation between $\{S_i\}$ and Brattka's function is depicted in Figure 2.

Similar infinite systems of contractions give rise to some other examples of Fine-continuous functions which are not locally uniformly Fine-continuous [12].

3 Self-similar Sets

We summarize here some known facts of the fractals which are defined to be the invariant sets for finite systems of contractions.

Definition 2 (Contraction map). *A map $S : \mathbb{R}^d \to \mathbb{R}^d$ is called a contraction map if*

$$|S(x) - S(y)| \leq c\,|x - y| \quad \text{for all } x, y \in \mathbb{R}^d$$

for some c, $0 < c < 1$. Such a c is called a contraction ratio for S.

Definition 3 (Invariant sets). *Let $\{S_k\}_{k=1}^n$ be a finite system of contractions. A set F is called invariant for $\{S_k\}_{k=1}^n$ if it satisfies the equation*

$$F = \bigcup_{k=1}^{n} S_k(F).$$

Definition 4 (Hausdorff measure and Hausdorff dimension)

a) *The s-dimensional Hausdorff measure $H^s(F)$ of a subset F of $\mathbb{R}^d$ is defined by*

$$H^s(F) = \lim_{\delta \downarrow 0} H^s_\delta(F),$$

$$H^s_\delta(F) = \inf\left\{ \sum_{i=1}^{\infty} |U_i|^s : \{U_i\} \text{ is a } \delta\text{-cover of } F \right\},$$

where $|U_i|$ denotes the diameter of U_i, and $\{U_i\}_{i=1}^\infty$ is a δ-cover of F if $F \subset \bigcup_{i=1}^\infty U_i$ and $0 < |U_i| \leq \delta$.

b) *The Hausdorff dimension $\dim_{\mathrm{H}}(F)$ of a subset F of $\mathbb{R}^d$ is defined by*

$$\dim_{\mathrm{H}}(F) = \inf\,\{s : H^s(F) = 0\} = \sup\,\{s : H^s(F) = \infty\}.$$

Definition 5 (Self-similar set)

(i) *A map $S : \mathbb{R}^d \to \mathbb{R}^d$ is called a contraction similarity if*

$$|S(x) - S(y)| = c\,|x - y| \quad \text{for all } x, y \in \mathbb{R}^d,$$

where $0 < c < 1$. c is called the contraction similarity ratio of S.

(ii) *Let $\{S_k\}_{k=1}^n$ be contraction similarities.*
A non-empty compact set F is called self-similar for $\{S_k\}_{k=1}^n$ if it is invariant for $\{S_k\}_{k=1}^n$, that is, it satisfies

$$F = \bigcup_{k=1}^{n} S_k(F),$$

and, for some $s > 0$,

$$0 < H^s(F) < \infty \quad \text{and} \quad H^s(S_i(F) \cap S_j(F)) = 0 \quad (i \neq j).$$

A well known example which is characterized by a finite number of contraction similarities is the Koch curve (cf. [4] and [5]).

Example 6 (Koch Curve). *The Koch curve is invariant for* S_1 *and* S_2, *where* S_i *maps the whole triangle to the smaller triangle for* $i = 1, 2$. *(See Figure 3.)*

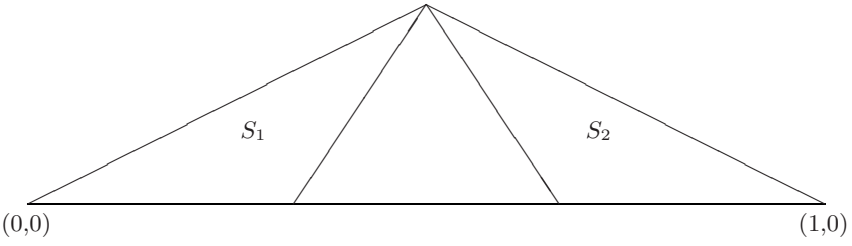


Fig. 3. The Koch curve is invariant for S_1 and S_2

The theorem below concerning the finite contraction system is well known.

Theorem 7 (Hutchinson; [8,4,5]). *Let* $S_1, \ldots, S_m$ *be contractions.*

(i) *There exists a unique non-empty compact set* F *which is invariant for* $\{S_i\}_{i=1}^m$.

(ii) *Assume furthermore that* $S_1, \ldots, S_m$ *are contraction similarities with contraction similarity ratios* $c_1, c_2, \ldots, c_m$ *respectively, which satisfy the open set condition, that is, there exists a non-empty bounded open set* V *such that*

$$V \supset \bigcup_{i=1}^{m} S_i(V)$$

holds with the union disjoint. Then the unique compact invariant set F *has the Hausdorff dimension* s, *where* s *is determined by*

$$\sum_{i=1}^{m} c_i^s = 1.$$

Moreover, F *is self-similar for* $S_1, \ldots, S_m$, *that is,*

$$0 < H^s(F) < \infty \quad \text{and} \quad H^s(S_i(F) \cap S_j(F)) = 0 \quad (i \neq j).$$

4 An Infinite System of Contractions

In this section we investigate the invariant set (a fractal) which is defined in terms of an infinite system of contractions. Fernau and some others (in [6], for example) have also treated the invariant set for the infinite system of contractions. They are mostly concerned with compact invariants, while we do not restrict ourselves only to compact invariants. This enables us to apply the results to the graphs such as the one in Example 1, which are not necessarily compact.

In the subsequent discussion, $\{S_i\}$, $\{S_i\}_{i=1}^{\infty}$ or $S_1, S_2, \cdots, S_m, \cdots$ will denote an infinite system of contractions.

Definition 8 (Invariant set). *A set F is called invariant for $\{S_i\}_{i=1}^{\infty}$ if*

$$F = \bigcup_{i=1}^{\infty} S_i(F).$$

Note that if F is compact then $S_i(F)$ is compact, but $\bigcup_{i=1}^{\infty} S_i(F)$ is not necessarily closed. So, we may not have a compact invariant set for the case of an infinite system of contractions.

However, we have the theorem below which follows from the next proposition.

Proposition 9. *Let $\{S_i\}$ be as above. Assume that $\sup_i c_i < 1$, where c_i is a contraction ratio of S_i, that is,*

$$|S_i(x) - S_i(y)| \leq c_i \ |x - y| \quad \text{for all } x, y \in \mathbb{R}^d.$$

Assume also that there is a non-empty bounded open set V such that

$$V \supset \bigcup_{i=1}^{\infty} S_i(V).$$

Then there is a "unique" non-empty $\{S_i\}$-invariant set H in the following sense. Suppose that V_1 and V_2 are non-empty bounded open sets which satisfy the above condition on V. Let

$$H_j = \bigcap_{k=0}^{\infty} S^k(\overline{V}_j), \ j = 1, 2,$$

where $S(A) = \bigcup_{i=1}^{\infty} S_i(A)$, and S^k denotes the k-th iterate of S. Then H_1 and H_2 are invariant for $\{S_i\}_{i=1}^{\infty}$, and

$$H_j \subset \overline{V}_j, \quad j = 1, 2.$$

Furthermore, $H_1 = H_2$.

Proof. The invariance of H_1 (also of H_2) for $\{S_i\}_{i=1}^{\infty}$ is obtained by the fact that $\{S^k(\overline{V}_1)\}_{k=0}^{\infty}$ is a decreasing sequence.

For the rest it suffices to show that for any two non-empty bounded open sets V and W which satisfy

$$V \subset W,\ V \supset \bigcup_{i=1}^{\infty} S_i(V),\ W \supset \bigcup_{i=1}^{\infty} S_i(W),$$

it holds that

$$\bigcap_{k=0}^{\infty} S^k(\overline{V}) = \bigcap_{k=0}^{\infty} S^k(\overline{W}).$$

For, if we let $W = V_1 \cup V_2$ and $V = V_1$ (or V_2), the equation above induces the equality

$$\bigcap_{k=0}^{\infty} S^k(\overline{W}) = \bigcap_{k=0}^{\infty} S^k(\overline{V_1})\ (= \bigcap_{k=0}^{\infty} S^k(\overline{V_2})).$$

Recall that the Hausdorff distance $d(E, F)$ between two sets E and F is defined by

$$d(E, F) = \inf\{\delta > 0 : E \subset F_\delta \ \text{ and } \ F \subset E_\delta\},$$

where F_δ is the δ-parallel body of F, that is,

$$F_\delta = \{x \in \mathbb{R}^d : |x - a| \le \delta \text{ for some } a \in F\}.$$

From the definition, we have

$$d(\cup_{i=1}^{\infty} E_i, \cup_{i=1}^{\infty} F_i) \le \sup\{d(E_i, F_i) : 1 \le i < \infty\}.$$

It follows that, for any sets A and B,

$$\begin{aligned} d(S(A), S(B)) &= d(\cup_{i=1}^{\infty} S_i(A), \cup_{i=1}^{\infty} S_i(B)) \\ &\le \sup\{d(S_i(A), S_i(B)) : 1 \le i < \infty\} \\ &\le (\sup_{1 \le i < \infty} c_i)\ d(A, B). \end{aligned}$$

Let $c = \sup_{1 \le i < \infty} c_i (< 1)$. Notice that $\{\bigcap_{k=0}^{m} S^k(\overline{V})\}_m$ and $\{\bigcap_{k=0}^{m} S^k(\overline{W})\}_m$ are decreasing sequences of sets. So

$$d(\bigcap_{k=0}^{m} S^k(\overline{V}),\ \bigcap_{k=0}^{m} S^k(\overline{W})) \le c^m d(\overline{V}, \overline{W}).$$

It then follows that

$$d(\bigcap_{k=0}^{\infty} S^k(\overline{V}),\ \bigcap_{k=0}^{\infty} S^k(\overline{W})) = 0.$$

Note also that

$$\overline{\bigcap_{k=0}^{\infty} S^k(\overline{W})} = \overline{\bigcap_{k=0}^{\infty} S^k(\overline{V})} \subset \overline{V},$$

and hence

$$S^m(\bigcap_{k=0}^{\infty} S^k(\overline{W})) \subset S^m(\overline{V}).$$

Then

$$\bigcap_{m=0}^{\infty} S^m(\bigcap_{k=0}^{\infty} S^k(\overline{W})) \subset \bigcap_{m=0}^{\infty} S^m(\overline{V}).$$

Recall that $\bigcap_{k=0}^{\infty} S^k(\overline{W})$ is invariant and so $\bigcap_{k=0}^{\infty} S^k(\overline{W}) \subset \bigcap_{m=0}^{\infty} S^m(\overline{V})$. This implies that $\bigcap_{k=0}^{\infty} S^k(\overline{W}) = \bigcap_{k=0}^{\infty} S^k(\overline{V})$, because $V \subset W$. ■

The theorem is stated for contraction similarities (cf. Definition 5, Theorem 7).

Theorem 10. *Let $S_1, \ldots, S_m, \ldots$ be contraction similarities with contraction similarity ratios $c_1, \ldots, c_m, \ldots$ respectively. Suppose they satisfy the open set condition, that is, there exists a non-empty bounded open set V such that*

$$V \supset \bigcup_{i=1}^{\infty} S_i(V)$$

with the union disjoint. Let

$$F = \bigcap_{k=0}^{\infty} S^k(\overline{V}).$$

Then the following hold.

(i) *F is invariant for $\{S_i\}_{i=1}^{\infty}$.*
(ii) *F has the Hausdorff dimension s, where s satisfies*

$$\sum_{i=1}^{\infty} c_i^s = 1.$$

(iii) *For the s in* (ii),

$$H^s(F) < \infty$$

where $H^s(F)$ is the s-dimensional Hausdorff measure of F.

Proof.
(i): The invariance of F is shown in Proposition 9.
(ii) and (iii): First notice that $\sum_{i=1}^{\infty}(c_i)^d \leq 1$, since $0 < c_i \leq \sup_i c_i < 1$ and $V \supset \bigcup_{i=1}^{\infty} S_i(V)$ with the union disjoint. (d is the Euclidean dimension of $\mathbb{R}^d$.) It follows that $\phi(t) := \sum_{i=1}^{\infty}(c_i)^t$ is a continuous and strictly decreasing function with $\phi(0) = \infty$ and $\phi(d) \leq 1$. Note that s in (ii) is the unique solution of $\phi(t) = 1$.

The fact that $H^s(F) < \infty$ is proved in a similar way to Theorem 8.6 in [4] or Theorem 9.3 in [5], and so the Hausdorff dimension of F is smaller than or equal to s.

We claim below that $H^{s'}(F) > 0$ for any s' such that $0 < s' < s$. This implies that the Hausdorff dimension of F is greater than or equal to s. We can thus conclude that s is the Hausdorff dimension of F.

As $\phi(s) = 1$ and $\phi(t)$ is decreasing, $\sum_{i=1}^{\infty}(c_i)^{s'} = \phi(s') > 1$. Fix an m so that $\sum_{i=1}^{m}(c_i)^{s'} > 1$. Then by the fact that $\sum_{i=1}^{m}(c_i)^t$ is decreasing and continuous in t, there exists a real number u $(s' < u < s)$ for which $\sum_{i=1}^{m}(c_i)^u = 1$. The finite system $\{S_i\}_{i=1}^m$ satisfies the open set condition. Let F_m be the self-similar set for $\{S_i\}_{i=1}^m$. By virtue of Theorem 7, $H^u(F_m) > 0$ holds. It also holds generally that $F_m = \bigcap_{k=0}^{\infty}(S_{(m)})^k(\overline{V})$, where $S_{(m)}(F) = \bigcup_{i=1}^{m} S_i(F)$. (See [4] and [5].) So $F_m = \bigcap_{k=0}^{\infty}(S_m)^k(\overline{V}) \subset \bigcap_{k=0}^{\infty} S^k(\overline{V}) = F$ holds, and this implies that $H^{s'}(F) \geq H^u(F) \geq H^u(F_m) > 0$. ■

Note 11. It is not currently known whether $0 < H^s(F)$ holds in general, but it *does* hold for our examples.

Example 1 (continued). $\{S_i\}_{i=1}^{\infty}$ denotes the system of contractions for Example 1 in Section 2. Let $V = (0,1) \times (0, \frac{3}{2})$. The graph G of Brattka's function has Hausdorff dimension 1, and the 1-dimensional Hausdorff measure of G is positive, because the projection of G to the x-axis is the unit interval. This fact will be reconfirmed in Section 5.

We give another example below.

Example 12. *Let*

$$S_1(x,y) = \left(\frac{x}{2}, \frac{y}{2}\right),\ S_2(x,y) = \left(\frac{y}{4} + \frac{1}{2}, \frac{x}{4}\right),\ S_3(x,y) = \left(\frac{x+2}{4}, \frac{y+1}{4}\right),$$

$$S_{2n}(x,y) = \left(\frac{y}{2^{n+1}} + 1 - \frac{1}{2^n}, -\frac{x}{2^{n+1}} + \frac{1}{2^n}\right) \text{ for } n = 2,3,\ldots,$$

$$S_{2n+1}(x,y) = \left(\frac{x}{2^{n+1}} + 1 - \frac{1}{2^n}, -\frac{y}{2^{n+1}} + \frac{1}{2^{n+1}}\right) \text{ for } n = 2,3,\ldots.$$

The maps $\{S_i\}$ and the invariant set are illustrated in Figure 4. The invariant set has Hausdorff dimension $s = \frac{\log(\sqrt{2}+1)}{\log 2}$, and the s-dimensional Hausdorff measure is positive. The last fact will be seen in Section 5.

5 Graph-Directed Sets

Graph-directed sets are investigated in, for example, Bedford [1], Mauldin and Williams [9] and Edgar [3]. In this section we discuss the preceding examples in light of the theory of graph-directed sets. More precisely, the invariant sets of these examples are characterized in terms of graph-directed sets. The closure of each example is also a graph-directed set, and is obtained from the original set by augmenting a countable set of points. Due to this fact, both the Hausdorff dimension and the Hausdorff measure with respect to the dimension for the original set are the same as those of its nonrectification.

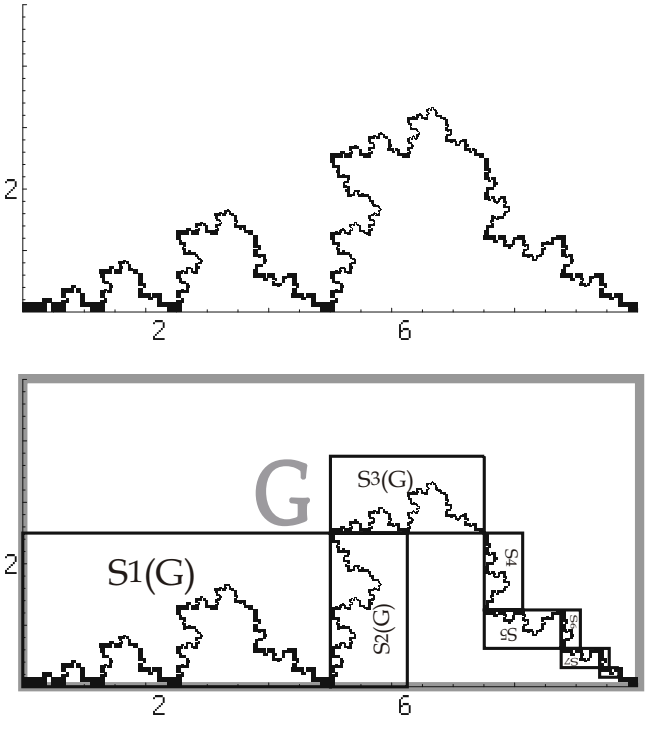


Fig. 4. Example 12

The definition of graph-directed sets is stated below.

(V, E) will denote a finite directed graph, that is, $V = \{1, \ldots, K\}$ is a set of vertices, $E_{i,j}$ is a finite set of edges from vertex i to vertex j and $E = \{E_{i,j}\}_{i,j \in V}$. Some $E_{i,j}$ may be empty.

Definition 13 (Graph-directed sets; [5]). *Let (V, E) be a directed graph. Assume that a contraction, say S_e, is associated with each $e \in E_{i,j}$. A K-tuple of non-empty sets $(F_1, F_2, \ldots, F_K)$ is called a family of graph-directed sets for (V, E) if*

$$F_i = \bigcup_{j=1}^{K} \bigcup_{e \in E_{i,j}} S_e(F_j), \quad i = 1, \ldots, K.$$

Remark 14. *The graph-directed sets in Definition 13 can be reformulated as follows.*

Assume $K \geqq 1$, and let

$$\mathcal{S} = \left\{ \begin{array}{cccc} \{S_i^{11}\}_{i=1}^{n_{11}} & \{S_i^{12}\}_{i=1}^{n_{12}} & \cdots & \{S_i^{1K}\}_{i=1}^{n_{1K}} \\ \cdots & \cdots & \cdots & \cdots \\ \{S_i^{K1}\}_{i=1}^{n_{K1}} & \{S_i^{K2}\}_{i=1}^{n_{K2}} & \cdots & \{S_i^{KK}\}_{i=1}^{n_{KK}} \end{array} \right\},$$

where each S_i^{kl} is a contraction and $n_{kl} \geq 0$. A K-tuple of sets $(F_1, \ldots, F_K)$ is called a family of graph-directed sets for $\mathcal{S}$ if

$$F_k = \bigcup_{i=1}^{n_{k1}} S_i^{k1}(F_1) \cup \cdots \cup \bigcup_{i=1}^{n_{kK}} S_i^{kK}(F_K), \quad k = 1, \ldots, K.$$

Note 15. The equivalence of these two definitions follows if we let

$$\{S_e : e \in E_{i,j}\} = \{S_l^{ij} : l = 1, \quad \ldots, n_{ij}\}, \quad i, j = 1, \ldots, K.$$

The open set condition for graph-directed sets can be expressed similarly to that of (ii) in Theorem 7. That is, a family of graph-directed sets is said to satisfy the open set condition if there exist non-empty bounded open sets $\{V_i\}_{i=1}^K$ such that

$$V_i \supset \bigcup_{j=1}^{K} \bigcup_{e \in E_{i,j}} S_e(V_j), \quad i = 1, \ldots, K,$$

with the unions disjoint.

Let $E_{i,j}^k$ be the set of sequences of k edges $(e_1, e_2, \ldots, e_k)$ which is a directed path from vertex i to vertex j. We say that the directed graph (V, E) is transitive if, for any i, j, there is a positive integer p such that $E_{i,j}^p$ is non-empty.

The following theorem is due to Mauldin and Williams. It claims the unique existence of a family of compact graph-directed sets.

Theorem 16 ([1,3,5,9]). *Let (V, E) be a directed graph and let S_e be a contraction for each $e \in E_{i,j}$. Then the following hold.*

(i) *There is a unique family $(F_1, F_2, \ldots, F_K)$ of non-empty compact sets which is a family of graph-directed sets for (V, E).*

(ii) *Assume furthermore that (V, E) is transitive, the contractions are similarities with contraction ratio c_e for each $e \in E_{i,j}$ $(i, j \in E)$, and the open set condition is satisfied. Let $\mathbf{c}(s)$ be a $K \times K$ matrix with the (i, j)-th entry*

$$\mathbf{c}(s)_{i,j} = \sum_{e \in E_{i,j}} c_e^s.$$

Let s be the unique real number that satisfies

$$\lambda(s) = 1,$$

where $\lambda(s)$ is the maximal eigenvalue of $\mathbf{c}(s)$. Then the Hausdorff dimensions of the sets $F_1, F_2, \ldots F_K$ in (i) *are all equal to s. Moreover,*

$$0 < H^s(F_i) < \infty, \quad i = 1, \ldots K.$$

The examples below concern non-compact graph-directed sets, which are obtained directly. The Hausdorff dimension and positivity of Hausdorff measure

of each set can be obtained by compactifying it, as has been remarked at the beginning of this section.

Example 1 (continued). The graph of Brattka's function can also be characterized in terms of graph-directed sets as follows. Let

$$S_1(x,y) = \left(\frac{x}{2}, \frac{y}{2}\right),\ T_1(x,y) = \left(\frac{x+1}{2}, \frac{y+2}{2}\right),\ T_2(x,y) = \left(\frac{x+3}{4}, \frac{y}{4}\right).$$

Then the graph of Brattka's function G is characterized by

$$\begin{cases} G = T_1(B) \cup B \\ B = S_1(G) \cup T_2(B), \end{cases}$$

where $B = G \cap [0,1) \times [0,1)$.

G and B also satisfy

$$\begin{cases} G = T_1 \circ S_1(G) \cup S_1(G) \cup T_1 \circ T_2(B) \cup T_2(B) \\ B = S_1(G) \cup T_2(B). \end{cases}$$

Note that B is invariant for $\{S_1,\ T_2,\ S_1 \circ T_1\}$ but G is not invariant for any finite system of contractions. The relation between the system of contractions $\{S_1,\ T_1,\ T_2\}$ and Brattka's function is illustrated in Figure 5.

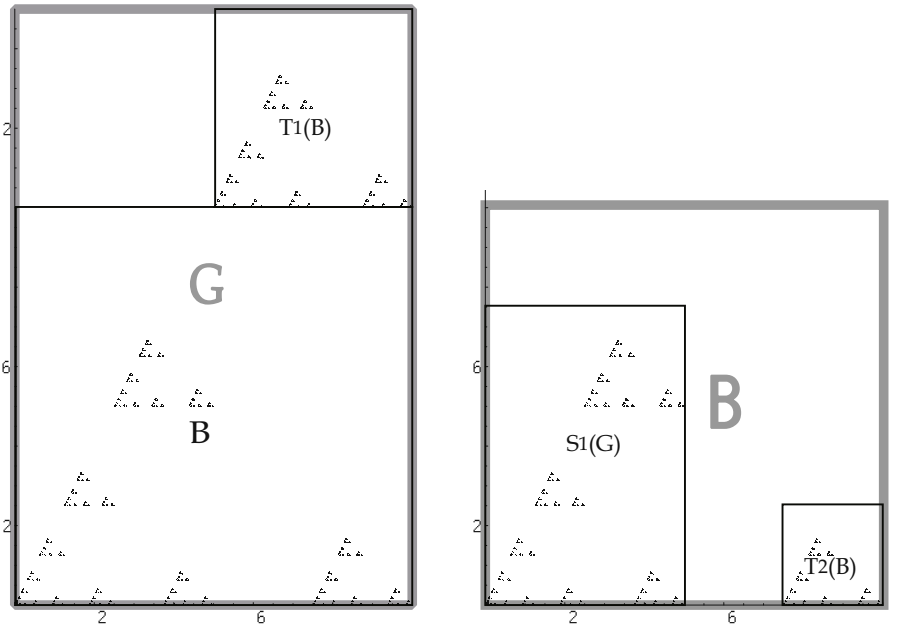


Fig. 5. Graph-directed sets and Brattka's function

Example 12 (continued). The invariant set for Example 12 can be characterized in terms of graph-directed sets as follows. Let

$$S_1(x,y)=\Big(\frac{x}{2},\frac{y}{2}\Big),\ S_2(x,y)=\Big(\frac{y}{4}+\frac{1}{2},\frac{x}{4}\Big),\ S_3(x,y)=\Big(\frac{x+2}{4},\frac{y+1}{4}\Big),$$

$$S_4(x,y)=\Big(\frac{y}{8}+\frac{3}{4},-\frac{x}{8}+\frac{1}{4}\Big),\ S_5(x,y)=\Big(\frac{x+6}{8},-\frac{y}{8}+\frac{1}{8}\Big),\ T(x,y)=\Big(\frac{x}{2}+\frac{1}{2},\frac{y}{2}\Big).$$

Then the closure C of the graph of Example 12 is characterized by

$$C=\bigcup_{i=1}^{3}S_i(C)\cup D,$$
$$D=\bigcup_{i=4}^{5}S_i(C)\cup T(D),$$

where $D=C\cap([\frac{3}{4},1)\times[0,\frac{1}{4}])$.

C and D also satisfy

$$C=\bigcup_{i=1}^{5}S_i(C)\cup T(D),$$
$$D=\bigcup_{i=4}^{5}S_i(C)\cup T(D).$$

6 Markov Self-similar Sets versus Graph-Directed Sets

Tsujii has defined "Markov-self-similar sets" and investigated their properties in [13].

Definition 17 (Markov-self-similar sets; [13]). *Let $\mathcal{T}$ be an N-tuple of N contraction similarities. It can be expressed as a matrix of the form*

$$\mathcal{T}=\left\{\begin{array}{cccc}T_{11}&T_{12}&\dots&T_{1N}\\T_{21}&T_{22}&\dots&T_{2N}\\\dots&\dots&\dots&\dots\\T_{N1}&T_{N2}&\dots&T_{NN}\end{array}\right\},$$

where each T_{kl} is a contraction similarity, and not all of $T_{k1},T_{k2},\dots,T_{kN}$ are the null map for each k. (Precisely speaking, the null map is not a contraction. We regard it here, however, as a contraction with the contraction ratio 0. Any proper contraction should have the positive contraction ratio.)

An N-tuple $(K_1,K_2,\dots,K_N)$ of non-empty sets is called Markov-self-similar for $\mathcal{T}$ if it satisfies the following conditions.

$$K_k = \bigcup_{i=1}^{N} T_{ki}(K_i) \quad \text{for} \quad k = 1, \ldots, N,$$

for some $s > 0$,

$$0 < H^s(K_i) < \infty \quad \text{for} \quad i = 1, \ldots, N,$$

and, for such an s,

$$H^s(T_{ki}(K_i) \cap T_{kj}(K_j)) = 0 \ (i \neq j) \quad \text{for} \quad k = 1, \ldots, N.$$

For a non-negative number t, we define an $N \times N$ non-negative matrix $R(t) = ((c_{ki})^t)$, where c_{ki} denotes the similarity ratio of T_{ki} for $k, i = 1, \ldots, N$.

Tsujii has proved the following theorem.

Theorem 18 (Tsujii; [13]). *Let* $\mathcal{T}$ *be as in Definition 17, satisfying further the following conditions.*

a) *There exists a non-empty open set* V *for which*

$$T_{ki}(V) \subset V \text{ and } T_{ki}(V) \cap T_{kj}(V) = \phi \text{ if } i \neq j \text{ for all } k, i, j = 1, \ldots, N.$$

b) *The matrix* $R(0) = ((c_{ki})^0)$ *is irreducible and the maximal eigenvalue* $\lambda(0)$ *of the matrix* $R(0) = ((c_{ki})^0)$ *is greater than 1, where* $0^0 = 0$.

Then the following hold.

(i) *There exists a unique N-tuple* $(K_1, K_2, \ldots, K_N)$, *called the family of Markov-self-similar sets for* $\mathcal{T}$.
(ii) *For any* $k = 1, \ldots, N$, *the Hausdorff dimension of* K_k *is the* s *for which the maximal eigenvalue* $\lambda(s)$ *of* $R(s)$ *is 1.*
(iii) *Let* $(x_1, \ldots, x_N)$ *be the positive eigenvector of* $R(s)$. *Then there exists a positive* c *such that*

$$H^s(K_k) = cx_k \quad \text{for} \quad k = 1, \ldots, N.$$

Theorem 19. *A family of graph-directed sets can be obtained as a family of Markov-self-similar sets. The converse also holds.*

Proof. a) We show that graph-directed sets $F_1, F_2, \ldots, F_K$ for (V, E) form a subset of a family of Markov-self-similar sets $K_1, K_2, \ldots, K_N$ for some matrix $\mathcal{T}$ as in Definition 17. Furthermore, $\{F_i\}$ and $\{K_j\}$ are identical sets.

Let $\mathcal{S}$ and $F_1, \ldots, F_K$ be as in Remark 14. Let

$$\begin{aligned} n_1 &= \max(n_{11}, n_{21}, \ldots, n_{K1}), \\ n_2 &= \max(n_{12}, n_{22}, \ldots n_{K2}), \\ &\cdots, \\ n_K &= \max(n_{1K}, n_{2K}, \ldots n_{KK}), \\ u_j &= n_1 + \cdots + n_j, \quad \text{for} \quad j = 1, \ldots, K. \end{aligned}$$

In order to transfer graph-directed sets to Markov-self-similar sets, we expand the index set V to:

$$I = \{1, 2, \ldots, u_K\}.$$

We construct maps $\{T_{ik}\}, 1 \leq i, j \leq u_K$ from the maps in $\mathcal{S}$ in Remark 14 as follows. For each i, k, $1 \leq i, j \leq u_1$, define

$$T_{ik} = \begin{cases} S_k^{11} & (k = 1, \ldots, n_{11}) \\ \emptyset & (k = n_{11}+1, \ldots, u_1) \\ S_{k-u_1}^{12} & (k = u_1+1, \ldots, u_1+n_{12}) \\ \emptyset & (k = u_1+n_{12}+1, \ldots, u_2) \\ \cdots & \cdots \\ S_{k-u_{K-1}}^{1K} & (k = u_{K-1}+1, \ldots, u_{K-1}+n_{1K}) \\ \emptyset & (k = u_{K-1}+n_{1K}+1, \ldots, u_K) \ . \end{cases}$$

For each i, $u_1 + 1 \leq i \leq u_2$, define

$$T_{ik} = \begin{cases} S_k^{21} & (k = 1, \ldots, n_{21}) \\ \emptyset & (k = n_{21}+1, \ldots, u_1) \\ S_{k-u_1}^{22} & (k = u_1+1, \ldots, u_1+n_{22}) \\ \emptyset & (k = u_1+n_{22}+1, \ldots, u_2) \\ \cdots & \cdots \\ S_{k-u_{K-1}}^{2K} & (k = u_{K-1}+1, \ldots, u_{K-1}+n_{2K}) \\ \emptyset & (k = u_{K-1}+n_{2K}+1, \ldots, u_K) \ . \end{cases}$$

Continue in this way, and finally for i, $u_{K-1} + 1 \leq i \leq u_K$, define

$$T_{ik} = \begin{cases} S_k^{K1} & (k = 1, \ldots, n_{K1}) \\ \emptyset & (k = n_{K1}+1, \ldots, u_1) \\ S_{k-u_1}^{K2} & (k = u_1+1, \ldots, u_1+n_{K2}) \\ \emptyset & (k = u_1+n_{K2}+1, \ldots, u_2) \\ \cdots & \cdots \\ S_{k-u_{K-1}}^{KK} & (k = u_{K-1}+1, \ldots, u_{K-1}+n_{KK}) \\ \emptyset & (k = u_{K-1}+n_{KK}+1, \ldots, u_K) \ . \end{cases}$$

$\mathcal{T} = (T_{ik})$ is a $u_K \times u_K$ square matrix consisting of contraction similarities which satisfy the condition in Definition 17. It also satisfies the conditions a) and b) in Theorem 18. By virtue of (i) and (ii) in Theorem 18, there is a family of Markov-self-similar sets $(K_1, \cdots, K_{u_k})$ for $\mathcal{T}$, and it is obvious from the definition $\mathcal{T}$ that

$$\begin{aligned} K_i &= F_1 \quad \text{for} \quad i = 1, \ldots, u_1, \\ K_i &= F_2 \quad \text{for} \quad i = u_1+1, \ldots, u_2, \\ &\ldots, \\ K_i &= F_K \quad \text{for} \quad i = u_{K-1}+1, \ldots, u_K. \end{aligned}$$

Now, the family of the graph-directed sets $F_1, \ldots, F_K$ for $\mathcal{S}$ is a sub-family of the family of Markov-self-similar sets $K_1, K_2, \ldots, K_{u_K}$ for $\mathcal{T}$, and $\{F_m\} = \{K_i\}$.

b) Markov-self-similar sets are graph-directed sets by definition. ■

Remark 20. *1) It is self-evident that the Hausdorff dimension of the graph-directed sets for $\mathcal{S}$ (Theorem 16) is equal to that of the Markov-self-similar sets for the corresponding $\mathcal{T}$.*
2) The matrix $(T_{ik})_{i,k}$ defined in the proof of Theorem 19 is singular.
3) It is easier to evaluate the eigenvalues with respect to $\mathcal{S}$ than with respect to $\mathcal{T}$.

References

1. Bedford, T.: Dimension and dynamics of fractal recurrent sets. J. Lond. Math. Soc. 33(2), 89–100 (1986)
2. Brattka, V.: Some Notes on Fine Computability. J. Univers. Comput. Sci. 8, 382–395 (2002)
3. Edgar, G.A.: Measure, Topology and Fractal Geometry. Undergraduate Texts in Mathematics. Springer, Heidelberg (1990)
4. Falconer, K.J.: The geometry of fractal sets. Cambridge University Press, Cambridge (1986)
5. Falconer, K.J.: Fractal Geometry. John Wiley & Sons, Chichester (1990)
6. Fernau, H.: Infinite iterated function systems. Math. Nachr. 170, 79–91 (1994)
7. Fine, N.J.: On the Walsh Functions. Trans. Amer. Math. Soc. 65, 373–414 (1949)
8. Hutchinson, J.E.: Fractals and self similarity. Indiana Univ. Math. J. 30, 713–747 (1981)
9. Mauldin, R.D., Williams, S.C.: Hausdorff dimension in graph directed constructions. Trans. Amer. Math. Soc. 309, 811–829 (1988)
10. Mori, T.: On the computability of Walsh functions. Theor. Comput. Sci. 284, 419–436 (2002)
11. Mori, T.: Computabilities of fine-continuous functions. In: Blank, J., Brattka, V., Hertling, P. (eds.) CCA 2000. LNCS, vol. 2064, pp. 200–221. Springer, Heidelberg (2001)
12. Mori, T., Tsujii, Y., Yasugi, M.: Fine-Computable Functions and Effective Fine-Convergence. Mathematics Applied in Science and Technology (accepted)
13. Tsujii, Y.: Markov-self-similar sets. Hiroshima Math. J. 21(3), 491–519 (1991)

Is P = PSPACE for Infinite Time Turing Machines?

Joost Winter*

Bovenover 215, 1025 JN Amsterdam, The Netherlands

1 Introduction

In the existing literature on infinite time Turing machines (ITTM), which were originally defined in [3], issues of time complexity have been widely considered. The question $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ for Infinite Time Turing Machines, and several variants on it, are treated in, e.g., [6], [2], and [4].

Besides time complexity, we may also try to look at issues of space complexity in ITTMs. However, because an ITTM contains tapes of length ω, and all non-trivial ITTM computations will use the entire, ω-length tape, simply measuring the space complexity by counting the portion of the tape used by the computation is not an option. In [5], therefore, an alternate notion of space complexity is provided, that is based on looking at the levels of Gödel's constructible hierarchy where the snapshots of the computation can be found.

With this notion of space complexity of ITTMs, we can consider questions such of the type $\mathbf{P} \stackrel{?}{=} \mathbf{PSPACE}$, analogous to the existing work on the question $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ for ITTMs. In this paper, we will look at some of these questions, in a manner analogous to the earlier work on ITTM time complexity classes.

2 Definitions

For any infinite time Turing machine computation, we define the following operations:

Definition 1. *(1) $q_\alpha^T(x)$ denotes the state a machine T is in at stage α, having started from the input x; (2) $h_\alpha^T(x)$ denotes the position of the head at stage α, having started from the input x; and (3) $c_{i,\alpha}^T(x)$, where $i \in \{1, 2, 3\}$, denotes the content of tape i at stage α, having started from the input x. Also, we will define another operation, $c_{i,n,\alpha}^T(x)$, which will be defined as* 1 *if* $n \in c_{i,\alpha}^T(x)$, *and* 0 *otherwise.*

Following [5] and earlier articles such as [6], the ITTM space and time complexity classes are defined as follows:

* (*Received by the editors.* 16 February 2008. *Revised.* 29 April 2008; 29 July 2008. *Accepted.* 8 September 2008.)

M. Archibald et al. (Eds.): ILC 2007, LNAI 5489, pp. 126–137, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Definition 2. *If T is a machine that eventually reaches the halting state q_{f}, and α is the unique ordinal such that $q_\alpha^T(x) = q_{\mathrm{f}}$, then we say that* $\mathbf{time}(x, T) = \alpha$.

Definition 3. *For any function $f : \mathbb{R} \to \mathrm{Ord}$ and any* ITTM *T, we say that T is a* ***time*** *f* ***machine*** *if* $\mathbf{time}(x, T)$ *is defined for all x and, for all $x \in \mathbb{R}$, we have* $\mathbf{time}(x, T) \leq f(x)$. *For any ordinal ξ, we say that T is a* ***time*** *ξ* ***machine*** *if T is a time f machine for the constant function f with $f(x) = \xi$ for all x.*

Definition 4. *The family of all sets of reals that are decidable by a time f machine is denoted by $\mathbf{P}_f$. For any ordinal ξ, $\mathbf{P}_\xi$ denotes the family of all sets of reals that are decidable by a time η machine for some $\eta < \xi$. The class* **P** *is defined as the class $\mathbf{P}_{\omega^\omega}$.*

Definition 5. *We define: (1) $\ell_\alpha^T(x) := \min\{\eta : c_{i,\alpha}^T \in \mathbf{L}_\eta[x]$ for $i \in \{1, 2, 3\}\}$, and (2)* $\mathbf{space}(x, T) := \sup\{\ell_\xi^T : \xi \leq \mathbf{time}(x, T)\}$.

Definition 6. *For any ITTM T, we say that T is a* ***space*** *f* ***machine*** *if, for all $x \in \mathbb{R}$, we have* $\mathbf{space}(x, T) \leq f(x)$. *For any ordinal ξ, we say that T is a* ***space*** *ξ* ***machine*** *if T is a space f machine for the constant function f such that $f(x) = \xi$ for all x.*

Definition 7. *For any function f,* $\mathbf{PSPACE}_f$ *denotes the class of all sets of reals decidable by a space f machine. For any ordinal ξ,* $\mathbf{PSPACE}_\xi$ *denotes the class of all sets of reals that are decidable by a space η machine for some $\eta < \xi$.* **PSPACE** *is defined as the class* $\mathbf{PSPACE}_{\omega^\omega}$.

Furthermore, we define the weak halting problem h and its relativized versions as follows:

Definition 8. *We let h denote the set $\{e : \phi_e(e) \downarrow 0\}$. Furthermore, for any α, we let h_α denote the set $\{e : \phi_e(e) \downarrow 0 \wedge \mathbf{time}(e, e) \leq \alpha\}$*

In the remainder of these papers, we will readily use some of the more standard terminology and definitions in the ITTM literature, such as *clockable* ordinals and *writable* reals and ordinals. We also will refer to a number of basic or easily proven results, a few of which will be listed below:

Lemma 1 (Welch). *For any real x, the supremum of x-clockable ordinals γ^x is equal to the supremum of x-writable ordinals λ^x.*

We also have the following result about the weak halting problem h, and its relativized variants h_α:

Proposition 1. *For every ordinal α, we have $h_\alpha \notin \mathbf{P}_{\alpha+1}$. Moreover, h is undecidable.*

Regarding the constructible hierarchy **L**, we have the follwing:

Lemma 2. *If $\alpha > \omega$, and $A \subset \omega$, and $A \in \mathbf{L}_\alpha[w]$ for a certain set w, and for a certain set B, $|(A \backslash B) \cup (B \backslash A)| < \omega$, then $B \in \mathbf{L}_\alpha[w]$.*

3 P ⊆ PSPACE for Infinite Time Turing Machines

First of all, we want to be sure that **P** is always a subset of **PSPACE**. Due to the somewhat unusual definition of **PSPACE**, however, it is not directly evident that $\mathbf{time}(x,T) \geq \mathbf{space}(x,T)$ for all x, T. It will be shown that this indeed is the case, by showing that we can represent any ITTM-computation of length α, starting from a set w, in level $\mathbf{L}_\alpha[w]$ of the constructible hierarchy. We will first define what we mean with a 'representation' of an ITTM computation:

Definition 9. *We use the notation $C^T_\alpha(w)$ for a representation of an ITTM computation of a machine T from w in α steps, of the form*

$$C^T_\alpha(w) = \{(\beta, q^T_\beta(w), h^T_\beta(w), c^T_{1,\beta}(w), c^T_{2,\beta}(2), c^T_{3,\beta}(w)) : \beta \leq \alpha\}$$

where β is an ordinal representing the stage, and $q^T_\beta(w)$, $h^T_\beta(w)$, $c^T_{i,\beta}(w)$ are as they were defined earlier.

Lemma 3. *When we have a suitable representation of Turing machines as finite sets, the notion 'a one step ITTM computation by a machine T from the state $t = (q, h, c_1, c_2, c_3)$ results in a state $t' = (q', h', c'_1, c'_2, c'_3)$ is representable by a formula of first order logic.*

Lemma 4. *The notion '$X = C^T_\alpha(w)$: X represents an ITTM computation from w in α steps' is representable in the language of set theory.*

Proof. Note that X represents an ITTM computation from w in α steps, if and only if all of the following five conditions hold: (1) Every element of X is of the form $(\beta, q, h, c_1, c_2, c_3)$ where β is an ordinal smaller than or equal to α; (2) If we have $(\beta, q_1, h_1, c_{1,1}, c_{2,1}, c_{3,1}) \in X$ and $(\beta, q_2, h_2, c_{1,2}, c_{2,2}, c_{3,2}) \in X$, then $q_1 = q_2$, $h_1 = h_2$, $c_{1,1} = c_{1,2}$, $c_{2,1} = c_{2,2}$ and $c_{3,1} = c_{3,2}$; (3) We have $(0, q_s, h_0, c_{1,0}, c_{2,0}, c_{3,0})$ where q_s is the initial state of the Turing machine, $h_0 = 0$, $c_{1,0} = w$, $c_{2,0} = c_{3,0} = \varnothing$; (4) If β is a successor ordinal $\gamma + 1$, and $\beta < \alpha$, then there are elements $(\gamma, q_1, h_1, c_{1,1}, c_{2,1}, c_{3,1})$ and $(\beta, q_2, h_2, c_{1,2}, c_{2,2}, c_{3,2})$ such that a one step Turing computation from the snapshot $(q_1, h_1, c_{1,1}, c_{2,1}, c_{3,1})$ results in $(q_2, h_2, c_{1,2}, c_{2,2}, c_{3,2})$; (5) If β is a limit ordinal, then there is an element $\beta, q, h, c_1, c_2, c_3$ such that q is the limit state of the Turing machine, $h = 0$, and for each of c_i, we have that $x \in c_i$ if and only if for every ordinal $\gamma < \beta$ there is an ordinal δ greater than or equal to γ and smaller than β such that if $(\delta, q', h', c'_1, c'_2, c'_3) \in X$, then $x \in c'_i$.

It should be clear that all of these notions are representable by a formula of first order logic in the language of set theory. ■

Now we will turn to the main theorem. The final aim will be to show, that all computations starting from an input w of length α can be carried out while only having tape contents slightly more complicated than in $\mathbf{L}_\alpha[w]$. The addition of 'slightly' here only signifies that one needs a small finite fixed extra number of steps—the construction made here assumes gives a crude upper bound of 12, although the precise number might be lower than that. First, we need an additional auxiliary lemma:

Lemma 5. *If $\beta, q, h, c_1, c_2, c_3$ are all in $\mathbf{L}_\alpha[w]$, then $(\beta, q, h, c_1, c_2, c_3)$ is in $\mathbf{L}_{\alpha+10}[w]$.*

Proof. We assume that (c_2, c_3) is defined as $\{\{c_2\}, \{c_2, c_3\}\}$. If c_2 and c_3 are in $\mathbf{L}_\alpha$, then $\{c_2\}$ and $\{c_2, c_3\}$ are definable subsets of $\mathbf{L}_\alpha$, and are thus in $\mathbf{L}_{\alpha+1}$. Consequently, $\{\{c_2\}, \{c_2, c_3\}\}$ is a definable subset of $\mathbf{L}_{\alpha+1}$, and is hence in $\mathbf{L}_{\alpha+2}$.

Assuming that $(\beta, q, h, c_1, c_2, c_3)$ is defined as $(\beta, (q, (h, (c_1, (c_2, c_3)))))$, it easily follows by repeating the above procedure that $(\beta, q, h, c_1, c_2, c_3)$ is in $\mathbf{L}_{\alpha+10}$. ■

Now we will turn to the main theorem, which will be proved using a kind of simultaneous induction:

Theorem 1. *The following propositions are true:*

1. *For any infinite ordinal α and any input set $w \subseteq \omega$, we have that $c^T_{1,\alpha}(w)$, $c^T_{2,\alpha}(w)$, $c^T_{3,\alpha}(w)$ are in $\mathbf{L}_{\alpha+1}[w]$. If α is a finite ordinal, however, we can only be sure that the sets are in $\mathbf{L}_{\alpha+2}[w]$.*
2. *If α is a successor ordinal above ω and is equal to $\beta + n$, where n is a natural number, then $c^T_{1,\alpha}(w)$, $c^T_{2,\alpha}(w)$, $c^T_{3,\alpha}(w)$ are in $\mathbf{L}_{\beta+1}[w]$. This is a strengthening of the above property.*
3. *For any ordinal $\alpha \geq \omega$ and any input set $w \subseteq \omega$, we have $(\alpha, q^T_\alpha(w), h^T_\alpha(w), c^T_{1,\alpha}(w), c^T_{2,\alpha}(w), c^T_{3,\alpha}(w)) \in \mathbf{L}_{\alpha+11}[w]$.*
4. *For any ordinal $\alpha \geq \omega$ and any input set $w \subseteq \omega$, we have $C^T_\alpha(w) \in \mathbf{L}_{\alpha+12}[w]$.*

Proof. The tactic here will be to first show that, for any α, if property (1) holds for all ordinals smaller than or equal to α, and property (4) holds for all ordinals strictly smaller than α, then properties (2), (3), and (4) hold for α. Then, we will show that if properties (1), (2), (3), and (4) hold for all ordinals strictly smaller than α, then property (1) also holds for α.

- $1 \rightarrow 2$: We use here the fact that, assuming $\alpha > \omega$, and $\alpha = \beta + n$, for each i in $\{1, 2, 3\}$, $c^T_{i,\beta}(w)$ and $c^T_{i,\alpha}(w)$ can only differ by finitely many elements. So it follows from $c^T_{i,\beta}(w) \in \mathbf{L}_{\beta+1}[w]$ that $c^T_{i,\alpha}(w) \in \mathbf{L}_{\beta+1}[w]$.
- $1 \rightarrow 3$: We have that $\alpha \in \mathbf{L}_{\alpha+1}[w]$ for any w. Because $q^T_\alpha(w)$ and $h^T_\alpha(w)$ are finite sets, they are also in $\mathbf{L}_{\alpha+1}[w]$ by the assumption that α is infinite, and the desired result follows from the assumption of (1) and an application of Lemma 5.
- $1\ (+3) \rightarrow 4$ for successor ordinals: If α is a successor ordinal $\beta + 1$, then we have that x is in $C^T_\alpha(w)$ if and only if $x \in C^T_\beta(w)$ or if x is equal to $(\alpha, q^T_\alpha(w), h^T_\alpha(w), c^T_{1,\alpha}(w), c^T_{2,\alpha}(w), c^T_{3,\alpha}(w))$. Because by the inductive hypothesis $C^T_\beta(w)$ and hence, by transitivity, every element of it, is in $\mathbf{L}_{\beta+12}[w]$, or $\mathbf{L}_{\alpha+11}[w]$, and because by (3)—which was already proven from (1)—$(\alpha, q^T_\alpha(w), h^t_\alpha(w), c^T_{1,\alpha}(w), c^T_{2,\alpha}(w), c^T_{3,\alpha}(w))$ is in $\mathbf{L}_{\alpha+11}$, it follows that $C^T_\alpha(w)$ is a definable subset of $\mathbf{L}_{\alpha+11}[w]$, and hence an element of $\mathbf{L}_{\alpha+12}[w]$.
- $1\ (+3) \rightarrow 4$ for limit ordinals: If α is a limit ordinal, note that, for any $\beta < \alpha$, we have by the inductive hypothesis that $C^T_\beta(w) \in \mathbf{L}_{\beta+12}[w]$, and hence also that $C^T_\beta(w) \in \mathbf{L}_\alpha[w]$. Now consider the following set $C^T_{<\alpha}(w)$:

$$\{X \in L_\alpha[w] : \exists\beta(\beta \in \alpha \wedge X \in C^T_\beta(w))\}$$

Note that $\beta \in \alpha$ basically just says, 'β is an ordinal smaller than α'. This set is clearly a set definable from $L_\alpha[w]$, and it is the union of all computations from w in less than α steps. To obtain $C^T_\alpha(w)$, we only need to add $(\alpha, q^T_\alpha(w), h^t_\alpha(w), c^T_{1,\alpha}(w), c^T_{2,\alpha}(w), c^T_{3,\alpha}(w))$ to this set, which can be done easily like in the above case of successor ordinals.

- Property (1) for finite ordinals: note that, at stage n, because the head starts at 0 and can only move forward one step at a time, the head can only be at a location between 0 and n. Because of this, the only positions of $c^T_{i,n}$ that can have changed by stage n are the positions in the range $[0, n-1]$. Because we have $c^T_{0,0} = w$ and $c^T_{1,0} = c^T_{2,0} = \varnothing$, we also have $c^T_{i,0} \in L_1[w]$.
  Now, we can define $c^T_{i,n}$ as $(c^T_{i,0} \backslash \{j_1, \ldots, j_l\}) \cup \{k_1, \ldots, k_m\}$, which is a definable subset of $L_n[w]$ as a result of the fact that all the j and k are not larger than $n-1$ and are thus elements of $L_n[w]$.
- Inductive case for successor ordinals larger than ω: If α is a successor ordinal $\beta+1$, for each $i \in \{1,2,3\}$ $c^T_{i,\alpha}(w)$ and $c^T_{i,\beta}(w)$ can only differ by one element. Taking the largest limit ordinal γ below α, such that $\alpha = \gamma + n$, an appeal to Lemma 2 (and possibly to case (2) for β, if β itself is a successor ordinal) suffices to show that for each $i \in \{1,2,3\}$, $c^T_{i,\alpha}$ is in $\mathbf{L}_{\gamma+1}[w]$ and hence also in $\mathbf{L}_{\alpha+1}[w]$.
- Inductive case for limit ordinals: if α is a limit ordinal, we have, by inductive assumption, that for all $\beta < \alpha$, $C^T_\beta(w) \in \mathbf{L}_{\beta+12}[w]$, and hence, also that $C^T_\beta(w) \in \mathbf{L}_\alpha[w]$.
  To start, let us find a way of identifying ordinals smaller than α in $\mathbf{L}_\alpha[w]$. If $\alpha \notin \mathbf{L}_\alpha[w]$, the set of ordinals in $L_\alpha[w]$ is exactly the set of ordinals smaller than α: in this case we can identify any ordinal smaller than α in $\mathbf{L}_\alpha[w]$ with the property $\mathrm{Ord}(x)$. If somehow we do have $\alpha \in \mathbf{L}_\alpha[w]$, we can identify any ordinal smaller than α in $L_\alpha[w]$ with the property $x \in \alpha$. Depending on which case we are in, let us take the appropriate property.
  Now note that the definition of cells at limit values implies that $x \in c^T_{i,\alpha}(w)$ if and only if for every ordinal $\beta < \alpha$, there is an ordinal $\gamma < \alpha$ with $\gamma \geq \beta$, such that $x \in c^T_{i\gamma}(w)$. We can now define $c^T_{i,\alpha}(w)$ in the following way:
  $c^T_{i,\alpha}(w)$ is the set of all $n \in \omega$, such that for every ordinal $\beta < \alpha$ there is an ordinal $\gamma < \alpha$ with $\gamma \geq \beta$, such that n is an element of the $i+3$th argument of $C^T_\gamma(w)$ (i.e., $n \in c^T_{i,\gamma}(w)$).

Although it should be noted that we have proved property (3) and (4) only for infinite ordinals, this does not pose a problem, because for any finite n, we have that $C^T_n(w)$ will occur at at least some finite stage of Gödel's Constructible hierarchy. As a result, we will have $C^T_n(w) \in \mathbf{L}_\omega[w]$ for any finite n, and the inductive steps will still work for the ω-case. ■

This gives us the following result about **P** and **PSPACE:**

Proposition 2. *For any* ITTM T *and any real* x, *if* $\mathbf{time}(x,T) > \omega$, *we have* $\mathbf{space}(x,T) \leq \mathbf{time}(x,T)$. *Also, if* $\mathbf{time}(x,T) \leq \omega$, *we have* $\mathbf{space}(x,T) \leq \omega$. *Hence, for any function* f *such that* $f(x) > \omega$ *for all* x, *every time* f *machine is a space* f *machine, and consequently we have* $\mathbf{P}_f \subseteq \mathbf{PSPACE}_f$.

Proof. Assume that $\alpha > \omega$ and $\alpha = \mathbf{time}(x, T)$. For any $\xi < \alpha$ with $\xi \geq \omega$, we have, by Theorem 1, $c^T_{i,\xi}(x) \in \mathbf{L}_{\xi+1}[x]$ (here we have $+1$ instead of $+2$ because $\xi \geq \omega$) for $i \in \{1, 2, 3\}$, and hence $c^T_{i,\xi}(x) \in \mathbf{L}_\alpha[x]$. For finite ξ, we have $c^T_{i,\xi}(x) \in \mathbf{L}_\omega[x]$ for $i \in \{1, 2, 3\}$, and again $c^T_{i,\xi}(x) \in \mathbf{L}_\alpha[x]$. Finally, we notice that α must be equal to $\beta + 1$ for some β, as ITTMs cannot halt at limit ordinal stages. It is also immediate from the definition of ITTMs that $c^T_{i,\beta}(x)$ and $c^T_{i,\beta+1}(x)$ can only differ by one element at most. Because of this, it follows from $c^T_{i,\beta}(x) \in \mathbf{L}_\alpha[x]$ that $c^T_{i,\beta+1}(x) \in \mathbf{L}_\alpha[x]$ using Lemma 2 and the fact that $\alpha > \omega$. So at all stages of the computations, the content of the tape is inside $\mathbf{L}_\beta[x]$, so we have $\mathbf{space}(x, T) \leq \beta$, and hence $\mathbf{space}(x, T) \leq \mathbf{time}(x, T)$.

If $\alpha \leq \omega$ and $\alpha = \mathbf{time}(x, T)$, we have for all $\xi < \alpha$, by Theorem 1, $c^T_{i,\xi}(x) \in \mathbf{L}_\omega[x]$ for $i \in \{1, 2, 3\}$, and $\mathbf{space}(x, T) \leq \omega$ now follows directly. ■

4 The Question Whether $\mathbf{P}_f = \mathbf{PSPACE}_f$

Now we have seen that, in all cases where the range of f contains only infinite ordinals, $\mathbf{P}_f$ is a subset of $\mathbf{PSPACE}_f$, one is inclined to wonder whether this inclusion can be shown to be proper. In this section, we will show that this, indeed, is the case at least for a number of functions f.

4.1 $\mathbf{P}_\alpha \neq \mathbf{PSPACE}_\alpha$ for Ordinals Up to ω^2

For the first result, the idea will be to construct a function that can be shown to be $\mathbf{PSPACE}_{\omega+2}$, but that, at the same time, can also be shown to be *not* $\mathbf{P}_\alpha$ for any $\alpha < \omega^2$. This will be done using the notion of arithmetical reals, which are defined more-or-less analogously to arithmetic sets of reals.

First, we observe the following fact about the location of recursive sets in Gödel's constructible hierarchy:

Lemma 6. *For any recursive set $w \in \mathbb{R}$, we can write w on the tape in ω steps, and thus we have that $w \in \mathbf{L}_{\omega+1}[\varnothing]$, and thus also $w \in \mathbf{L}_{\omega+1}[x]$ for any $x \in \mathbb{R}$.*

Proof. Because w is recursive, there is a Turing machine T that halts on all inputs $n \in \mathbb{N}$, such that $\phi_T(n) = 1$ if and only if $n \in w$. Now consider an ITTM that, in turn, for each number n, simulates the machine T, and then writes the correct number to the nth cell of the output tape. Because every individual computation will terminate in a finite number of steps, this can be done in ω steps, and we can go directly from the limit state to the halting state. ■

We will now consider the notion of arithmetical sets of natural numbers: a set $S \subseteq \mathbb{N}$ is arithmetical if and only if there is a formula ϕ of PA such that $\phi(x) \iff x \in N$.

Lemma 7. *The set $A := \{x \in \mathbb{R} : x \text{ is arithmetical}\}$, is not arithmetical.*

Proof. This is shown in Example 13.1.9 in [1]. ■

This gives us the following result:

Lemma 8. *For any ordinal $\alpha \leq \omega^2$, the set $A := \{x \in \mathbb{R} : x \text{ is arithmetical}\}$ is not in $\mathbf{P}_\alpha$.*

Proof. By Theorem 2.6 in [3], the arithmetic sets are exactly the sets which can be decided in time $\omega \cdot n$ for some $n \in \mathbb{N}$. Because A is not arithmetical, it cannot be decided by any algorithm using a bounded finite number of limits, and hence, it is not in $\mathbf{P}_\alpha$ for any $\alpha \leq \omega^2$. ■

It turns out, however, that this set A is in $\mathbf{PSPACE}_{\omega+2}$:

Theorem 2. *The set $A := \{x \in \mathbb{R} : x \text{ is arithmetical}\}$ is in $\mathbf{PSPACE}_{\omega+2}$.*

Proof. We will make use of an ITTM with three scratch tapes (see [7, Proposition 4.28] for a justification why this can be done without affecting the space complexity): the first is used to enumerate over all possible formulas, the second is used to enumerate over, and store the choices of the quantifiers occurring in each formula, and the third is used for the actual evaluation of the resulting, quantifier-free, formulas.

First, we can note that we can, without any problems, enumerate over all possible formulae that determine arithmetic sets, on one of the scratch tapes: every formula can be coded by a set which is finite, and hence in $\mathbf{PSPACE}_\omega$. Now, we may also assume that this enumeration only gives formulae in a normal form, with all quantifiers at the front. By enumerating over the formulas in such a way that they are enumerated over by increasing length, we can be sure that no infinite (and hence, no nonrecursive) set will appear on the tape before the very end of the computation; by switching all cells with content 0 to 1 and back during the enumeration process, we can furthermore ensure that the tape, at the very end, will be filled with 1s, and hence again contain a recursive set.

Given a formula ϕ, we can also determine for any x whether $\phi(x)$ holds, by only writing down finite sets on a scratch tape. That this is the case can be shown by induction: if ϕ is quantifier free, we can simply evaluate the formula; if ϕ has an existential quantifier at the front and is of the form $\exists x_1 \psi$, we can write down every possible value n for x_1 on the second scratch tape (right after any earlier quantifiers that may have been written there), coded by n 1s followed by a single 0, recursively evaluate $\psi[n/x_1]$ (which is possible by the inductive assumption) using the rest of the second, as well as the third scratch tape; succeed if this is successful, and fail otherwise; if ϕ has a universal quantifier at the front and is of the form $\forall x_1 \phi$, we again write down every possible value n for x_1 at the front of the scratch tape, recursively evaluate $\psi[n/x_1]$, and now fail if this fails, and continue otherwise.

At limit stages during this process, we will, after a fixed prefix of sequences of 1s followed by a single 0, end up with an infinite sequence of 1s on the second scratch tape: this is a set containing only finitely many 0s, and is, hence, clearly recursive. During the actual evaluation of formulas, on the third tape, we can switch every cell that is accessed and that has value 0 to 1 and back again: this

way, we also easily prevent the accidental occurrence of nonrecursive sets on the third scratch tape.

Now, the strategy for the whole function will be:

1. enumerate over all possible formulae in the aforementioned normal form;
2. for each such formula, test for each x whether the formula holds for x if and only if the xth position on the input tape is a 1;
3. succeed if we have found such a formula, and fail otherwise.

This can, as a result of some of the precautions, be done while writing out only recursive sets. ■

This gives us the following result on **P** and **PSPACE:**

Theorem 3. *For any α such that $\alpha \geq \omega + 2$ and $\alpha \leq \omega^2$, we have that $\mathbf{P}_\alpha \subsetneq \mathbf{PSPACE}_\alpha$. Also, $\mathbf{PSPACE}_{\omega+2}$ contains elements that are not in $\mathbf{P}_\alpha$.*

Proof. This follows directly from Theorem 8 and Theorem 2. ■

4.2 The Case of Recursive Ordinals

We will now show that the inequality $\mathbf{P}_\alpha \neq \mathbf{PSPACE}_\alpha$ holds for much wider range of ordinals. The strategy in showing this will be to show, that, for all recursive ordinals, $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$ which, together with the already known fact that $h_\alpha \notin \mathbf{P}_{\alpha+1}$, gives the desired result.

From Proposition 1, we know that for all α, $h_\alpha \notin \mathbf{P}_{\alpha+1}$. It turns out, however, that for all recursive ordinals α, we do have that $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$:

Proposition 3. *For any recursive ordinal α, such that $\alpha \geq \omega + 1$, we have $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$.*

Proof. We can compute h_α in the following way, by making use of an ITTM with several scratch tapes: to start, we will check whether the input corresponds to a natural number; if it does not, then we output 0, and if it does, we continue. Then, we will look if this natural number corresponds to a coding of an ITTM_0—again, we output 0 if this is not the case, and if it is, we continue. Note that the computation so far can be performed by writing only finite sets on the scratch tapes.

Now the real work can begin. First we write down the ordinal α, on the first scratch tape. Because α is recursive, we can do this in ω steps, so we are sure that the content of the first scratch tape, at all times, will be inside $\mathbf{L}_{\omega+1}[x]$ and, because $\alpha \geq \omega + 1$, also inside $\mathbf{L}_\alpha[x]$.

Once this is done, we will check for all ordinals β smaller than α (which can be easily found simply by restricting the ordinal written on the tape to all the elements smaller than a certain element, without affecting the space complexity in any way), whether the eth ITTM_0 has reached the halt state by stage β. If it turns out that this is the case, we look at the output: if the output is 0, then we will finish the computation by writing 1 on the output tape, and otherwise we

finish by writing 0 on the output tape. If we reach stage β during the simulation without having halted, we go on checking with the next ordinal. Finally, if we have exhausted all ordinals $\beta < \alpha$, we again finish by writing 0 on the output tape.

In the case where α is a limit ordinal, we are now done, because we know that no machine can halt at any limit ordinal stage. In the case where α is a successor ordinal $\eta + 1$, however, we will additionally check whether the computation has finished at α itself.

For any $\beta < \alpha$, we know that $c^T_{i,\beta}[x] \in \mathbf{L}_\alpha[x]$ for $i \in \{1, 2, 3\}$. Furthermore, in the case where α is a successor ordinal $\eta + 1$, we know that $c^T_{i,\eta}[x]$ and $c^T_{i,\alpha}[x]$ can only differ by one element at most, and because α and η are known to be infinite, we obtain $\mathbf{L}_\alpha[x]$ from $\mathbf{L}_\eta[x]$ using Lemma 2.

It follows that this computation never writes a set on any of the tapes that is not in $\mathbf{L}_\alpha[x]$. This proves that $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$ indeed holds. ■

Hence we have:

Theorem 4. *For every recursive successor ordinal $\alpha \geq \omega+1$, $\mathbf{P}_\alpha \subsetneq \mathbf{PSPACE}_\alpha$ holds.*

Unfortunately, however, this process cannot be easily extended to work for limit ordinals: given a limit ordinal α, we can still compute h_α by only writing out information of space complexity less than α on tape, but it seems hard, if not impossible, to do this bounded by a specific ordinal β below α.

4.3 The Case of Clockable Ordinals

It is, however, possible, to extend the above process to many writable successor ordinals. The strategy here is essentially the same as in the case of recursive ordinals: we know that h_α cannot be in $\mathbf{P}_{\alpha+1}$, and then we show that $h_\alpha \in \mathbf{PSPACE}_{\alpha+1}$. In the case of clockable ordinals, we can make use of the following theorem due to Philip Welch (Lemma 15 in [2]):

Theorem 5. *If α is a clockable ordinal, then every ordinal up to the next admissible beyond α is writable in time $\alpha + \omega$.*

Besides this, we already know that $\alpha \in \mathbf{L}_{\alpha+1}[0]$. However, we will also have $\alpha \cup \{\{\{\{\varnothing\}\}, \eta\} : \eta \leq \beta\} \in \mathbf{L}_{\alpha+1}[0]$ for any $\beta \leq \omega$. Because these sets $\{\{\{\varnothing\}\}, \eta\}$ are not ordinals, it is immediate that α and $\{\{\{\{\varnothing\}\}, \eta\} : \eta \leq \beta\}$ are always disjoint. Thus, we can consider these sets $\alpha \cup \{\{\{\{\varnothing\}\}, \eta\} : \eta \leq \beta\}$ as alternative representations for the ordinals up to $\alpha + \omega$, that are within $\mathbf{L}_{\alpha+1}[0]$. This way, using this 'alternative' representation of the ordinals between α and $\alpha + \omega$, instead of the regular ones, in combination with the fact that the snapshots $c^T_{i,\alpha+n}[x]$ can only differ from $c^T_{i,\alpha}[x]$ by finitely many elements, we can represent computations of length $\alpha+\omega$ from x within $\mathbf{L}_{\alpha+2}[x]$ using a construction similar to that in section 5.1.

This gives us that, for clockable α, every ordinal up to the next admissible ordinal beyond α can be written on the tape with the content of the tape inside $\mathbf{L}_{\alpha+2}[0]$ at all stages. This gives us the following theorem:

Theorem 6. *If β is a clockable ordinal, and α is a successor ordinal between $\beta + 3$ and the next admissible after β, then we have $\mathbf{P}_\alpha \subsetneq \mathbf{PSPACE}_\alpha$.*

Proof. This goes largely analogous to the case of Proposition 3, with the major difference that we can now write α on the tape while staying inside $\mathbf{L}_{\beta+2}[0]$. As a result from this, if $\alpha = \eta + 1$, we get from $\eta \geq \beta + 2$ that $h_\eta \in \mathbf{PSPACE}_{\eta+1}$, whereas $h_\eta \notin \mathbf{P}_{\eta+1}$, giving the desired result. ■

4.4 The Case of Suitable Functions f

We can extend the above results from ordinals to suitable functions as defined in [2]. There, a suitable function is defined as follows:

Definition 10. *A function or a function-like operation f from $\mathbb{R}$ to the ordinals is called* suitable *whenever, for all reals x and y, $x \leq_\mathrm{T} y$ implies $f(x) \leq f(y)$, and if we have, for all x, $f(x) \geq \omega + 1$. The symbol $\leq_\mathrm{T}$ here stands for ordinary Turing reducibility as defined in, e.g., [1].*

It turns out that some important functions and operations are in fact suitable:

Proposition 4. *For any ordinal $\alpha > \omega + 1$, the constant function $f_0(x) = \alpha$ is suitable. Furthermore, the functions λ^x, and ζ^x, and Σ^x are all suitable.*

Proof. For constant functions, we have $f_0(x) = f_0(y)$, and hence $f_0(x) \leq f_0(y)$ in all cases, so also in the specific case where $x \leq_\mathrm{T} y$. Hence, all constant functions are suitable.

For the function λ^x, assume that $x \leq_\mathrm{T} y$, and assume that $\alpha < \lambda^x$, or, in other words, that α is a x-writable ordinal. We can now write α from y, by first computing x from y, and then going on with the computation that writes α from x. Hence, $\alpha < \lambda^y$, and $\lambda^x \leq \lambda^y$ follows directly. The cases of the functions ζ^x and Σ^x go very similarly. ■

We have the following results about suitable functions:

Theorem 7. *For any suitable function f and any set A of natural numbers,*

(i) $A \in \mathbf{P}_f$ if and only if $A \in \mathbf{P}_{f(0)+1}$
(ii) $A \in \mathbf{PSPACE}_f$ if and only if $A \in \mathbf{PSPACE}_{f(0)+1}$

Proof. (i) was originally proven in [2, Theorem 26]; here we will provide a slightly modified version of the proof. Because for any natural number n, we have $0 =_\mathrm{T} n$, we get $f(0) = f(n)$ by the assumption of suitability; also, we have $0 =_\mathrm{T} \mathbb{N}$, so $f(0) = f(\mathbb{N})$; moreover, for any real number x, we have $0 \leq_T x$, and hence $f(0) \leq f(x)$. Now consider the constant function g such that $g(x) = f(0)$ for all x. As a direct result of the definition, we obtain $\mathbf{P}_{f(0)+1} = \mathbf{P}_g$. We also have $g(x) \leq f(x)$ for all x, so the result $\mathbf{P}_g \subseteq \mathbf{P}_f$ is immediate.

For the converse, assume that $A \in \mathbf{P}_f$, and that T is a time f machine deciding A. We now construct a time g machine T', which performs the same computation

as T, while, during the first ω steps of the computation, simultaneously checking if the input actually codes a natural number or the entire set $\mathbb{N}$. Because a natural number n, when considered as a real, is equal to the set $\{0, \ldots, n-1\}$, and any such set corresponds to a natural number n, it follows that a real x does *not* correspond to a natural number or the complete set $\mathbb{N}$ if and only if the string 01 occurs in it. We now ensure that T', during the first ω steps, searches for the string 01, and halts whenever this string is encountered, while simultaneously simulating T. After we first reach the limit state, we know that the input did not contain the string 01, and we continue the original computation of T. If no 01 is encountered, the input x must be either a natural number n, or the complete set of natural numbers $\mathbb{N}$, and it will finish within time $f(x) = f(0)$. If a 01 is encountered in the input x, it is encountered within the first ω steps, and hence $f(x) < \omega < f(0)$. So T' is a time $f(0)$-machine deciding A, and hence, $A \in \mathbf{P}_{f(0)+1}$.

(ii) can be proven similarly. It again follows directly that $\mathbf{PSPACE}_{f(0)+1} \subseteq \mathbf{PSPACE}_f$. The converse now is a bit simpler. If T is a space f machine deciding A, consider the following machine T': on input x, we first check, without making any modifications (and thus, while staying within $\mathbf{L}_0[x]$), whether x is a natural number. We output 0 if it does not, and if it does, we continue the computation of which we now know that $\mathbf{space}(x, T) < f(x) = f(0)$. Because at the start of this computation, the tape is still unchanged, and the algorithm performed after the check if x is a natural number, is identical, we also obtain $\mathbf{space}(x, T) < f(0)$. It is now clear that $\mathbf{space}(x, T') < f(0)$ for all x, so T' is a space $f(0)$-machine deciding A, so $A \in \mathbf{PSPACE}_{f(0)+1}$. ■

Now, because for any α, the set h_α always consists of only natural numbers, we can directly extend the earlier results about ordinals to results about suitable functions f:

Theorem 8. *If f is a suitable function, and $f(0)$ is either recursive, or a clockable ordinal, such that there is an ordinal β such that $f(0)$ is between $\beta + 2$ and the next admissible ordinal after β, then we have $\mathbf{P}_f \subsetneq \mathbf{PSPACE}_f$.*

Proof. On one hand, we have $h_{f(0)+1} \notin \mathbf{P}_{f(0)+2}$ from Proposition 1, which, by Theorem 7 gives us $h_{f(0)+1} \notin \mathbf{P}_{f+1}$. On the other hand, we have $h_{f(0)+1} \in \mathbf{PSPACE}_{f(0)+2}$ from either Proposition 3 or Proposition 6, which gives us $h_{f(0)+1} \in \mathbf{PSPACE}_{f+1}$. Hence we have $\mathbf{P}_f \neq \mathbf{PSPACE}_f$, and the result follows. ■

4.5 However, $\mathbf{P}_f = \mathbf{PSPACE}_f$ for 'Almost All' f

So far, we have shown that, for certain classes of 'low' functions f and ordinals α, there is a strict inclusion $\mathbf{P}_f \subset \mathbf{PSPACE}_f$. This brings us to wonder we can also find functions and ordinals where this strict inclusion does not hold, and instead we have an equality $\mathbf{P}_f = \mathbf{PSPACE}_f$.

It is easy to see that we will have this equality, at least for very high, non-countable, ordinals and functions: if we have $\alpha > \omega_1$, then we must have

$\mathbf{P}_\alpha = \mathbf{Dec}$, and because we also have $\mathbf{PSPACE}_\alpha \subseteq \mathbf{Dec}$ and $\mathbf{P}_\alpha \subseteq \mathbf{PSPACE}_\alpha$, we indeed obtain $\mathbf{P}_\alpha = \mathbf{PSPACE}_\alpha$. However, we can generalize this towards a wider range of functions:

Proposition 5. *If f satisfies $f(x) \geq \lambda^x$ for all x, then we have $\mathbf{P}_f = \mathbf{PSPACE}_f$.*

Proof. We clearly have $\mathbf{P}_f = \mathbf{Dec}$ because f is, on all x, larger than the supremum of halting times of ITTM computable functions on input x, as a result of the fact that $\gamma^x = \lambda^x$ for all x. Also, we have $\mathbf{P}_f \subseteq \mathbf{PSPACE}_f$, as well as $\mathbf{PSPACE}_f \subseteq \mathbf{Dec}$. Hence, we have $\mathbf{P}_f = \mathbf{Dec} = \mathbf{PSPACE}_f$. ■

As this class of functions f is a superset of the class of functions f—called 'almost all f' there—for which it is shown, in [4], that $\mathbf{P}_f \neq \mathbf{NP}_f$, we can, with a little wink, indeed say that we have $\mathbf{P}_f = \mathbf{PSPACE}_f$ for 'almost all' f.

References

1. Barry Cooper, S.: Computability Theory. Chapman & Hall/CRC (2004)
2. Deolalikar, V., Hamkins, J.D., Schindler, R.-D.: $\mathbf{P} \neq \mathbf{NP} \cap$ co-$\mathbf{NP}$ for infinite time Turing machines. J. Log. Comput. 15, 577–592 (2005)
3. Hamkins, J.D., Lewis, A.: Infinite time Turing machines. J. Symb. Log. 65, 567–604 (2000)
4. Hamkins, J.D., Welch, P.D.: $\mathbf{P}_f \neq \mathbf{NP}_f$ for almost all f. Math. Log. Q. 49(5), 536–540 (2003)
5. Löwe, B.: Space bounds for infinitary computation. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 319–329. Springer, Heidelberg (2006)
6. Schindler, R.: $P \neq NP$ for infinite time Turing machines. Monatsh. Math. 139, 335–340 (2003)
7. Winter, J.: Space Complexity in infinite time Turing Machines, ILLC Publications, MoL-2007-14 (2007)

Author Index

Bekker, Wilmari 2
Busi, Nadia 21

Chatterjee, Krishnendu 32

Duparc, Jacques 46

Facchini, Alessandro 46
Finkel, Alain 56

Goranko, Valentin 2

Henzinger, Thomas A. 32

Louchard, Guy 87
Lozes, Étienne 56

Mihoubi, Douadi 97
Mori, Takakazu 109

Prodinger, Helmut 87

Sangnier, Arnaud 56

Tsuiki, Hideki 109
Tsujii, Yoshiki 109

Winter, Joost 126

Yasugi, Mariko 109

Zandron, Claudio 1, 21