Games for Verification: Algorithmic Issues

Marcin Jurdziński

Ph.D. Dissertation

BRICS

BRICS Ph.D. School
Department of Computer Science
University of Aarhus
Denmark

October 2000 **Supervisor:** Mogens Nielsen

Games for Verification: Algorithmic Issues

A dissertation
presented to the Faculty of Science
of the University of Aarhus
in partial fulfilment of the requirements for the
Ph.D. degree

by
Marcin Jurdziński
9 October 2000

Abstract

This dissertation deals with a number of algorithmic problems motivated by computer aided formal verification of finite state systems. The goal of formal verification is to enhance the design and development of complex systems by providing methods and tools for specifying and verifying correctness of designs. The success of formal methods in practice depends heavily on the degree of automation of development and verification process. This motivates development of efficient algorithms for problems underlying many verification tasks.

Two paradigmatic algorithmic problems motivated by formal verification that are in the focus of this thesis are model checking and bisimilarity checking. In the thesis game theoretic formulations of the problems are used to abstract away from syntactic and semantic peculiarities of formal models and specification formalisms studied. This facilitates a detailed algorithmic analysis, leading to two novel model checking algorithms with better theoretical or practical performance, and to an undecidability result for a notion of bisimilarity.

The original technical contributions of this thesis are collected in three research articles whose revised and extended versions are included in the dissertation.

In the first two papers the computational complexity of deciding the winner in parity games is studied. The problem of solving parity games is polynomial time equivalent to the modal mu-calculus model checking. The modal mu-calculus plays a central role in the study of logics for specification and verification of programs. The model checking problem is extensively studied in literature on computer aided verification. The question whether there is a polynomial time algorithm for the modal mu-calculus model checking is one of the most challenging and fascinating open questions in the area.

In the first paper a new algorithm is developed for solving parity games, and hence for the modal mu-calculus model checking. The design and analysis of the algorithm are based on a semantic notion of a progress measure. The worst-case running time of the resulting algorithm matches the best worst-case running time bounds known so far for the problem, achieved by the algorithms due to Browne at al., and Seidl. Our algorithm has better space complexity: it works in small polynomial space; the other two algorithms have exponential worst-case space complexity.

In the second paper a novel approach to model checking is pursued, based

on a link between parity games and discounted payoff and stochastic games, established and advocated by Puri. A discrete strategy improvement algorithm is given for solving parity games, thereby proving a new procedure for the modal mu-calculus model checking. Known strategy improvement algorithms, as proposed for stochastic games by Hoffman and Karp, and for discounted payoff games and parity games by Puri, work with real numbers and require solving linear programming instances involving high precision arithmetic. The present algorithm for parity games avoids these difficulties by efficient manipulation of carefully designed discrete valuations. A fast implementation is given for a strategy improvement step. Another advantage of the approach is that it provides a better conceptual understanding of the underlying discrete structure and gives hope for easier analysis of strategy improvement algorithms for parity games. However, so far it is not known whether the algorithm works in polynomial time. The long standing problem whether parity games can be solved in polynomial time remains open.

In the study of concurrent systems it is common to model concurrency by non-determinism. There are, however, some models of computation in which concurrency is represented explicitly; elementary net systems and asynchronous transition systems are well-known examples. History preserving and hereditary history preserving bisimilarities are behavioural equivalence notions taking into account causal relationships between events of concurrent systems. Checking history preserving bisimilarity is known to be decidable for finite labelled elementary nets systems and asynchronous transition systems. Its hereditary version appears to be only a slight strengthening and it was conjectured to be decidable too. In the third paper it is proved that checking hereditary history preserving bisimilarity is undecidable for finite labelled asynchronous transition systems and elementary net systems. This solves a problem open for several years. The proof is done in two steps. First an intermediate problem of deciding the winner in domino bisimulation games for origin constrained tiling systems is introduced and its undecidability is shown by a reduction from the halting problem for 2-counter machines. Then undecidability of hereditary history preserving bisimilarity is shown by a reduction from the problem of domino bisimulation games.

Acknowledgements

I am grateful to my Ph.D. supervisor Mogens Nielsen for his guidance and support during my studies in Aarhus. His contagious enthusiasm was an invaluable source of encouragement, stimulation, and energy. Research collaboration with Mogens was an exciting and rewarding experience for me. At the same time I enjoyed a lot of freedom in the choice of other research topics to pursue. Mogens' cheerful encouragement and prudent advice helped me keep on a steady course throughout my graduate studies.

Special thanks are due to my advisors and colleagues Damian Niwiński and Igor Walukiewicz. They have guided my first steps in research and have given valuable advice and feedback all along the way. Damian attracted me to work on modal μ-calculus and parity games and my first research paper on games was co-authored by Igor. Their influence on my research and on this thesis is hard to overestimate.

I thank Wolfgang Thomas for letting me spend four months in his group at RWTH Aachen as a research student. I am indebted to him for his hospitality and all the time, effort, and resources he spent to make my stay possible and pleasant. I have enjoyed and benefited a lot, both scientifically and personally, from his open-mindedness, wise advice, and discreet guidance.

My collaboration with Jens Vöge was one of the most exciting research projects I have participated in. Long brain-storming sessions with Jens, his insightful ideas, immediate feedback and constructive criticism, sudden leaps of progress in our understanding made our joint work an exhilarating experience.

I wish to thank my dear friends Stefan and Paola for their sense of humour, vitality, and generosity. Without their constant support, entertainment and friendliness I would probably burn out long before finishing this thesis.

BRICS and DAIMI are an exceptionally inspiring and resourceful working environment for graduate studies and research in theoretical computer science. I feel privileged to have had the opportunity to study and work here. Special thanks go to BRICS/DAIMI secretaries, Janne Christiansen and Karen Møller, for all the help they provide to foreign students.

Last but not least I thank my brother Tomek for attracting me to computer science and my parents for steady support and encouragement.

Contents

Chapter 1

Introduction

This dissertation illustrates how game theoretic arguments help to study some fundamental algorithmic problems in the area of computer aided verification of finite state non-terminating systems, such as the modal μ-calculus model checking and bisimilarity checking. Adopting game theoretic formulations helps to abstract away from syntactic and semantic peculiarities of modelling and specification formalisms and makes the computational problems in question more easily amendable to algorithmic analysis.

1.1 Formal methods

Non-terminating computing systems involving multiple, distributed, and interacting agents abound today and can be found in environments as varied as household appliances, medical equipment, industrial control systems, flight control systems in airplanes, etc. Failures caused by design faults may be very costly and they should be avoided as much as possible. Behaviour of such systems is typically very complex which makes their design and validation a challenge. Formal methods try to address this challenge by developing formal models of such systems, and methods to specify and reason about their properties. A formal method is of particular interest if it offers not only a rigorous and unambiguous way to describe systems and their intended behaviour, but also provides efficient algorithms allowing to automate (parts of) design and validation tasks. We believe that software tools based on such algorithms are one of the keys for the industrial success of formal methods. Two important trends in formal methods are verification and synthesis, both aiming to design efficient algorithmic techniques supporting development and analysis of complex systems are verification and synthesis.

In verification, we are typically given a formal model of a system, and a formal specification of its intended behaviour, and we are asked to check whether the system implements the specification. The two paradigmatic problems studied in the verification community are model checking and equivalence checking. In the model checking problems we are typically given a transition system, and a temporal logic formula, and we are to decide whether the behaviours of the system satisfy the formula. The input to an equivalence checking procedure are usually two transition systems, one can be often thought as a specification and the other as an implementation; the task is to decide if they are equivalent with respect to some notion of behavioural equivalence. Synthesis achieves a more ambitious goal. Instead of building the system from scratch by hand, and then trying to validate it, an automatic procedure is given to synthesize the system which is guaranteed to satisfy the specification of its intended behaviour.

1.2 Model checking

Model checking is one of the most successful automatic methods for formal verification of hardware and software systems [CGP99]. The approach is to model systems as labelled transition systems or Kripke structures and to specify properties of their behaviours with formulas of some logic of programs. The verification task then boils down to model checking, i.e., deciding for a given labelled transition system and a formula of the logic, whether the transition system satisfies the formula. One of the reasons for the success of the model checking approach to verification stems from establishing a proper compromise between expressibility of the specification formalisms (propositional temporal logics), and efficiency of their model checkers (low polynomial time complexity). Another attractive feature of model checking is a high degree of automation of the verification process. It is a common observation that model checking often requires less skill and effort to apply than verification methods based on (computer aided) theorem proving. Last but not least, model checkers often provide useful diagnostic information in the case when a system under consideration does not satisfy the intended property. Such information is of great help for the designer in identifying and correcting errors.

The modal μ-calculus introduced by Kozen [Koz83] is a very expressive logic capable of specifying a variety of correctness properties of non-terminating concurrent systems, e.g., safety, liveness, fairness, etc. Moreover, most of the studied temporal logics of programs can be succinctly encoded in a modest fragment of the modal μ-calculus [EL86, EJS93]. Importantly, efficient model checking algorithms are known for the relevant fragments of the logic [EL86, AC88, CS91, EJS93]. Even though formulas of the modal μ-calculus are notoriously hard to understand for humans, the logic has been widely accepted as the common low-level language into which other specification formalisms are automatically translated in order to perform model checking. This approach is compatible with symbolic model checking [BCM$^+$92, CGL94] which has been devised to alleviate the so-called state explosion problem in model checking.

An intriguing question is that of the computational complexity of the modal μ-calculus model checking. The problem is known to be in NP $\cap$ co-NP [EJS93], so it is considered unlikely to be NP-complete. A considerable research effort has been carried out to devise a polynomial time algorithm for the problem (see for example [EL86, EJS93, BCJ$^+$97, Sei96, Jur00, VJ00b] and references there), but no such algorithm has been found so far. From the remarks above about the model checking approach to verification it is clear that establishing the exact complexity of the model checking for the whole modal μ-calculus is not of big practical importance, but it is still a fascinating theoretical challenge.

There is a very close connection between the modal μ-calculus and other temporal and modal logics of programs with automata on infinite words and trees. The essence of the so-called automata theoretic method is, for every formula of a logic, to associate a finite automaton recognizing exactly those infinite structures in which the formula holds. Then the algorithmic verification problems for the logic, such as satisfiability and model checking can be rephrased as the problem of checking non-emptiness of automata. The advantage of this approach is the separation of the logical and combinatorial aspects. The translation from logic to automata often allows to abstract away from syntactic and semantic peculiarities of the logic. The automata theoretic problem of checking non-emptiness is more easily amendable to algorithmic analysis and it can be studied by people who are not experts in formal methods.

The central algorithmic problem of checking non-emptiness of automata on infinite trees can be rephrased as the problem of deciding the winner in infinite duration path-forming games played by two players on a finite graph. The winner in these games is determined using the infinitary acceptance condition of the automaton. In this thesis we study the computational complexity of the modal μ-calculus model checking using this game theoretic formulation of the problem. This facilitates a transfer of new algorithmic techniques for the model checking problem from literature on other games: mean payoff, discounted payoff [ZP96, Pur95], and stochastic games [HK66, Con93].

1.3 Bisimilarity checking

Bisimilarity [Par81, Mil80] is a behavioural equivalence notion that has attracted most attention in concurrency theory. It is often used to provide semantics for numerous models of concurrent systems: two systems are considered to have the same meaning if they are bisimilar. The prominent role that bisimilarity plays is owing a lot to the pleasant properties it enjoys.

Two systems are bisimilar if and only if there is a witness, called a bisimulation, relating them. There may be many bisimulations relating two systems and different insights can guide a choice between witnesses. Proving bisimilarity of systems by exhibiting a bisimulation relating them is considered as a convenient proof rule for showing equivalence of programs.

A process of checking whether two systems are bisimilar has a simple yet elegant interpretation as a two player game which can be seen as an Ehrenfeucht-Fraïssé game for modal logic [Tho93, Sti97]. One of the players tries to prove the system not to be bisimilar, while the other player defends the hypothesis that the systems are bisimilar. In other words, two systems are bisimilar if and only if they are indistinguishable by formulas of modal logic [HM85].

A notable property of bisimilarity is its computational feasibility for a wide range of finite and infinite state models of computation. For example, for finite state systems there are algorithms for checking bisimilarity running in low polynomial time, while the classical language equivalence is PSPACE-complete, i.e., intractable. For a class of infinite state systems generated by context free grammars bisimilarity is decidable, while language equivalence is not. Algorithmic tractability makes bisimilarity attractive for automatic verification of concurrent systems.

The essence of bisimilarity "is that the behaviour of a program is determined by how it communicates with an observer" [HM85]. Therefore, a notion of bisimilarity crucially depends on what is observable of a behaviour of a system. Joyal et al. [JNW96] formalize this idea by giving an abstract definition of bisimilarity parameterized by a notion of observable behaviours. For a category of models where objects are behaviours and morphisms capture a notion of simulation, and given a subcategory of observable behaviours, the abstract definition yields a notion of bisimilarity for all behaviours with respect to observable ones.

It is common to model concurrency by non-determinism: if events can occur independently of each other in parallel then it is reflected in the model of the system by allowing all linearizations, also called interleavings, of occurrences of concurrent events. In order to model concurrency more faithfully several models have been studied that make explicit the distinction between events that can occur concurrently and those that are causally related. A selection of fundamental models including trace languages, Petri nets, asynchronous transition systems, event structures, and their mutual relationships are surveyed in the chapter by Winskel and Nielsen [WN95]. In the models with an explicit representation of concurrency it is quite natural to extend the notion of observable behaviour from sequences, i.e., linear orders of occurrences of events to partial orders describing their causal relationships. If we take labelled partial orders as observable behaviours then the above abstract definition of bisimilarity yields hereditary history preserving bisimilarity [Bed91, JNW96] which is a back-and-forth version of history preserving bisimilarity [RT88, GG89]. In this thesis we show that hereditary history preserving bisimilarity is undecidable for finite state elementary net systems [JN00].

1.4 Outline of dissertation

This dissertation consists of two parts. Part II contains our original technical contributions to the area of formal verification of non-terminating computational systems. In Part I we provide the context for the results of Part II with a high-level overview of the relevant theory: we define the basic notions and survey related work.

Part I consists of two chapters. Chapter 2 contains an overview of the modal μ-calculus and the role it plays in the theory of computer aided formal verification of non-terminating finite state systems. In particular we focus on the connection between logic and automata. We survey the application of the so-called automata theoretic method for solving satisfiability and model checking problems for the modal μ-calculus. We also mention the relation of the modal μ-calculus model checking problem to solving infinite parity games. Finally we survey model checking algorithms and we compare them with algorithms for model checking derived from algorithms for solving parity games. Chapter 3 is concerned with models with explicit representation of concurrency and with a number of behavioural equivalences for these models. We recall two classical models: elementary net systems and asynchronous transition systems. Then we mention a few bisimilarity notions, recall their logical and game theoretic characterizations, and discuss decidability of checking bisimilarity for finite state asynchronous transition systems and elementary net systems.

Part II contains revised and extended versions of the following three articles.

- Chapter 4: Marcin Jurdziński. Small Progress Measures for Solving Parity Games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, Lille, France, February 2000. Springer-Verlag [Jur00].

- Chapter 5: Jens Vöge and Marcin Jurdziński. A Discrete Strategy Improvement Algorithm for Solving Parity Games (Extended Abstract). In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215, Chicago, IL, USA, July 2000. Springer-Verlag [VJ00b].

- Chapter 6: Marcin Jurdziński and Mogens Nielsen. Hereditary History Preserving Bisimilarity Is Undecidable. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 358–369, Lille, France, February 2000. Springer-Verlag [JN00].

For completeness we mention other published contributions we have made to the theory of formal verification and synthesis of concurrent programs. The results contained in the two articles listed below do not form a part of this dissertation; they are included in the author's M.Sc. thesis [Jur97].

- Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How Much Memory Is Needed to Win Infinite Games? In *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 99–110, Warsaw, Poland, July 1997. IEEE Computer Society Press [DJW97].

- Marcin Jurdziński. Deciding the winner in parity games is in UP $\cap$ co-UP. *Information Processing Letters*, 68(3):119–124, November 1998 [Jur98].

Part I

Context

Chapter 2

Modal μ-calculus, model checking, games

This chapter contains an overview of the modal μ-calculus and the role it plays in the theory of computer aided formal verification of non-terminating finite state systems. In particular we focus on the connection between logic and automata. We survey the application of the so-called automata theoretic method for solving satisfiability and model checking problems for the modal μ-calculus. We also discuss the relation of the modal μ-calculus model checking problem to solving infinite parity games. Finally we survey model checking algorithms and we compare them with algorithms for model checking derived from algorithms for solving parity games.

2.1 Modal μ-calculus

Modal μ-calculus is a powerful logic used for specification and verification of computer systems. Consult papers by Emerson [Eme96], Niwiński [Niw97], and Bradfield and Stirling [BS00], for recent comprehensive surveys on μ-calculi and related topics.

Syntax

Let `Prop` be a set of atomic propositions and let `Var` be a set of variables. The set of formulas of modal μ-calculus [Koz83] is defined by the following grammar:

$$X \mid p \mid \neg p \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid \langle \cdot \rangle \alpha \mid [\cdot]\alpha \mid \mu X.\alpha \mid \nu X.\alpha$$

where X ranges the set `Var` of variables, p ranges the set `Prop` of propositions, and α, β range over formulas. Formulas of the form $\mu X.\alpha$ and $\nu X.\alpha$ are called fixpoint formulas; symbols μ and ν are called least and greatest fixpoint operators, respectively.

Semantics

A Kripke structure $K = (S, R, L)$ consists of a set of states S, transition relation $R \subseteq S \times S$, and a labelling function $L : S \to \wp(\texttt{Prop})$ that maps to every state the set of propositions true in it. A rooted Kripke structure (K, r) consists of a Kripke structure and a state $r \in S$ called the root.

The meaning of a formula α in a Kripke structure K with respect to a valuation $V : \texttt{Var} \to \wp(S)$ is the set of states $[\![\alpha]\!](K, V)$, defined in the following way:

$$
\begin{aligned}
[\![X]\!](K,V) &= V(X) \\
[\![p]\!](K,V) &= \{\, s \in S \,:\, p \in L(s) \,\} \\
[\![\neg p]\!](K,V) &= \{\, s \in S \,:\, p \notin L(s) \,\} \\
[\![\alpha \vee \beta]\!](K,V) &= [\![\alpha]\!](K,V) \,\cup\, [\![\beta]\!](K,V) \\
[\![\alpha \wedge \beta]\!](K,V) &= [\![\alpha]\!](K,V) \,\cap\, [\![\beta]\!](K,V) \\
[\![\langle\cdot\rangle\alpha]\!](K,V) &= \big\{\, s \in S \,:\, \text{there is } t \in S \text{, s.t. } (s,t) \in R \text{, and } t \in [\![\alpha]\!](K,V) \,\big\} \\
[\![[\cdot]\alpha]\!](K,V) &= \big\{\, s \in S \,:\, \text{for all } t \in S \text{, s.t. } (s,t) \in R \text{, we have } t \in [\![\alpha]\!](K,V) \,\big\} \\
[\![\mu X.\alpha]\!](K,V) &= \texttt{LFP}\big(S' \mapsto [\![\varphi]\!](K, V[S'/X])\big) \\
[\![\nu X.\alpha]\!](K,V) &= \texttt{GFP}\big(S' \mapsto [\![\varphi]\!](K, V[S'/X])\big)
\end{aligned}
$$

where $\texttt{LFP}(f)$ and $\texttt{GFP}(f)$ are the least and the greatest, respectively, fixed points of a function $f : \wp(S) \to \wp(S)$. Note that we allow negation only in front of atomic propositions. By Knaster-Tarski theorem [Tar55], the meaning is well defined for all formulas of the modal μ-calculus.

Let α be a modal μ-calculus sentence, i.e., a formula without free variables. We say that α holds in a state s of a Kripke structure K if $s \in [\![\alpha]\!](K, \emptyset)$; we write $K, s \models \alpha$ to denote this. A sentence α holds in a rooted Kripke structure (K, r) if $K, r \models \alpha$.

Alternation depth

Niwiński [Niw86] introduced a hierarchy of fixpoint terms based on the number of alternations between least and greatest fixpoint operators. Emerson and Lei [EL86] defined a similar notion of alternation depth of a formula; consult Niwiński's paper [Niw97] for a comparison of the two notions. We follow Niwiński's definition here.

We introduce classes Σ_i^μ and Π_i^μ of modal μ-calculus formulas, for all $i \in \mathbb{N}$, in the following way. We define $\Sigma_0^\mu = \Pi_0^\mu$ to be the set of modal μ-calculus formulas without an occurrence of a fixed point operator. We define the class Σ_{i+1}^μ (Π_{i+1}^μ) to be the smallest set containing Π_i^μ (Σ_i^μ) and closed on the following rules:

1. if $\alpha, \beta \in \Sigma_{i+1}^\mu$ (Π_{i+1}^μ) then $\alpha \vee \beta, \alpha \wedge \beta, \langle\cdot\rangle\alpha, [\cdot]\alpha \in \Sigma_{i+1}^\mu$ (Π_{i+1}^μ),
2. if $\alpha(X), \beta \in \Sigma_{i+1}^\mu$ (Π_{i+1}^μ) then $\alpha(\beta) \in \Sigma_{i+1}^\mu$ (Π_{i+1}^μ), provided that no free variable in β is bound by a fixed point operator in α,

3. if $\alpha \in \Sigma^{\mu}_{i+1}$ (Π^{μ}_{i+1}) then $\mu X.\alpha \in \Sigma^{\mu}_{i+1}$ ($\nu X.\alpha \in \Pi^{\mu}_{i+1}$).

We define the alternation depth of a modal μ-calculus formula α to be the smallest number d, such that α can be obtained from formulas in $\Sigma^{\mu}_{d} \cup \Pi^{\mu}_{d}$ by application of rules 1. and 2. above.

Expressiveness

The fixpoint operators of the modal μ-calculus give it an immense expressive power. It subsumes most studied logics of programs, for example PDL [FL79], LTL [Pnu77], CTL [CE81], and CTL* [EH86]. Translations of these logics into the modal μ-calculus [EL86, Dam94] are formulas of alternation depth at most 2.

On the infinite k-ary trees the modal μ-calculus is equal in expressive power to Rabin tree automata [Niw88, EJ91, Niw97]. Therefore, by the results of Rabin [Rab69], modal μ-calculus is equivalent to monadic second order logic on the infinite k-ary tree, which is known to be one of the most expressive decidable mathematical theories.

The question whether the hierarchy of properties based on alternation depth of the formulas is strict and infinite has been positively resolved by Bradfield [Bra98] and Lenzi [Len96]; Arnold [Arn99] gave an elegant automata-theoretic proof by diagonalization.

Theorem 2.1 ([Bra98, Len96])
For all $i \in \mathbb{N}$, there is a formula in Σ^{μ}_{i} which is not equivalent to a formula in Π^{μ}_{i}. Therefore, the modal μ-calculus alternation depth hierarchy is infinite and strict.

Decidability and complexity

The two key algorithmic problems for the automatic verication of specifications written in the modal μ-calculus are satisfiability and model checking. Satisfiability problem is, given a modal μ-calculus sentence α, to decide whether there is a rooted Kripke structure (K, r), such that α holds in (K, r). Model checking problem is, given a modal μ-calculus sentence α and a rooted Kripke structure (K, r), to decide whether α holds in (K, r). Both problems are decidable and of manageable computational complexity.

Theorem 2.2 ([SE89, EJ99])
Modal μ-calculus satisfiability problem is DEXPTIME-complete.

Theorem 2.3 ([EJS93, EL86])
Modal μ-calculus model checking problem is in NP $\cap$ co-NP.

A substantial part of this thesis is dedicated to a more detailed study of the computational complexity of the model checking problem for the modal μ-calculus.

Problem 2.4 ([EJS93]) Is there a polynomial time algorithm for the modal μ-calculus model checking?

In Chapters 4 and 5 of this thesis we propose two new algorithms for the polynomial-time equivalent problem of solving parity games but the above challenging question remains open.

We mention another challenging open problem in this area.

Problem 2.5 Is the modal μ-calculus alternation hierarchy decidable? More precisely, is there an algorithm for the following problem: given a number $i \in \mathbb{N}$, and a modal μ-calculus formula α, decide whether there is a formula in Σ_i^μ equivalent to α.

The problem is known to be decidable in the very special cases: for $i = 0$ [Ott99], and for $i = 1$ [Wil99a, Wal99].

2.2 Automata on infinite trees

Automata on infinite trees have been introduced by Rabin [Rab69]; for comprehensive and accessible surveys of the subject see the papers by Thomas [Tho90, Tho96]. In the context of the modal μ-calculus automata are meant to run on variable arity labelled trees obtained by unfolding of rooted Kripke structures. For simplicity, following the tradition [Tho90], we consider automata on infinite binary trees here. There are several proposals for more adequate automata running on variable arity trees [JW95, Wil99b, KVW00]; the differences between them are not very important for our considerations.

Non-deterministic automata

A Σ-labelled infinite binary tree $T = (\{1,2\}^*, L)$ consists of the set of nodes $\{1,2\}^*$ and the labelling function $L : \{1,2\}^* \to \Sigma$. A non-deterministic tree automaton $A = (Q, \Sigma, q_0, \delta, \mathcal{W})$ consists of:

- a finite set of states Q,
- a finite alphabet Σ,
- an initial state $q_0 \in Q$,
- a transition function $\delta : Q \times \Sigma \to \wp(Q \times Q)$,
- an infinitary acceptance condition $\mathcal{W}$ that specifies a subset of Q^ω; we describe several types of acceptance conditions below.

The behaviour of an automaton on a Σ-labelled tree $T = (\{1,2\}^*, L)$ is captured by the notion of a run. A run of an automaton A on T is a Q-labelled tree $(\{1,2\}^*, R)$, such that:

- $R(\varepsilon) = q_0$, and
- $\big(R(x \cdot 1), R(x \cdot 2)\big) \in \delta\big(R(x), L(x)\big)$, for all $x \in \{1,2\}^*$.

A run is accepting if all its infinite paths satisfy the acceptance condition $\mathcal{W}$. In order to define it more precisely we need the following notation. Let $(\{1,2\}^*, R)$ be a run; if $P = \langle x_1, x_2, \ldots \rangle$ is an infinite path in a binary tree then we write $\mathtt{Inf}(P)$ for the set of states which appear infinitely often on path P in the run, i.e., the set of states occurring infinitely often in $\langle R(x_1), R(x_2), \ldots \rangle$. We consider the following acceptance conditions.

- Muller conditions. A path P satisfies a Muller condition $\mathcal{F} \subseteq \wp(Q)$ if and only if $\mathtt{Inf}(P) \in \mathcal{F}$.
- Rabin conditions. A path P satisfies a Rabin condition
$$\mathcal{R} = \big\{(G_1, R_1), (G_2, R_2), \ldots, (G_m, R_m)\big\},$$
where for $i \in \{1, 2, \ldots, m\}$, we have $G_i \subseteq Q$ and $R_i \subseteq Q$, if and only if there is $i \in \{1, 2, \ldots, m\}$, such that $\mathtt{Inf}(P) \cap G_i \neq \emptyset$ and $\mathtt{Inf}(P) \cap R_i = \emptyset$.
- Streett conditions. A path P satisfies a Street condition
$$\mathcal{S} = \big\{(G_1, R_1), (G_2, R_2), \ldots, (G_m, R_m)\big\},$$
where for $i \in \{1, 2, \ldots, m\}$, we have $G_i \subseteq Q$ and $R_i \subseteq Q$, if and only if for all $i \in \{1, 2, \ldots, m\}$, we have that $\mathtt{Inf}(P) \cap G_i \neq \emptyset$ implies $\mathtt{Inf}(P) \cap R_i \neq \emptyset$.
- Parity conditions. A path P satisfies a parity condition $p : Q \to \{0, 1, \ldots, k\}$ if and only if $\max \big\{ \, p(q) : q \in \mathtt{Inf}(P) \, \big\}$ is even.

We say that a Σ-labelled tree T is accepted by an automaton A if there exists an accepting run of A on T. We write $\mathcal{L}(A)$ for the set of Σ-labelled trees accepted by A.

We say that a parity automaton $A = (Q, \Sigma, q_0, \delta, p)$ has index (l, k) [NW98] if the parity condition p is a function $Q \to \{l, l+1, \ldots, k\}$. What matters is the size of the range of p and whether the smallest number is even or odd, and hence it suffices to consider indices of the form $(0, k)$ and $(1, k)$, for $k \in \mathbb{N}$.

Complexity of non-emptiness problem

The key algorithmic problem for automata on infinite trees in the context of applications to the modal μ-calculus and other program logics is the non-emptiness problem. Non-emptiness problem is, given an automaton A, to decide whether $\mathcal{L}(A) \neq \emptyset$.

Theorem 2.6
Non-emptiness problem for non-deterministic automata on trees is:

- *NP-complete for Rabin and co-NP-complete for Streett automata [EJ99],*

- *in NP $\cap$ co-NP for parity automata [EJS93].*

There is an algorithm for checking non-emptiness of Rabin/Streett automata running in time $O\big((n \cdot m)^{3m}\big)$, where n is the number of state of the automaton and m is the number of Rabin/Streett pairs in the acceptance condition [EJ99, PR89a].

2.3 Automata theoretic method

An intimate relationship between logic and automata has been discovered by Büchi [Büc60] and Elgot [Elg61]. Büchi [Büc62] and Rabin [Rab69] have successfully applied the "automata theoretic method" in order to prove decidability of satisfiability for the monadic second order logic on the infinite words and trees, respectively. They did it by developing the theory of automata on infinite words and trees, respectively, establishing effective translations of monadic second order formulas into automata on words and trees, thus reducing the satisfiability problem for the logic to the non-emptiness problem for automata. See the chapter by Thomas [Tho96] for a recent comprehensive survey on the connection between logic and automata.

The automata theoretic method turned out to be fruitful in the later studies on program logics carried out in the '80s and '90s, often providing optimal decision procedures for both satisfiability and model checking problems. It is particularly valued for providing a clean separation of logical and algorithmic aspects of program logics. Automata are seen as useful normal forms of logical formulas, abstracting away from syntactical and semantical peculiarities of program logics. We mention selected results concerning the application of the automata theoretic method to the modal μ-calculus.

Theorem 2.7 ([SE89])
A modal μ-calculus sentence α is satisfiable if and only if it holds in some n-ary tree, where $n = |\alpha|$. There is an exponential time algorithm which given a modal μ-calculus sentence α, constructs a non-deterministic Rabin tree automaton N_α, such that $\mathcal{L}(N_\alpha)$ is the set of n-ary trees in which α holds. The automaton N_α is of size exponential in n, and has only $O(n)$ pairs in the Rabin acceptance condition.

Note that combining Theorems 2.6 and 2.7 we get an automata theoretic procedure for checking satisfiability of modal μ-calculus of optimal complexity; see Theorem 2.2.

The automata theoretic method has been also successfully applied to the model checking problem for the modal μ-calculus.

Theorem 2.8 ([EJS93])
The modal μ-calculus model checking problem is polynomial time equivalent to non-emptiness for non-deterministic parity tree automata.

An alternative approach to get automata theoretic procedures for satisfiability and model checking problems for the modal μ-calculus and other branching-time logics was taken by Kupferman et al. [KVW00]. They provide a linear-time translation of a modal μ-calculus sentence α into an equivalent alternating parity tree automaton A_α, following the ideas of Niwiński [Niw88, Niw97], and Emerson and Jutla [EJ91]. Then they use the same automaton for satisfiability and model checking in the following way. In order to check satisfiability it suffices to test for non-emptiness of the automaton A_α. The non-emptiness problem for alternating parity tree automata is in DEXPTIME. In order to do model checking on a rooted Kripke structure (K, r) Kupferman et al. build a product automaton $A_\alpha \times A_{(K,r)}$, where $A_{(K,r)}$ is a trivial automaton generating the unfolding of the Kripke structure K from the root r. The product automaton is a 1-letter alternating parity automaton on infinite words of size $|K| \cdot \alpha$. Kupferman et al. show that the emptiness problem for such automata is linear-time equivalent to the emptiness problem of non-deterministic parity tree automata.

2.4 Infinite games and modal μ-calculus

In the study of automata on infinite trees and the modal μ-calculus a technically useful metaphor is that of playing infinite duration two-player games. Primary examples are simplified proofs of Rabin's complementation lemma [Rab69] for automata on infinite trees due to Gurevich and Harrington [GH82], and Emerson and Jutla [EJ91]. See the chapter by Thomas [Tho96] for a streamlined exposition of a proof of Rabin's complementation lemma based on memoryless determinacy of parity games.

In the context of the modal μ-calculus both satisfiability [NW96b] and model checking [Sti95] problems can be interpreted as determining whether a player has a winning strategy in an infinite duration game. This point of view has the following advantage. If a formula is satisfiable then a winning strategy in a game defined by Niwiński and Walukiewicz [NW96b] gives a model in which the formula holds. Similarly, a winning strategy in a game defined by Stirling [Sti95] can be seen as a proof that a sentence holds in a given rooted Kripke structure. Interestingly, if a formula is not satisfiable or it does not hold, then by determinacy of games the other player has a winning strategy and this strategy can serve as a witness that a formula is not satisfiable or it does not hold, respectively.

In this thesis we adopt this game theoretic metaphor for the study of the computational complexity of the modal μ-calculus model checking problem.

Infinite games on graphs

We consider infinite duration games played by two players (player 0 and player 1) on finite directed graphs. A game $G = \big(V, E, (M_0, M_1), \mathcal{W}\big)$ consists of a finite directed graph (V, E), a partition $M_0 \cup M_1 = V$ of the set of vertices V, and an infinitary winning condition $\mathcal{W}$ that specifies a subset of V^ω. We consider the same ways for specifying a winning condition as for infinitary acceptance conditions in automata: Muller, Rabin, Street, and parity. We say that a parity game has index (ℓ, k) if its parity condition p is a function $V \to \{\ell, \ell+1, \dots, k\}$.

A position in a game is a finite path in the game graph. We write $\texttt{Pos}(G)$ for the set of positions in game G. The initial position of a play starting from a vertex $v_0 \in V$ is the path $\langle v_0 \rangle$ consisting only of vertex v_0. Let $\langle v_0, v_1, \dots, v_i \rangle$ be the current position of a play. If $v_i \in M_0$ then it is the turn of player 0 to make a move, otherwise player 1 moves. A player makes a move by choosing a successor v_{i+1} of v_i and the new position is $\langle v_0, v_1, \dots, v_{i+1} \rangle$. A play is either an infinite path $\pi = \langle v_0, v_1, \dots \rangle$ in the game graph, or a finite path $\pi = \langle v_0, v_1, \dots, v_k \rangle$, such that there is no edge going out of v_k in the game graph, i.e., $(v_k, v) \notin E$, for all $v \in V$.

A finite play $\pi = \langle v_0, v_1, \dots, v_k \rangle$ is winning for player 0 if $v_k \in M_1$; otherwise play π is winning for player 1. In other words, a finite play π is lost by the player who is stuck in position π. An infinite play π is winning for player 0 if π is in the winning condition specified by $\mathcal{W}$. Otherwise an infinite play π is winning for player 1.

A strategy for player 0 is a function $\zeta : \texttt{Pos}(G) \to V$, which given the current position of a play specifies which move player 0 should make in this position. A play $\pi = \langle v_0, v_1, \dots \rangle$ is consistent with a strategy ζ if $\zeta\big(\langle v_0, v_1, \dots, v_i \rangle\big) = v_{i+1}$, for all $v_i \in M_0$. A strategy for player 0 is a winning strategy from a vertex v if every play staring from v and consistent with the strategy is winning for player 0. The winning set of player 0 is the set of vertices from which there is a winning strategy for player 0. Strategies, winning strategies, and the winning set are defined similarly for player 1. A strategy ζ is memoryless if its value depends only on the last vertex in a position of a game, i.e., if $\zeta\big(\langle v_0, v_1, \dots, v_i \rangle\big)$ is uniquely determined by v_i for every position $\langle v_0, v_1, \dots, v_i \rangle$.

Theorem 2.9 (Memoryless determinacy [EJ91, Mos91])
Parity games are determined, i.e., from every vertex in a parity game one of the players has a winning strategy. Moreover, both players have memoryless winning strategies from their winning sets.

In Section 2.8 we discuss some results on bounded memory determinacy for games with arbitrary Muller conditions [GH82, McN93, Zie98, DJW97].

Deciding the winner in infinite games

The problem of deciding the winner is, given a game G and an initial vertex v_0, to determine whether player 0 has a winning strategy in game G from vertex v_0. The following fact is easy to prove.

Proposition 2.10 Non-emptiness problem for non-deterministic parity (Muller, Rabin, Streett) automata is linear-time equivalent to the problem of deciding the winner in parity (Muller, Rabin, Streett) games.

For the automata-to-games translation make player 0 choose letters labelling nodes of a tree and transitions of the automaton, and make player 1 choose directions in the tree in the game of constructing a tree and an accepting run of the automaton on this tree. In this construction player 0 constructs a tree and a run, and player 1 checks that the run is accepting. By Proposition 2.10 and Theorem 2.8 we get the following.

Corollary 2.11 ([Sti95, EJS93]) The modal μ-calculus model checking problem is polynomial time equivalent to deciding the winner in parity games.

Model checking games

In Chapters 4 and 5 of this thesis we develop two new algorithms for deciding the winner in parity games and thus for the modal μ-calculus model checking. In order to facilitate the analysis of performance of these model checking algorithms and to be able to compare them with other algorithms we state the details of the reduction to parity games.

Theorem 2.12 ([EJS93, Sti95])
There is an algorithm that given a rooted Kripke structure (K, r) with the set of states S, and a modal μ-calculus sentence α of alternation depth d, constructs a parity game $\mathcal{G}(K, r, \alpha)$ of index $(0, d)$, with $s \cdot f$ vertices and $k \cdot f$ edges, where $s = |S|$, $k = |K|$, and $f = |\alpha|$, and such that α holds in (K, r) if and only if player 0 has a winning strategy from the initial vertex in $\mathcal{G}(K, r, \alpha)$. The running time of the algorithm is $O\big(k \cdot f\big)$.

2.5 Model checking algorithms

Modal μ-calculus model checking was first studied by Emerson and Lei [EL86]. They have observed how monotonicity of operations defined by modal μ-calculus formulas can be used to speed up the successive approximation of fixpoint expressions without alternation of least and greatest fixpoint operators.

Theorem 2.13 ([EL86])
There is an algorithm for the modal μ-calculus model checking with running time $O\big((k \cdot f)^{d+1}\big)$, where k is the combined size of the Kripke structure, f is the size of the modal μ-calculus sentence, and d is its alternation depth.

Arnold and Crubille [AC88], and Cleaveland and Steffen [CS91] gave linear-time algorithms for model checking alternation-free formulas, i.e., formulas of alternation depth 1. Andersen [And94] and Cleaveland et al. [CKS92] developed algorithms with slightly better complexity than Emerson and Lei's algorithm; their running times are $O(s \cdot k^{d-1} \cdot f^d)$, and $O\big(k \cdot f \cdot (s \cdot f/d)^{d-1}\big)$, respectively, where s is the number of states of the Kripke structure.

A substantial improvement in the complexity has been obtained by Browne et al. [BCJ+97], by a more refined monotonicity arguments than those of Emerson and Lei, obtaining running time that is roughly the square root of the running time of the previously known algorithms.

Theorem 2.14 ([BCJ+97])
There is an algorithm for the modal μ-calculus model checking with running time $O\big(k \cdot (s \cdot f)^{d/2}\big)$. In worst case the space requirements of the algorithm are as big as the running time estimates.

Seidl [Sei96] gave a simplified algorithm achieving the running time bounds of the algorithm of Browne et al. In fact, the analysis that Seidl does for his elegant algorithm gives slightly better running time $O\big(k \cdot (s \cdot f/d)^{d/2}\big)$ and also exponential space.

2.6 Complexity of solving parity games

McNaughton [McN93] proposed an elegant recursive algorithm for solving games with Muller winning conditions; he also adapted it to the special case of parity games. The following is not hard to prove; we omit a proof since it would require describing the algorithm in detail which is out of the scope of this thesis.

Theorem 2.15
Zielonka's variant [Zie98] of McNaughton's algorithm for parity games has running time $O\big(m \cdot \big(\frac{n+d}{d}\big)^d\big)$, where n is the number of vertices and m is the number of edges in the game graph, and $(0, d)$ is the index of the parity game.

Note that combining it with Theorem 2.12 we get a model checking algorithm with running time roughly $O\big(k \cdot f \cdot \big(s \cdot f/d\big)^d\big)$, which is comparable with the algorithm of Cleaveland et al. [CKS92].

In Chapter 4 of this thesis we give a new algorithm for deciding the winner in parity games with running time $O\big(m \cdot \big(\frac{n+d}{d}\big)^{d/2}\big)$ and using only $O(d \cdot n)$ space. Therefore we get a modal μ-calculus model checking algorithm matching the running time bound of Browne et al. and Seidl's algorithms, and using only small polynomial space instead of exponential space.

Theorem 2.16 ([Jur00])
There is an algorithm for the modal μ-calculus model checking with running time $O\big(k \cdot (s \cdot f/d)^{d/2}\big)$ and working in $O(d \cdot s \cdot f)$ space.

New insights into the problem of deciding the winner in parity games come from a connection to discounted payoff and simple stochastic games [Con92, ZP96] fist established by Puri [Pur95] and Jerrum [Sti95]. In 1966, Hoffman and

Karp [HK66] proposed a strategy improvement algorithm for solving stochastic games. The algorithm has gained a reputation of having a very fast convergence rate in practice but very little is proved about its performance. In particular, it is an open problem whether it is a polynomial time algorithm [Con93]. Puri [Pur95] has adapted the algorithm of Hoffman and Karp to discounted payoff games and advocated its use for solving parity games. The drawback of Puri's algorithm is that it requires solving linear programming instances and manipulates real numbers which makes implementations slow and blurs understanding of the algorithm.

In chapter 5 we propose a purely discrete strategy improvement algorithm for parity games [VJ00b]. We provide an efficient discrete algorithm for performing a strategy improvement step. The algorithm has been recently implemented [SV00]. The following remains an open question.

Problem 2.17 Does a strategy improvement algorithm for parity games terminate in polynomial number of steps?

We hope that the discrete structure revealed in our work might help in resolving this question.

2.7 A UP $\cap$ co-UP upper bound for parity games

In the following two sections we briefly review our earlier contributions to the study of the complexity of infinite games [Jur98, DJW97]. Some of these results were presented in the author's M.Sc. thesis [Jur97]. The results surveyed in the following two sections are included here only for completeness and do not form a part of the original technical contributions of this Ph.D. thesis. The result sketched in this section [Jur98] was published during the author's graduate studies at BRICS in Aarhus.

This section is devoted to a UP $\cap$ co-UP upper bound for the problem of deciding the winner in parity games [Jur98]. First we sketch a simple polynomial time reduction from parity games to mean payoff games. Then we mention how a UP $\cap$ co-UP upper bound follows for discounted payoff games from the results of Zwick and Paterson [ZP96].

Reduction of parity games to mean payoff games

A mean payoff game $G = (V, E, (M_0, M_1), d, w)$ consists of a directed graph (V, E), a partition $M_0 \cup M_1 = V$ of the set of vertices V, a number $d \in \mathbb{N}$, and a weight function $w : E \to \{-d, \ldots, -1, 0, 1, \ldots, d\}$. The two players: player 0 and player 1 play in the same way as in the infinite games of Section 2.4. An infinite play $\langle v_0, v_1, v_2, \ldots \rangle$ is winning for player 0 if

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} w(v_{i-1}, v_i) \geq 0.$$

Given a mean payoff game $G = (V, E, (M_0, M_1), d, w)$ and a real number $0 < \lambda < 1$, we define the discounted payoff game G_λ to have the same game graph as G, and the following winning condition: a play $\langle v_0, v_1, v_2, \ldots\rangle$ is winning for player 0 if

$$(1-\lambda)\sum_{i=1}^{\infty} \lambda^i \cdot w(v_{i-1}, v_i) \geq 0.$$

Below we sketch a simple reduction from parity games to mean payoff games. Essentially the same reduction has been given by Puri [Pur95]. A reduction of parity games to mean payoff games has been also obtained by Jerrum [Sti95].

Our reduction relies on memoryless determinacy of both parity and mean payoff games. Parity games have memoryless winning strategies by Theorem 2.9. Memoryless determinacy for mean payoff games has been shown by Ehrenfeucht and Mycielski [EM79] and independently by Gurvich et al. [GKK88].

Theorem 2.18 ([Pur95, Sti95, Jur98])
The problem of deciding the winner in parity games reduces in polynomial time to the problem of deciding the winner in mean payoff games.

Let $G = (V, E, (M_0, M_1), p)$ be a parity game. Given G and a starting vertex $v_0 \in V$ we construct a mean payoff game $H = (V, E, (M_0, M_1), d, w)$ with the same game graph, such that player 0 has a winning strategy from v_0 in G if and only if she has a winning strategy from v_0 in H. We give the reduction here; see [Jur98] for a proof of its correctness. Let $r = \max\{\ p(v)\ :\ v \in V\ \}$, and $m = |V|$. To define the mean payoff game H we set $d = m^r$, and for every edge $e = (v, u) \in E$ we let $w(e) = (-m)^{p(v)}$.

Observe that in the above reduction the weight function w has values exponential with respect to the size of the parity game G, but their binary represetations are of polynomial size and can be computed in polynomial time. Interestingly, having a reduction with polynomial values of the weight function would imply that solving parity games is in P, since a polynomial time algorithm for mean payoff games with polynomial weights is given by Zwick and Paterson [ZP96].

The UP $\cap$ co-UP upper bound

We argue now that deciding the winner in mean payoff games is in UP $\cap$ co-UP. Let $G = (V, E, (M_0, M_1), d, w)$ be a mean payoff game. Ehrenfeucht and Mycielski [EM79] and independently Gurvich et al. [GKK88] have proved that for every vertex $v_0 \in V$, there is a number $\nu(v_0)$, called the value of G at v_0, such that the following two conditions hold:

1. player 0 has a memoryless strategy, such that for every play $\langle v_0, v_1, v_2, \ldots\rangle$ consistent with this strategy, we have $\liminf_{n\to\infty} 1/n \sum_{i=1}^{n} w(v_{i-1}, v_i) \geq \nu(v_0)$,

2. player 1 has a memoryless strategy, such that for every play $\langle v_0, v_1, v_2, \ldots \rangle$ consistent with this strategy, we have $\limsup_{n\to\infty} 1/n \sum_{i=1}^{n} w(v_{i-1}, v_i) \leq \nu(v_0)$.

We call strategies satisfying those conditions optimal. Clearly, given the values of a game it is straightforward to decide the winner: it suffices to check whether $\nu(v_0) \geq 0$. Hence the values of a mean payoff game may seem to be a plausible candidate for unique short certificates. Unfortunately, we have failed to devise a polynomial time algorithm to check whether a vector $\langle x_v \rangle_{v \in V}$ is indeed the vector of the values of the game. Moreover, whereas it is straightforward to extract a winning strategy in a parity game from the canonical signature assignment [Wal96], we do not know how to do it efficiently given just the values of the mean payoff game.

In order to get the UP ∩ co-UP upper bound on the complexity of mean payoff games we use the following result of Zwick and Paterson [ZP96].

Theorem 2.19 ([ZP96])
The problem of deciding the winner in a mean payoff game reduces in polynomial time to the problem of deciding the winner in a discounted payoff game.

Hence it suffices to provide unique short certificates for winning strategies in discounted payoff games. Zwick and Paterson [ZP96] have shown that optimal memoryless strategies for both players exist also in discounted payoff games. In the two conditions above one has to replace the term $1/n \sum_{i=1}^{n} w(v_{i-1}, v_i)$ with $(1-\lambda)\sum_{i=1}^{n} \lambda^i \cdot w(v_{i-1}, v_i)$, and the number $\nu(v_0)$ with $\nu_\lambda(v_0)$, which we call the value of the discounted payoff game G_λ at v_0.

The vector of the values of a discounted payoff game is the unique certificate we are after. The crucial property which allows us to quickly check whether some $\langle x_v \rangle_{v \in V}$ is the vector of the values of a discounted payoff game is the following characterisation due to Zwick and Paterson [ZP96].

Theorem 2.20 ([ZP96])
The vector $\overline{\nu} = \langle \nu_\lambda(v) \rangle_{v \in V}$ of the values of the discounted payoff game G_λ is the unique solution *of the following system of equations*

$$x_v = \begin{cases} \max_{(v,u)\in E} \left\{ (1-\lambda)\cdot w(v,u) + \lambda \cdot x_u \right\} & \text{if } v \in M_0, \\ \min_{(v,u)\in E} \left\{ (1-\lambda)\cdot w(v,u) + \lambda \cdot x_u \right\} & \text{if } v \in M_1. \end{cases}$$

It only remains to show that the values of a discounted payoff games are *short*, i.e., can be written using a number of bits polynomial in $N = |G_\lambda|$, the size of the binary representation of the game G_λ. This can be done by standard techniques (see [Jur98] for details).

This settles the UP upper bound for the problem of deciding the winner in discounted payoff games. The co-UP upper bound follows easily, because by the existence of optimal strategies [ZP96] player 0 does *not* have a winning strategy from a vertex v_0 in the game G_λ if and only if $\nu_\lambda(v_0) < 0$, so it can be directly read from the vector of the values of the game.

Putting the UP $\cap$ co-UP upper bound for discounted payoff games, together with the reductions of Theorems 2.18 and 2.19, we get the UP $\cap$ co-UP upper bound.

Theorem 2.21 ([Jur98])
The problems of deciding the winner in parity, mean payoff and discounted payoff games are in UP $\cap$ co-UP.

2.8 On memory needed to win infinite games

This section summarizes the results on the size of memory needed for winning strategies in games with arbitrary Muller winning conditions [DJW97].

Already in the early '60s there was some interest in verification and synthesis of switching circuits [Chu63]. Büchi and Landweber [BL69] have pioneered algorithmic solutions for the synthesis problem. Their result amounts to saying that certain infinite games are determined, both players have finite state winning strategies, and there is in fact an algorithm to synthesize a winning finite state strategy. On the other hand there was a commonly shared opinion that games provide an appealing and natural model for parallel, concurrent and reactive, computation [CKS81, RP80, NYY92, NRY96, AHK97], and strategies in these games can be viewed as reactive programs. Among these, papers [NYY92, NRY96] give examples how both the system and a specification can be represented as a two player game, where synthesizing a reactive program satisfying the specification amounts to finding a winning strategy for one of the players.

Thomas [Tho95] in his survey article claims that whereas the verification problem has been extensively studied, refined, and extended in past decades, synthesis has been neglected. In fact, the complexity of the algorithm of Büchi and Landweber seems to be very high, and other authors studying the synthesis problem [ALW89, PR89a, PR89b] rejected the game theoretic approach because of this reason. McNaughton [McN93] has revived interest in it by giving an elegant algorithm for solving finite state infinite games with Muller winning conditions, and an estimate of its complexity. Variations of this algorithm have been worked out by Thomas [Tho95] and Zielonka [Zie98]. The common and remarkable feature of these algorithms is that the strategies constructed by them use one version or another of the LAR (latest appearance record) data structure, due to Büchi [Büc83] and Gurevich and Harrington [GH82]. An obvious drawback of all these procedures is that LAR's themselves are of exponential size with respect to the size of the game. Note that the memory size of a winning strategy corresponds to the size of the synthesized reactive program. However, as observed by many authors [McN93, Kla94, Les95, NRY96, Zie98], there are special cases when much less memory is needed.

Following a popular demand [McN93, Tho95, Les95, NRY96, Zie98] we have been looking for a classification of winning conditions according to how much memory is really needed for winning strategies. We have come up with a quite general criterion allowing to estimate the size of memory for winning strategies, based only on the structure of the Muller winning condition, as captured by the so called split tree [DJW97, Zie98]. By a careful analysis of Zielonka's version of McNaughton's algorithm [Zie98] we have obtained a non-trivial upper bound, for which a matching lower bound holds [DJW97]. The bound itself can be easily computed from the split tree [Zie98] of the winning condition. In particular, we have shown that there are winning conditions for which LAR's are essentially optimal.

Let $G = (V, E, (M_0, M_1), C, \chi, \mathcal{F})$ be a game, where C is a finite set of colors, $\chi : V \to C$ is a coloring function, and $\mathcal{F} \subseteq \wp(C)$ is a Muller winning condition expressed in terms of colors. An infinite play $\pi = \langle v_0, v_1, v_2, \ldots \rangle$ is winning for player 0 if and only if $\mathrm{Inf}(\pi)$, i.e., the set of colors appearing infinitely often in $\langle \chi(v_0), \chi(v_1), \chi(v_2), \ldots \rangle$, is in $\mathcal{F}$.

A strategy has finite memory if its values depend only on the last vertex in a position and a bounded information about the rest of the position. More precisely, a strategy for player 0 with memory M is given by an element $m_0 \in M$, and a pair of functions (ζ_M, ζ_V): a memory update function $\zeta_M : M \times V \to M$, and a next move function $\zeta_V : M_0 \times M \to V$. The first function is used to determine the contents of the memory in a position of the game:

$$\zeta_M(v_0, \ldots, v_k) = \zeta_M(v_k, \zeta_M(v_{k-1}, \cdots \zeta_M(v_1, \zeta_M(v_0, m_0)) \cdots)).$$

The second function defines the strategy $\zeta : \mathtt{Pos}(G) \to V$ by

$$\zeta(v_0, \ldots, v_k) = \zeta_V(v_k, \zeta_M(v_0, \ldots, v_k)).$$

One may think of a strategy with memory as an input/output automaton computing the strategy. This automaton inputs the moves taken by the opponent (player 1), keeps track of the memory in its finite control using the memory update function, and outputs the moves for the player (player 0) using the next move function.

Observe that if we take for M the set of all positions (i.e., all finite paths in the arena), and for ζ_M the identity function, then a strategy with memory is just a strategy as defined before. The notion of a strategy with memory is interesting in the case when the cardinality of M is smaller than the cardinality of the set of positions of the game. In particular, if we take M to be a one element set, then we obtain the notion of a memoryless strategy.

Gurevich and Harrington [GH82] have proved that bounded memory suffices for winning strategies in games with arbitrary Muller winning conditions.

Theorem 2.22 (Forgetful determinacy [GH82, McN93])
Games with Muller (Rabin, Streett) winning conditions are determined. Moreover, both players have winning strategies from their winning sets with finite memory of size $|C|!$, i.e., factorial in the number of colors.

Notation. Let $\mathcal{F} \subseteq \wp(C)$ be a winning condition. Define $\mathcal{F} \upharpoonright D \subseteq \wp(D)$ as the set $\{ D' \in \mathcal{F} : D' \subseteq D \}$.

Definition 2.23 (Split tree of a winning condition)
We define the split tree of $\mathcal{F} \subseteq \wp(C)$, denoted by $\mathcal{Z}_{\mathcal{F},C}$, inductively.

1. If $C \notin \mathcal{F}$ then $\mathcal{Z}_{\mathcal{F},C} = \mathcal{Z}_{\overline{\mathcal{F}},C}$ where $\overline{\mathcal{F}} = \wp(C) \setminus \mathcal{F}$.
2. If $C \in \mathcal{F}$ then the root of $\mathcal{Z}_{\mathcal{F},C}$ is labelled with C. Let $C_0, C_1, \ldots, C_{k-1}$ be all the maximal sets in $\{ X \notin \mathcal{F} : X \subseteq C \}$. Then we attach to the root, as its subtrees, the split trees of $\mathcal{F} \restriction C_i$, i.e., $\mathcal{Z}_{\mathcal{F}\restriction C_i, C_i}$, for $i = 0, 1, \ldots, k-1$.

[Definition 2.23] □

For every $\mathcal{F} \subseteq \wp(C)$, we define a number $m_{\mathcal{F}}$, such that for every game with $\mathcal{F}$ as the winning condition, a memory of size $m_{\mathcal{F}}$ is sufficient for a winning strategy for player 0 in the game. The number $m_{\mathcal{F}}$ is determined by the structure of a split tree of the winning condition $\mathcal{F}$.

Definition 2.24 (The number $m_{\mathcal{F}}$)
Let $\mathcal{F} \subseteq \wp(C)$ be a winning condition and $\mathcal{Z}_{\mathcal{F}_0,C_0}, \ldots, \mathcal{Z}_{\mathcal{F}_{k-1},C_{k-1}}$ be the subtrees attached to the root of the tree $\mathcal{Z}_{\mathcal{F},C}$. By $\mathcal{F}_i$ we denote the condition $\mathcal{F} \restriction C_i \subseteq \wp(C_i)$ for $i = 0, \ldots, k-1$. We define the number $m_{\mathcal{F}}$ as follows

$$m_{\mathcal{F}} = \begin{cases} 1 & \text{if } \mathcal{Z}_{\mathcal{F},C} \text{ does not have any subtrees,} \\ \max\{ m_{\mathcal{F}_0}, \ldots, m_{\mathcal{F}_{k-1}} \} & \text{if } C \notin \mathcal{F}, \\ \sum_{i=0}^{k-1} m_{\mathcal{F}_i} & \text{if } C \in \mathcal{F}. \end{cases}$$

It is not hard to see that for every $\mathcal{F} \subseteq \wp(C)$ we have that $m_{\mathcal{F}} \leq |C|!$. By a careful analysis of Zielonka's version [Zie98] of McNaughton's proof [McN93] of forgetful determinacy for games with Muller winning conditions we get an upper bound on the size of memory for winning strategies.

Theorem 2.25 ([DJW97])
In every game with a Muller winning condition $\mathcal{F}$, player 0 has a winning strategy with memory of size $m_{\mathcal{F}}$ on her winning set.

Interestingly, this upper bound is tight.

Theorem 2.26 ([DJW97])
For every $\mathcal{F} \subseteq \wp(C)$, there is a game with Muller winning condition $\mathcal{F}$, such that every winning strategy for player 0 in this game requires memory of size at least $m_{\mathcal{F}}$.

A drawback of this lower bound is that the sizes of game graphs we construct are as big as $m_{\mathcal{F}}$, and hence they can be exponential in the number of colors. Using different examples we have proved, however, that even on game graphs of size linear in the number of colors the latest appearance records are optimal.

Theorem 2.27 ([DJW97])
There is a family of games $\langle G_n \rangle_{n \in \mathbb{N}}$, such that G_n is of size $O(n)$ and every winning strategy for player 0 in G_n has memory of size at least $n!$.

Chapter 3

Independence models and bisimilarity

This chapter briefly surveys a selection of models with explicit representation of concurrency and a number of behavioural equivalences for these models. We recall two classical models: elementary net systems and asynchronous transition systems. Then we define a few bisimilarity notions, mention their logical and game theoretic characterizations, and discuss decidability of checking bisimilarity for finite state asynchronous transition systems and elementary net systems.

3.1 Models

Definition 3.1 (Asynchronous transition system)
A labelled asynchronous transition system [Shi85, Bed88, WN95]

$$A = (S, i, E, \rightarrow, I, L, \lambda)$$

consists of:

- a set of states S, and an initial state $i \in S$,
- a set of events E,
- a transition relation $\rightarrow \subseteq S \times E \times S$,
- an independence relation $I \subseteq E \times E$ which is symmetric and irreflexive,
- a set of labels L, and a labelling function $\lambda : E \rightarrow L$.

We often write $s \xrightarrow{e} s'$ instead of $(s, e, s') \in \rightarrow$. An asynchronous transition system must satisfy the following conditions:

1. if $s \xrightarrow{e} s'$ and $s \xrightarrow{e} s''$ then $s' = s''$,

2. if $(e, e') \in I$, $s \stackrel{e}{\rightarrow} s'$, and $s' \stackrel{e'}{\rightarrow} t$, then $s \stackrel{e'}{\rightarrow} s''$, and $s'' \stackrel{e}{\rightarrow} t$ for some $s'' \in S$.

An asynchronous transition system is *coherent* if it satisfies the following extra condition:

3. if $(e, e') \in I$, $s \stackrel{e}{\rightarrow} s'$, and $s \stackrel{e'}{\rightarrow} s''$, then $s' \stackrel{e'}{\rightarrow} t$, and $s'' \stackrel{e}{\rightarrow} t$ for some $t \in S$.

An asynchronous transition system is *prime* if it is acyclic and satisfies the following extra condition:

4. if $s \stackrel{e}{\rightarrow} t$ and $s' \stackrel{e'}{\rightarrow} t$ then $(e, e') \in I$. [Definition 3.1] □

Condition 1. states that an occurrence of an event in a state least to a unique state. Conditions 2. and 3. express properties of independence which is meant to model concurrency. Condition 2. demands that if two independent events can occur immediately one after another then they should be able to occur in the other order as well, i.e., all interleavings of concurrent events are possible. Condition 3. requires that if two independent events can occur at a state then they should be able to occur together in any order and thus reach a common state. Condition 4. demands that if occurrences of two events result in a common state then the events must be independent.

Definition 3.2 (Elementary net system)
A labelled elementary net system [Thi87]

$$N = (C, E, \mathtt{pre}, \mathtt{post}, M_0, L, \lambda)$$

consists of:

- a set C of conditions,
- a set E of events,
- functions $\mathtt{pre} : E \to \wp(E)$ and $\mathtt{post} : E \to \wp(E)$ specifying pre-conditions and post-conditions of events, respectively,
- an initial marking $M_0 \subseteq C$,
- a set of labels L, and a labelling function $\lambda : E \to L$. [Definition 3.2] □

For $e \in E$ we sometimes write ${}^\bullet e$ for the set $\mathtt{pre}(e)$ of pre-conditions of event e, and we write $e^\bullet$ for the set $\mathtt{post}(e)$ of post-conditions of e. We write ${}^\bullet e^\bullet$ for the set ${}^\bullet e \cup e^\bullet$, called the neighbourhood of e.

Conditions can be thought of as local states of an elementary net system (or, for short, a net). Markings of a net are sets of conditions and can be thought of as global states of the net. Independence of events is a derived notion in nets: events $e, f \in E$ are independent if and only if ${}^\bullet e^\bullet \ \cap \ {}^\bullet f^\bullet = \emptyset$, i.e., neighbourhoods of events e and f are disjoint.

The dynamics of a net is formalized by the notion of firing an event. An event $e \in E$ can be fired in a marking $M \subseteq C$ if ${}^\bullet e \subseteq M$, and $e^\bullet \cap M = \emptyset$.

The result of firing an event $e \in E$ in a marking $M \subseteq C$ is the marking $M' = (M \setminus {}^\bullet e) \cup e^\bullet$, we denote it by $M \xrightarrow{e}_N M'$. Note that a firing of an event e changes the global state (i.e., the marking) by changing local states (i.e., conditions) only in the neighbourhood of e, i.e., we have that $M \cap (C \setminus {}^\bullet e^\bullet) = M' \cap (C \setminus {}^\bullet e^\bullet)$

Relating the models

A thorough survey of models for concurrency can be found in a chapter by Winskel and Nielsen [WN95]. They show in detail, using notions of category theory, formal relationships between several established models for concurrency such as trace languages, event structures, Petri nets, and asynchronous transition systems.

For our purposes it suffices to recall the obvious translation of an elementary net system into an asynchronous transition system. Given an elementary net system $N = (C, E, \mathtt{pre}, \mathtt{post}, M_0)$ we define a labelled asynchronous transition system $na(N) = (S, i, E, \rightarrow, I, L, \lambda)$, where:

- the set of states S is the set of markings $\wp(C)$,
- the initial state i is the initial marking M_0,
- the set of events E and the labelling are inherited from the net N,
- transition relation $\rightarrow$ is the relation $\rightarrow_N$ of firing an event in the net,
- we define $(e, f) \in E$ to hold if and only if ${}^\bullet e^\bullet \cap {}^\bullet f^\bullet = \emptyset$.

Proposition 3.3 If N is a labelled elementary net system then $na(N)$ is a labelled asynchronous transition system.

Behaviours of asynchronous transition systems

Let $A = (S, s^{\mathrm{ini}}, E, \rightarrow, I, L, \lambda)$ be a labelled asynchronous transition system. A sequence of events $\overline{e} = \langle e_1, e_2, \ldots, e_n \rangle \in E^*$ is a *run* of A if there are states $s_1, s_2, \ldots, s_{n+1} \in S$, such that $s_1 = s^{\mathrm{ini}}$, and for all $i \in \{1, 2, \ldots, n\}$, we have $s_i \xrightarrow{e_i} s_{i+1}$. We write $\mathtt{Runs}(A)$ to denote the set of runs of A. We extend the labelling function λ to runs in the standard way.

Let $\overline{e} = \langle e_1, e_2, \ldots, e_n \rangle \in \mathtt{Runs}(A)$. We say that the k-th event, $1 \leq k < n$, is *swappable* in $\overline{e}$ if $(e_k, e_{k+1}) \in I$. We define $\mathrm{Swap}(\overline{e})$ to be the set of numbers of swappable events in $\overline{e}$. We write $\overline{e} \otimes k$ to denote the result of *swapping* the k-th event of $\overline{e}$ with the $(k+1)$-st, i.e., the sequence $\langle e_1, \ldots, e_{k-1}, e_{k+1}, e_k, \ldots, e_n \rangle$. Note that if $k \in \mathrm{Swap}(\overline{e})$ then $\overline{e} \otimes k \in \mathtt{Runs}(A)$; it follows from condition 2. of definition of an asynchronous transition system.

A run of a transition system models a finite *sequential* behaviour of a system: a sequence of occurrences of events. In order to model *concurrent* behaviours of a system we define an equivalence relation on the set of runs of an asynchronous transition system. We define the equivalence relation $\cong_A$ on $\texttt{Runs}(A)$ to be the reflexive, symmetric, and transitive closure of

$$\{ (\overline{e}, \overline{e} \otimes k) \ : \ \overline{e} \in \texttt{Runs}(A) \text{ and } k \in \mathrm{Swap}(\overline{e}) \}.$$

In other words, we have that $\overline{e_1} \cong_A \overline{e_2}$, for $\overline{e_1}, \overline{e_2} \in \texttt{Runs}(A)$, if and only if $\overline{e_2}$ can be obtained from $\overline{e_1}$ by a finite number of swaps of swappable events. A run is a linear order of occurrences of events. An equivalence class of the relation $\cong_A$ can be seen as a partial order of occurrences of events. A run $\overline{e} = \langle e_1 \cdot e_2 \cdots e_n \rangle$ determines a partial order on $\{1, 2, \ldots, n\}$ which is the reflexive and transitive closure of the relation $\lessdot$, where $i \lessdot j$ is defined to hold if $i < j$ and $(e_i, e_j) \notin I$. It can be shown that the $\cong_A$-equivalence class of a run is the set of all linearizations of this partial order.

We define an unfolding operation of asynchronous transition systems into prime asynchronous transition systems. The states of the unfolding of an asynchronous transition system A are meant to represent all concurrent behaviours of a system, just like the states of a synchronization tree represent all sequential behaviours of a system.

Definition 3.4 (Unfolding)
Let $A = (S, i, E, \rightarrow, I, L, \lambda)$ be an asynchronous transition system. The unfolding $\texttt{Unf}(A)$ of A is an asynchronous transition system with the same set of events, the labelling function, and the independence relation as A. The set of states, the initial state, and the transition relation of $\texttt{Unf}(A)$ are defined as follows:

- the set of states $S_{\texttt{Unf}(A)}$ of $\texttt{Unf}(A)$ is defined to be $\texttt{Runs}(A)/_{\cong_A}$, i.e., the set of concurrent behaviours of A,
- the initial state $i_{\texttt{Unf}(A)}$ of $\texttt{Unf}(A)$ is $[\varepsilon]_{\cong_A}$, i.e., the $\cong_A$-equivalence class of the empty run,
- the set of transitions $\rightarrow_{\texttt{Unf}(A)}$ of $\texttt{Unf}(A)$ consists of transitions of the form $\big([\overline{e}]_{\cong_A}, e, [\overline{e} \cdot e]_{\cong_A}\big)$, for all $\overline{e} \in E^*$, and $e \in E$, such that $\overline{e} \cdot e \in \texttt{Runs}(A)$.

[Definition 3.4] □

The following proposition follows easily from definition of $\texttt{Unf}(A)$.

Proposition 3.5 If A is an asynchronous transition system then its unfolding $\texttt{Unf}(A)$ is a prime asynchronous transition system.

3.2 Behavioural equivalences

The notion of behavioural equivalence which has attracted most attention in concurrency theory is bisimilarity, introduced by Park [Par81] and Milner [Mil80]. A notion of equivalence is meant to provide semantics for concurrent systems:

two systems are considered to have the same meaning if and only their unfoldings (representing all possible behaviours of systems) are equivalent, e.g., bisimilar.

Definition 3.6 (Bisimulation)
Let $A_1 = (S_1, i_1, E_1, \to_1, I_1, L, \lambda_1)$ and $A_2 = (S_2, i_2, E_2, \to_2, I_2, L, \lambda_2)$ be labelled asynchronous transition systems. A relation $B \subseteq \mathtt{Runs}(A_1) \times \mathtt{Runs}(A_2)$ is a bisimulation relating A_1 and A_2 if the following conditions are satisfied:

1. $(\varepsilon, \varepsilon) \in B$,

and if $(\overline{e_1}, \overline{e_2}) \in B$ then $\lambda_1(\overline{e_1}) = \lambda_2(\overline{e_2})$, and:

2. for all $e_1 \in E_1$, if $\overline{e_1} \cdot e_1 \in \mathtt{Runs}(A_1)$, then there exists $e_2 \in E_2$, such that $\overline{e_2} \cdot e_2 \in \mathtt{Runs}(A_2)$, and $\lambda_1(e_1) = \lambda_2(e_2)$, and $(\overline{e_1} \cdot e_1, \overline{e_2} \cdot e_2) \in B$,

3. for all $e_2 \in E_2$, if $\overline{e_2} \cdot e_2 \in \mathtt{Runs}(A_2)$, then there exists $e_1 \in E_1$, such that $\overline{e_1} \cdot e_1 \in \mathtt{Runs}(A_1)$, and $\lambda_1(e_1) = \lambda_2(e_2)$, and $(\overline{e_1} \cdot e_1, \overline{e_2} \cdot e_2) \in B$.

A bisimulation B is a back-and-forth bisimulation if the following extra condition is satisfied:

4. if $(\overline{e_1} \cdot e_1, \overline{e_2} \cdot e_2) \in B$ for $e_1 \in E_1$ and $e_2 \in E_2$, then $(\overline{e_1}, \overline{e_2}) \in B$.

[Definition 3.6] □

Two asynchronous transition systems A_1, and A_2 are *bisimilar*, if there is a bisimulation relating them. They are back-and-forth bisimilar if there is a back-and-forth bisimulation relating them.

Theorem 3.7 ([HS85])
Transition systems A_1 and A_2 are bisimilar if and only if they are back-and-forth bisimilar.

Note that the above notions of bisimilarity do not refer to the independence relation and hence capture only sequential behaviour of transition systems.

Definition 3.8 ((Hereditary) history preserving bisimulation)
A bisimulation relation $B \subseteq \mathtt{Runs}(A_1) \times \mathtt{Runs}(A_2)$ is a history preserving bisimulation (hp-bisimulation) if apart from conditions 1.–3. above the following extra condition holds:

5. if $(\overline{e_1}, \overline{e_2}) \in B$ then $\mathrm{Swap}(\overline{e_1}) = \mathrm{Swap}(\overline{e_2})$, and for all $k \in \mathrm{Swap}(\overline{e_1})$, we have $(\overline{e_1} \otimes k, \overline{e_2} \otimes k) \in B$.

A relation $B \subseteq \mathtt{Runs}(A_1) \times \mathtt{Runs}(A_2)$ is a hereditary history preserving bisimulation (hhp-bisimulation) if it satisfies conditions 1.–5. In other words, an hp-bisimulation is hereditary if it is a back-and-forth bisimulation. [Definition 3.8] □

We say that asynchronous transition systems A_1 and A_2 are history preserving bisimilar (hp-bisimilar) if there is an hp-bisimulation relating them. They are hereditary history preserving bisimilar (hhp-bisimilar) if there is a hhp-bisimulation relating them.

Hp-bisimilarity has been introduced by Rabinovich and Trakhtenbrot [RT88] and van Glabbeek and Goltz [GG89]. Hhp-bisimilarity has been proposed by Bednarczyk [Bed91] and independently by Joyal et al. [JNW96].

Unlike in the sequential case, by adding the back-and-forth property to the definition of hp-bisimulation we get a stronger notion of bisimilarity.

Proposition 3.9 ([Bed91, JNW96]) There are asynchronous transition systems which are hp-bisimilar but not hhp-bisimilar.

Bisimilarity, logic and games

Hennessy and Milner [HM85] have established that bisimilarity captures indistinguishability with respect to modal logic (often called Hennessy-Milner logic.)

Theorem 3.10 ([HM85])
Two labelled (finitely branching) transition systems satisfy the same modal logic formulas if and only if they are bisimilar.

Another related characterization of bisimilarity is the game theoretic one. Stirling [Sti97] has popularized the view of bisimilarity checking as a game played on a pair of transition systems by two players, one of them challenging bisimilarity of the transition systems and the other trying to prove that they are bisimilar. This game can be also seen as a version of Ehrenfeucht-Fraïssé games for modal logic [Tho93].

Logical and game theoretic characterizations have been generalized to hhp-bisimilarity by Nielsen and Clausen [NC95]. They introduce a modal logic with backwards modalities and interpret it over unfoldings of asynchronous transition systems.

Theorem 3.11 ([NC95])
(Unfoldings) of two labelled (finitely branching) asynchronous transition systems satisfy the same formulas of the modal logic with backwards modalities if and only if they are hhp-bisimilar.

Nielsen and Clausen [NC95] also introduced a natural bisimilarity checking games with backwards moves played on unfoldings of asynchronous transition systems.

Bisimilarity from open maps

The essence of bisimilarity, quoting [HM85], "is that the behaviour of a program is determined by how it communicates with an observer." Therefore, the notion of what can be observed of a behaviour of a system affects the notion of bisimilarity. An abstract definition of bisimilarity for arbitrary categories of models due to Joyal et al. [JNW96] formalizes this idea.

Given a category of models where objects are behaviours and morphisms represent a form of simulation, and given a subcategory of observable behaviours, the abstract definition yields a notion of bisimilarity for all behaviours with respect to observable behaviours. More concretely, Joyal et al. define open maps which are morphisms that, roughly speaking, reflect as well as preserve behaviour. Then two systems are (open maps) bisimilar if and only if there is a span of open maps between their behaviours. For example, for rooted labelled transition systems, taking synchronization trees [Mil80] into which they unfold as their behaviours, and sequences of actions as the observable behaviours, we recover the standard strong bisimilarity of Park and Milner [JNW96].

Theorem 3.12 ([JNW96])
Two labelled transition systems are (back-and-forth) bisimilar if and only if they are open maps bisimilar with sequences of labels as the observable behaviours.

Taking sequences, i.e., linear orders as observable behaviours results in not distinguishing between non-deterministic choices and concurrency. For models where concurrency is represented explicitly a natural choice is to replace sequences, i.e., linear orders as the observable behaviours, by labelled partial orders of occurrences of events with causality as the ordering relation. For example, taking unfoldings of labelled asynchronous transition systems into prime asynchronous transition systems (or event structures) as the behaviours, and labelled partial orders as the observations, Joyal et al. [JNW96] obtained hhp-bisimilarity from their abstract definition of open maps bisimilarity.

Theorem 3.13 ([JNW96])
Two labelled asynchronous transition systems are hhp-bisimilar if and only if they are open maps bisimilar with labelled partial orders as the observable behaviours.

Decidability issues

The problem of checking bisimilarity of transition systems is, given two transition systems, to decide whether they are bisimilar.

Theorem 3.14 ([PT87])
Checking bisimilarity can be done in time $O(m \cdot \log n)$, where n is the sum of numbers of states and m is the sum of sizes of transition relations of the input transition systems.

The problem of checking (h)hp-bisimilarity of asynchronous transition systems is, given two asynchronous transition systems, to decide whether they are (h)hp-bisimilar. The problem of checking (h)hp-bisimilarity of elementary net systems is, given two elementary net systems N_1 and N_2, to decide whether asynchronous transition systems $na(N_1)$ and $na(N_2)$ are (h)hp-bisimilar.

Theorem 3.15 ([Vog91])
The problem of checking hp-bisimilarity for elementary net systems is decidable.

By applying a technique similar to that used by Jategaonkar and Meyer [JM96] it can be shown that checking hp-bisimilarity of arbitrary asynchronous transition systems can be done in deterministic exponential time.

Hhp-bisimilarity, the back-and-forth version of hp-bisimilarity, seems to be only a slight strengthening of hp-bisimilarity [JNW96], and hence many attempts have been made to extend the above mentioned algorithms to the case of hhp-bisimilarity. However, decidability of hhp-bisimilarity has remained open, despite several attempts over the years [NC95, NW96a, CS96, FH99].

Fröschle and Hildebrandt [FH99] have discovered an infinite hierarchy of bisimilarity notions refining hp-bisimilarity, and coarser than hhp-bisimilarity, such that hhp-bisimilarity is the intersection of all the bisimilarities in the hierarchy. They have shown all these bisimilarities to be decidable for 1-safe Petri nets. Fröschle [Frö00] has shown hhp-bisimilarity to be decidable for BPP-processes, a class of infinite state systems.

In chapter 6 of this dissertation we finally settle the question of decidability of hhp-bisimilarity by showing it to be undecidable for finite elementary net systems.

Theorem 3.16 ([JN00])
The problem of checking hhp-bisimilarity for elementary net systems is undecidable.

In order to make the proof more transparent we first introduce an intermediate problem of domino bisimilarity and show its undecidability by a direct reduction from the halting problem of 2-counter machines. Domino bisimilarity and domino bisimulation games we introduce are a novel variation of the classical domino problems [Har85], and different from domino games of Grädel [Grä90] and domino snakes [EHM94].

Part II

Papers

Chapter 4

Small progress measures for PG's

This chapter contains a revised version of [Jur00].

Abstract. In this paper we develop a new algorithm for deciding the winner in parity games, and hence also for the modal μ-calculus model checking. The design and analysis of the algorithm is based on a notion of game progress measures: they are witnesses for winning strategies in parity games. We characterize game progress measures as pre-fixed points of certain monotone operators on a complete lattice. As a result we get the existence of the least game progress measures and a straightforward way to compute them. The worst-case running time of our algorithm matches the best worst-case running time bounds known so far for the problem, achieved by the algorithms due to Browne et al., and Seidl. Our algorithm has better space complexity: it works in small polynomial space; the other two algorithms have exponential worst-case space complexity.

4.1 Introduction

A parity game is an infinite path-forming game played by two players, player $\diamond$ and player $\Box$, on a graph with integer priorities assigned to vertices. In order to determine the winner in an infinite play we check the parity of the lowest priority occurring infinitely often in the play: if it is even then player $\diamond$ wins, otherwise player $\Box$ is the winner. The problem of deciding the winner in parity games is, given a parity game and an initial vertex, to decide whether player $\diamond$ has a winning strategy from the vertex.

There are at least two motivations for the study of the complexity of deciding the winner in parity games. One is that the problem is polynomial time equivalent to the modal μ-calculus model checking [EJS93, Sti95], hence developing better algorithms for parity games may lead to better model checking tools, which is a major objective in computer aided verification. The other is that the problem has an interesting status from the point of view of structural complexity theory. It is known to be in **NP** $\cap$ **co-NP** [EJS93] (and even in **UP** $\cap$ **co-UP** [Jur98]), and hence it is very unlikely to be **NP**-complete, but at the same time it is not known to be in **P**, despite substantial effort of the community (see [EJS93, BCJ+97, Sei96, ZP96] and references therein).

Progress measures [KK91] are decorations of graphs whose local consistency guarantees some global, often infinitary, properties of graphs. Progress measures have been used successfully for complementation of automata on infinite words and trees [Kla91, Kla94]; they also underlie a translation of alternating parity automata on infinite words to weak alternating automata [KV98]. A similar notion, a signature, occurs in the study of the modal μ-calculus [SE89]. Signatures were used to prove memoryless determinacy of parity games [EJ91, Wal96].

Our algorithm for parity games is based on the notion of game parity progress measures; Walukiewicz [Wal96] calls them consistent signature assignments. Game parity progress measures are witnesses for winning strategies in parity games. We provide an upper bound on co-domains of progress measures; this reduces the search space of potential witnesses. Then we provide a characterization of game parity progress measures as pre-fixed points of certain monotone operators on a finite complete lattice. This characterization implies that the least game parity progress measures exist, and it also suggests an easy way to compute them.

The modal μ-calculus model checking problems is, given a formula φ of the modal μ-calculus and a Kripke structure K with a set of states S, to decide whether the formula is satisfied in the initial state of the Kripke structure. The problem has been studied by many researchers; see for example [EL86, EJS93, BCJ+97, Sei96, LRS98] and references therein. The algorithms with the best proven worst-case running time bounds so far are due to Browne et al. [BCJ+97], and Seidl [Sei96]. Their worst-case running time bounds are roughly $O\big(d^2 \cdot m \cdot n^{\lceil d/2 \rceil}\big)$ and $O\big(d \cdot m \cdot \big((n+d)/d\big)^{\lceil d/2 \rceil}\big)$, respectively, where n and m are some numbers depending on φ and K, such that $n \leq |S| \cdot |\varphi|$, $m \leq |K| \cdot |\varphi|$, and d is the alternation depth of the formula φ.

In fact, number n above is the number of vertices in the parity game obtained from the formula and the Kripke structure via the standard reduction of the modal μ-calculus model checking to parity games, and m is the number of edges in the game graph; see for example [EJS93, Sti95]. Moreover, the reduction can be done in such a way that the number of different priorities in the parity game is equal to the alternation depth d of the formula. Our algorithm has worst-case running time $O\big(d \cdot m \cdot (n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor}\big)$, and it can be made to work in time $O\big(d \cdot m \cdot \big((n+d)/d\big)^{\lceil d/2 \rceil}\big)$, hence it matches the bounds of the other two algorithms. Moreover, it works in space $O(d \cdot n)$ while the other two algorithms have exponential worst-case space complexity. Our algorithm can be seen as a generic algorithm allowing many different evaluation policies; good heuristics can potentially improve performance of the algorithm. However, we show a family of examples for which worst-case running time occurs for all evaluation policies.

Among algorithms for parity games it is worthwhile to mention the algorithm of McNaughton [McN93] and its slight modification by Zielonka [Zie98]. In the extended version of this paper we show that McNaughton/Zielonka's algorithm can be implemented to work in time roughly $O\big(m \cdot (n/d)^d\big)$, and we also provide a family of examples for which the algorithm needs this time. Zielonka's algorithm works in fact for games with more general Muller winning conditions. By a careful analysis of the algorithm for games with Rabin (Streett) winning conditions we get a running time bound $O\big(m \cdot n^{2k}\big)$, where k is the number of pairs in the Rabin (Streett) condition. The algorithm also works in small polynomial space. This compares favourably with other algorithms for the linear-time equivalent problem of checking non-emptiness of non-deterministic Rabin (Streett) tree automata [EJ88, PR89a, KV98], and makes it the best algorithm known for this **NP**-complete (co-**NP**-complete) [EJ88] problem.

4.2 Parity games

Notation: For all $n \in \mathbb{N}$, by $[n]$ we denote the set $\{0, 1, 2, \ldots, n-1\}$. If (V, E) is a directed graph and $W \subseteq V$, then by $(V, E) \upharpoonright W$ we denote the subgraph (W, F) of (V, E), where $F = E \cap W^2$. [Notation] □

A *parity graph* $G = (V, E, p)$ consists of a directed graph (V, E) and a priority function $p : V \to [d]$, where $d \in \mathbb{N}$. A *parity game* $\Gamma = \big(V, E, p, (V_\Diamond, V_\Box)\big)$ consists of a parity graph $G = (V, E, p)$, called the *game graph* of Γ, and of a partition $(V_\Diamond, V_\Box)$ of the set of vertices V. For technical convenience we assume that all game graphs have the property that every vertex has at least one out-going edge. We also restrict ourselves throughout this paper to games with finite game graphs.

A parity game is played by two players: player $\Diamond$ and player $\Box$, who form an infinite path in the game graph by moving a token along edges. They start by placing the token on an initial vertex and then they take moves indefinitely in the following way. If the token is on a vertex in $V_\Diamond$ then player $\Diamond$ moves the token along one of the edges going out of the vertex. If the token is on a vertex in $V_\Box$ then player $\Box$ takes a move. In the result players form an infinite path $\pi = \langle v_1, v_2, v_3, \ldots\rangle$ in the game graph; for brevity we refer to such infinite paths as *plays*. The winner in a play is determined by referring to priorities of vertices in the play. Let $\mathrm{Inf}(\pi)$ denote the set of priorities occurring infinitely often in $\big\langle p(v_1), p(v_2), p(v_3), \ldots \big\rangle$. A play π is a *winning play* for player $\Diamond$ if $\min\big(\mathrm{Inf}(\pi)\big)$ is even, otherwise π is a winning play for player $\Box$.

A function $\sigma : V_\Diamond \to V$ is a *strategy* for player $\Diamond$ if $\big(v, \sigma(v)\big) \in E$ for all $v \in V_\Diamond$. A play $\pi = \langle v_1, v_2, v_3, \ldots\rangle$ is *consistent* with a strategy σ for player $\Diamond$ if $v_{\ell+1} = \sigma(v_\ell)$, for all $\ell \in \mathbb{N}$, such that $v_\ell \in V_\Diamond$. A strategy σ is a *winning strategy* for player $\Diamond$ from set $W \subseteq V$, if every play starting from a vertex in W and consistent with σ is winning for player $\diamond$. Strategies and winning strategies are defined similarly for player $\Box$.

Theorem 4.1 (Memoryless Determinacy [EJ91, Mos91])
For every parity game, there is a unique partition $(W_\diamond, W_\Box)$ of the set of vertices of its game graph, such that there is a winning strategy for player $\diamond$ from $W_\diamond$, and a winning strategy for player $\Box$ from $W_\Box$.

We call the sets $W_\diamond$ and $W_\Box$ the *winning sets* of player $\diamond$ and player $\Box$, respectively. The problem of *deciding the winner* in parity games is, given a parity game and a vertex in the game graph, to determine whether the vertex is in the winning set of player $\diamond$.

Before we proceed we mention a simple characterization of winning strategies for player $\diamond$ in terms of simple cycles in a subgraph of the game graph associated with the strategy. We say that a strategy σ for player $\diamond$ is *closed* on a set $W \subseteq V$ if for all $v \in W$, we have:

- if $v \in V_\diamond$ then $\sigma(v) \in W$, and
- if $v \in V_\Box$ then $(v, w) \in E$ implies $w \in W$.

Note that if a strategy σ for player $\diamond$ is closed on W then every play starting from a vertex in W and consistent with σ stays within W.

If σ is a strategy for player $\diamond$ then by G_σ we denote the parity graph (V, E_σ, p) obtained from game graph $G = (V, E, p)$ by removing from E all edges (v, w) such that $v \in V_\diamond$ and $\sigma(v) \neq w$.

We say that a cycle in a parity graph is an *i-cycle* if i is the smallest priority of a vertex occurring in the cycle. A cycle is an even cycle if it is an i-cycle for some even i, otherwise it is an odd cycle. The following proposition is not hard to prove.

Proposition 4.2 Let σ be a strategy for player $\diamond$ closed on W. Then σ is a winning strategy for player $\diamond$ from W if and only if all simple cycles in $G_\sigma \restriction W$ are even.

4.3 Small progress measures

In this section we study a notion of progress measures. Progress measures play a key role in the design and analysis of our algorithm for solving parity games.

First we define parity progress measures for parity graphs, and we show that there is a parity progress measure for a parity graph if and only if all cycles in the graph are even. In other words, parity progress measures are witnesses for the property of parity graphs having only even cycles. The proof of the 'if' part also provides an upper bound on the size of the co-domain of a parity progress measure. Then we define game parity progress measures for parity games, we argue that they are witnesses for winning strategies for player $\diamond$, and we show that the above-mentioned upper bound holds also for game parity progress measures.

Notation: If $\alpha \in \mathbb{N}^d$ is a d-tuple of non-negative integers then we number its components from 0 to $d-1$, i.e., we have $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{d-1})$. When applied to tuples of natural numbers, the comparison symbols $<, \leq, =, \neq, \geq$, and $>$ denote the lexicographic ordering. When subscripted with a number $i \in \mathbb{N}$ (e.g., $<_i, =_i, \geq_i$), they denote the lexicographic ordering on $\mathbb{N}^i$ applied to the arguments truncated to their first i components. For example, $(2,3,0,0) >_2 (2,2,4,1)$, but $(2,3,0,0) =_0 (2,2,4,1)$. [Notation] □

Definition 4.3 (Parity progress measure)
Let $G = \big(V, E, p : V \to [d]\big)$ be a parity graph. A function $\varrho : V \to \mathbb{N}^d$ is a parity progress measure for G if for all $(v,w) \in E$, we have $\varrho(v) \geq_{p(v)} \varrho(w)$, and the inequality is strict if $p(v)$ is odd. [Definition 4.3] □

Proposition 4.4 If there is a parity progress measure for a parity graph G then all cycles in G are even.

Proof: Let $\varrho : V \to \mathbb{N}^d$ be a parity progress measure for G. For the sake of contradiction suppose that there is an odd cycle $v_1, v_2, \ldots, v_\ell$ in G, and let $i = p(v_1)$ be the smallest priority on this cycle. Then by the definition of a progress measure we have $\varrho(v_1) >_i \varrho(v_2) \geq_i \varrho(v_2) \geq_i \cdots \geq_i \varrho(v_\ell) \geq_i \varrho(v_1)$, and hence $\varrho(v_1) >_i \varrho(v_1)$, a contradiction. [Proposition 4.4] ■

If $G = \big(V, E, p : V \to [d]\big)$ is a parity graph then for every $i \in [d]$, we write V_i to denote the set $p^{-1}(i)$ of vertices with priority i in parity graph G. Let $n_i = |V_i|$, for all $i \in [d]$. Define M_G to be the following finite subset of $\mathbb{N}^d$: if d is even then

$$M_G = [1] \times [n_1 + 1] \times [1] \times [n_3 + 1] \times \cdots \times [1] \times [n_{d-1} + 1];$$

for odd d we have $\cdots \times [n_{d-2}+1] \times [1]$ at the end. In other words, M_G is the finite set of d-tuples of integers with only zeros on even positions, and non-negative integers bounded by $|V_i|$ on every odd position i.

Theorem 4.5 (Small parity progress measure)
If all cycles in a parity graph G are even then there is a parity progress measure $\varrho : V \to M_G$ for G.

Proof: The proof goes by induction on the number of vertices in $G = \big(V, E, p : V \to [d]\big)$. For the induction to go through we slightly strengthen the statement of the theorem: we additionally claim, that if $p(v)$ is odd then $\varrho(v) >_{p(v)} (0, \ldots, 0)$. The statement of the theorem holds trivially if G has only one vertex.

Without loss of generality we may assume that $V_0 \cup V_1 \neq \emptyset$; otherwise we can scale down the priority function of G by two, i.e., replace the priority function p by the function $p - 2$ defined by $\big(p-2\big)(v) = p(v) - 2$, for all $v \in V$. Suppose first that $V_0 \neq \emptyset$. By induction hypothesis there is a parity progress measure $\varrho : \big(V \setminus V_0\big) \to M_G$ for the subgraph $G \restriction \big(V \setminus V_0\big)$. Setting $\varrho(v) = (0, \ldots, 0) \in M_G$, for all $v \in V_0$, we get a parity progress measure for G.

Suppose that $V_0 = \emptyset$ then $V_1 \neq \emptyset$. We claim that there is a non-trivial partition (W_1, W_2) of the set of vertices V, such that there is no edge from W_1 to W_2 in G.

Let $u \in V_1$; define $U \subseteq V$ to be the set of vertices to which there is a non-trivial path from u in G. If $U = \emptyset$ then $W_1 = \{u\}$ and $W_2 = V \setminus \{u\}$ is a desired partition of V. If $U \neq \emptyset$ then $W_1 = U$ and $W_2 = V \setminus U$ is a desired partition. The partition is non-trivial (i.e., $V \setminus U \neq \emptyset$) since $u \notin U$: otherwise a non-trivial path from u to itself gives a 1-cycle because $V_0 = \emptyset$, contradicting the assumption that all cycles in G are even.

Let $G_1 = G \restriction W_1$, and $G_2 = G \restriction W_2$ be subgraphs of G. By induction hypothesis there are parity progress measures $\varrho_1 : W_1 \to M_{G_1}$ for G_1, and $\varrho_2 : W_2 \to M_{G_2}$ for G_2. Let $n'_i = \big|V_i \cap W_1\big|$, and let $n''_i = \big|V_i \cap W_2\big|$, for $i \in [d]$. Clearly $n_i = n'_i + n''_i$, for all $i \in [d]$. Recall that there are no edges from W_1 to W_2 in G. From this and our additional claim applied to ϱ_2 it follows that the function $\varrho = \varrho_1 \cup \big(\varrho_2 + (0, n'_1, 0, n'_3, \ldots)\big) : V \to M_G$ is a parity progress measure for G. [Theorem 4.5] ■

Let $\Gamma = \big(V, E, p, (V_\Diamond, V_\Box)\big)$ be a parity game and let $G = (V, E, p)$ be its game graph. We define $M_G^\top$ to be the set $M_G \cup \{\top\}$, where $\top$ is an extra element. We use the standard comparison symbols (e.g., $<, =, \geq$, etc.) to denote the order on $M_G^\top$ which extends the lexicographic order on M_G by taking $\top$ as the biggest element, i.e., we have $m < \top$, for all $m \in M_G$. Moreover, for all $m \in M_G$ and $i \in [d]$, we set $m <_i \top$, and $\top =_i \top$. If $\varrho : V \to M_G^\top$ and $(v, w) \in E$ then

> by $\mathrm{Prog}(\varrho, v, w)$ we denote the least $m \in M_G^\top$, such that $m \geq_{p(v)} \varrho(w)$, and if $p(v)$ is odd then either the inequality is strict, or $m = \varrho(w) = \top$.

Definition 4.6 (Game parity progress measure)
A function $\varrho : V \to M_G^\top$ is a game parity progress measure if for all $v \in V$, we have:

- if $v \in V_\Diamond$ then $\varrho(v) \geq_{p(v)} \mathrm{Prog}(\varrho, v, w)$ for some $(v, w) \in E$, and
- if $v \in V_\Box$ then $\varrho(v) \geq_{p(v)} \mathrm{Prog}(\varrho, v, w)$ for all $(v, w) \in E$;

by $\|\varrho\|$ we denote the set $\big\{\, v \in V \;:\; \varrho(v) \neq \top \,\big\}$. [Definition 4.6] □

For every game parity progress measure ϱ we define a strategy $\widetilde{\varrho} : V_\Diamond \to V$ for player $\Diamond$, by setting $\widetilde{\varrho}(v)$ to be a successor w of v, which minimizes $\varrho(w)$.

Corollary 4.7 If ϱ is a game parity progress measure then $\widetilde{\varrho}$ is a winning strategy for player $\Diamond$ from $\|\varrho\|$.

Proof: Note first that ϱ restricted to $\|\varrho\|$ is a parity progress measure on $G_{\widetilde{\varrho}} \restriction \|\varrho\|$. Hence by Proposition 4.4 all simple cycles in $G_{\widetilde{\varrho}} \restriction \|\varrho\|$ are even.

It also follows easily from definition of a game parity progress measure that strategy $\widetilde{\varrho}$ is closed on $\|\varrho\|$. Therefore, by Proposition 4.2 we get that $\widetilde{\varrho}$ is a winning strategy for player $\diamond$ from $\|\varrho\|$. [Corollary 4.7] ■

Corollary 4.8 (Small game parity progress measure)
There is a game progress measure $\varrho : V \to M_G^\top$ such that $\|\varrho\|$ is the winning set of player $\diamond$.

Proof: It follows from Theorem 4.1 that there is a winning strategy σ for player $\diamond$ from her winning set $W_\diamond$, which is closed on $W_\diamond$. Therefore by Proposition 4.2 all cycles in parity graph $G_\sigma \restriction W_\diamond$ are even, hence by Theorem 4.5 there is a parity progress measure $\varrho : W_\diamond \to M_G$ for $G_\sigma \restriction W_\diamond$. It follows that setting $\varrho(v) = \top$ for all $v \in V \setminus W_\diamond$, makes ϱ a game parity progress measure. [Corollary 4.8] ■

4.4 The algorithm

In this section we present a simple algorithm for solving parity games based on the notion of a game parity progress measure. We characterize game parity progress measures as (pre-)fixed points of certain monotone operators in a finite complete lattice. By Knaster-Tarski theorem it implies existence of the *least* game progress measure μ, and a simple way to compute it. It then follows from Corollaries 4.8 and 4.7 that $\|\mu\|$ is the winning set of player $\diamond$.

Before we present the algorithm we define an ordering, and a family of $\mathrm{Lift}(\cdot, v)$ operators for all $v \in V$, on the set of functions $V \to M_G^\top$. Given two functions $\mu, \varrho : V \to M_G^\top$, we define $\mu \sqsubseteq \varrho$ to hold if $\mu(v) \le \varrho(v)$ for all $v \in V$. The ordering relation $\sqsubseteq$ gives a complete lattice structure on the set of functions $V \to M_G^\top$. We write $\mu \sqsubset \varrho$ if $\mu \sqsubseteq \varrho$, and $\mu \ne \varrho$. Define $\mathrm{Lift}(\varrho, v)$ for $v \in V$ as follows:

$$\mathrm{Lift}\big(\varrho, v\big)(u) = \begin{cases} \varrho(u) & \text{if } u \ne v, \\ \max\big\{\varrho(v), \min_{(v,w)\in E} \mathrm{Prog}(\varrho, v, w)\big\} & \text{if } u = v \in V_\diamond, \\ \max\big\{\varrho(v), \max_{(v,w)\in E} \mathrm{Prog}(\varrho, v, w)\big\} & \text{if } u = v \in V_\square. \end{cases}$$

The following propositions follow immediately from definitions of a game parity progress measure, and of the $\mathrm{Lift}(\cdot, v)$ operators.

Proposition 4.9 For every $v \in V$, the operator $\mathrm{Lift}(\cdot, v)$ is $\sqsubseteq$-monotone.

Proposition 4.10 A function $\varrho : V \to M_G^\top$ is a game parity progress measure, if and only if is it is a simultaneous pre-fixed point of all $\mathrm{Lift}(\cdot, v)$ operators, i.e., if $\mathrm{Lift}(\varrho, v) \sqsubseteq \varrho$ for all $v \in V$.

From Knaster-Tarski theorem it follows that the $\sqsubseteq$-*least* game parity progress measure exists, and it can be obtained by running the following simple procedure computing the least simultaneous (pre-)fixed point of operators $\mathrm{Lift}(\cdot, v)$, for all $v \in V$.

```
ProgressMeasureLifting
    μ := λv ∈ V.(0, ..., 0)
    while μ ⊏ Lift(μ, v) for some v ∈ V do μ := Lift(μ, v)
```

Theorem 4.11 (The algorithm)
Given a parity game, procedure `ProgressMeasureLifting` *computes winning sets for both players and a winning strategy for player $\diamond$ from her winning set; it works in space $O(d \cdot n)$, and its running time is*

$$O\Big(dm \cdot \Big(\frac{n}{\lfloor d/2 \rfloor}\Big)^{\lfloor d/2 \rfloor}\Big),$$

where n is the number of vertices, m is the number of edges, and d is the maximum priority in the parity game.

Proof: The result of running `ProgressMeasureLifting` on a parity game is the $\sqsubseteq$-least game progress measure μ. Let $W_\diamond$ be the winning set of player $\diamond$. By minimality of μ and by Corollary 4.8 it follows that $W_\diamond \subseteq \|\mu\|$. Moreover, Corollary 4.7 implies that $\widetilde{\mu}$ is a winning strategy for player $\diamond$ from $\|\mu\|$, and hence by Theorem 4.1 we get that $\|\mu\| \subseteq W_\diamond$, i.e., $\|\mu\| = W_\diamond$.

Procedure `ProgressMeasureLifting` algorithm works in space $O(d \cdot n)$ because it only needs to maintain a d-tuple of integers for every vertex in the game graph. The $\mathrm{Lift}(\cdot, v)$ operator, for every $v \in V$, can be implemented to work in time $O\big(d \cdot \text{out-deg}(v)\big)$, where out-deg$(v)$ is the out-degree of v. Every vertex can be "lifted" only $|M_G|$ many times, hence the running time of procedure `ProgressMeasureLifting` is bounded by

$$O\Big(\sum_{v \in V} d \cdot \text{out-deg}(v) \cdot |M_G|\Big) = O\big(d \cdot m \cdot |M_G|\big).$$

To get the claimed time bound it suffices to notice that

$$|M_G| = \prod_{i=1}^{\lfloor d/2 \rfloor} (n_{2i-1} + 1) \le \Big(\frac{n}{\lfloor d/2 \rfloor}\Big)^{\lfloor d/2 \rfloor},$$

because $\sum_{i=1}^{\lfloor d/2 \rfloor}(n_{2i-1} + 1) \le n$ if $n_i \neq 0$ for all $i \in [d]$, which we can assume without loss of generality; if $n_i = 0$ for some $i \in [d]$ then we can scale down the priorities bigger than i by two. [Theorem 4.11] ■

Remark: Our algorithm for solving parity games can be made to have

$$O\Big(d \cdot m \cdot \Big(\frac{n+d}{d}\Big)^{\lceil d/2 \rceil}\Big)$$

as its worst-case running time bound, which is better than $O\big(d \cdot m \cdot (n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor}\big)$ for even d, and for odd d if $2 \log_{1/(1-\varepsilon)} n \le d \le (1-\varepsilon)n$, for some $0 < \varepsilon < 1$.

If $\sum_{0\le 2i-1<d}(n_{2i-1}+1) \le (n+d)/2$ then we have

$$|M_G| \;=\; \prod_{0\le 2i-1<d} (n_{2i-1}+1) \;\le\; \Big(\frac{(n+d)/2}{\lfloor d/2\rfloor}\Big)^{\lfloor d/2\rfloor} \;\le\; \Big(\frac{n+d}{d-1}\Big)^{\lfloor d/2\rfloor}.$$

Otherwise, we have $\sum_{0\le 2i<d}(n_{2i}+1) \le (n+d)/2$. In this case it suffices to run procedure `ProgressMeasureLifting` on the dual game $\overline{G}$, i.e., the game obtained by scaling all priorities in game G up by one, and swapping sets in the $(V_\Diamond, V_\Box)$ partition. The winning set of player $\Diamond$ in the original game is the winning set of player $\Box$ in the dual game. Note that

$$|M_{\overline{G}}| \;=\; \prod_{0\le 2i<d} (n_{2i}+1) \;\le\; \Big(\frac{(n+d)/2}{\lceil d/2\rceil}\Big)^{\lceil d/2\rceil} \;\le\; \Big(\frac{n+d}{d}\Big)^{\lceil d/2\rceil}.$$

To conclude it suffices to verify that $\big((n+d)/(d-1)\big)^{\lfloor d/2\rfloor} \le \big((n+d)/d\big)^{\lceil d/2\rceil}$, for $1 \le d \le n$. [Remark] $\Box$

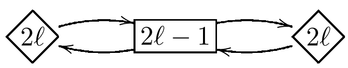
Note that in order to make `ProgressMeasureLifting` a fully deterministic algorithm one has to fix a policy of choosing vertices at which the function μ is being "lifted". Hence it can be considered as a generic algorithm whose performance might possibly depend on supplying heuristics for choosing the vertices to lift. Unfortunately, as we show in the next section, there is a family of examples on which the worst case performance of the algorithm occurs for all vertex lifting policies.

4.5 Worst-case behaviour

Theorem 4.12 (Worst-case behaviour)
For all $d, n \in \mathbb{N}$ such that $d \le n$, there is a game of size $O(n)$ with priorities not bigger than d, on which procedure `ProgressMeasureLifting` *performs at least $\big(\lceil n/d\rceil\big)^{\lceil d/2\rceil}$ many lifts, for all lifting policies.*

Proof: We define the family of games $H_{\ell,b}$, for all $\ell, b \in \mathbb{N}$. The game graph of $H_{\ell,b}$ consists of ℓ "levels", each level contains b "blocks". There is one "odd" level, and $\ell-1$ "even" levels.

The basic building block of the odd level is the following subgraph.



The numbers in vertices are their priorities. The odd level of $H_{\ell,b}$ consists of b copies of the above block assembled together by identifying the left-hand vertex with priority 2ℓ of the a-th block, for every $a \in \{1, 2, \ldots, b-1\}$, with the right-hand vertex with priority 2ℓ of the $(a+1)$-st block. For example the odd level of $H_{4,3}$ is the following.

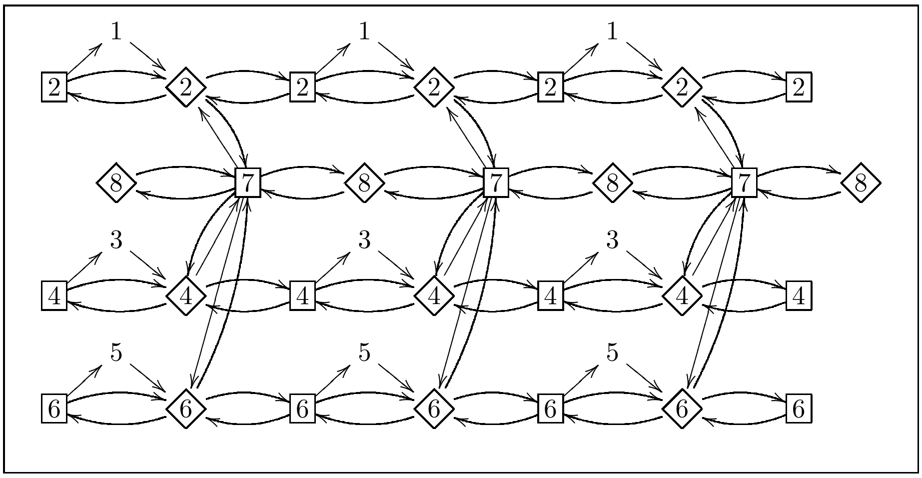
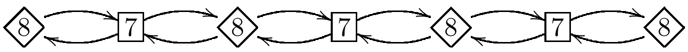
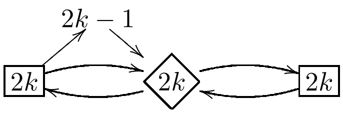


Figure 4.1: The game $H_{4,3}$.



In all our pictures vertices with a diamond-shaped frame are meant to belong to $V_\Diamond$, i.e., they are vertices where player $\Diamond$ moves; vertices with a box-shaped frame belong to $V_\Box$. Some vertices have no frame; for concreteness let us assume that they belong to $V_\Diamond$, but including them to $V_\Box$ would not change our reasoning, because they all have only one successor in the game graph of $H_{\ell,b}$.

The basic building block of the k-th even level, for $k \in \{1, 2, \ldots, \ell - 1\}$, is the following subgraph.



Every even level is built by putting b copies of the above block together in a similar way as for the odd level.

To assemble the game graph of $H_{\ell,b}$ we connect all $\ell - 1$ even levels to the odd level, by introducing edges in the following way. For every even level $k \in \{1, 2, \ldots, \ell - 1\}$, and for every block $a \in \{1, 2, \ldots, b\}$, we introduce edges in both directions between the box vertex with priority $2\ell - 1$ from the a-th block of the odd level, and the diamond vertex with priority $2k$ from the a-th block of the k-th even level. See Figure 4.1 for an example: the game $H_{4,3}$.

Claim 4.13 Every vertex with priority $2\ell - 1$ in game $H_{\ell,b}$ is lifted $(b+1)^\ell$ many times by procedure `ProgressMeasureLifting`.

Proof: Note that in game $H_{\ell,b}$ player $\Diamond$ has a winning strategy from all vertices in even levels, and player $\Box$ has a winning strategy from all vertices in the odd level; see Figure 4.1. Therefore, the value of the least progress measure in all vertices with priority $2\ell - 1$ is $\top \in M^{\top}_{H_{\ell,b}}$. Hence it suffices to show that every vertex with priority $2\ell - 1$ can be lifted only to its immediate successor in the order on $M^{\top}_{H_{\ell,b}}$. Then it is lifted $\left|M_{H_{\ell,b}}\right| = (b+1)^{\ell}$ many times, because

$$M_{H_{\ell,b}} = \underbrace{[1] \times [b+1] \times [1] \times [b+1] \times \cdots \times [b+1] \times [1]}_{2\ell+1 \text{ components}}.$$

Let v be a vertex with priority $2\ell - 1$ in the odd level of $H_{\ell,b}$, and let w be a vertex, such that there is an edge from v to w in the game graph of $H_{\ell,b}$. Then there is also an edge from w to v in the game graph of $H_{\ell,b}$; see Figure 4.1. Therefore, function μ maintained by the algorithm satisfies $\mu(w) \leq \mu(v)$, because w is a diamond vertex with even priority, so $\mathrm{Prog}(\mu, w, v) =_{p(w)} \mu(v)$, and $\big(\mathrm{Prog}(\mu, w, v)\big)_i = 0$ for all $i > p(w)$. It follows that $\mathrm{Lift}(\cdot, v)$ operator can only lift $\mu(v)$ to the immediate successor of $\mu(v)$ in the order on $M_{H_{\ell,b}}$, because the priority of v is $2\ell - 1$. [Claim 4.13] ■

Theorem 4.12 follows from the above claim by taking the game $H_{\lfloor d/2 \rfloor, \lceil n/d \rceil}$.
[Theorem 4.12] ■

4.6 Optimizations

Even though procedure `ProgressMeasureLifting` as presented above admits the worst-case performance, there is some room for improvements in its running time. Let us just mention here two proposals for optimizations, which should be considered when implementing the algorithm.

One way is to get better upper bounds on the values of the least game parity progress measure than the one provided by Corollary 4.8, taking into account the structure of the game graph. This would allow to further reduce the "search space" where the algorithm is looking for game progress measures. For example, let $G^{\geq i}$ be the parity graph obtained from the game graph G by removing all vertices with priorities smaller than i. One can show that if $v \in \|\mu\|$ for the least game progress measure μ then for odd i's the i-th component of $\mu(v)$ is bounded by the number of vertices of priority i reachable from v in graph $G^{\geq i}$. It requires further study to see whether one can get considerable improvements by pre-computing better bounds for the values of the least game parity progress measure.

Another simple but important optmization is to decompose game graphs into maximal strongly connected components. Note that every infinite play eventually stays within a strongly connected component, so it suffices to apply expensive procedure for solving parity games to the maximal strongly connected components separately. In fact, we need to proceed bottom up in the partial order of maximal strongly connected components. Each time one of the bottom components has been solved, we can also remove from the rest of the game the sets of vertices from which respective players have a strategy to force in a finite number of moves to their so far computed winning sets.

The above optimizations should considerably improve performance of our algorithm in practice, but they do not, as such, give any asymptotic worst-case improvement: see the examples $H_{\ell,b}$ from Section 4.5.

Chapter 5

Discrete strategy improvement for PG's

This chapter contains a revised and extended version of [VJ00b]. It is joint work with Jens Vöge. Other accounts of these results can be found in the RTWH technical report [VJ00a] and in Jens Vöge's Ph.D. dissertation [Vög00].

Abstract. A discrete strategy improvement algorithm is given for constructing winning strategies in parity games, thereby providing also a new solution of the model-checking problem for the modal μ-calculus. Known strategy improvement algorithms, as proposed for stochastic games by Hoffman and Karp in 1966, and for discounted payoff games and parity games by Puri in 1995, work with real numbers and require solving linear programming instances involving high precision arithmetic. In the present algorithm for parity games these difficulties are avoided by the use of discrete vertex valuations in which information about the relevance of vertices and certain distances is coded. An efficient implementation is given for a strategy improvement step. Another advantage of the present approach is that it provides a better conceptual understanding and easier analysis of strategy improvement algorithms for parity games. However, so far it is not known whether the present algorithm works in polynomial time. The long standing problem whether parity games can be solved in polynomial time remains open.

5.1 Introduction

The study of the computational complexity of solving parity games has two main motivations. One is that the problem is polynomial time equivalent to the modal μ-calculus model checking [EJS93, Sti95], and hence better algorithms for parity games may lead to better model checkers, which is a major objective in computer aided verification.

The other motivation is the intriguing status of the problem from the point of view of structural complexity theory. It is one of the few natural problems which is in **NP** $\cap$ **co-NP** [EJS93] (and even in **UP** $\cap$ **co-UP** [Jur98]), and is not known to have a polynomial time algorithm, despite substantial effort of the community (see [EJS93, BCJ$^+$97, Sei96, Jur00] and references there). Other notable examples of such problems include simple stochastic games [Con92, Con93], mean payoff games [EM79, ZP96], and discounted payoff games [ZP96]. There are polynomial time reductions of parity to mean payoff games [Pur95, Jur98], mean payoff to discounted payoff games [ZP96], and discounted payoff to simple stochastic games [ZP96]. Parity games, as the simplest of them all, seem to be the most plausible candidate for trying to find a polynomial time algorithm.

A strategy improvement algorithm has been proposed for solving stochastic games by Hoffman and Karp [HK66] in 1966. Puri in his PhD thesis [Pur95] has adapted the algorithm for discounted payoff games. Puri also provided a polynomial time reduction of parity games to mean payoff games, and advocated the use of the algorithm for solving parity games, and hence for the modal μ-calculus model checking.

In our opinion Puri's strategy improvement algorithm for parity games has two drawbacks.

- The algorithm uses high precision arithmetic, and needs to solve linear programming instances: both are typically costly operations. An implementation (by the first author) of Puri's algorithm, using a linear programming algorithm of Meggido [Meg83], proved to be prohibitively slow.

- Solving parity games is a discrete, graph theoretic problem, but the crux of the algorithm is manipulation of real numbers, and its analysis is crucially based on continuous methods, such as Banach's fixed point theorem.

The first one makes the algorithm inefficient in practice, the other one obscures understanding of the algorithm.

Our discrete strategy improvement algorithm remedies both shortcomings of Puri's algorithm mentioned above, while preserving the overall structure of the generic strategy improvement algorithm. We introduce discrete values (such as tuples of: vertices, sets of vertices and natural numbers denoting lengths of paths in the game graph) which are being manipulated by the algorithm, instead of their encodings into real numbers. (One can show a precise relationship between behaviour of Puri's and our algorithms; we will treat this issue elsewhere.)

The first advantage of our approach is that we avoid solving linear programming instances involving high precision arithmetic. Instead, a shortest paths instance needs to be solved in every strategy improvement step of the algorithm. The shortest paths instances occurring in this context have discrete weights recording relevance of vertices and distances in the game graph. We develop an algorithm exploiting the special structure of these instances instead of using standard shortest paths algorithms. Our algorithm gives an efficient implementation of a single improvement step of the strategy improvement algorithm. Its running time is $O(n \cdot m)$, where n is the number of vertices, and m is the number of edges in the game graph. In comparison, a naive application of Bellman-Ford algorithm [CLR90] gives $O(n^2 \cdot m)$ running time.

The other advantage is more subjective: we believe that it is easier to analyze the discrete data maintained by our algorithm, rather than its subtle encodings into real numbers involving infinite geometric series [Pur95]. The classical continuous reasoning involving Banach fixed point theorem gives an elegant proof of correctness of the algorithm in a more general case of discounted payoff games [Pur95], but we think that in the case of parity games it blurs an intuitive understanding of the underlying discrete structure. However, the long standing open question whether a strategy improvement algorithm works in polynomial time [Con93] remains unanswered. Nevertheless, we hope that our discrete analysis of the algorithm may help either to find a proof of polynomial time termination, or to come up with a family of examples on which the algorithm requires exponential number of steps. Any of those results would mark a substantial progress in understanding the computational complexity of parity games.

So far, for all families of examples we have considered the strategy improvement algorithm needs only linear number of strategy improvement steps. Notably, a linear number of strategy improvements suffices for several families of difficult examples for which other known algorithms need exponential time.

The rest of this chapter is organized as follows. In section 5.2 we define the infinite parity games and we establish their equivalence to finite cycle-domination games. In section 5.3 we sketch the idea of a generic strategy improvement algorithm and we state general postulates which guarantee correctness of the algorithm. Then in section 5.4 we give a specific proposal for the ingredients of a generic strategy improvement algorithm. In section 5.5 we prove that these ingredients satisfy the postulates of section 5.3. In this way we get a purely discrete strategy improvement algorithm for solving parity games. In section 5.6 we give a specialized shortest paths algorithm for the shortest paths instances occurring in our strategy improvement algorithm. In this way we obtain an efficient $O(n \cdot m)$ implementation of a strategy improvement step, where n is the number of vertices, and m is the number of edges in the game graph. Finally, in section 5.7 we discuss partial results and open questions concerning the time complexity of strategy improvement algorithms.

5.2 Parity games

5.2.1 Infinite parity games

A *game graph* $G = \big(V, E, (M_{\text{Even}}, M_{\text{Odd}}), p\big)$ of a parity game consists of a directed graph (V, E), a partition $(M_{\text{Even}}, M_{\text{Odd}})$ of the set of vertices V, and a *priority function* $p : V \to \{0, 1, \ldots, k\}$ for some $k \in \mathbb{N}$. We restrict ourselves to finite parity game graphs and for technical convenience we assume that every vertex has at least one out-going edge.

An *infinite parity game* G^∞ is a two-person infinite duration path-forming game played on graph G. The two players, Even and Odd, keep moving a token along edges of the game graph: player Even moves the token from vertices in M_{Even}, and player Odd moves the token from vertices in M_{Odd}. A *play* of G^∞ is an infinite path $\langle v_0, v_1, v_2, \ldots \rangle$ in the game graph G arising in this way. A play $P = \langle v_0, v_1, v_2, \ldots \rangle$ is *winning* for player Even if the biggest priority occurring infinitely often in P (i.e., the biggest number occurring infinitely often in $\langle p(v_0), p(v_1), p(v_2), \ldots \rangle$) is even; otherwise P is winning for player Odd.

A *memoryless strategy* for player Even is a function $\sigma : M_{\text{Even}} \to V$ such that $\big(v, \sigma(v)\big) \in E$ for all $v \in M_{\text{Even}}$. (We consider only memoryless strategies here so for brevity we just write strategies to denote memoryless strategies throughout the paper.) A play $\langle v_0, v_1, v_2, \ldots \rangle$ is *consistent* with a strategy σ for player Even if $v_{\ell+1} = \sigma(v_\ell)$ for all $\ell \in \mathbb{N}$, such that $v_\ell \in M_{\text{Even}}$. Strategies for player Odd are defined analogously. If σ is a strategy for player Even (Odd) then we write G_σ to denote the game graph obtained from G by removing all the edges (v, w), such that $v \in M_{\text{Even}}$ ($v \in M_{\text{Odd}}$) and $\sigma(v) \neq w$; we write E_σ for the set of edges of G_σ. If σ is a strategy for player Even and τ is a strategy for player Odd then we write $G_{\sigma\tau}$ for $(G_\sigma)_\tau$; we write $E_{\sigma\tau}$ for its set of edges.

Note that if σ is a strategy for player Even and τ is a strategy for player Odd then for every vertex v, there is a unique play $P_{\sigma\tau}(v)$ starting from v and consistent with both σ and τ. We say that a strategy σ for player Even is a *winning strategy* from a vertex v if for every strategy τ for player Odd, the unique play $P_{\sigma\tau}(v)$ starting from v and consistent with both σ and τ is winning for player Even. A strategy σ is a winning strategy from a set of vertices W if it is winning from every vertex in W. Winning strategies for player Odd are defined analogously.

Remark. In literature on parity games (see for example [Tho95, Zie98, Jur98]) a different definition of a winning strategy is used; here we call it "a strategy winning against arbitrary strategies." We say that a strategy σ for player Even (Odd) is a *strategy winning against arbitrary strategies* from a vertex v if every play starting from v and consistent with σ is winning for player Even (Odd). We argue that the two definitions are equivalent for parity games.

Proposition 5.1 A strategy σ for player Even (Odd) is a winning strategy from a vertex v if and only if σ is a strategy winning against arbitrary strategies from v.

Proof. The "if" part is obvious.

We prove the "only if" part for player Even; the proof for player Odd is analogous. Suppose that σ is a winning strategy for player Even from v. Then the biggest priority on every cycle in G_σ reachable from v in G_σ is even. Let P be an infinite play consistent with σ. We argue that the biggest priority occurring infinitely often on P is even. Using a simple stacking technique (see for example [ZP96], page 347, or [Jur98], page 122) we can decompose play P into a finite path and an infinite number of simple cycles in G_σ. (The decomposition may be different if the game graph is infinite, but the proposition can be proved in a very similar way also for parity games with infinite game graphs.) Note that biggest priorities in all simple cycles in the decomposition are even because all the cycles are cycles in G_σ reachable from v in G_σ. Hence the maximal priority occurring infinitely often in P is even, i.e., P is a winning play for player Even. [Proposition 5.1] ■ [Remark] □

Theorem 5.2 (Memoryless Determinacy [EJ91, Mos91])
For every parity game graph G, there is a unique partition $(W_{\text{Even}}, W_{\text{Odd}})$ of the set of vertices of G, such that there is a winning strategy for player Even from W_{Even} in G^∞ and a winning strategy for player Odd from W_{Odd} in G^∞.

We call sets W_{Even} and W_{Odd} the *winning sets* of player Even and player Odd, respectively. The problem of *deciding the winner in parity games* is, given a parity game graph and a vertex of the graph, to determine whose winning set the vertex belongs to. By *solving* a parity game we mean finding the partition $(W_{\text{Even}}, W_{\text{Odd}})$.

5.2.2 Finite cycle-domination games

For the purpose of development and reasoning about our discrete strategy improvement algorithm for solving parity games it is technically convenient to reformulate a bit the standard definition of parity games outlined above. Below we define finite cycle-domination games and we give a simple translation of every parity game into an equivalent cycle-domination game.

A game graph $G = \big(V, E, (M_\oplus, M_\ominus), (R_+, R_-)\big)$ of a cycle-domination game consists of a directed graph (V, E), where $V = \{1, 2, \ldots, n\}$ for some $n \in \mathbb{N}$, and of two partitions (R_+, R_-) and $(M_\oplus, M_\ominus)$ of the set of vertices V. We sometimes refer to the numbers identifying vertices as their *priorities*, and we use the standard $\leq$ order on natural numbers to compare them.

A *finite cycle-domination game G^ω* is a two-person finite path-forming game played on graph G. The two players, player $\oplus$ and player $\ominus$, play similarly to players Even and Odd in infinite parity games by moving a token along edges of the game graph G: player $\oplus$ moves the token from vertices in $M_\oplus$, and player $\ominus$ moves the token from vertices in $M_\ominus$. The difference from infinite parity games is that a play is finished as soon as the path constructed so far by the players contains a cycle. In other words, a play in G^ω is a finite path $P = \langle v_0, v_1, \ldots, v_\ell\rangle$ in the game graph G, such that $v_i \neq v_j$ for all $1 \leq i < j < \ell$, and there is a $k < \ell$, such that $v_k = v_\ell$. It follows that $\langle v_k, v_{k+1}, \ldots, v_\ell\rangle$ is the only cycle in play P; we write $\mathtt{Cycle}(P)$ to denote this unique cycle, and we call it the cycle of P. We define the *cycle dominating value* $\lambda(P)$ of play P to be $\max^{\leq}\big(\mathtt{Cycle}(P)\big) = \max^{\leq}\{v_k, v_{k+1}, \ldots, v_\ell\}$, i.e., the vertex with biggest priority in the cycle of P. We say that a play P in G^ω is winning for player $\oplus$ if

$\lambda(P) \in R_+$, otherwise $\lambda(P) \in R_-$ and play P is winning for player $\ominus$. Strategies and winning strategies for both players in G^ω are defined similarly as in infinite parity games.

Given a parity game graph $G = \big(V, E, p, (M_{\text{Even}}, M_{\text{Odd}})\big)$ we construct a cycle-domination game graph $\overline{G} = \big(V, E, (R_+, R_-), (M_\oplus, M_\ominus)\big)$, such that the games G^∞ and $\overline{G}^\omega$ are equivalent in the following sense.

Proposition 5.3 A strategy σ is a winning strategy for player Even (player Odd) from a vertex v in G^∞ if and only if σ is a winning strategy for player $\oplus$ (player $\ominus$) from v in $\overline{G}^\omega$.

Construction of $\overline{G}$: Let $M_\oplus = M_{\text{Even}}$, $M_\ominus = M_{\text{Odd}}$, and

$$R_+ = \big\{\, v \in V \ :\ p(v) \text{ is even} \,\big\}, \text{ and } R_- = \big\{\, v \in V \ :\ p(v) \text{ is odd} \,\big\}.$$

We introduce a total order relation $\leq$ on V called the *relevance* order. Let the relevance order $\leq$ be an arbitrary total order extending the pre-order induced by priorities, i.e., such that $p(v) \leq p(w)$ implies $v \leq w$ for all $v, w \in V$. Note that we use the same symbol "$\leq$" for the standard order on natural numbers and for the relevance order; in fact we identify vertices in V with integers in the set $\{1, 2, \ldots, |V|\}$ via the unique order-isomorphism between the total orders $(V, \leq)$ and $\big(\{1, 2, \ldots, |V|\}, \leq\big)$.

Proof (of Proposition 5.3). The first condition, i.e., that σ is a winning strategy for player Even (player Odd) in G^∞ is equivalent to saying that for every cycle in G_σ reachable from v in G_σ, the biggest priority occurring on the cycle is even (odd). The other condition, i.e., that σ is a winning strategy for player $\oplus$ (player $\ominus$) in $\overline{G}^\omega$ is equivalent to saying that for every cycle in $\overline{G}_\sigma$, the most relevant vertex in the cycle belongs to R_+ (R_-). It follows immediately from the construction of $\overline{G}$ that the biggest priority occurring on a cycle in G_σ is even (odd), if and only if the most relevant vertex in the same cycle in $\overline{G}_\sigma$ belongs to R_+ (R_-), and so we are done. [Proposition 5.3] ■

5.3 Generic strategy improvement algorithm

In order to develop a strategy improvement algorithm for cycle-domination games we define the problem of solving cycle-domination games as an "optimization problem." Suppose we have a pre-order $\sqsubseteq$ on the set $\texttt{Strategies}_\oplus \subseteq (M_\oplus \to V)$ of strategies for player $\oplus$, satisfying the following two postulates.

P1. There is a maximum element in the pre-order $(\texttt{Strategies}_\oplus, \sqsubseteq)$, i.e., there is a strategy $\sigma \in \texttt{Strategies}_\oplus$, such that $\kappa \sqsubseteq \sigma$ for all $\kappa \in \texttt{Strategies}_\oplus$.

P2. If σ is a maximum element in the pre-order $(\texttt{Strategies}_\oplus, \sqsubseteq)$ then σ is a winning strategy for player $\oplus$ from every vertex of her winning set.

The problem of *solving a cycle-domination game with respect to* $\sqsubseteq$ is, given a cycle-domination game, to find a strategy for player $\oplus$ which is a maximum element in the pre-order $(\texttt{Strategies}_{\oplus}, \sqsubseteq)$.

Suppose we also have an operator $\texttt{Improve} : \texttt{Strategies}_{\oplus} \to \texttt{Strategies}_{\oplus}$, satisfying the following two postulates:

I1. If σ is not a maximum element in the pre-order $(\texttt{Strategies}_{\oplus}, \sqsubseteq)$ then $\sigma \sqsubseteq \texttt{Improve}(\sigma)$ and $\texttt{Improve}(\sigma) \not\sqsubseteq \sigma$.

I2. If σ is a maximum element in the pre-order $(\texttt{Strategies}_{\oplus}, \sqsubseteq)$ then we have $\texttt{Improve}(\sigma) = \sigma$.

A generic strategy improvement algorithm is the following procedure.

Strategy improvement algorithm
`pick a strategy` σ `for player` $\oplus$
while $\sigma \neq \texttt{Improve}(\sigma)$ **do** $\sigma := \texttt{Improve}(\sigma)$

Note that postulate I1. guarantees that the algorithm terminates, because there are only finitely many strategies for player $\oplus$. Postulate I2. implies that when the algorithm terminates then the strategy σ is a maximum element in the pre-order $(\texttt{Strategies}_{\oplus}, \sqsubseteq)$. Altogether, we get the following.

Theorem 5.4
If a pre-order $(\texttt{Strategies}_{\oplus}, \sqsubseteq)$ *satisfies postulates P1. and P2., and an* $\texttt{Improve}$ *operator satisfies postulates I1. and I2. then strategy improvement algorithm is a correct algorithm for solving cycle-domination games with respect to* $\sqsubseteq$.

5.4 Discrete strategy improvement algorithm

In what follows we give a particular proposal for a pre-order $(\texttt{Strategies}_{\oplus}, \sqsubseteq)$ satisfying postulates P1. and P2., and an $\texttt{Improve}$ operator satisfying postulates I1. and I2. These definitions are based on discrete valuations assigned to strategies and hence give rise to a purely discrete strategy improvement algorithm for solving cycle-domination games.

5.4.1 Play values

For every $w \in V$, we define $V_{>w} = \{ v \in V \ : \ v > w \}$, and $V_{<w} = \{ v \in V \ : \ v < w \}$.

Let $P = \langle v_1, v_2, \ldots, v_\ell \rangle$ be a play, and let $v_k = \lambda(P)$, i.e., the cycle value of P is the k-th element of P. Let $\texttt{Prefix}(P) = \{v_1, v_2, \ldots, v_{k-1}\}$, i.e., $\texttt{Prefix}(P)$ is the set of vertices in play P occurring before the loop value of P. We define the *primary path value* $\pi(P)$ of play P to be $\texttt{Prefix}(P) \cap V_{>\lambda(P)}$, i.e., the set of vertices occurring in P with priorities bigger than $\lambda(P)$. We define the *secondary path value* $\#(P)$ of play P to be $|\texttt{Prefix}(P)|$, i.e., the length of the path from the initial vertex of P to $\lambda(P)$. The *path value* of play P is defined to be the ordered pair $\big(\pi(P), \#(P)\big) \in \wp(V) \times \mathbb{N}$. The *play value* $\Theta(P)$ of play P is defined to be the ordered pair $\Big(\lambda(P), \big(\pi(P), \#(P)\big)\Big) \in V \times \big(\wp(V) \times \mathbb{N}\big)$. We write `PlayValues` to denote the image of the function $\Theta : \texttt{Plays} \to V \times \big(\wp(V) \times \mathbb{N}\big)$. Note that

$$\texttt{PlayValues} \subseteq \left\{ \big(v, (B, k)\big) \ : \ B \subseteq V_{>v} \right\}. \tag{5.1}$$

5.4.2 Linear order $\trianglelefteq$ on `PlayValues`

For all $v \in V$, we define

$$\texttt{Reward}(v) = \begin{cases} -v & \text{if } v \in R_-, \\ v & \text{if } v \in R_+. \end{cases}$$

For all $v, w \in V$, we define $v \preceq w$ to hold if and only if $\texttt{Reward}(v) \leq \texttt{Reward}(w)$. We write $v \prec w$ if $v \preceq w$ and $v \neq w$. Note that $\preceq$ is a linear order on V, and $m \prec p$ for all $m \in R_-$ and $p \in R_+$.

For $B, C \in \wp(V)$, such that $B \neq C$, we define

$$\texttt{MaxDiff}(B, C) = \max^{\leq}\big((B \setminus C) \cup (C \setminus B)\big).$$

We define $B \prec C$ to hold if and only if either

- $\texttt{MaxDiff}(B, C) \in (B \setminus C) \cap R_-$, or
- $\texttt{MaxDiff}(B, C) \in (C \setminus B) \cap R_+$.

We write $B \preceq C$ if $B \prec C$ or $B = C$. For $\ell \in V$, we write $B \preceq_\ell C$ if $B \cap V_{>\ell} \preceq C \cap V_{>\ell}$. We write $B =_\ell C$ if $B \preceq_\ell C$ and $C \preceq_\ell B$. We write $B \prec_\ell C$ if $B \preceq_\ell C$ and $C \not\preceq_\ell B$.

For all $\ell \in V$, and $(B, k), (B', k') \in \wp(V) \times \mathbb{N}$, we define $(B, k) \trianglelefteq_\ell (B', k')$ to hold if and only if either

- $B \prec_\ell B'$, or
- $B =_\ell B'$ and either
  - $\ell \in R_-$ and $k \leq k'$, or
  - $\ell \in R_+$ and $k' \leq k$.

We write $(B, k) \lhd_\ell (B', k')$ if $(B, k) \trianglelefteq_\ell (B', k')$ and $(B', k') \not\trianglelefteq_\ell (B, k)$.

For all $\big(\ell, (B, k)\big), \big(\ell', (B', k')\big) \in V \times \big(\wp(V) \times \mathbb{N}\big)$, we define $\big(\ell, (B, k)\big) \trianglelefteq \big(\ell', (B', k')\big)$ to hold if and only if either

- $\ell \prec \ell'$, or
- $\ell = \ell'$ and $(B, k) \trianglelefteq_\ell (B', k')$.

For $\xi, \xi' \in V \times \big(\wp(V) \times \mathbb{N}\big)$, we write $\xi \lhd \xi'$ if $\xi \unlhd \xi'$ and $\xi' \not\unlhd \xi$.

Note that (5.1) guarantees that $\unlhd$ is a linear order on `PlayValues`. It is a linear pre-order on $V \times \big(\wp(V) \times \mathbb{N}\big)$.

Example. Note that $\xi_1 = \big(3, \{1\}, 2\big)$ and $\xi_2 = (3, \{2\}, 2)$ are not play values, because $\{1\} \not\subseteq V_{>3}$ and $\{2\} \not\subseteq V_{>3}$, respectively. We have $\xi_1 \unlhd \xi_2$ and $\xi_2 \unlhd \xi_1$, even though $\xi_1 \neq \xi_2$. [Example] □

5.4.3 Pre-order $\sqsubseteq$ on `Strategies`$_\oplus$

We write `Valuations` to denote the set ($V \to$ `PlayValues`) of functions assigning a play value to every vertex of the game graph. We extend the $\unlhd$ order to a partial order on `Valuations` by defining it point-wise, i.e., for $\Xi, \Xi' \in$ `Valuations` we define $\Xi \unlhd \Xi'$ to hold if $\Xi(v) \unlhd \Xi'(v)$ for all $v \in V$. We write $\Xi \lhd \Xi'$ if $\Xi \unlhd \Xi'$ and $\Xi \neq \Xi'$.

Valuation Ω_σ. For every strategy $\sigma \in$ `Strategies`$_\oplus$ we define $\Omega_\sigma \in$ `Valuations` in the following way:

$$\Omega_\sigma(v) \;=\; \min{}^{\unlhd}\big\{\, \Theta(P) \;:\; P \in \mathtt{Plays}_\sigma(v) \,\big\},$$

where $\mathtt{Plays}_\sigma(v)$ is the set of plays starting from vertex v and consistent with σ. Finally, for $\sigma, \sigma' \in$ `Strategies`$_\oplus$ we define $\sigma \sqsubseteq \sigma'$ to hold if and only if $\Omega_\sigma \unlhd \Omega_{\sigma'}$. We write $\sigma \sqsubset \sigma'$ if $\Omega_\sigma \lhd \Omega_{\sigma'}$.

5.4.4 An `Improve` operator

We say that a set $I \subseteq E$ is unambiguous if $(v, w) \in I$ and $(v, u) \in I$ imply that $w = u$. For every unambiguous set of edges I, we define an operator $\mathtt{Switch}_I : \mathtt{Strategies}_\oplus \to \mathtt{Strategies}_\oplus$ as follows:

$$\big[\mathtt{Switch}_I(\sigma)\big](v) = \begin{cases} w & \text{if } (v, w) \in I \text{ for some } w \in V, \\ \sigma(v) & \text{otherwise.} \end{cases}$$

We say that an edge $(v, w) \in E$ is an *improvement* for a strategy σ if

$$\Omega_\sigma\big(\sigma(v)\big) \lhd \Omega_\sigma(w). \tag{5.2}$$

We define a (non-deterministic) operator $\mathtt{Improve} : \mathtt{Strategies}_\oplus \to \mathtt{Strategies}_\oplus$ as follows:

$$\mathtt{Improve}(\sigma) = \begin{cases} \mathtt{Switch}_I(\sigma) & \text{for some set } I \neq \emptyset \text{ of improvements for } \sigma, \\ \sigma & \text{if there are no improvements for } \sigma. \end{cases}$$

Let $I_1, I_2, \ldots, I_t \subseteq E$, and let σ be a strategy for player $\oplus$. Define $\sigma_0 = \sigma$, and $\sigma_k = \mathtt{Switch}_{I_k}(\sigma_{k-1})$ for $k > 0$. We say that $P = \langle I_1, I_2, \ldots, I_t \rangle$ is a *strategy improvement policy* for σ if:

- for all $k \leq t$, we have that $I_k \neq \emptyset$ is an unambiguous set of improvements for σ_{k-1}, and
- there are no improvements for σ_t.

5.4.5 A few technical definitions

Pre-orders $\trianglelefteq_\ell$ on $\mathcal{M}(V) \times \mathbb{N}$. For technical reasons, for every $\ell \in V$, we extend the pre-order $\trianglelefteq_\ell$ on $\wp(V) \times \mathbb{N}$ to $\mathcal{M}(V) \times \mathbb{N}$, where $\mathcal{M}(V)$ is the set of multi-sets of elements of V. The definitions of $\preceq_\ell$, and $\trianglelefteq_\ell$ are the same as for $\wp(V) \times \mathbb{N}$; the only difference is that `MaxDiff` for $B, C \in \mathcal{M}(V)$ is defined by

$$\texttt{MaxDiff}(B, C) = \max^{\leq}\big((B \setminus_{\mathcal{M}} C) \cup (C \setminus_{\mathcal{M}} B)\big),$$

where $\setminus_{\mathcal{M}}$ is the multiset difference, and $B \prec C$ is defined to holds if and only if either

- $\texttt{MaxDiff}(B, C) \in (B \setminus_{\mathcal{M}} C) \cap R_-$, or
- $\texttt{MaxDiff}(B, C) \in (C \setminus_{\mathcal{M}} B) \cap R_+$.

Example. If $3 \in R_-$ then we have $\{1, 3, 3\} \preceq \{1, 2, 3\}$ because

$$\texttt{MaxDiff}\big(\{1, 3, 3\}, \{1, 2, 3\}\big) = 3$$

and $3 \in \big(\{1, 3, 3\} \setminus_{\mathcal{M}} \{1, 2, 3\}\big) \cap R_-$. [Example] □

Pre-orders $\widetilde{\trianglelefteq}_\ell$ on $\mathcal{M}(V) \times \mathbb{N}$. We define the following weakening of the $\trianglelefteq_\ell$ pre-order. For $\ell \in V$ and $(B, k), (B', k') \in \mathcal{M}(V) \times \mathbb{N}$, we define $(B, k) \widetilde{\trianglelefteq}_\ell (B', k')$ to hold if and only if $B \preceq_\ell B'$. We write $(B, k) \cong_\ell (B', k')$ if $(B, k) \widetilde{\trianglelefteq}_\ell (B', k')$ and $(B', k') \widetilde{\trianglelefteq}_\ell (B, k)$. Note that $\widetilde{\trianglelefteq}_\ell$ is a pre-order on $\mathcal{M}(V) \times \mathbb{N}$, and that $\widetilde{\trianglelefteq}_\ell$ is coarser than $\trianglelefteq_\ell$, i.e., $(B, k) \trianglelefteq_\ell (B', k')$ implies $(B, k) \widetilde{\trianglelefteq}_\ell (B', k')$. Note, for example, that if $2 \in R_+$ then we have $\big(\{2, 3\}, 1\big) \cong_2 \big(\{1, 3\}, 2\big)$, but $\big(\{2, 3\}, 1\big) \not\trianglelefteq_2 \big(\{1, 3\}, 2\big)$.

Operation $\boxplus$ on $\mathcal{M}(V) \times \mathbb{N}$. For $(B, k), (B', k') \in \mathcal{M}(V) \times \mathbb{N}$, we define $(B, k) \boxplus (B', k') = \big(B \cup_{\mathcal{M}} B', k + k'\big)$, where $\cup_{\mathcal{M}}$ is the multi-set union. We also use the following shorthand: if $C \in \wp(V)$ then $(B, k) \boxplus C$ is defined to be equal to $(B, k) \boxplus \big(C, |C|\big)$, i.e., we have that $(B, k) \boxplus C = \big(B \cup_{\mathcal{M}} C, k + |C|\big)$. Moreover, if $v \in V$ then we often write $(B, k) \boxplus v$ instead of $(B, k) \boxplus \{v\}$. In other words, we have that $(B, k) \boxplus v = (B \cup_{\mathcal{M}} \{v\}, k + 1)$.

5.5 Correctness of the algorithm

In order to prove correctness of our discrete strategy improvement algorithm, i.e., to establish Theorem 5.4, it suffices to argue that the definitions of subsection 5.4 satisfy postulates P1., P2., I1., and I2.

5.5.1 Valuation Ω_σ and shortest paths

Let $\Omega_\sigma = (\Lambda_\sigma, \Pi_\sigma) : V \to V \times \big(\wp(V) \times \mathbb{N}\big)$. We establish the following characterization of Ω_σ.

1. Let $T_\sigma = \{ t \in V \ : \ \text{there is a cycle } C \text{ in } G_\sigma, \text{ such that } \max^{\leq}(C) = t \}$. Then for all $v \in V$, we have that
$$\Lambda_\sigma(v) = \min\nolimits^{\preceq}\{ t \in T_\sigma \ : \ \text{there is a path from } v \text{ to } t \text{ in } G_\sigma \}. \tag{5.3}$$

2. For $v, t \in V$, we define $\mathtt{Path}_\sigma(v,t)$ to be the set of sets of vertices $S \in \wp(V)$, such that $S \cup \{t\}$ is the set of vertices occurring on a (simple) path from v to t in G_σ. Let $v \in V$ and let $\ell = \Lambda_\sigma(v)$. Then we have
$$\Pi_\sigma(v) = \min\nolimits^{\unlhd_\ell}\Big\{ \big(S \cap V_{>\ell}, |S|\big) \ : \ S \in \mathtt{Path}_\sigma(v,\ell) \Big\}. \tag{5.4}$$

Clause 1. is straightforward. Note that for all $(v,w) \in E_\sigma$ we have
$$\Lambda_\sigma(v) \preceq \Lambda_\sigma(w). \tag{5.5}$$

We argue that clause 2. holds. Let $P \in \mathtt{Plays}_\sigma(v)$, such that $\Theta(P) = \Omega_\sigma(v)$. Let $R = \mathtt{Prefix}(P)$ and let $C = \mathtt{Cycle}(P)$. Then $\Lambda_\sigma(v) = \lambda(P)$ and
$$\Pi_\sigma(v) \ = \ \big(\pi(P), \#(P)\big) \ = \ \big((R \cup C) \cap V_{>\ell}, |R|\big),$$
i.e., $\Pi_\sigma(v) = \big(R \cap V_{>\ell}, |R|\big)$; the last equality holds because $\max^{\leq}(C) = \lambda(P)$. Therefore, we have that
$$\begin{aligned} \Pi_\sigma(v) \ &= \ \min\nolimits^{\unlhd_\ell}\Big\{ \big(\pi(P), \#(P)\big) \ : \ P \in \mathtt{Plays}_\sigma(v) \text{ and } \lambda(P) = \ell \Big\} \\ &= \ \min\nolimits^{\unlhd_\ell}\Big\{ \big(S \cap V_{>\ell}, |S|\big) \ : \ S \in \mathtt{Path}_\sigma(v,\ell) \Big\}. \end{aligned}$$

As a result of the above we get a characterization of $\Pi_\sigma(v)$ as a solution of the following shortest paths problem instance. For $\ell \in T_\sigma$ consider the subgraph $G^\ell_\sigma = (V^\ell, E^\ell_\sigma)$ of G_σ induced by $V^\ell = \{ v \in V \ : \ \Lambda_\sigma(v) = \ell \}$. With every edge $(v,w) \in E^\ell_\sigma$ we associate the weight $\big(\{v\} \cap V_{>\ell}, 1\big) \in \mathcal{M}(V) \times \mathbb{N}$. The set of weights of paths in G^ℓ_σ is $\mathcal{M}(V) \times \mathbb{N}$ with $\boxplus$ as the operation of adding weights. Shortest paths are taken with respect to the $\unlhd_\ell$ pre-order on $\mathcal{M}(V) \times \mathbb{N}$.

Remark. Section 26.4 of the book by Cormen et al. [CLR90] describes an algebraic structure called a closed semi-ring that allows to define a shortest paths problem and devise algorithms for finding shortest paths. We leave it as an exercise to the reader to verify that for every $\ell \in V$, the set $\mathcal{M}(V) \times \mathbb{N}$ with the *extension operator* $\boxplus$ and the *summation operator* $\max^{\unlhd_\ell}$ forms a closed semi-ring. [Remark] □

Note that from $\Lambda_\sigma(v) = \ell$ for all $v \in V^\ell$, it follows that all cycles in G^ℓ_σ have "non-negative" weight, i.e., they have weight $\unlhd_\ell$-bigger than $(\emptyset, 0)$ if $\ell \in R_-$, and $\trianglelefteq_\ell$-bigger than $(\emptyset, +\infty)$ if $\ell \in R_+$. Therefore, shortest paths to ℓ exist from each vertex $v \in V^\ell$, they are simple paths, and $\Pi_\sigma(v)$ is the weight of the shortest path from v to ℓ in G^ℓ_σ.

Observe, that for $v \neq \ell$, the characterization of $\Pi_\sigma(v)$ as the weight of a shortest path from v to ℓ implies, that if $w \in V^\ell$ is a successor of v on a shortest path from v to ℓ in G^ℓ_σ then

$$\Pi_\sigma(v) \;=_\ell \Pi_\sigma(w) \;\boxplus\; v, \tag{5.6}$$

and for all $(v, u) \in E^\ell_\sigma$ we have

$$\Pi_\sigma(v) \;\trianglelefteq_\ell \Pi_\sigma(u) \;\boxplus\; v. \tag{5.7}$$

Note that we have

$$\Pi_\sigma(\ell) = (\emptyset, 0). \tag{5.8}$$

Moreover, observe that by definition of $\ell \in T_\sigma$, there is a cycle C in G^ℓ_σ, such that $\max^{\leq}(C) = \ell$, and there are no cycles with "negative" weight. Therefore, if w is the successor of ℓ with a shortest path to ℓ then $\Pi_\sigma(w) = (\emptyset, k)$ for some $k \in \mathbb{N}$, and hence we have

$$\Pi_\sigma(\ell) \;\cong_\ell \Pi_\sigma(w) \;\boxplus\; \ell, \tag{5.9}$$

and for all $(\ell, u) \in E^\ell_\sigma$ we have

$$\Pi_\sigma(\ell) \;\tilde{\trianglelefteq}_\ell \Pi_\sigma(u) \;\boxplus\; \ell. \tag{5.10}$$

5.5.2 Locally progressive valuations

Strategy $\overline{\sigma}$ for player $\ominus$. Let σ be a strategy for player $\oplus$. We define a strategy $\overline{\sigma}$ for player $\ominus$ in the following way. For every $v \in M_\ominus$, we set:

$$\overline{\sigma}(v) = w, \text{ such that } \Omega_\sigma(w) \trianglelefteq \Omega_\sigma(u) \text{ for all } u \in \mathrm{succ}(v),$$

i.e., $\overline{\sigma}(v)$ is defined to be a successor of v which minimizes the value of Ω_σ with respect to $\trianglelefteq$. Note that from the characterization of Ω_σ from the previous subsection it follows that for all $(v, w) \in E_{\sigma\overline{\sigma}}$, we have $\Lambda_\sigma(v) = \Lambda_\sigma(w)$. Moreover, claims (5.6) and (5.9) hold, respectively, for all $(v, w) \in E_{\sigma\overline{\sigma}}$, depending on whether $\Lambda_\sigma(v) \neq v$ or $\Lambda_\sigma(v) = v$, respectively. These observations together with claim (5.8) motivate the following notion of a locally progressive valuation.

Locally progressive valuation. Let $\Xi = (\Lambda, \Pi) : V \to V \times \big(\wp(V) \times \mathbb{N}\big)$ be a valuation. We define $\texttt{Prog}(\Xi, e)$ to hold for $e = (v, w) \in E$, if and only if:

1. $\Lambda(v) = \Lambda(w)$, and
2. if $\Lambda(v) \neq v$ then $\Pi(v) =_{\Lambda(v)} \Pi(w) \;\boxplus\; v$, and
3. if $\Lambda(v) = v$ then $\Pi(v) \cong_{\Lambda(v)} \Pi(w) \;\boxplus\; v$ and $\Pi(v) = (\emptyset, 0)$.

We say that Ξ is a *locally progressive valuation* for strategies σ and τ if and only if $\texttt{Prog}(\Xi, e)$ holds, for all $e \in E_{\sigma\tau}$.

The following fact follows immediately from the definition of strategy $\overline{\sigma}$ and from (5.3), (5.6), (5.8), and (5.9).

Proposition 5.5 Valuation Ω_σ is a locally progressive valuation for σ and $\overline{\sigma}$.

Valuation $\Theta_{\sigma\tau}$. Note that if σ and τ are strategies for player $\oplus$ and for player $\ominus$, respectively, then for every vertex $v \in V$, there is a unique play $P_{\sigma\tau}(v)$ starting from v and consistent with σ and τ. We write $\Theta_{\sigma\tau}$ for a valuation defined by $\Theta_{\sigma\tau}(v) = \Theta\big(P_{\sigma\tau}(v)\big)$.

In the following lemma we establish that a locally progressive valuation for σ and τ is a local characterization of the valuation $\Theta_{\sigma\tau}$.

Lemma 5.6 A valuation Ξ is a locally progressive valuation for σ and τ if and only if $\Xi = \Theta_{\sigma\tau}$.

Proof. Let $v \in V$ and let $P_{\sigma\tau}(v) = \langle v_1, v_2, \ldots, v_\ell \rangle$ be the unique play starting from v and consistent with σ and τ. It is easy to verify that $\Theta_{\sigma\tau}$ is a locally progressive valuation for σ and τ. We show that if Ξ is a locally progressive valuation for σ and τ then $\Xi(v) = \Theta\big(P_{\sigma\tau}(v)\big)$.

Let $C = \{v_k, v_{k+1}, \ldots, v_{\ell-1}\}$ be the cycle of $P_{\sigma\tau}(v)$, i.e., let $v_k = v_\ell$ and $k < \ell$. Let $\Xi = (\Lambda, \Pi)$, where $\Lambda : V \to V$ and $\Pi : V \to \big(\mathcal{M}(V) \times \mathbb{N}\big)$. By definition of a locally progressive valuation we get that $\Lambda(v_1) = \Lambda(v_2) = \cdots = \Lambda(v_\ell)$. Let $\ell = \Lambda(v)$. We claim that $\lambda\big(P_{\sigma\tau}(v)\big) = \Lambda(v)$, i.e., that $\max^{\leq}(C) = \ell$.

By definition of a locally progressive valuation we get that $\Pi(v_i) \cong_\ell \Pi(v_{i+1}) \boxplus v_i$ for all $i \in \{k, k+1, \ldots, \ell-1\}$. This implies that

$$\Pi(v_k) \;\cong_\ell\; \Pi(v_\ell) \boxplus \{v_k, v_{k+1}, \ldots, v_{\ell-1}\} \;=\; \Pi(v_k) \boxplus C.$$

It follows that $\max^{\leq}(C) \leq \ell$. Note that $\max^{\leq}(C) < \ell$ implies that $\Pi(v_i) =_\ell \Pi(v_{i+1}) \boxplus v_i$ for all $i \in \{k, k+1, \ldots, \ell-1\}$, i.e., that $\Pi(v_k) =_\ell \Pi(v_k) \boxplus C$ holds. This, however, is impossible because $|C| > 0$. Therefore, we get that $\max^{\leq}(C) = \ell$.

Now we argue that $\Pi(v)$ is equal to the path value of $P_{\sigma\tau}(v)$. By definition of a locally progressive valuation we get that $\Pi(v_i) =_\ell \Pi(v_{i+1}) \boxplus v_i$ for all $i \in \{1, 2, \ldots, k-1\}$, and that $\Pi(v_k) = (\emptyset, 0)$. Therefore, we get that

$$\Pi(v) \;=_\ell\; (\emptyset, 0) \;\boxplus\; \texttt{Prefix}\big(P_{\sigma\tau}(v)\big) \;=_\ell\; \Big(\pi\big(P_{\sigma\tau}(v)\big), \#\big(P_{\sigma\tau}(v)\big)\Big).$$

[Lemma 5.6] ■

A best counter-strategy against σ. Note that by Proposition 5.5, an immediate corollary of Lemma 5.6 is that

$$\Omega_\sigma = \Theta_{\sigma\overline{\sigma}}. \tag{5.11}$$

Hence by definition of Ω_σ we get that $\Theta_{\sigma\overline{\sigma}} \trianglelefteq \Theta_{\sigma\tau}$, for all strategies τ for player $\ominus$. In other words, $\overline{\sigma}$ is a best counter-strategy against σ.

5.5.3 Locally under-progressive valuations

Motivated by conditions (5.5), (5.7), (5.8), and (5.10) satisfied by Ω_σ, we introduce a relaxation of the notion of a locally progressive valuation called a *locally under-progressive valuation*.

Locally under-progressive valuation. Let $\Xi = (\Lambda, \Pi) : V \to V \times \big(\wp(V) \times \mathbb{N}\big)$ be a valuation. We define $\texttt{UnderProg}(\Xi, e)$ to hold for $e = (v, w) \in E$, if and only if:

1. $\Lambda(v) \preceq \Lambda(w)$, and

if $\Lambda(v) = \Lambda(w)$ then:

2. if $\Lambda(v) \neq v$ then $\Pi(v) \trianglelefteq_{\Lambda(v)} \Pi(w) \boxplus v$, and
3. if $\Lambda(v) = v$ then $\Pi(v) \mathrel{\widetilde{\trianglelefteq}}_{\Lambda(v)} \Pi(w) \boxplus v$ and $\Pi(v) = (\emptyset, 0)$.

We say that Ξ is a *locally under-progressive valuation* for strategy σ if and only if $\texttt{UnderProg}(\Xi, e)$ holds, for all $e \in E_\sigma$.

The following fact follows immediately from (5.5), (5.7), (5.8), and (5.10).

Proposition 5.7 Valuation Ω_σ is a locally under-progressive valuation for strategy σ.

In the following proposition we collect a couple of simple facts about locally under-progressive valuations and relations $\trianglelefteq$ and $\mathrel{\widetilde{\trianglelefteq}}_\ell$.

Proposition 5.8 Let $\Xi = (\Lambda, \Pi) : V \to V \times \big(\wp(V) \times \mathbb{N}\big)$ be a valuation.

1. If $\texttt{UnderProg}\big(\Xi, (v, w)\big)$ holds and $\Lambda(v) = \Lambda(w)$ then $\Pi(v) \mathrel{\widetilde{\trianglelefteq}}_{\Lambda(v)} \Pi(w) \boxplus v$.
2. If $\texttt{UnderProg}\big(\Xi, (v, w)\big)$ holds and $\Xi(w) \trianglelefteq \Xi(u)$ then $\texttt{UnderProg}\big(\Xi, (v, u)\big)$ holds.

In the next lemma we establish that a locally under-progressive valuation Ξ for a strategy σ is a witness that all plays starting from a vertex v and consistent with σ, have value at least as big as $\Xi(v)$ with respect to the $\trianglelefteq$ order.

Lemma 5.9 If Ξ is a locally under-progressive valuation for a strategy σ then $\Xi \trianglelefteq \Omega_\sigma$.

Before we prove Lemma 5.9 we collect the following important properties of relations $\trianglelefteq_\ell$ and $\mathrel{\widetilde{\trianglelefteq}}_\ell$.

Proposition 5.10 Let $\ell \in V$, and $B, C \in \mathcal{M}(V)$, and $k \in \mathbb{N}$.

1. If $\max^{\leq}(C) \geq \ell$ then

$$\max^{\leq}(C) \succeq \ell \quad \text{if and only if} \quad (B, k) \mathrel{\widetilde{\trianglelefteq}}_\ell (B, k) \boxplus C.$$

2. If $\max^{\leq}(C) \neq \ell$ then

$$\max^{\leq}(C) \succ \ell \quad \text{if and only if} \quad (B,k) \lhd_\ell (B,k) \boxplus C.$$

Proof. Assume first that $\max^{\leq}(C) > \ell$. Then $(B,k) \mathrel{\widetilde{\lhd}_\ell} (B,k) \boxplus C$ holds if and only if $B \prec_\ell B \cup_{\mathcal{M}} C$ holds, if and only if $\max^{\leq}(C) \in R_+$ holds, if and only if $\max^{\leq}(C) \succ \ell$ holds. Observe also, that $(B,k) \lhd_\ell (B,k) \boxplus C$ holds if and only if $B \prec_\ell B \cup_{\mathcal{M}} C$ holds, if and only if $(B,k) \mathrel{\widetilde{\lhd}_\ell} (B,k) \boxplus C$ holds.

Assume that $\max^{\leq}(C) = \ell$. Then $(B,k) \mathrel{\widetilde{\unlhd}_\ell} (B,k) \boxplus C$ holds because $B =_\ell B \cup_{\mathcal{M}} C$; and $\max^{\leq}(C) \succeq \ell$ obviously holds.

Assume finally that $\max^{\leq}(C) < \ell$. Then $B =_\ell B \cup_{\mathcal{M}} C$ and therefore by definition of $\unlhd_\ell$ we have that $(B,k) \lhd_\ell (B,k) \boxplus C$ holds if and only if $\ell \in R_-$, if and only if $\max^{\leq}(C) \succ \ell$. [Proposition 5.10] ■

Proof (of Lemma 5.9)**.** Let $v \in V$, and let P be a play starting from v and consistent with σ. It suffices to show that $\Xi(v) \unlhd \Theta(P)$.

Let $\Xi = (\Lambda, \Pi) : V \to V \times \big(\mathcal{M}(V) \times \mathbb{N}\big)$. First we show that $\Lambda(v) \preceq \lambda(P)$. From definition of a locally under-progressive valuation it follows that the values of Λ on vertices in play P are non-decreasing with respect to $\preceq$ order, so they are all equal on the cycle $\texttt{Cycle}(P)$ of P. Hence, if we define ℓ to be the value of Λ on vertices in $\texttt{Cycle}(P)$ then we have $\Lambda(v) \preceq \ell$. Therefore, it suffices to prove that $\ell \preceq \lambda(P)$.

If $\ell \notin \texttt{Cycle}(P)$ then applying the definition of a locally under-progressive valuation around the cycle of P we get that

$$\Pi(w) \unlhd_\ell \Pi(w) \boxplus \texttt{Cycle}(P),$$

for some vertex $w \in \texttt{Cycle}(P)$. Note that from $\ell \notin \texttt{Cycle}(P)$ it follows that $\lambda(P) \neq \ell$, and therefore clause 2. of Proposition 5.10, with $C = \texttt{Cycle}(P)$, implies that $\lambda(P) = \max^{\leq}\big(\texttt{Cycle}(P)\big) \succeq \ell$.

If $\ell \in \texttt{Cycle}(P)$ then applying the definition of a locally under-progressive valuation around the cycle of P, together with clause 1. of Proposition 5.8, we get that

$$\Pi(w) \mathrel{\widetilde{\unlhd}_\ell} \Pi(w) \boxplus \texttt{Cycle}(P),$$

for some vertex $w \in \texttt{Cycle}(P)$. Note that from $\ell \in \texttt{Cycle}(P)$ it follows that $\lambda(P) \geq \ell$, and therefore clause 1. of Proposition 5.10 implies that $\lambda(P) = \max^{\leq}\big(\texttt{Cycle}(P)\big) \succeq \ell$.

If $\Lambda(v) \prec \lambda(P)$ then we have $\Xi(v) \lhd \Theta(P)$ and we are done. Assume then that $\ell = \Lambda(v) = \lambda(P)$. We show that in this case $\Pi(v) \lhd_\ell \big(\pi(P), \#(P)\big)$, which immediately implies that $\Xi(v) \unlhd \Theta(P)$. By applying the definition of a locally under-progressive valuation along $\texttt{Prefix}(P)$ we get

$$\Pi(v) \unlhd_\ell \Pi(\ell) \boxplus \texttt{Prefix}(P),$$

and we also have $\Pi(\ell) = (\emptyset, 0)$ because $\Lambda(\ell) = \ell$, and hence

$$\Pi(v) \unlhd_\ell \Big(\texttt{Prefix}(P), \big|\texttt{Prefix}(P)\big|\Big) =_\ell \big(\pi(P), \#(P)\big).$$

[Lemma 5.9] ■

5.5.4 Strategy improvement

Lemma 5.11 For every strategy σ for player $\oplus$, we have that $\sigma \sqsubseteq \texttt{Improve}(\sigma)$.

Proof. It suffices to show that Ω_σ is a locally under-progressive valuation for $\texttt{Improve}(\sigma)$. Then by Lemma 5.9 we get that $\Omega_\sigma \trianglelefteq \Omega_{\texttt{Improve}(\sigma)}$, i.e., that $\sigma \sqsubseteq \texttt{Improve}(\sigma)$.

We argue that Ω_σ is a locally under-progressive valuation for $\texttt{Improve}(\sigma)$. By Proposition 5.7 we have that Ω_σ is a locally under-progressive valuation for σ. Therefore, it suffices to check that for all $v \in M_\oplus$, the predicate $\texttt{UnderProg}$ holds for Ω_σ along the edge $\big(v, [\texttt{Improve}(\sigma)](v)\big)$. Note that by definition of the $\texttt{Improve}$ operator we have

$$\Omega_\sigma\big([\texttt{Improve}(\sigma)](v)\big) \trianglerighteq \Omega_\sigma\big(\sigma(v)\big),$$

for all $v \in M_\oplus$. Note that the $\texttt{UnderProg}$ predicate holds for Ω_σ along the edges $\big(v, \sigma(v)\big)$, for all $v \in M_\oplus$, because Ω_σ is a locally progressive valuation for σ. It then follows from clause 2. of Proposition 5.8 that the $\texttt{UnderProg}$ predicate for Ω_σ holds along every edge $\big(v, [\texttt{Improve}(\sigma)](v)\big)$, for all $v \in M_\oplus$. [Lemma 5.11] ■

5.5.5 Maximum strategies

Lemma 5.12 For every strategy σ for player $\oplus$, the following are equivalent:

1. strategy σ is not a maximum element in $(\texttt{Strategies}_\oplus, \sqsubseteq)$,
2. $\texttt{Improve}(\sigma) \neq \sigma$,
3. $\texttt{Improve}(\sigma) \not\sqsubseteq \sigma$.

Locally over-progressive valuation. Let $\Xi = (\Lambda, \Pi) : V \to V \times \big(\wp(V) \times \mathbb{N}\big)$ be a valuation. We define $\texttt{OverProg}(\Xi, e)$ to hold for $e = (v, w) \in E$ if and only if:

1. $\Lambda(v) \succeq \Lambda(w)$, and

if $\Lambda(v) = \Lambda(w)$ then:

2. if $\Lambda(v) \neq v$ then $\Pi(v) \trianglerighteq_{\Lambda(v)} \Pi(w) \boxplus v$, and
3. if $\Lambda(v) = v$ then $\Pi(v) \mathrel{\widetilde{\trianglerighteq}}_{\Lambda(v)} \Pi(w) \boxplus v$ and $\Pi(v) = (\emptyset, 0)$.

We say that Ξ is a *locally over-progressive valuation* for strategy τ for player $\ominus$ if and only if $\texttt{OverProg}(\Xi, e)$ holds for all $e \in E_\tau$.

Valuation $\mho_\sigma$. For every strategy $\tau \in \texttt{Strategies}_\ominus$ we define $\mho_\tau \in \texttt{Valuations}$ in the following way:

$$\mho_\tau(v) = \max{}^{\trianglelefteq}\big\{\, \Theta(P) \;:\; P \in \texttt{Plays}_\tau(v) \,\big\},$$

where $\texttt{Plays}_\tau(v)$ is the set of plays starting from vertex v and consistent with τ.

In the following lemma we show that a locally over-progressive valuation Ξ for a strategy τ is a witness that all plays starting from vertex v and consistent with τ have value at most as big as $\Xi(v)$ with respect to the $\trianglelefteq$ order on valuations.

Lemma 5.13 If Ξ is a locally over-progressive valuation for a strategy τ then $\mho_\tau \trianglelefteq \Xi$.

The above lemma is proved similarly to Lemma 5.9, using the following simple facts instead of Proposition 5.8.

Proposition 5.14 Let $\Xi = (\Lambda, \Pi) : V \to V \times \big(\wp(V) \times \mathbb{N}\big)$ be a valuation.

1. If $\texttt{OverProg}\big(\Xi, (v, w)\big)$ holds and $\Lambda(v) = \Lambda(w)$ then $\Pi(v) \mathrel{\widetilde{\trianglerighteq}}_{\Lambda(v)} \Pi(w) \boxplus v$.

2. If $\texttt{OverProg}\big(\Xi, (v, w)\big)$ holds and $\Xi(w) \trianglerighteq \Xi(u)$ then $\texttt{OverProg}\big(\Xi, (v, u)\big)$ holds.

Proof (of Lemma 5.12).
$1 \Rightarrow 2$. We claim that it suffices to argue that if $\texttt{Improve}(\sigma) = \sigma$ then Ω_σ is a locally over-progressive valuation for $\overline{\sigma}$. By Lemma 5.13 we then have

$$\mho_{\overline{\sigma}} \trianglelefteq \Omega_\sigma. \tag{5.12}$$

This, however, implies that $\kappa \sqsubseteq \sigma$ for all strategies κ for player $\oplus$. It is so because $\Omega_\kappa \trianglelefteq \mho_\tau$ holds for all strategies κ and τ, hence in particular $\Omega_\kappa \trianglelefteq \mho_{\overline{\sigma}}$ holds, so altogether we get $\Omega_\kappa \trianglelefteq \mho_{\overline{\sigma}} \trianglelefteq \Omega_\sigma$.

We argue that Ω_σ is a locally over-progressive valuation for $\overline{\sigma}$. By Proposition 5.5 we know that $\texttt{Prog}$ predicate for Ω_σ holds along all edges $(v, w) \in E$, such that $\sigma(v) = w$ or $\overline{\sigma}(v) = w$. Therefore, it suffices to check that the $\texttt{OverProg}$ predicate for Ω_σ holds along all edges $(v, w) \in E$, such that $v \in M_\oplus$. Note that from $\texttt{Improve}(\sigma) = \sigma$ it follows that

$$\Omega_\sigma\big(\sigma(v)\big) \trianglerighteq \Omega_\sigma(w),$$

for all $v \in M_\oplus$ and $w \in \text{succ}(v)$, and hence by clause 2. of Proposition 5.14 we get that $\texttt{OverProg}\big(\Omega_\sigma, (v, w)\big)$ holds.

$2 \Rightarrow 3$. Note that if $\texttt{Improve}(\sigma) \neq \sigma$ then for some $v \in M_\oplus$, the predicate $\texttt{Prog}$ does not hold for Ω_σ along the edge $\big(v, [\texttt{Improve}(\sigma)](v)\big)$. Therefore, by Lemma 5.6 we have that $\Omega_\sigma \neq \Theta_{\texttt{Improve}(\sigma)\overline{\texttt{Improve}(\sigma)}}$, i.e., by (5.11) we have that $\Omega_\sigma \neq \Omega_{\texttt{Improve}(\sigma)}$. Recall that $\trianglelefteq$ is a partial order on the set of valuations, hence it must be the case that $\Omega_{\texttt{Improve}(\sigma)} \not\trianglelefteq \Omega_\sigma$ because by Lemma 5.11 we have $\Omega_\sigma \trianglelefteq \Omega_{\texttt{Improve}(\sigma)}$.

$3 \Rightarrow 1$. Straightforward. [Lemma 5.12] ■

5.5.6 Proving the postulates P1., P2., I1., and I2.

P1. Lemmas 5.11 and 5.12 imply that after a finite number of iterations of the strategy improvement algorithm it must be the case that $\texttt{Improve}(\sigma) = \sigma$. By Lemma 5.12 such a strategy σ is a maximum element in the pre-order $(\texttt{Strategies}_{\oplus}, \sqsubseteq)$.

P2. Let σ be a maximum element in $(\texttt{Strategies}_{\oplus}, \sqsubseteq)$. Then from Lemma 5.12 it follows that $\texttt{Improve}(\sigma) = \sigma$. By definition of Ω_σ, strategy σ is a winning strategy for player $\oplus$ from the set

$$W_{\oplus}^{\sigma} = \left\{\, v \in V \ :\ \Omega_\sigma(v) = \big(\ell, (B, k)\big) \text{ and } \ell \in R_+ \,\right\}.$$

We claim that $W_{\oplus}^{\sigma}$ is the winning set for player $\oplus$. It suffices to argue that player $\ominus$ has a winning strategy from the set $V \setminus W_{\oplus}^{\sigma}$. Note that

$$V \setminus W_{\oplus}^{\sigma} = \left\{\, v \in V \ :\ \Omega_\sigma(v) = \big(\ell, (B, k)\big) \text{ and } \ell \in R_- \,\right\},$$

and by (5.12) we have $\mho_{\overline{\sigma}} \trianglelefteq \Omega_\sigma$. This means, however, that $\overline{\sigma}$ is a winning strategy for player $\ominus$ from $V \setminus W_{\oplus}^{\sigma}$.

I1. Immediate from Lemmas 5.11 and 5.12.

I2. Immediate from Lemma 5.12.

5.6 Efficient implementation

We are given a graph $G = (V, E)$ with a linear order $\leq$ on vertices, and a vertex $t \in V$, such that:

$$\text{for every } v \in V \text{, there is a path from } v \text{ to } t \text{ in } G, \tag{5.13}$$

and

$$\text{for every cycle } C \text{ in } G \text{, we have that } \max^{\leq}(C) \succeq t. \tag{5.14}$$

Every edge $(v, w) \in E$ has the weight $\big(\{v\} \cap V_{>t}, 1\big) \in \wp(V_{>t}) \times \mathbb{N}$. The weights are ordered by $\trianglelefteq_t$ and added to each other with $\boxplus$. The task is for all vertices $v \in V$, to determine (the weight of) a shortest path from v to t in G. For all $v \in V$, let $\Sigma(v) \subseteq \wp(V_{>t})$ be the set of vertices with priority bigger than t occurring in a shortest path from v to t in G.

We use the following algorithm to compute the shortest paths.

Shortest paths

1\. $E := E \setminus \big(\{t\} \times V\big)$
2\. **for all** $r \in V_{>t}$ in $\leq$-descending order
3\. **if** $r \in R_+$ **then**
4\. $W := \big\{\, w \in V$: there is a path from w to t in $(V \setminus r, E) \,\big\}$
5\. $E := E \setminus \big((W \cup \{r\}) \times (V \setminus W)\big)$
6\. **if** $r \in R_-$ **then**
7\. $U := \big\{\, u \in V$: there is a path from u to r in $(V, E) \,\big\}$
8\. $E := E \setminus \big((U \setminus \{r\}) \times (V \setminus U)\big)$
9\. **if** $t \in R_+$ **then** find longest distances from every vertex to t
10\. **if** $t \in R_-$ **then** find shortest distances from every vertex to t

The algorithm works as follows. First we remove edges going out of t since we are only interested in paths from all vertices to t (line 1.). Then in every iteration of the **for all** loop (lines 2.–8.) we remove edges that cannot possibly belong to a shortest path from a vertex v to t. After the **for all** loop has been executed, from every vertex there are only paths to t containing the same set of vertices in $V_{>t}$ as a shortest path. Therefore, in order to find shortest paths from every vertex to t it suffices to find longest distances to t if $t \in R_+$, and shortest distances to t if $r \in R_-$ (lines 9. or 10.). Finding longest distances in the case when $t \in R_+$ can be done efficiently because in this case the graph is acyclic.

Let us analyze the first iteration of the **for all** loop, i.e., let $r = \max^{\leq}(V)$ and let $r > t$. There are two cases to consider: $r \in R_+$ and $r \in R_-$.

Suppose that $r \in R_+$. Then for all $v \in V$, by the definition of $\trianglelefteq_t$ and by the maximality of r with respect to $\leq$, a shortest path from v to t should avoid visiting r if possible. More formally, for every $v \in V$, we have that:

$r \notin \Sigma(v)$ if and only if there is a path from v to t in G in which r does not occur. (5.15)

Therefore, the set W computed in line 4. contains all vertices $v \in V$, such that r does not occur on a shortest path from v to t. On the other hand, the set $V \setminus W$ consists of vertices from which a visit to r is "unavoidable" on a shortest path to t. It is then not hard to see that by removing in line 5. all the edges leading from W to $V \setminus W$ we only disconnect paths which cannot be shortest paths to t. Let G' be the graph after removing edges in line 5. Therefore, for every $v \in V$, we have that:

G' contains a shortest path from v to t in G. (5.16)

Moreover, after performing the deletion of edges in line 5., no path from W to t contains r, and all the paths from $V \setminus W$ to t contain r. In other words, we have that:

if $r \in \Sigma(v)$ then r occurs on every path from v to t in G', (5.17)

and

if $r \notin \Sigma(v)$ then r does not occur on any path from v to t in G'. (5.18)

Observe also, that by performing the deletion in line 5. we disconnect all cycles containing r, because then r can only be reached from vertices in $V \setminus W$, and only vertices in W are reachable form r, i.e., we have:

$$\text{there is no cycle in } G' \text{ containing } r. \tag{5.19}$$

We omit considering the other case, i.e., $r = \max^{\leq}(V_{>t}) \in R_-$. Instead, motivated by claims (5.16), (5.17), (5.18), and (5.19), established above for $r = \max^{\leq}(V_{>t}) \in R_+$, we argue that the following invariant is maintained by all iterations of the **for all** loop.

For $x \in V_{>t}$, let $G(x)$ be the graph after the body of **for all** loop has been performed for all $r \in V_{>t}$, such that $r > x$.

Proposition 5.15 Let $r \in V_{>t}$. For all $s \in V_{>t}$, such that $s > r$, and for all $v \in V$, we have:

1. $G(r)$ contains a shortest path from v to t in G,
2. if $s \in \Sigma(v)$ then s occurs on every path from v to t in $G(r)$,
3. if $s \notin \Sigma(v)$ then s does not occur on any path from v to t in $G(r)$,
4. there is no cycle in $G(r)$ containing s.

Proof. We prove the proposition by induction on r. Claims 1.–4. hold trivially for $r = \max^{\leq}(V)$. Assume as the induction hypothesis that clauses 1.–4. hold for some $r \in V_{>t}$. We show then, that clauses 1.–4. hold for $r' = \max^{\leq}(V_{<r})$, by analyzing what happens when the body of the **for all** loop is performed. We consider two cases.

- $r \in R_+$.

  Note that by clauses 1.-3. of the induction hypothesis, and by definition of $\trianglelefteq_t$, we have:

  $$r \in \Sigma(v) \text{ if and only if } r \text{ occurs on every path from } v \text{ to } t \text{ in } G(r).$$

  With this analogue of claim (5.15), the arguments needed to prove clauses 1.–4. for r' are the same as the ones we have used to establish claims (5.16)–(5.19).

- $r \in R_-$.

  Note that by clauses 1.-3. of the induction hypothesis, and by definition of $\trianglelefteq_t$, we have:

  $$r \in \Sigma(v) \text{ if and only if } r \text{ occurs on some path from } v \text{ to } t \text{ in } G(r). \tag{5.20}$$

  We argue that:

  $$\text{there is no cycle in } G(r) \text{ containing } r. \tag{5.21}$$

Suppose the contrary is the case, i.e., that there is a cycle C containing r in $G(r)$. Then by clause 4. of the induction hypothesis we have that $\max^{\leq}(C) = r$, which together with $r \in R_-$ implies that $\max^{\leq}(C) \prec t$, a contradiction with (5.14). Note that (5.21) establishes clause 4. for r'.

A corollary of (5.21) is that if $r \in \Sigma(v)$ then all paths from v to t and containing r are simple paths. Moreover, by clause 1. of the induction hypothesis there is a path from r to t in $G(r)$, and hence, claim (5.20) implies the following:

$$r \in \Sigma(v) \text{ if and only if there is a path from } v \text{ to } r \text{ in } G(r). \tag{5.22}$$

Therefore, the set U computed in line 7. contains all vertices $v \in V$, such that r occurs on a shortest path from v to t, and the set $V \setminus U$ contains all vertices from which a visit to r is not possible on a shortest path to t. Observe, that by removing edges leading from $U \setminus \{r\}$ to $V \setminus U$ we only disconnect paths which cannot be shortest paths to t. This establishes clause 1. for r'.

Note also, that by definition of U no path from $V \setminus U$ to t contains r, and after deletions of edges in line 8. we have that all paths from U to t contain r. This establishes clauses 3. and 2. for r'. [Proposition 5.15] ■

Theorem 5.16
An improvement step of our discrete strategy improvement algorithm can be performed in time $O(n \cdot m)$.

5.7 Time complexity

In the analysis of the running time of a strategy improvement algorithm there are two parameters of major interest:

1. the time needed to perform a single strategy improvement step,
2. the number of strategy improvement steps needed.

By Theorem 5.16 our discrete strategy improvement algorithm achieves a satisfactory bound on the former parameter.

A satisfactory analysis of the latter parameter is missing in our work. Despite the long history of strategy improvement algorithms for stochastic and payoff games [HK66, Con93, Pur95] very little is known about the number of strategy improvement steps needed. The best upper bounds are exponential [Con93, Pur95] but to our best knowledge no examples are known which require more than linear number of improvement steps. We believe that our purely discrete description of strategy improvement gives new insights into the behaviour of the algorithm in the special case of parity games. The two long standing questions: whether there is a polynomial time algorithm for solving parity games [EJS93], and, more concretely, whether a strategy improvement algorithm for parity games terminates in polynomial time [Con93, Pur95], remain open. Below we discuss some disjoint observations we have come up with so far, and some questions which we believe are worth pursuing.

Proposition 5.17 For every initial strategy, there is a strategy improvement policy of length at most n. Moreover, there is such a policy switching exactly one edge in every improvement step.

Proof. Let us fix a strategy κ which is a maximum element in the partial order $(\mathtt{Strategies}_{\oplus}, \sqsubseteq)$. Note that it suffices to show that for every strategy σ which is not a maximum element in $(\mathtt{Strategies}_{\oplus}, \sqsubseteq)$, there is an improvement $(v, w) \in E$ for σ, such that $w = \kappa(v)$.

Claim 5.18 If $I \subseteq E$ contains no improvement for σ then $\mathtt{Switch}_I(\sigma) \sqsubseteq \sigma$.

We argue how the proposition follows from the claim. Suppose for the sake of contradiction that for all $v \in M_{\oplus}$, such that $\sigma(v) \neq \kappa(v)$, we have $\Omega_{\sigma}\big(\kappa(v)\big) \trianglelefteq \Omega_{\sigma}\big(\sigma(v)\big)$. Let $I = \big\{ \big(v, \kappa(v)\big) : v \in M_{\oplus} \text{ and } \sigma(v) \neq \kappa(v) \big\}$. Note that $\mathtt{Switch}_I(\sigma) = \kappa$. Hence by Claim 5.18 we get that $\kappa \sqsubseteq \sigma$ which contradicts the assumption that κ is a maximum element in $(\mathtt{Strategies}_{\oplus}, \sqsubseteq)$ and that σ is not.

Proof (of Claim 5.18)**.** Note that for all $(v, w) \in I$ we have that

$$\Omega_{\sigma}(w) \trianglelefteq \Omega_{\sigma}\big(\sigma(v)\big). \tag{5.23}$$

By Proposition 5.5 we have that $\mathrm{Prog}(\Omega_{\sigma}, e)$ holds for all edges e in the graph of strategies σ and $\overline{\sigma}$. From (5.23) and from clause 2. of Proposition 5.14 it follows that $\mathtt{OverProg}(\Omega_{\sigma}, e)$ holds for all edges e in the graph of strategies $\mathtt{Switch}_I(\sigma)$ and $\overline{\sigma}$. Let $\sigma' = \mathtt{Switch}_I(\sigma)$. Note that in the (one-player) game $G_{\sigma'}$ we have $\mho_{\overline{\sigma}} = \Theta_{\sigma'\overline{\sigma}}$. Therefore, by applying Lemma 5.13 to $G_{\sigma'}$ we get that $\Theta_{\sigma'\overline{\sigma}} \trianglelefteq \Omega_{\sigma}$. Note that by definition of $\Omega_{\sigma'}$ we have that $\Omega_{\sigma'} \trianglelefteq \Theta_{\sigma'\overline{\sigma}}$, and hence we get $\Omega_{\sigma'} \trianglelefteq \Omega_{\sigma}$ which concludes the proof. [Claim 5.18] ■ [Proposition 5.17] ■

This contrasts with an algorithm for solving parity games based on progress measures [Jur00], for which there are families of examples on which every policy requires an exponential number of steps.

Melekopoglou and Condon [MC94] exhibit families of examples of simple stochastic games for which several natural strategy improvement policies switching only one switchable vertex in every strategy improvement step have exponential length.

Problem 5.19 Are there families of examples of parity games for which there exist strategy improvement policies of super-polynomial length?

Examples of Melekopoglou and Condon [MC94] are Markov decision processes, i.e., one-player simple stochastic games [Con92]. It is an open question whether the strategy improvement algorithm using the standard policy, i.e., switching all switchable vertices in every strategy improvement step, works in polynomial time for one-player simple stochastic games [MC94]. In contrast, our discrete strategy improvement algorithm terminates in polynomial time for one-player parity games.

Proposition 5.20 The discrete strategy improvement algorithm terminates after $O(n^3)$ strategy improvement steps for one-player parity games.

Most algorithms for solving parity games studied in literature have roughly $O\big((n/d)^d\big)$ or $O\big((n/d)^{d/2}\big)$ worst-case running time bounds, where d is the number of different priorities assigned to vertices. The best upper bound we can give at the moment for the number of strategy improvement steps needed by our discrete strategy improvement algorithm is the trivial one, i.e., the number of different strategies for player 0, which can be $2^{\Omega(n)}$.

Proposition 5.21 The discrete strategy improvement algorithm terminates after $\prod_{v \in V_0} \text{out-deg}(v)$ many strategy improvement steps.

There is, however, a variation of the strategy improvement algorithm for parity games, for which the number of strategy improvement steps is bounded by $O\big((n/d)^d\big)$.

Proposition 5.22 There is a strategy improvement algorithm for parity games for which all policies have length $O\big((n/d)^d\big)$, and every strategy improvement step can be performed in $n^{O(1)}$ time.

Note that in every strategy improvement step the current valuation Ω_σ strictly improves with respect to $\trianglelefteq$ in at least one vertex. We say that a strategy improvement step is *substantial* if in the current valuation the loop value for some vertex strictly improves. Observe that there can be at most $O(n^2)$ substantial strategy improvement steps. It follows that in search for superpolynomial examples one has to manufacture gadgets allowing long sequences of non-substantial strategy improvement steps.

We have collected a little experimental evidence that in practice most improvement steps are non-substantial. There are few interesting scalable families of hard examples of parity games known in literature. Using an implementation of our discrete strategy improvement algorithm by Schmitz and Vöge [SV00] we have run some experiments on families of examples taken from [BLV96] and from [Jur00], and on a family of examples mentioned in [Jur00] which make Zielonka's version [Zie98] of the McNaughton's algorithm [McN93] work in exponential time. For all these families only linear number of strategy improvement steps were needed and, interestingly, the number of non-substantial strategy improvement steps was in all cases constant, i.e., not dependent of the size of the game graph.

We conclude with a number of questions to pursue.

Problem 5.23 Does our discrete algorithm, with a policy switching in every strategy improvement step a maximal set of improvements (i.e., switching all switchable vertices), terminate in polynomial time? If not, exhibit families of examples for which there are policies of exponential length.

Problem 5.24 Are there polynomial time computable heuristics for constructing policies of polynomial length?

Problem 5.25 Define and study other partial orders on the set of strategies and other strategy improvement operators.

Problem 5.26 Develop other algorithms than a strategy improvement algorithm for solving the optimization problem of solving cycle-domination games with respect to a partial order on the set of strategies.

Chapter 6

Hhp-bisimilarity is undecidable

This chapter contains a revised version of [JN00]. It is joint work with Mogens Nielsen.

Abstract. History preserving bisimilarity (hp-bisimilarity) and hereditary history preserving bisimilarity (hhp-bisimilarity) are behavioural equivalences taking into account causal relationships between events of concurrent systems. Their prominent feature is being preserved under action refinement, an operation important for the top-down design of concurrent systems. We show that—unlike hp-bisimilarity—checking hhp-bisimilarity for finite labelled asynchronous transition systems is not decidable, by a reduction from the halting problem of 2-counter machines. To make the proof more transparent we introduce an intermediate problem of checking domino bisimilarity for origin constrained tiling systems, whose undecidability is interesting in its own right. We also argue that the undecidability of hhp-bisimilarity holds for finite labelled elementary net systems.

6.1 Introduction

The notion of behavioural equivalence that has attracted most attention in concurrency theory is bisimilarity, originally introduced by Park [Par81] and Milner [Mil80]; concurrent programs are considered to have the same meaning if they are bisimilar. The prominent role of bisimilarity is due to many pleasant properties it enjoys; we mention a few of them here.

A process of checking whether two transition systems are bisimilar can be seen as a two player game which is in fact an Ehrenfeucht-Fraïssé type of game for modal logic. More precisely, there is a winning strategy for a player who wants to show that the systems are bisimilar if and only if the systems cannot be distinguished by the formulas of the logic; the result due to Hennessy and Milner [HM85].

Another notable property of bisimilarity is its computational feasibility; see for example the overview note [MS95]. Let us illustrate this on the examples of finite transition systems and a class of infinite-state transition systems generated by context free grammars. For finite transition systems there are very efficient polynomial time algorithms for checking bisimilarity [KS90, PT87], in sharp contrast to **PSPACE**-completeness of the classical language equivalence. For transition systems generated by context free grammars, while language equivalence is undecidable, bisimilarity is decidable [CHS95], and if the grammar has no redundant nonterminals, even in polynomial time [HJM96]. Furthermore, as the results of [GH94] indicate, bisimilarity has a very rare status of being a decidable equivalence for context free grammars: all the other equivalences in the linear/branching time hierarchy [Gla90] are indeed undecidable. The algorithmic tractability makes bisimilarity especially attractive for automatic verification of concurrent systems.

The essence of bisimilarity, quoting [HM85], "is that the behaviour of a program is determined by how it communicates with an observer." Therefore, the notion of what can be observed of a behaviour of a system affects the notion of bisimilarity. An abstract definition of bisimilarity for arbitrary categories of models due to Joyal et al. [JNW96] formalizes this idea. Given a category of models where objects are behaviours and morphisms correspond to extension of behaviours, and given a subcategory of observable behaviours, the abstract definition yields a notion of bisimilarity for all behaviours with respect to observable behaviours. For example, for rooted labelled transition systems, taking synchronization trees [Mil80] into which they unfold as their behaviours, and sequences of actions as the observable behaviours, we recover the standard strong bisimilarity of Park and Milner [JNW96].

In order to model concurrency more faithfully several models have been introduced (see [WN95] for a survey) that make explicit the distinction between events that can occur concurrently, and those that are causally related. Then a natural choice is to replace sequences, i.e., linear orders as the observable behaviours, by partial orders of occurrences of events with causality as the ordering relation. For example, taking unfoldings of labelled asynchronous transition systems into event structures as the behaviours, and labelled partial orders as the observations, Joyal et al. [JNW96] obtained from their abstract definition the hereditary history preserving bisimilarity (hhp-bisimilarity), independently introduced and studied by Bednarczyk [Bed91].

A similar notion of bisimilarity has been studied before, namely history preserving bisimilarity (hp-bisimilarity), introduced by Rabinovich and Trakhtenbrot [RT88] and van Glabbeek and Goltz [GG89]. For the relationship between hp- and hhp-bisimilarity see for example [Bed91, JNW96, FH99].

One of the important motivations to study partial order based equivalences was the discovery that hp-bisimilarity has a rare status of being preserved under action refinement [GG89], an operation important for the top-down design of concurrent systems. Bednarczyk [Bed91] has extended this result to hhp-bisimilarity.

There is a natural logical characterization of hhp-bisimilarity checking games as shown by Nielsen and Clausen [NC95]: they are characteristic games for an extension of modal logic with backwards modalities, interpreted over event structures.

Hp-bisimilarity has been shown to be decidable for 1-safe Petri nets by Vogler [Vog91], and to be **DEXP**-complete by Jategaonkar, and Meyer [JM96]; let us just mention here that 1-safe Petri nets can be regarded as a proper subclass of finite asynchronous transition systems (see [WN95] for details), and that decidability of hp-bisimilarity can be easily extended to all finite asynchronous transition systems using the methods of [JM96].

Hhp-bisimilarity appears to be only a slight strengthening of hp-bisimilarity [JNW96], and hence many attempts have been made to extend the above mentioned algorithms to the case of hhp-bisimilarity. However, decidability of hhp-bisimilarity has remained open, despite several attempts over the years [NC95, NW96a, CS96, FH99]. Fröschle and Hildebrandt [FH99] have discovered an infinite hierarchy of bisimilarity notions refining hp-bisimilarity, and coarser than hhp-bisimilarity, such that hhp-bisimilarity is the intersection of all the bisimilarities in the hierarchy. They have shown all these bisimilarities to be decidable for 1-safe Petri nets. Fröschle [Frö00] has shown hhp-bisimilarity to be decidable for BPP-processes, a class of infinite state systems.

In this paper, we finally settle the question of decidability of hhp-bisimilarity by showing it to be undecidable for finite labelled elementary net systems. In order to make the proof more transparent we first introduce an intermediate problem of domino bisimilarity and show its undecidability by a direct reduction from the halting problem of 2-counter machines.

6.2 Hereditary history preserving bisimilarity

Definition 6.1 (Labelled asynchronous transition system)
A *labelled asynchronous transition system* is a tuple $A = (S, s^{\text{ini}}, E, \to, L, \lambda, I)$, where S is its set of *states*, $s^{\text{ini}} \in S$ is the *initial state*, E is the set of *events*, $\to \subseteq S \times E \times S$ is the set of *transitions*, L is the set of labels, and $\lambda : E \to L$ is the *labelling function*, and $I \subseteq E^2$ is the *independence relation* which is irreflexive and symmetric. We often write $s \xrightarrow{e} s'$, instead of $(s, e, s') \in \to$. Moreover, the following conditions have to be satisfied:

1. if $s \xrightarrow{e} s'$ and $s \xrightarrow{e} s''$ then $s' = s''$,

2. if $(e, e') \in I$, $s \xrightarrow{e} s'$, and $s' \xrightarrow{e'} t$, then $s \xrightarrow{e'} s''$, and $s'' \xrightarrow{e} t$ for some $s'' \in S$.

An asynchronous transition system is *coherent* if it satisfies the following condition:

3. if $(e, e') \in I$, $s \xrightarrow{e} s'$, and $s \xrightarrow{e'} s''$, then $s' \xrightarrow{e'} t$, and $s'' \xrightarrow{e} t$ for some $t \in S$.

An asynchronous transition system is *prime* if it is acyclic and satisfies the following condition:

4. if $s \xrightarrow{e} t$ and $s' \xrightarrow{e'} t$ then $(e, e') \in I$. [Definition 6.1] □

Winskel and Nielsen [WN95, NW96a] give a thorough survey and establish formal relationships between asynchronous transition systems and other models for concurrency, such as Petri nets, and event structures. The independence relation is meant to model concurrency: independent events can occur concurrently, while those that are not independent are causally related or in conflict.

Let $A = (S, s^{\mathrm{ini}}, E, \rightarrow, L, \lambda, I)$ be a labelled asynchronous transition system. A sequence of events $\overline{e} = \langle e_1, e_2, \ldots, e_n \rangle \in E^*$ is a *run* of A if there are states $s_1, s_2, \ldots, s_{n+1} \in S$, such that $s_1 = s^{\mathrm{ini}}$, and for all $i \in \{1, 2, \ldots, n\}$, we have $s_i \xrightarrow{e_i} s_{i+1}$. We write $\mathtt{Runs}(A)$ to denote the set of runs of A. We extend the labelling function λ to runs in the standard way.

Let $\overline{e} = \langle e_1, e_2, \ldots, e_n \rangle \in \mathtt{Runs}(A)$. We say that the k-th event, $1 \leq k < n$, is *swappable* in $\overline{e}$ if $(e_k, e_{k+1}) \in I$. We define $\mathrm{Swap}(\overline{e})$ to be the set of numbers of swappable events in $\overline{e}$. We write $\overline{e} \otimes k$ to denote the result of *swapping* the k-th event of $\overline{e}$ with the $(k+1)$-st, i.e., the sequence $\langle e_1, \ldots, e_{k-1}, e_{k+1}, e_k, \ldots, e_n \rangle$. Note that if $k \in \mathrm{Swap}(\overline{e})$ then $\overline{e} \otimes k \in \mathtt{Runs}(A)$; it follows from condition 2. of definition of an asynchronous transition system.

A run of a transition system models a finite *sequential* behaviour of a system: a sequence of occurrences of events. In order to model *concurrent* behaviours of a system we define an equivalence relation on the set of runs of an asynchronous transition system. We define the equivalence relation $\cong_A$ on $\mathtt{Runs}(A)$ to be the reflexive, symmetric, and transitive closure of

$$\left\{ \, (\overline{e}, \overline{e} \otimes k) \; : \; \overline{e} \in \mathtt{Runs}(A) \text{ and } k \in \mathrm{Swap}(\overline{e}) \, \right\}.$$

In other words, we have that $\overline{e_1} \cong_A \overline{e_2}$, for $\overline{e_1}, \overline{e_2} \in \mathtt{Runs}(A)$, if and only if $\overline{e_2}$ can be obtained from $\overline{e_1}$ by a finite number of swaps of swappable events.

We define an unfolding operation on asynchronous transition systems into prime asynchronous transition systems. The states of the unfolding of an asynchronous transition system A are meant to represent all concurrent behaviours of a system, just like the states of a synchronization tree represent all sequential behaviours of a system.

Definition 6.2 (Unfolding)
Let $A = (S, s^{\mathrm{ini}}, E, \rightarrow, L, \lambda, I)$ be an asynchronous transition system. The unfolding $\mathtt{Unf}(A)$ of A is an asynchronous transition system with the same set of events, the labelling function, and the independence relation as A. The set of states, the initial state, and the transition relation of $\mathtt{Unf}(A)$ are defined as follows:

- the set of states $S_{\mathtt{Unf}(A)}$ of $\mathtt{Unf}(A)$ is defined to be $\mathtt{Runs}(A)/_{\cong_A}$, i.e., the set of concurrent behaviours of A,

- the initial state $s^{\mathrm{ini}}_{\mathtt{Unf}(A)}$ of $\mathtt{Unf}(A)$ is $[\varepsilon]_{\cong_A}$, i.e., the $\cong_A$-equivalence class of the empty run,
- the set of transitions $\rightarrow_{\mathtt{Unf}(A)}$ of $\mathtt{Unf}(A)$ consists of transitions of the form $\big([\overline{e}]_{\cong_A}, e, [\overline{e}\cdot e]_{\cong_A}\big)$, for all $\overline{e} \in E^*$, and $e \in E$, such that $\overline{e}\cdot e \in \mathtt{Runs}(A)$.

[Definition 6.2] □

The following proposition follows easily from definition of $\mathtt{Unf}(A)$.

Proposition 6.3 If A is an asynchronous transition system then its unfolding $\mathtt{Unf}(A)$ is a prime asynchronous transition system.

Let $\overline{e} = \langle e_1, e_2, \ldots, e_n\rangle \in \mathtt{Runs}(A)$. We say that the k-th event, $1 \leq k \leq n$, is *most recent* in $\overline{e}$ if and only if $(e_k, e_\ell) \in I$, for all ℓ, such that $k < \ell \leq n$. We define $\mathrm{MR}(\overline{e})$ to be the set of numbers of most recent events in $\overline{e}$. We write $\overline{e} \oslash k$ to denote the result of removing the k-th event from $\overline{e}$, i.e., the sequence $\langle e_1, \ldots, e_{k-1}, e_{k+1}, \ldots, e_n\rangle$. Note that if $k \in \mathrm{MR}(\overline{e})$ then $\overline{e} \oslash k \in \mathtt{Runs}(A)$; it follows from condition 2. of definition of an asynchronous transition system.

Definition 6.4 (Hereditary history preserving bisimulation)
Let $A_i = (S_i, s_i^{\mathrm{ini}}, E_i, \rightarrow_i, L, \lambda_i, I_i)$ for $i \in \{1,2\}$ be labelled asynchronous transition systems. A relation $B \subseteq \mathtt{Runs}(A_1) \times \mathtt{Runs}(A_2)$ is a *hereditary history preserving* (hhp-) bisimulation relating A_1 and A_2 if the following conditions are satisfied:

1. $(\varepsilon, \varepsilon) \in B$,

and if $(\overline{e_1}, \overline{e_2}) \in B$ then $\lambda_1(\overline{e_1}) = \lambda_2(\overline{e_2})$, and:

2. for all $i \in \{1,2\}$ and $e_i \in E_i$, if $\overline{e_i}\cdot e_i \in \mathtt{Runs}(A_i)$, then there exists $e_{3-i} \in E_{3-i}$, such that $\overline{e_{3-i}}\cdot e_{3-i} \in \mathtt{Runs}(A_{3-i})$, and $\lambda_1(e_1) = \lambda_2(e_2)$, and $(\overline{e_1}\cdot e_1, \overline{e_2}\cdot e_2) \in B$,
3. $\mathrm{MR}(\overline{e_1}) = \mathrm{MR}(\overline{e_2})$,
4. if $k \in \mathrm{MR}(\overline{e_1}) = \mathrm{MR}(\overline{e_2})$ then $(\overline{e_1} \oslash k, \overline{e_2} \oslash k) \in B$. [Definition 6.4] □

Two asynchronous transition systems A_1, and A_2 are hereditary history preserving (hhp-) *bisimilar*, if there is an hhp-bisimulation relating them.

The following proposition is straightforward since every asynchronous transition system A and its unfolding $\mathtt{Unf}(A)$ have the same set of runs and the same independence relation.

Proposition 6.5 Asynchronous transition systems A_1 and A_2 are hhp-bisimilar if and only if their unfoldings $\mathtt{Unf}(A_1)$ and $\mathtt{Unf}(A_2)$ are hhp-bisimilar.

The main result of this paper is the following theorem proved in section 6.4.

Theorem 6.6 (Undecidability of hhp-bisimilarity)
Hhp-bisimilarity is undecidable for finite labelled asynchronous transition systems.

The process of checking hhp-bisimilarity of asynchronous transition systems is conveniently viewed as a game played on runs of the systems by two players: *Challenger* and *Duplicator*. Duplicator aims to prove the systems to be bisimilar while Challenger intends otherwise [Sti97, Tho93, NC95].

Definition 6.7 (Hhp-bisimilarity checking game)
Let $A_i = (S_i, s_i^{\mathrm{ini}}, E_i, \to_i, L, \lambda_i, I_i)$ for $i \in \{1, 2\}$ be labelled asynchronous transition systems. *Configurations* of the *hhp-bisimilarity checking game* $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$ are elements of the set $\mathtt{Runs}(A_1) \times \mathtt{Runs}(A_2)$. Game $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$ is played by two players: *Challenger* and *Duplicator*. The *initial configuration* is the pair of empty runs $(\varepsilon, \varepsilon)$. In each *move* the players change the current configuration $(\overline{e_1}, \overline{e_2})$ of $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$ in one of the following ways chosen by Challenger.

- Forward move:

  1. Challenger chooses an $i \in \{1, 2\}$ and an event $e_i \in E_i$, such that $\overline{e_i} \cdot e_i \in \mathtt{Runs}(A_i)$;
  2. Duplicator responds by choosing an event $e_{3-i} \in E_{3-i}$, such that $\overline{e_{3-i}} \cdot e_{3-i} \in \mathtt{Runs}(A_{3-i})$, and $\lambda_1(e_1) = \lambda_2(e_2)$;

  the pair $(\overline{e_1} \cdot e_1, \overline{e_2} \cdot e_2)$ becomes the current configuration.

- Backward move:

  1. Challenger chooses an $i \in \{1, 2\}$ and a $k \in \mathrm{MR}(\overline{e_i})$;
  2. Duplicator can only respond if $k \in \mathrm{MR}(\overline{e_{3-i}})$; otherwise Duplicator gets stuck;

  if $k \in \mathrm{MR}(\overline{e_1})$, and $k \in \mathrm{MR}(\overline{e_2})$ then $(\overline{e_1} \oslash k, \overline{e_2} \oslash k)$ becomes the current configuration.

A *play* of $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$ if a *maximal* sequence of configurations formed by players making moves in the fashion described above. Duplicator is the winner in every infinite play; a finite play is lost by the player who is stuck. Note that Challenger gets stuck only if both transition systems have no transitions going out from their initial states. [Definition 6.7] □

We avoid tedious details of formalizing notions of strategies and winning strategies for either of the players. The following standard fact is proved by arguing that an hhp-bisimulation is a good formalization of the notion of a winning strategy for Duplicator in an hhp-bisimilarity checking game [NC95].

Proposition 6.8 Asynchronous transition systems A_1 and A_2 are hhp-bisimilar if and only if there is a winning strategy for Duplicator in hhp-bisimilarity checking game $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$.

It is particularly easy to see how an hhp-bisimulation $B \subseteq \mathtt{Runs}(A_1) \times \mathtt{Runs}(A_2)$ can serve as a winning strategy for Duplicator in $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$. Intuitively, the hhp-bisimulation B contains all configurations of $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$ which can be become current configurations when Duplicator is following the strategy determined by B. The strategy is defined as follows. Let $(\overline{e_1}, \overline{e_2})$ be the current configuration of $\mathcal{B}_{\mathrm{hhp}}(A_1, A_2)$. If Challenger chooses event e_i of A_i in a forward move, then Duplicator responds by choosing an event e_{3-i} of A_{3-i}, such that $(\overline{e_1} \cdot e_1, \overline{e_2} \cdot e_2) \in B$. If Challenger makes a backward move then response of Duplicator is unique. This strategy contains the initial configuration $(\varepsilon, \varepsilon)$ by condition 1. of definition of an hhp-bisimulation, and it is well defined by conditions 2.–4.

6.3 Domino bisimilarity is undecidable

6.3.1 Domino bisimilarity

Definition 6.9 (Origin constrained tiling system)
An *origin constrained tiling system* $T = \big(D, D^{\mathrm{ori}}, (H, H^0), (V, V^0), L, \lambda\big)$ consists of a set D of *dominoes*, its subset $D^{\mathrm{ori}} \subseteq D$ called the *origin constraint*, two *horizontal compatibility* relations $H, H^0 \subseteq D^2$, two *vertical compatibility* relations $V, V^0 \subseteq D^2$, a set L of *labels*, and a *labelling function* $\lambda : D \to L$.

[Definition 6.9] □

A *configuration* of T is a triple $(d, x, y) \in D \times \mathbb{N} \times \mathbb{N}$, such that if $x = y = 0$ then $d \in D^{\mathrm{ori}}$. In other words, in the "origin" position $(x, y) = (0, 0)$ of the non-negative integer grid only dominoes from the origin constraint D^{ori} are allowed.

Let (d, x, y), and (d', x', y') be configurations of T such that $|x' - x| + |y' - y| = 1$, i.e., the positions (x, y), and (x', y') are neighbours in the non-negative integer grid. Without loss of generality we may assume that $x + y < x' + y'$. We say that configurations (d, x, y), and (d', x', y') are *compatible* if either of the two conditions below holds:

- $x' = x$, and $y' = y + 1$, and
  if $y = 0$, then $(d, d') \in V^0$, and if $y > 0$, then $(d, d') \in V$, or
- $x' = x + 1$, and $y' = y$, and
  if $x = 0$, then $(d, d') \in H^0$, and if $x > 0$, then $(d, d') \in H$.

Definition 6.10 (Domino bisimulation)
Let $T_i = \big(D_i, D_i^{\mathrm{ori}}, (H_i, H_i^0), (V_i, V_i^0), L_i, \lambda_i\big)$ for $i \in \{1, 2\}$ be origin constrained tiling systems. A relation $B \subseteq D_1 \times D_2 \times \mathbb{N} \times \mathbb{N}$ is a *domino bisimulation* relating T_1 and T_2, if $(d_1, d_2, x, y) \in B$ implies that $\lambda_1(d_1) = \lambda_2(d_2)$, and the following conditions are satisfied for all $i \in \{1, 2\}$:

1. for all $d_i \in D_i^{\mathrm{ori}}$, there is $d_{3-i} \in D_{3-i}^{\mathrm{ori}}$, so that $\lambda_1(d_1) = \lambda_2(d_2)$, and $(d_1, d_2, 0, 0) \in B$,
2. for all $x, y \in \mathbb{N}$, such that $(x, y) \neq (0, 0)$, and $d_i \in D_i$, there is $d_{3-i} \in D_{3-i}$, such that $\lambda_1(d_1) = \lambda_2(d_2)$, and $(d_1, d_2, x, y) \in B$,

3. if $(d_1, d_2, x, y) \in B$, then for all neighbours $(x', y') \in \mathbb{N} \times \mathbb{N}$ of (x, y), and $d'_i \in D_i$, if configurations (d_i, x, y), and (d'_i, x', y') of T_i are compatible, then there exists $d'_{3-i} \in D_{3-i}$, such that $\lambda_1(d'_1) = \lambda_2(d'_2)$, and configurations (d_{3-i}, x, y), and (d'_{3-i}, x', y') of T_{3-i} are compatible, and $(d'_1, d'_2, x', y') \in B$. [Definition 6.10] □

We say that two tiling systems are *domino bisimilar* if and only if there is a domino bisimulation relating them.

The main result of this section is the following theorem proved in subsection 6.3.3.

Theorem 6.11 (Undecidability of domino bisimilarity)
Domino bisimilarity is undecidable for origin constrained tiling systems.

The proof is a reduction from the halting problem for deterministic 2-counter machines. For a deterministic 2-counter machine M we define in section 6.3.3 two origin constrained tiling systems T_1, and T_2, enjoying the following property.

Proposition 6.12 Machine M does not halt, if and only if there is a domino bisimulation relating T_1 and T_2.

The process of checking domino bisimilarity of origin constrained tiling systems is conveniently viewed as a game played on an infinite grid by two players: Challenger and Duplicator. As in the case of hhp-bisimilarity checking games Duplicator aims to prove the tiling systems to be bisimilar while Challenger intends otherwise.

Definition 6.13 (Origin constrained domino bisimilarity checking game)
Let T_1, and T_2 be origin constrained tiling systems. *Configurations* of the *origin constrained domino bisimilarity checking game* $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$ are elements of the set $D_1 \times D_2 \times \mathbb{N} \times \mathbb{N}$. Game $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$ is played by two players *Challenger* and *Duplicator*.

- First the players fix an *initial configuration*:

  1. Challenger chooses an $i \in \{1, 2\}$, and a configuration (d_i, x, y) of T_i,
  2. Duplicator responds by choosing a domino $d_{3-i} \in D_{3-i}$, such that (d_{3-i}, x, y) is a configuration of T_{3-i}, and $\lambda_1(d_1) = \lambda_2(d_2)$;

  if both players were able to make their choices then the tuple (d_1, d_2, x, y) becomes the *current configuration* of $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$.

- In each *move* of the game the players change the current configuration (d_1, d_2, x, y):

  1. Challenger chooses an $i \in \{1, 2\}$, and a configuration (d'_i, x', y') of T_i compatible with configuration (d_i, x, y),

2. Duplicator responds by choosing a domino $d'_{3-i} \in D_{3-i}$ such that (d'_{3-i}, x', y') is a configuration of T_{3-i}, and $\lambda_1(d_1) = \lambda_2(d_2)$, and configurations (d_{3-i}, x, y) and (d'_{3-i}, x', y') of T_{3-i} are compatible;

if both players were able to make their choices then the tuple (d'_1, d'_2, x', y') becomes the *current configuration* of $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$.

A *play* of $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$ is a *maximal* sequence of configurations formed by players making moves in the fashion described above. Duplicator is the winner in every infinite play; a finite play is lost by the player who is stuck. [Definition 6.13] □

We avoid tedious details of formalizing notions of strategies and winning strategies for either of the players. The following simple fact is proved by arguing that a domino bisimulation is a good formalization of a winning strategy for Duplicator in a domino bisimulation checking game.

Proposition 6.14 Origin constrained tiling systems T_1 and T_2 are domino bisimilar if and only if Duplicator has a winning strategy in the domino bisimulation game $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$.

6.3.2 Counter machines

A 2-counter machine M consists of a finite program with the set L of instruction labels, and instructions of the form:

- ℓ: `c`$_i$ `:= c`$_i$ `+ 1; goto` m
- ℓ: `if c`$_i$ `= 0 then c`$_i$ `:= c`$_i$ `+ 1; goto` m
  `else c`$_i$ `:= c`$_i$ `- 1; goto` n
- `halt:`

where $i = 1, 2$; $\ell, m, n \in L$, and $\{\texttt{start}, \texttt{halt}\} \subseteq L$. A configuration of M is a triple $(\ell, x, y) \in L \times \mathbb{N} \times \mathbb{N}$, where ℓ is the label of the current instruction, and x, and y are the values stored in counters $\mathtt{c}_1$, and $\mathtt{c}_2$, respectively; we denote the set of configurations of M by $\mathtt{Confs}(M)$. The semantics of 2-counter machines is standard: let $\vdash_M \subseteq \mathtt{Confs}(M) \times \mathtt{Confs}(M)$ be the usual one-step derivation relation on configurations of M; by $\vdash_M^+$ we denote the reachability (in at least one step) relation for configurations, i.e., the transitive closure of $\vdash_M$.

Before we give a reduction from the halting problem of 2-counter machines to origin constrained domino bisimilarity let us take a look at the directed graph $(\mathtt{Confs}(M), \vdash_M)$, with configurations of M as vertices, and edges denoting derivation in one step. Since machine M is deterministic, for each configuration there is at most one outgoing edge; moreover only halting configurations have no outgoing edges. It follows that connected components of the graph $(\mathtt{Confs}(M), \vdash_M)$ are either trees with edges going to the root which is the unique halting configuration in the component, or have no halting configuration at all. This observation is formalized in the following proposition.

Proposition 6.15 Let M be a 2-counter machine. The following conditions are equivalent:

1. machine M halts on input $(0,0)$, i.e., $(\texttt{start}, 0, 0) \vdash_M^+ (\texttt{halt}, x, y)$ for some $x, y \in \mathbb{N}$,
2. $(\texttt{start}, 0, 0) \sim_M (\texttt{halt}, x, y)$ for some $x, y \in \mathbb{N}$, where the relation $\sim_M \subseteq \texttt{Confs}(M) \times \texttt{Confs}(M)$ is the symmetric and transitive closure of $\vdash_M$.

6.3.3 The reduction

Now we go for a proof of Proposition 6.12. The idea is to design a tiling system which "simulates" behaviour of a 2-counter machine.

Let M be a 2-counter machine. We construct a tiling system T_M with the set L of instruction labels of M as the set of dominoes, and the identity function on L as the labelling function. Note that this implies that all tuples belonging to a domino bisimulation relating copies of T_M are of the form (ℓ, ℓ, x, y), so we can identify them with configurations of M, i.e., sometimes we will make no distinction between (ℓ, ℓ, x, y) and $(\ell, x, y) \in \texttt{Confs}(M)$ for $\ell \in L$.

We define the horizontal compatibility relations $H_M, H_M^0 \subseteq L \times L$ of the tiling system T_M as follows:

- $(\ell, m) \in H_M$ if and only if either of the instructions below is an instruction of machine M:
  - ℓ: `c`$_1$ `:= c`$_1$ `+ 1; goto` m
  - m: `if c`$_1$ `= 0 then c`$_1$ `:= c`$_1$ `+ 1; goto` n
    `else c`$_1$ `:= c`$_1$ `- 1; goto` ℓ
- $(\ell, m) \in H_M^0$ if and only if $(\ell, m) \in H_M$, or the instruction below is an instruction of machine M:
  - ℓ: `if c`$_1$ `= 0 then c`$_1$ `:= c`$_1$ `+ 1; goto` m
    `else c`$_1$ `:= c`$_1$ `- 1; goto` n

Vertical compatibility relations V_M, and V_M^0 are defined in the same way, with `c`$_1$ instructions replaced with `c`$_2$ instructions. We also take $D_M^{\rm ori} = L$, i.e., all dominoes are allowed in position $(0,0)$. Note that the identity function is a 1-1 correspondence between configurations of M, and configurations of the tiling system T_M; from now on we will hence identify configurations of M and T_M. It follows immediately from the construction of T_M, that two configurations $c, c' \in \texttt{Confs}(M)$ are compatible as configurations of T_M, if and only if $c \vdash_M c'$, or $c' \vdash_M c$, i.e., compatibility relation of T_M coincides with the symmetric closure of $\vdash_M$. By $\approx_M$ we denote the symmetric and transitive closure of the compatibility relation of configurations of T_M. The following proposition is then straightforward.

Proposition 6.16 The two relations $\sim_M$ and $\approx_M$ coincide.

Now we are ready to define the two origin constrained tiling systems T_1, and T_2, postulated in Proposition 6.12. The idea is to have two independent and slightly pruned copies of T_M in T_2: one without the initial configuration $(\texttt{start}, 0, 0)$, and the other without any halting configurations $(\texttt{halt}, x, y)$. The other tiling system T_1 is going to have three independent copies of T_M: the two of T_2, and moreover, another full copy of T_M.

More formally we define $D_2 = \big(L \times \{1, 2\}\big) \setminus \big\{(\texttt{halt}, 2)\big\}$, and $D_2^{\mathrm{ori}} = D_2 \setminus \big\{(\texttt{start}, 1)\big\}$, and $V_2 = \big((V_M \otimes 1) \cup (V_M \otimes 2)\big) \cap (D_2 \times D_2)$, where for a binary relation R we define $R \otimes i$ to be the relation $\big\{ \big((a, i), (b, i)\big) \,:\, (a, b) \in R \big\}$. The other compatibility relations V_2^0, H_2, and H_2^0 are defined analogously from the respective compatibility relations of T_M.

The tiling system T_1 is obtained from T_2 by adding yet another independent copy of T_M, this time a complete one: $D_1 = D_2 \cup (L \times \{3\})$, and $D_1^{\mathrm{ori}} = D_2^{\mathrm{ori}} \cup (L \times \{3\})$, and $V_1 = V_2 \cup (V_M \otimes 3)$, etc.. The labelling functions of T_1, and T_2 are defined as $\lambda_i\big((\ell, i)\big) = \ell$.

In order to show Proposition 6.12, and hence conclude the proof of Theorem 6.11, it suffices to establish the following two claims.

Claim 6.17 If machine M halts on input $(0, 0)$ then origin constrained tiling systems T_1 and T_2 are not domino bisimilar.

Proof: By Proposition 6.14 it suffices to show that if machine M halts on input $(0, 0)$ then Challenger has a winning strategy in the game $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$. Challenger starts by choosing the configuration $\big((\texttt{start}, 3), 0, 0\big)$ of T_1. Duplicator has to respond with domino $(\texttt{start}, 2)$ of T_2 since $(\texttt{start}, 1) \not\in D_2^{\mathrm{ori}}$. Then Challenger "simulates" the finite computation of M on input $(0, 0)$ in the following way. If $\big((\ell, 3), (\ell, 2), x, y\big)$ is the current configuration of the game then Challenger chooses the configuration $\big((\ell', 3), x', y'\big)$ of T_1, such that $\big(\ell, x, y\big) \vdash_M (\ell', x', y')$. This move is allowed thanks to Proposition 6.16. Then Duplicator can only respond with domino $(\ell', 2)$ of T_2, and $\big((\ell', 3), (\ell', 2), x', y'\big)$ becomes the current configuration of the game. In the last step of the simulation Challenger chooses a configuration $\big((\texttt{halt}, 3), x', y'\big)$ for some $x', y' \in \mathbb{N}$ which makes Duplicator stuck because $(\texttt{halt}, 2) \not\in D_2$. [Claim 6.17] ■

Claim 6.18 If machine M does not halt on input $(0, 0)$ then origin constrained tiling systems T_1 and T_2 are domino bisimilar.

Proof: By Proposition 6.14 it suffices to show that if machine M does not halt on input $(0, 0)$ then Duplicator has a winning strategy in the game $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$. We claim that the following is a winning strategy for Duplicator.

If in the first step Challenger chooses a configuration $\big((\ell, j), x, y\big)$ of T_1 or T_2 for $j \in \{1, 2\}$, then Duplicator responds with the domino (ℓ, j) of the other tiling system. It is obvious that then Duplicator can respond to all moves of Challenger because both players play on identical pruned copies of T_M.

If instead Duplicator chooses a configuration $\big((\ell, 3), x, y\big)$ of T_1 in the first step then Duplicator responds with:

- domino $(\ell, 1)$ of T_2 if $(\ell, x, y) \sim_M (\texttt{halt}, x', y')$ for some $x', y' \in \mathbb{N}$, and
- domino $(\ell, 2)$ of T_2 if $(\ell, x, y) \not\sim_M (\texttt{halt}, x', y')$ for all $x', y' \in \mathbb{N}$.

In the first case the only way Challenger can make Duplicator stuck is to be able to choose configuration $\big((\texttt{start}, 3), 0, 0\big)$ of T_1 since the only difference between copy 3 of T_M in T_1 and copy 1 of T_M in T_2 is that the latter does not have the triple $(\texttt{start}, 0, 0)$ as a configuration. Hence in order to prove that Duplicator has a winning strategy from the initial configuration $\big((\ell, 3), (\ell, 1), x, y\big)$, it suffices to show that $(\ell, x, y) \not\approx_M (\texttt{start}, 0, 0)$. Assume for the sake of contradiction that $(\ell, x, y) \approx_M (\texttt{start}, 0, 0)$. By Proposition 6.16 we then have $(\ell, x, y) \sim_M (\texttt{start}, 0, 0)$. This, by our assumption that $(\ell, x, y) \sim_M (\texttt{halt}, x', y')$ for some $x', y' \in \mathbb{N}$, implies that $(\texttt{start}, 0, 0) \sim_M (\texttt{halt}, x', y')$ for some $x', y' \in \mathbb{N}$. Then Proposition 6.15 implies that $(\texttt{start}, 0, 0) \vdash_M^+ (\texttt{halt}, x', y')$, which contradicts the assumption of the claim that machine M does not halt on input $(0, 0)$.

The argument in the other case is similar. It suffices to show that $(\ell, x, y) \not\approx_M (\texttt{halt}, x', y')$ for all $x', y' \in \mathbb{N}$, because the only difference between copy 3 of T_M in T_1 and copy 2 of T_M in T_2 is that the latter has no triple $(\texttt{halt}, x', y')$ as a configuration. By applying Proposition 6.16 to our assumption that $(\ell, x, y) \not\sim_M (\texttt{halt}, x', y')$ for all $x', y' \in \mathbb{N}$, we immediately get that $(\ell, x, y) \not\approx_M (\texttt{halt}, x', y')$ for all $x', y' \in \mathbb{N}$. [Claim 6.18] ■

6.4 Hhp-bisimilarity is undecidable

The proof of Theorem 6.6 is a reduction from the problem of deciding domino bisimilarity for origin constrained tiling systems. A method of encoding a tiling system on an infinite grid in the unfolding of a finite asynchronous transition system is due to Madhusudan and Thiagarajan [MT98]. For each origin constrained tiling system T, we define a finite asynchronous transition system $A(T)$, such that the following proposition holds.

Proposition 6.19 Origin constrained tiling systems T_1 and T_2 are domino bisimilar if and only if asynchronous transition systems $A(T_1)$ and $A(T_2)$ are hhp-bisimilar.

6.4.1 Asynchronous transition system $A(T)$

Let $T = \big(D, D^{\mathrm{ori}}, (H, H^0), (V, V^0), L, \lambda\big)$ be an origin constrained tiling system. The infinite grid structure is modelled by the unfolding of the asynchronous transition system shown in Figure 6.1. The set of events of this asynchronous transition system is $E = \{\texttt{x}_0, \texttt{x}_1, \texttt{x}_2, \texttt{x}_3, \texttt{x}_4, \texttt{y}_0, \texttt{y}_1, \texttt{y}_2, \texttt{y}_3, \texttt{y}_4\}$. The independence relation I is the symmetric closure of $\big\{ (\texttt{x}_i, \texttt{y}_j) \;:\; i, j \in \{0, 1, 2, 3, 4\} \big\}$.

We identify the states of the asynchronous transition system in Figure 6.1 with pairs of numbers $(i, j) \in \{0, 1, 2, 3, 4\}^2$, where i is the horizontal coordinate and j is the vertical coordinate. The state in the one in the bottom-left

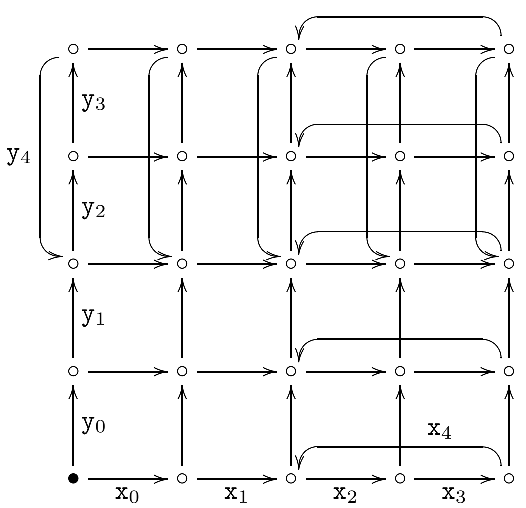


Figure 6.1: Modelling the infinite grid.

corner in Figure 6.1 is $(0,0)$. For all $n \in \mathbb{N}$, define:

$$\widehat{n} = \begin{cases} n & \text{if } n \leq 4, \\ 2 + \big((n-2) \bmod 3\big) & \text{if } n > 4. \end{cases}$$

A position $(n,m) \in \mathbb{N}^2$ of the infinite grid is represented by state $(\widehat{n}, \widehat{m})$ in the asynchronous transition system.

Configurations of the tiling system T are modelled by extra transitions going out of states of the grid structure in Figure 6.1, and labelled by events of the form d_{ij}, for $d \in D$ and $i,j \in \{0,1,2,3\}$. We define a set of events E_D as follows:

$$E_D = \{\ d_{ij}\ :\ d \in D;\ \text{and } i,j \in \{0,1,2,3\};\ \text{and } i=j=0 \text{ implies } d \in D^{\text{ori}}\ \}.$$

The idea is, for every $d \in D$, to have a transition going out of each state $(i,j) \in \{0,1,2,3,4\}^2$ labelled with d_{ij}, provided that (d,i,j) is a configuration of T. In fact, for a technical reason we need to use events d_{i1} and d_{1j} at states $(i,4)$ and $(4,j)$, respectively, instead of d_{i4} and d_{4j}. In order to avoid special treatment of this case throughout the rest of the paper we adopt the following notation, for all $n \in \mathbb{N}$:

$$\widetilde{n} = \begin{cases} n & \text{if } n \leq 3, \\ 1 + \big((n-1) \bmod 3\big) & \text{if } n > 3. \end{cases}$$

Horizontal and vertical compatibility of configurations of the tiling system T are modelled by an independence relation I_D on E_D, according to which events d_{ij} and $e_{k\ell}$ corresponding to "neighbouring" configurations are independent if and only if the configurations are compatible. More precisely, we define I_D to be the symmetric closure of the following set:

$$\begin{aligned}
&\big\{\ (d_{0j}, e_{1j})\ :\ j \in \{0,1,2,3\} \text{ and } (d,e) \in H_0\ \big\}\ \cup \\
&\big\{\ (d_{ij}, e_{\widehat{(i+1)}j})\ :\ i \in \{1,2,3\}, j \in \{0,1,2,3\}, \text{ and } (d,e) \in H\ \big\}\ \cup \\
&\big\{\ (d_{i0}, e_{i1})\ :\ i \in \{0,1,2,3\} \text{ and } (d,e) \in V_0\ \big\}\ \cup \\
&\big\{\ (d_{ij}, e_{i\widehat{(j+1)}})\ :\ i \in \{0,1,2,3\}, j \in \{1,2,3\}, \text{ and } (d,e) \in V\ \big\}.
\end{aligned}$$

For all $i, j \in \{0,1,2,3,4\}$ and $d \in D$, we have up to four transitions going out of state (i,j) and labelled by the following events in E_D: $d_{\widetilde{i}\,\widetilde{j}}$, $d_{(i-1)\widetilde{j}}$, if $i > 0$, $d_{\widetilde{i}(j-1)}$ if $j > 0$, and $d_{(i-1)(j-1)}$ if $i, j > 0$. We write $(i, j, \{d_{i'j'}\})$ to denote the state reached by the transition labelled by the event $d_{i'j'}$ going out of state (i,j). In other words, for all $i, j \in \{0,1,2,3,4\}$ we have the following transitions:

- $(i,j) \xrightarrow{d_{\widetilde{i}\,\widetilde{j}}} \big(i, j, \{d_{\widetilde{i}\,\widetilde{j}}\}\big)$,
- $(i,j) \xrightarrow{d_{(i-1)\widetilde{j}}} \big(i, j, \{d_{(i-1)\widetilde{j}}\}\big)$ if $i > 0$,
- $(i,j) \xrightarrow{d_{\widetilde{i}(j-1)}} \big(i, j, \{d_{\widetilde{i}(j-1)}\}\big)$ if $j > 0$,
- $(i,j) \xrightarrow{d_{(i-1)(j-1)}} \big(i, j, \{d_{(i-1)(j-1)}\}\big)$ if $i, j > 0$.

Moreover, if there are transitions:

- $(i,j) \xrightarrow{d_{k\ell}} \big(i, j, \{d_{k\ell}\}\big)$, and
- $(i,j) \xrightarrow{e_{k'\ell'}} \big(i, j, \{e_{k'\ell'}\}\big)$,

and $(d_{k\ell}, e_{k'\ell'}) \in I_D$, then there is also a state $\big(i, j, \{d_{k\ell}, e_{k'\ell'}\}\big)$ and transitions:

- $\big(i, j, \{d_{k\ell}\}\big) \xrightarrow{e_{k'\ell'}} \big(i, j, \{d_{k\ell}, e_{k'\ell'}\}\big)$, and
- $\big(i, j, \{e_{k'\ell'}\}\big) \xrightarrow{d_{k\ell}} \big(i, j, \{d_{k\ell}, e_{k'\ell'}\}\big)$.

Finally, there are transitions:

- $\big(i, j, \{d_{\widetilde{i}\,\widetilde{j}}\}\big) \xrightarrow{\mathrm{x}_i} \big(\widehat{i+1}, j, \{d_{\widetilde{i}\,\widetilde{j}}\}\big)$, if $\big(i, j, \{d_{\widetilde{i}\,\widetilde{j}}\}\big)$ is a state, and
- $\big(i, j, \{d_{\widetilde{i}(j-1)}\}\big) \xrightarrow{\mathrm{x}_i} \big(\widehat{i+1}, j, \{d_{\widetilde{i}(j-1)}\}\big)$, if $\big(i, j, \{d_{\widetilde{i}(j-1)}\}\big)$ is a state,

and transitions:

- $\big(i, j, \{d_{\widetilde{i}\,\widetilde{j}}\}\big) \xrightarrow{\mathrm{y}_j} \big(i, \widehat{j+1}, \{d_{\widetilde{i}\,\widetilde{j}}\}\big)$, if $\big(i, j, \{d_{\widetilde{i}\,\widetilde{j}}\}\big)$ is a state, and
- $\big(i, j, \{d_{(i-1)\widetilde{j}}\}\big) \xrightarrow{\mathrm{y}_j} \big(i, \widehat{j+1}, \{d_{(i-1)\widetilde{j}}\}\big)$, if $\big(i, j, \{d_{(i-1)\widetilde{j}}\}\big)$ is a state.

The sets of states $S_{A(T)}$ and transitions $\rightarrow_{A(T)}$ of the asynchronous transition system $A(T) = (S_{A(T)}, s^{\mathrm{ini}}_{A(T)} E_{A(T)}, \rightarrow_{A(T)}, \lambda_{A(T)}, I_{A(T)})$ are as described above. The set of events is defined by $E_{A(T)} = E \cup E_D$. The initial state is $s^{\mathrm{ini}}_{A(T)} = (0,0)$. The independence relation $I_{A(T)}$ is defined as the symmetric closure of the set:

$$I \cup I_D \cup \left\{ (\mathtt{x}_i, d_{\widetilde{i}\,\widetilde{j}}), (\mathtt{y}_j, d_{\widetilde{i}\,\widetilde{j}}) \; : \; i,j \in \{0,1,2,3,4\} \text{ and } d_{\widetilde{i}\,\widetilde{j}} \in E_D \right\}.$$

Finally, the labelling function $\lambda_{A(T)}$ is an identity on E, and for elements of E_D it replaces the dominoes with their labels in the tiling system T, i.e.,

$$\lambda_{A(T)}(e) = \begin{cases} e & \text{if } e \in E, \\ \big(\lambda(d)\big)_{ij} & \text{if } e \in E_D \text{ and } e = d_{ij}. \end{cases}$$

Proposition 6.20 The labelled transition systems $A(T)$ is a labelled asynchronous transition system.

6.4.2 The unfolding of $A(T)$

In this subsection we sketch the structure of the unfolding $\mathtt{Unf}\big(A(T)\big)$ of asynchronous transition system $A(T)$ defined in the previous subsection.

For notational convenience we will write $(i,j,\emptyset)$ for a state (i,j) of $A(T)$. In order to avoid a heavy use of notations $\widehat{n}$ and $\widetilde{m}$ we adopt the following conventions:

- we write $\mathtt{x}_n$ and $\mathtt{y}_m$, for all $n,m \in \mathbb{N}$, to denote events $\mathtt{x}_{\widehat{n}}, \mathtt{y}_{\widehat{m}} \in E$, respectively.
- we write d_{nm} to denote an event $d_{\widetilde{n}\widetilde{m}} \in E_D$, for all $n,m \in \mathbb{N}$.

Proposition 6.21 The set of states of $\mathtt{Unf}\big(A(T)\big)$ reachable from the initial state $(0,0,\emptyset)$ consists of triples $(n,m,C) \in \mathbb{N} \times \mathbb{N} \times \wp(E_D)$, such that either:

- $C = \emptyset$; or
- $C = \{d_{n'm'}\}$ such that $d_{n'm'} \in E_D$, and $n' \in \{n-1, n\}$, and $m' \in \{m-1, m\}$; or
- $C = \{d_{(n-1)m'}, e_{nm'}\}$ such that $d_{(n-1)m'}, e_{nm'} \in E_D$, and $m' \in \{m-1, m\}$, and configurations $(d, n-1m')$ and (e, n, m') of T are compatible; or
- $C = \{d_{n'(m-1)}, e_{n'm}\}$ such that $d_{n'(m-1)}, e_{n'm} \in E_D$, and $n' \in \{n-1, n\}$, and configurations $(d, n', m-1)$ and (e, n', m) of T are compatible.

States of the first category above represent positions on the infinite grid; in particular the state $(n,m,\emptyset)$ represents the position $(n,m) \in \mathbb{N} \times \mathbb{N}$. States of the second category above represent configurations of the tiling system T; in particular configuration $(d,n,m) \in D \times \mathbb{N} \times \mathbb{N}$ is represented by states

$(n', m', \{d_{nm}\})$ for $n' \in \{n, n+1\}$ and $m' \in \{m, m+1\}$. States of the third and forth categories above are used to "check compatibility" of neighbouring configurations of tiling system T.

Proposition 6.22 The set of transitions of $\mathtt{Unf}(A(T))$ consists of the following:

- $(n, m, C) \xrightarrow{\mathtt{x}_n}_{\mathtt{Unf}(A(T))} (n+1, m, C)$
for $C = \emptyset$, or $C = \{d_{nm'}\}$ for $m' \in \{m-1, m\}$,
- $(n, m, C) \xrightarrow{\mathtt{y}_m}_{\mathtt{Unf}(A(T))} (n, m+1, C)$
for $C = \emptyset$, or $C = \{d_{n'm}\}$ for $n' \in \{n-1, n\}$,
- $(n, m, \emptyset) \xrightarrow{d_{n'm'}}_{\mathtt{Unf}(A(T))} (n, m, \{d_{n'm'}\})$
for $n' \in \{n-1, n\}$, and $m' \in \{m-1, m\}$, and $d_{n'm'} \in E_D$,
- $(n, m, \{d_{(n-1)m'}\}) \xrightarrow{e_{nm'}}_{\mathtt{Unf}(A(T))} (n, m, \{d_{(n-1)m'}, e_{nm'}\})$ and
$(n, m, \{e_{nm'}\}) \xrightarrow{d_{(n-1)m'}}_{\mathtt{Unf}(A(T))} (n, m, \{d_{(n-1)m'}, e_{nm'}\})$,
for $m' \in \{m-1, m\}$ if configurations $(d, n-1, m')$ and (e, n, m') of T are compatible,
- $(n, m, \{d_{n'(m-1)}\}) \xrightarrow{e_{n'm}}_{\mathtt{Unf}(A(T))} (n, m, \{d_{n'(m-1)}, e_{n'm}\})$ and
$(n, m, \{e_{n'm}\}) \xrightarrow{d_{n'(m-1)}}_{\mathtt{Unf}(A(T))} (n, m, \{d_{n'(m-1)}, e_{n'm}\})$
for $n' \in \{n-1, n\}$, if configurations $(d, n', m-1)$ and (e, n', m) of T are compatible.

6.4.3 Translations between hhp- and domino bisimulations

By Proposition 6.5 it follows that in order to prove Proposition 6.19 it suffices to demonstrate that a domino bisimulation relating T_1 and T_2 gives rise to an hhp-bisimulation relating $\mathtt{Unf}(A(T_1))$ and $\mathtt{Unf}(A(T_2))$, and vice versa. In other words, it suffices to argue that a winning strategy for Duplicator in $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$ can be translated to a winning strategy for her in $\mathcal{B}_{\mathrm{hhp}}(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2)))$, and vice versa. In what follows, in order to keep the arguments from becoming too dull or cumbersome, we are mixing freely at our convenience the two ways of talking about bisimulations: as relations, and as winning strategies in bisimilarity checking games.

For notational convenience we introduce the following convention for writing elements of an hhp-bisimulation relating $\mathtt{Unf}(A(T_1))$ and $\mathtt{Unf}(A(T_2))$, or equivalently, for configurations of game $\mathcal{B}_{\mathrm{hhp}}(\mathtt{Unf}(A(T_1)), \mathtt{Unf}(A(T_2)))$. Note that if a pair of runs $(\overline{e_1}, \overline{e_2}) \in \mathtt{Runs}(\mathtt{Unf}(A(T_1))) \times \mathtt{Runs}(\mathtt{Unf}(A(T_2)))$ belongs to an hhp-bisimulation then the states reached by these runs are of the forms (n, m, C_1) and (n, m, C_2) for some $n, m \in \mathbb{N}$, respectively. In what follows we write (n, m, C_1, C_2) to denote such a pair $(\overline{e_1}, \overline{e_2})$. This notation is a bit sloppy because it is not 1-1. For example, $(1, 1, \emptyset)$ is used to denote both $(\mathtt{x}_0\mathtt{y}_0, \mathtt{x}_0\mathtt{y}_0)$ and $(\mathtt{y}_0\mathtt{x}_0, \mathtt{y}_0\mathtt{x}_0)$. It is not hard to see that this sloppiness is not a problem here.

From domino to hhp-bisimulation. Let $B \subseteq D_1 \times D_2 \times \mathbb{N} \times \mathbb{N}$ be a domino bisimulation relating T_1 and T_2. We define a winning strategy for Duplicator in game $\mathcal{B}_{\text{hhp}}(\texttt{Unf}(A(T_1)), \texttt{Unf}(A(T_2)))$ in the following way.

If Challenger makes a backward move then the response of Duplicator is determined uniquely. Moreover, this response can always be performed because asynchronous transition systems $\texttt{Unf}(A(T_1))$ and $\texttt{Unf}(A(T_2))$ have the property that every pair of runs with equal labelling sequences has equal sets of most recent events. If Challenger makes a forward move by choosing an event $\texttt{x}_n$ or $\texttt{y}_m$, for $n, m \in \mathbb{N}$, then Duplicator responds with the same event in the other transition system.

The only non-trivial responses of Duplicator are the ones to be made when Challenger makes a forward move by choosing an event of the form d_{nm}, where d is a domino and $n, m \in \mathbb{N}$. We define these responses by referring to the domino bisimulation B. The strategy for Duplicator we define below has the following property.

Property 6.23 Suppose that a configuration (n, m, C_1, C_2) of an hhp-bisimilarity checking game $\mathcal{B}_{\text{hhp}}(\texttt{Unf}(A(T_1)), \texttt{Unf}(A(T_2)))$ can be reached from the initial configuration while Duplicator is playing according to the strategy. Then $d_{n'm'} \in C_1$ and $e_{n'm'} \in C_2$ for $n' \in \{n-1, n\}$ and $m' \in \{m-1, m\}$ imply that $(d, e, n', m') \in B$

Suppose without loss of generality that Challenger makes a move in $\texttt{Unf}(A(T_1))$; the other case is symmetric. We consider several cases depending on the current configuration of $\mathcal{B}_{\text{hhp}}(\texttt{Unf}(A(T_1)), \texttt{Unf}(A(T_2)))$.

- The current configuration of the game $\mathcal{B}_{\text{hhp}}(\texttt{Unf}(A(T_1)), \texttt{Unf}(A(T_2)))$ is $(n, m, \emptyset, \emptyset)$ for some $n, m \in \mathbb{N}$. Challenger can choose an event $d_{n'm'}$, such that $n' \in \{n-1, n\}$ and $m' \in \{m-1, m\}$. Then Duplicator responds with an event $e_{n'm'}$ in $\texttt{Unf}(A(T_2))$, such that $(d, e, n', m') \in B$.

- The current configuration of the game $\mathcal{B}_{\text{hhp}}(\texttt{Unf}(A(T_1)), \texttt{Unf}(A(T_2)))$ is $(n, m, \{d_{n'm'}\}, \{e_{n'm'}\})$, such that $n' \in \{n-1, n\}$ and $m' \in \{m-1, m\}$. Challenger can choose an event $d'_{k\ell}$, such that either $k = n'$ and $\{m', \ell\} = \{m-1, m\}$, or $\ell = m'$ and $\{n', k\} = \{n-1, n\}$. In both cases Duplicator responds with an event $e'_{k\ell}$, such that configurations (e, n', m') and (e', k, ℓ) of T_2 are compatible, and $(d', e', k, \ell) \in B$.

Note that all the responses we have defined above are indeed possible due to Property 6.23 and definition of a domino bisimulation, and moreover, they maintain Property 6.23.

From hhp- to domino bisimulation. Let B be an hhp-bisimulation relating $\texttt{Unf}(A(T_1))$ and $\texttt{Unf}(A(T_2))$. We define a winning strategy for Duplicator in game $\mathcal{B}_{\text{d}}(T_1, T_2)$. The strategy for Duplicator we define below has the following property.

Property 6.24 If configuration (d, e, n, m) of $\mathcal{B}_{\rm d}(T_1, T_2)$ can be reached while Duplicator is playing according to the strategy then $\big(n, m, \{d_{nm}\}, \{e_{nm}\}\big) \in B$.

Suppose without loss of generality that Challenger makes a move in T_1; the other case is symmetric. We consider the two kinds of moves possible in a domino bisimulation game.

- In order to fix an initial configuration of $\mathcal{B}_{\rm d}(T_1, T_2)$ Challenger chooses a configuration (d, n, m) of T_1. Note that for all $n, m \in \mathbb{N}$, we have that $(n, m, \emptyset, \emptyset) \in B$. Let e_{nm} be Duplicator's response if Challenger makes a forward move in $\mathcal{B}_{\rm hhp}\big(\mathtt{Unf}\big(A(T_1)\big), \mathtt{Unf}\big(A(T_2)\big)\big)$ by choosing event d_{nm} in configuration $(n, m, \emptyset, \emptyset)$. Then we take e to be Duplicator's response to Challenger's choice of configuration (d, n, m).

- Let (d, e, n, m) be the current configuration of $\mathcal{B}_{\rm d}(T_1, T_2)$. In a next move Challenger can choose a configuration (d', n', m') of T_1 compatible with (d, n, m). We consider cases when $(n', m') = (n-1, m)$ and $(n', m') = n, m+1)$; the other two cases are analogous. Note that by Property 6.24 we have that
$$\big(n, m, \{d_{nm}\}, \{e_{nm}\}\big) \in B. \tag{6.1}$$

  - Let $(n', m') = (n-1, m)$. Since configurations (d, n, m) and $(d', n-1, m)$ of T_1 are compatible, by applying condition 2. of the definition of an hhp-bisimulation to (6.1) we get that there is a domino e' of T_2, such that configurations (e, n, m) and $(e', n-1, m)$ of T_2 are compatible, and
  $$\big(n, m, \{d'_{(n-1)m}, d_{nm}\}, \{e'_{(n-1)m}, e_{nm}\}\big) \in B. \tag{6.2}$$
  We define event e' to be Duplicator's response in $\mathcal{B}_{\rm d}(T_1, T_2)$ for Challenger's move consisting of choosing configuration $(d', n-1, m)$ of T_1. By applying condition 4. of definition of an hhp-bisimulation to (6.2) twice we get that
  $$\big(n-1, m, \{d'_{(n-1)m}\}, \{e'_{(n-1)m}\}\big) \in B.$$

  - Let $(n', m') = (n, m+1)$. By applying condition 2. of definition of an hhp-bisimulation to (6.1) we get that
  $$\big(n, m+1, \{d_{nm}\}, \{e_{nm}\}\big) \in B. \tag{6.3}$$
  Since configurations (d, n, m) and $(d', n, m+1)$ of T_1 are compatible, by applying condition 2. of definition of an hhp-bisimulation to (6.3) we get that there is a domino e' of T_2, such that configurations (e, n, m) and $(e', n, m+1)$ of T_2 are compatible, and
  $$\big(n, m+1, \{d_{nm}, d'_{n(m+1)}\}, \{e_{nm}, e'_{n(m+1)}\}\big) \in B. \tag{6.4}$$

We define event e' to be Duplicator's response in $\mathcal{B}_{\mathrm{d}}(T_1, T_2)$ for Challenger's move consisting of choosing configuration $(d', n, m+1)$ of T_1. By applying condition 4. of definition of an hhp-bisimulation to (6.4) we get

$$\big(n, m+1, \{d'_{n(m+1)}\}, \{e'_{n(m+1)}\}\big) \in B.$$

Note that all the responses we have defined above are indeed possible due to Property 6.24 and definition of a domino bisimulation, and moreover, they maintain Property 6.24.

6.4.4 Finite elementary net system $N(T)$

In this subsection we argue that undecidability of hhp-bisimilarity for finite elementary net systems follows as a corollary of our proof for finite asynchronous transition systems.

Given a tiling system T we define an elementary net system $N(T)$ and we argue that $A(T)$ is isomorphic to the asynchronous transition system $na\big(N(T)\big)$ corresponding to the net $N(T)$. This immediately implies the following fact.

Theorem 6.25
Hhp-bisimilarity is undecidable for finite labelled elementary net systems.

The elementary net system $N(T) = (P_{N(T)}, E_{N(T)}, \mathtt{pre}_{N(T)}, \mathtt{post}_{N(T)}, M_{N(T)})$ is shown in Figure 6.2 and it consists of the following:

- the set of conditions

$$P_{N(T)} = \big\{\mathtt{a}_i, \mathtt{b}_i : i \in \{0,1,2,3,4\}\big\} \cup \big\{\mathtt{a}^j_{i(i+1)}, \mathtt{b}^i_{j(j+1)} : i, j \in \{0,1,2,3\}\big\} \cup E_D;$$

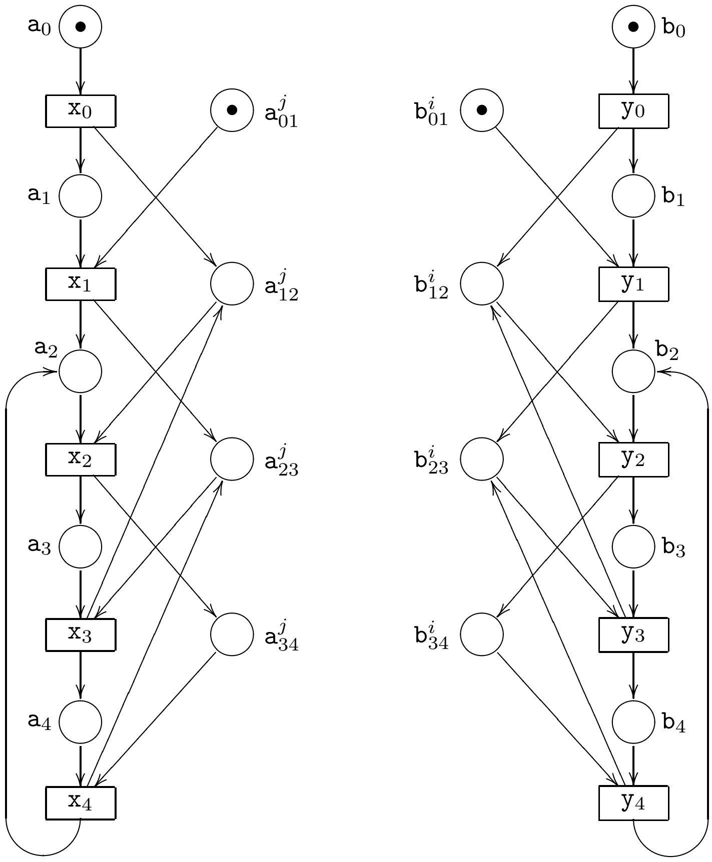
- the set of events $E_{N(T)} = E_{A(T)}$;

- the function $\mathtt{pre}_{N(T)} : E_{N(T)} \to \wp(P_{N(T)})$ specifying the set of places in the pre-condition of an event:

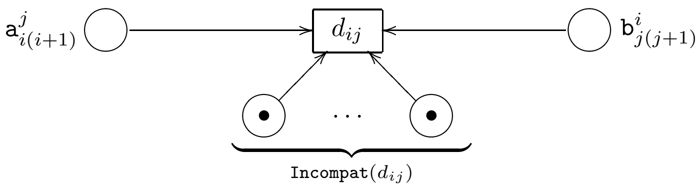
$$\mathtt{pre}_{N(T)}(e) = \begin{cases} \{\mathtt{a}_0\} & \text{if } e = \mathtt{x}_0, \\ \{\mathtt{b}_0\} & \text{if } e = \mathtt{y}_0, \\ \{\mathtt{a}_i\} \cup \mathtt{A}_{(i-1)i} & \text{if } e = \mathtt{x}_i \text{ for } i \in \{1,2,3,4\}, \\ \{\mathtt{b}_j\} \cup \mathtt{B}_{(j-1)j} & \text{if } e = \mathtt{y}_j \text{ for } j \in \{1,2,3,4\}, \\ \{\mathtt{a}^j_{i(i+1)}, \mathtt{b}^i_{j(j+1)}\} \cup \mathtt{Incompat}(d_{ij}) & \text{if } e = d_{ij} \in E_D, \end{cases}$$

where for $i, j \in \{0,1,2,3\}$, we define $\mathtt{A}_{i(i+1)} = \big\{\ \mathtt{a}^k_{i(i+1)}\ :\ k \in \{0,1,2,3\}\ \big\}$ and $\mathtt{B}_{j(j+1)} = \big\{\ \mathtt{b}^k_{j(j+1)}\ :\ k \in \{0,1,2,3\}\ \big\}$, and for $d_{ij} \in E_D$, we define

$$\mathtt{Incompat}(d_{ij}) = \{\ e_{k\ell} \in E_D\ :\ (d_{ij}, e_{k\ell}) \notin I_D\ \};$$



Variables i and j range over $\{0,1,2,3\}$. Therefore we have four copies of each place in the second column, for $j \in \{0,1,2,3\}$, and four copies of each place in the third column, for $i \in \{0,1,2,3\}$.



The preconditions of each $d_{ij} \in E_D$, are the two places $\mathtt{a}^j_{i(i+1)}$ and $\mathtt{b}^i_{j(j+1)}$, and all the places in $\mathtt{Incompat}(d_{ij}) = \{ e_{k\ell} \in E_D \ : \ (d_{ij}, e_{k\ell}) \notin I_D \}$.

Figure 6.2: The elementary net system $N(T)$.

- the function $\mathtt{post}_{N(T)} : E_{N(T)} \to \wp(P_{N(T)})$ specifying the set of places in the post-condition of an event:

$$\mathtt{post}_{N(T)}(e) = \begin{cases} \{\mathtt{a}_{i+1}\} \cup \mathtt{A}_{(i+1)(i+2)} & \text{if } e = \mathtt{x}_i \text{ for } i \in \{0,1,2\}, \\ \{\mathtt{a}_4\} \cup \mathtt{A}_{12} & \text{if } e = \mathtt{x}_3, \\ \{\mathtt{a}_2\} \cup \mathtt{A}_{23} & \text{if } e = \mathtt{x}_4, \\ \{\mathtt{b}_{j+1}\} \cup \mathtt{B}_{(j+1)(j+2)} & \text{if } e = \mathtt{y}_j \text{ for } j \in \{0,1,2\}, \\ \{\mathtt{b}_4\} \cup \mathtt{B}_{12} & \text{if } e = \mathtt{y}_3, \\ \{\mathtt{b}_2\} \cup \mathtt{B}_{23} & \text{if } e = \mathtt{y}_4, \\ \emptyset & \text{if } e \in E_D; \end{cases}$$

- the initial marking $M_{N(T)} = \{\mathtt{a}_0, \mathtt{b}_0\} \cup \mathtt{A}_{01} \cup \mathtt{B}_{01} \cup E_D$.

Proposition 6.26 The asynchronous transition system $A(T)$ is isomorphic to $na\big(N(T)\big)$.

Proof. We define a function $\Xi : S_{A(T)} \to \wp(P_{N(T)})$ as follows:

$$\Xi\big((i,j,C)\big) = \{\mathtt{a}_i, \mathtt{b}_j\} \cup \mathtt{X}_i \cup \mathtt{Y}_j \cup E_D \setminus \bigcup_{c \in C} \mathtt{pre}_{N(T)}(c),$$

where

$$\mathtt{X}_i = \begin{cases} \mathtt{A}_{01} & \text{if } i = 0, \\ \mathtt{A}_{(i-1)i} \cup \mathtt{A}_{i(i+1)} & \text{if } i \in \{1,2,3\}, \\ \mathtt{A}_{34} \cup \mathtt{A}_{12} & \text{if } i = 4, \end{cases}$$

and similarly

$$\mathtt{Y}_j = \begin{cases} \mathtt{B}_{01} & \text{if } j = 0, \\ \mathtt{B}_{(j-1)j} \cup \mathtt{B}_{j(j+1)} & \text{if } j \in \{1,2,3\}, \\ \mathtt{B}_{34} \cup \mathtt{B}_{12} & \text{if } j = 4. \end{cases}$$

In order to argue that Ξ is an isomorphism of asynchronous transition systems $A(T)$ and $na\big(N(T)\big)$ it suffices to establish the following:

1. $\Xi(s^{\mathrm{ini}}_{A(T)}) = M_{N(T)}$, i.e., the initial state of $A(T)$ is mapped by Ξ to the initial marking of $N(T)$,
2. for all $s \in S_{A(T)}$,
   (a) if $s \xrightarrow{e}_{A(T)} t$ then $\Xi(s) \xrightarrow{e}_{na(N(T))} \Xi(t)$,
   (b) if $\Xi(s) \xrightarrow{e}_{na(N(T))} M$ then there is $t \in S_{A(T)}$, such that $s \xrightarrow{e}_{A(T)} t$ and $M = \Xi(t)$,
3. $(e,f) \in I_{A(T)}$ if and only if ${}^\bullet e^\bullet \cap {}^\bullet f^\bullet = \emptyset$.

It is a routine exercise to verify that clauses 1.–3. hold. [Proposition 6.26] ■

Bibliography

[AC88] André Arnold and Paul Crubille. A linear algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29(2):57–66, 1988.

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, Florida, October 1997.

[ALW89] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium*, volume 372 of *LNCS*, pages 1–17, Stresa, Italy, 11–15 July 1989. Springer-Verlag.

[And94] Henrik Reif Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, 1994. A preliminary version appeared in Proc. of ESOP'92.

[Arn99] André Arnold. The μ-calculus alternation hierarchy is strict on binary trees. *Theoretical Informatics and Applications*, 33:329–339, 1999.

[BCJ$^+$97] A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science*, 178(1–2):237–255, May 1997. A preliminary version appeared in Proceedings of CAV'94, volume 818 of LNCS, Springer-Verlag.

[BCM$^+$92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: $10^2 0$ states and beyond. *Information and Computation*, 98(2):142–170, 1992. An earlier version of the paper appears in Proceedings of LICS'90.

[Bed88] Marek A. Bednarczyk. *Categories of Asynchronous Systems*. PhD thesis, University of Sussex, 1988.

[Bed91] Marek A. Bednarczyk. Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical report, Polish Academy of Sciences, Gdańsk, April 1991. Available at `http://www.ipipan.gda.pl/~marek`.

[BL69] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[BLV96] Nils Buhrke, Helmut Lescow, and Jens Vöge. Strategy construction in infinite games with Streett and Rabin chain winning conditions. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96*, volume 1055 of *LNCS*, pages 207–224, Passau, Germany, 27–29 March 1996. Springer-Verlag.

[Bra98] J. C. Bradfield. The modal mu-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195(2):133–153, 30 March 1998. A preliminary version appeared in Proceedings of CONCUR'96.

[BS00] Julian Bradfield and Colin Stirling. Modal logics and mu-calculi: an introduction. To appear in Handbook of Process Algebra. Available from `http://www.dcs.ed.ac.uk/home/jcb/Research/`, February 2000.

[Büc60] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Logic Grundl. Math.*, 6:66–92, 1960.

[Büc62] J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. 1960 International Congress for Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[Büc83] J. R. Büchi. State-strategies for games in $F_{\sigma\delta} \cap G_{\delta\sigma}$. *Journal of Symbolic Logic*, 48:1171–1198, 1983.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs Workshop*, volume 131 of *LNCS*, pages 52–71, IBM Yorktown Heights, New York, May 1981. Springer-Verlag.

[CGL94] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, Proceedings*, volume 803 of *LNCS*, pages 124–175, Noordwijkerhout, The Netherlands, June 1994. Springer-Verlag.

[CGP99] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[CHS95] Søren Christensen, Hans Hüttel, and Colin Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121(2):143–148, 1995.

[Chu63] A. Church. Logic, arithmetic and automata. In *Proc. Intern. Congr. Math.*, pages 21–35, Uppsala, 1963.

[CKS81] A. K. Chandra, D. C. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[CKS92] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking in the modal mu-calculus. In G.v. Bochmann and D.K. Probst, editors, *Computer Aided Verification, 4th International Workshop, CAV'92, Proceedings*, volume 663 of *LNCS*, pages 410–422, Montreal, Canada, 1992. Springer-Verlag.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[Con92] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.

[Con93] Anne Condon. On algorithms for simple stochastic games. In Jin-Yi Cai, editor, *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.

[CS91] Rance Cleaveland and Bernhard Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In Kim Guldstrand Larsen and Arne Skou, editors, *Computer Aided Verification, 3rd International Workshop, CAV '91, Proceedings*, volume 575 of *LNCS*, pages 48–58, Aalborg, Denmark, 1–4 July 1991. Springer-Verlag.

[CS96] Gian Luca Cattani and Vladimiro Sassone. Higher dimensional transition systems. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 55–62, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

[Dam94] Mads Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126(1):77–96, 11 April 1994.

[DJW97] Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 99–110, Warsaw, Poland, 29 June–2 July 1997. IEEE Computer Society Press.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EHM94] Yael Etzion-Petruschka, David Harel, and Dale Myers. On the solvability of domino snake problems. *Theoretical Computer Science*, 131:243–269, 1994.

[EJ88] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. In *29th Annual Symposium on Foundations of Computer Science*, pages 328–337, White Plains, New York, 24–26 October 1988. IEEE Computer Society Press.

[EJ91] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (Extended abstract). In *32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, 1–4 October 1991. IEEE Computer Society Press.

[EJ99] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal of Computing*, 29(1):132–158, 1999. A preliminary version appeared in Proc. of FOCS'88.

[EJS93] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In Costas Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV'93*, volume 697 of *LNCS*, pages 385–396, Elounda, Greece, June/July 1993. Springer-Verlag.

[EL86] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (Extended abstract). In *Proceedings, Symposium on Logic in Computer Science*, pages 267–278, Cambridge, Massachusetts, 16–18 June 1986. IEEE.

[Elg61] C. C. Elgot. Decision problems of finite automata design and related arithemtics. *Transactions of American Mathematical Society*, 98:21–52, 1961.

[EM79] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979.

[Eme96] E. A. Emerson. Automated temporal reasoning about reactive systems. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043, pages 41–101. Springer-Verlag, 1996.

[FH99] Sibylle Fröschle and Thomas Hildebrandt. On plain and hereditary history-preserving bisimulation. In Mirosław Kutyłowski, Leszek Pacholski, and Tomasz Wierzbicki, editors, *Mathematical*

Foundations of Computer Science 1999, 24th International Symposium, MFCS'99, volume 1672 of *LNCS*, pages 354–365, Szklarska Poręba, Poland, 6–10 September 1999. Springer-Verlag.

[FL79] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Science*, 18:194–211, 1979.

[Frö00] Sibylle Fröschle. Decidability of plain and hereditary history-preserving bisimilarity for BPP. In Ilaria Castellani and Björn Victor, editors, *Proceedings of EXPRESS'99 the 6th International Workshop on Expressiveness in Concurrency*, volume 27 of *Electronic Notes in Theoretical Computer Science*, 2000.

[GG89] Rob van Glabbeek and Ursula Goltz. Equivalence notions for concurrent systems and refinement of actions (Extended abstract). In A. Kreczmar and G. Mirkowska, editors, *Mathematical Foundations of Computer Science 1989*, volume 379 of *LNCS*, pages 237–248, Porąbka-Kozubnik, Poland, August/September 1989. Springer-Verlag.

[GH82] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 60–65, San Francisco, California, 5–7 May 1982. ACM Press.

[GH94] Jan Friso Groote and Hans Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):354–371, 1994.

[GKK88] V. A. Gurvich, A. V. Karzanov, and L. G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 28(5):85–91, 1988.

[Gla90] R. J. van Glabbeek. The linear time-branching time spectrum (Extended abstract). In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension*, volume 458 of *LNCS*, pages 278–297, Amsterdam, The Netherlands, 27–30 August 1990. Springer-Verlag.

[Grä90] Erich Grädel. Domino games and complexity. *SIAM Journal on Computing*, 19(5):787–804, 1990.

[Har85] David Harel. Recurring dominos: Making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.

[HJM96] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158(1–2):143–159, 1996.

[HK66] A. Hoffman and R. Karp. On nonterminating stochastic games. *Management Science*, 12:359–370, 1966.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[HS85] Matthew Hennessy and Colin Stirling. The power of the future perfect in program logics. *Information and Control*, 67(1–3):23–52, 1985.

[JM96] Lalita Jategaonkar and Albert R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154:107–143, 1996.

[JN00] Marcin Jurdziński and Mogens Nielsen. Hereditary history preserving bisimilarity is undecidable. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 358–369, Lille, France, February 2000. Springer-Verlag.

[JNW96] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996. A preliminary version appeared in *Proceedings of Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 418–427, Montreal, Canada, June 1993. IEEE Computer Society Press.

[Jur97] Marcin Jurdziński. O złożoności obliczeniowej gier na grafach (in Polish). Master's thesis, Instytut Informatyki, Uniwersytet Warszawski, Warszawa, Poland, June 1997. Title translation: On the Computational Complexity of Games on Graphs.

[Jur98] Marcin Jurdziński. Deciding the winner in parity games is in UP $\cap$ co-UP. *Information Processing Letters*, 68(3):119–124, November 1998.

[Jur00] Marcin Jurdziński. Small progress measures for solving parity games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, Lille, France, February 2000. Springer-Verlag.

[JW95] David Janin and Igor Walukiewicz. Automata for the modal mu-calulus and related results. In Jirí Wiedermann and Petr Hájek, editors, *Mathematical Foundations of Computer Science 1995, 20th International Symposium, MFCS'95, Proceedings*, volume 969 of *LNCS*, pages 552–562, Prague, Czech Republic, 28 August–1 September 1995. Springer-Verlag.

[KK91] Nils Klarlund and Dexter Kozen. Rabin measures and their applications to fairness and automata theory. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 256–265, Amsterdam, The Netherlands, 15–18 July 1991. IEEE Computer Society Press.

[Kla91] Nils Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *32nd Annual Symposium on Foundations of Computer Science*, pages 358–367, San Juan, Puerto Rico, 1–4 October 1991. IEEE Computer Society Press.

[Kla94] Nils Klarlund. Progress measures, immediate determinacy, and a subset construction for tree automata. *Annals of Pure and Applied Logic*, 69(2–3):243–268, 1994. A preliminary version appeared in Proceedings of LICS'92, IEEE Computer Society Press.

[Koz83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[KS90] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[KV98] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 224–233, Dallas, Texas, USA, 23–26 May 1998. ACM Press.

[KVW00] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000. A preliminary version appeared in Proc. of CAV'94.

[Len96] Giacomo Lenzi. A hierarchy theorem for the μ-calculus. In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *Automata, Languages and Programming, 23rd International Colloquium*, 1996.

[Les95] H. Lescow. On polynomial-size programs winning finite-state games. In P. Wolper, editor, *Computer Aided Verification, 7th International Conference, CAV'95*, volume 939 of *LNCS*, pages 239–252, Liege, Belgium, July 1995. Springer-Verlag.

[LRS98] Xinxin Liu, C. R. Ramakrishnan, and Scott A. Smolka. Fully local and efficient evaluation of alternating fixed points. In Bernhard Steffen, editor, *Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS '98*, volume 1384 of *LNCS*, pages 5–19, Lisbon, Portugal, 28 March–4 April 1998. Springer-Verlag.

[MC94] Mary Melekopoglou and Anne Condon. On the complexity of the policy improvement algorithm for Markov decision processes. *ORSA Journal of Computing*, 6(2):188–192, 1994. Operations Research Society of America.

[McN93] Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[Meg83] Nimrod Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM Journal on Computing*, 12:347–353, 1983.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, 1980.

[Mos91] A. W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdańsk, 1991.

[MS95] Faron Moller and Scott A. Smolka. On the computational complexity of bisimulation. *ACM Computing Surveys*, 27(2):287–289, 1995.

[MT98] P. Madhusudan and P. S. Thiagarajan. Controllers for discrete event systems via morphisms. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR'98, Concurrency Theory, 9th International Conference, Proceedings*, volume 1466 of *LNCS*, pages 18–33, Nice, France, September 1998. Springer-Verlag.

[NC95] Mogens Nielsen and Christian Clausen. Games and logics for a noninterleaving bisimulation. *Nordic Journal of Computing*, 2(2):221–249, 1995.

[Niw86] Damian Niwiński. On fixed-point clones (Extended abstract). In Laurent Kott, editor, *Automata, Languages and Programming, 13th International Colloquium, ICALP'86*, volume 226 of *LNCS*, pages 464–473, Rennes, France, 15–19 July 1986. Springer-Verlag.

[Niw88] Damian Niwiński. Fixed points vs. infinite generation. In *Proceedings, Third Annual Symposium on Logic in Computer Science*, pages 402–409, Edinburgh, Scotland, 5–8 July 1988. IEEE Computer Society Press.

[Niw97] Damian Niwiński. Fixed point characterization of infinite behavior of finite-state systems. *Theoretical Computer Science*, 189(1–2):1–69, 1997.

[NRY96] Anil Nerode, Jeffrey B. Remmel, and Alexander Yahknis. McNaughton games and extracting strategies for concurrent programs. *Annals of Pure and Applied Logic*, 78:203–242, 1996.

[NW96a] Mogens Nielsen and Glynn Winskel. Petri nets and bisimulation. *Theoretical Computer Science*, 153(1–2):211–244, 1996.

[NW96b] Damian Niwiński and Igor Walukiewicz. Games for the mu-calculus. *Theoretical Computer Science*, 163(1–2):99–116, 1996.

[NW98] Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1373 of *LNCS*, pages 320–331, Paris, France, 25–27 February 1998. Springer-Verlag.

[NYY92] Anil Nerode, Alexander Yakhnis, and Vladimir Yakhnis. Concurrent programs as strategies in games. In Y. N. Moschovakis, editor, *Logic from Computer Science, Proceedings of a Workshop*, volume 21 of *Mathematical Sciences Research Institute Publications*, pages 405–479. Springer-Verlag, 13–17November 1992.

[Ott99] Martin Otto. Eliminating recursion in the μ-calculus. In Christoph Meinel and Sophie Tison, editors, *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1563, pages 531–540, Trier, Germany, 4–6 March 1999.

[Par81] D. M. R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical Computer Science: 5th GI-Conference*, volume 104 of *LNCS*, pages 167–183. Springer-Verlag, 1981.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Providence, Rhode Island, 31 October–2 November 1977. IEEE Computer Society Press.

[PR89a] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL '89)*, pages 179–190, Austin, Texas, January 1989. ACM Press.

[PR89b] Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium*, volume 372 of *LNCS*, pages 652–671, Stresa, Italy, 11–15 July 1989. Springer-Verlag.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[Pur95] Anuj Puri. *Theory of Hybrid Systems and Discrete Event Systems*. PhD thesis, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, December 1995. Memorandum No. UCB/ERL M95/113.

[Rab69] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of American Mathematical Society*, 141:1–35, 1969.

[RP80] John H. Reif and Gary L. Peterson. A dynamic logic of multiprocessing with incomplete information. In *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages*, pages 193–202, Las Vegas, Nevada, 28–30 January 1980. ACM Press.

[RT88] A. Rabinovich and B. Trakhtenbrot. Behaviour structures and nets of processes. *Fundamenta Informaticae*, 11:357–404, 1988.

[SE89] Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81(3):249–264, 1989.

[Sei96] Helmut Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, September 1996.

[Shi85] M. W. Shields. Concurrent machines. *Computer Journal*, 28:449–465, 1985.

[Sti95] Colin Stirling. Local model checking games (Extended abstract). In Insup Lee and Scott A. Smolka, editors, *CONCUR'95: Concurrency Theory, 6th International Conference*, volume 962 of *LNCS*, pages 1–11, Philadelphia, Pennsylvania, 21–24 August 1995. Springer-Verlag.

[Sti97] Colin Stirling. Bisimulation, model checking and other games. Notes for Mathfit Instructional Meeting on Games and Computation, available at `http://www.dcs.ed.ac.uk/home/cps`, 1997.

[SV00] Dominik Schmitz and Jens Vöge. Implementation of a strategy improvement algorithm for finite-state parity games (Extended abstract). Fifth International Conference on Implementation and Application of Automata, CIAA 2000, Proceedings, London, Ontario, Canada, July 2000. Available at `http://www-i7.informatik.rwth-aachen.de/~voege/omega/`.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Thi87] P. S. Thiagarajan. Elementary net systems. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties*, volume 254 of *LNCS*, pages 26–59. Springer-Verlag, 1987.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers, 1990.

[Tho93] Wolfgang Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science. In M.-C. Gaudel and J.-P. Jouannaud, editors, *TAPSOFT'93: Theory and Practice of Software Development, 4th International Joint Conference CAAP/FASE*, volume 668 of *LNCS*, pages 559–568, Orsay, France, April 1993. Springer-Verlag.

[Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *12th Annual Symposium on Theoretical Aspects of Computer Science*, volume 900 of *LNCS*, pages 1–13, Munich, Germany, 2–4 March 1995. Springer-Verlag.

[Tho96] Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer-Verlag, 1996.

[VJ00a] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games. Aachener Informatik-Berichte 2000-2, RWTH Aachen, Fachgruppe Informatik, Aachen, Germany, February 2000.

[VJ00b] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games (Extended abstract). In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215, Chicago, IL, USA, July 2000. Springer-Verlag.

[Vog91] Walter Vogler. Deciding history preserving bisimilarity. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium, ICALP'91*, volume 510 of *LNCS*, pages 493–505, Madrid, Spain, 8–12 July 1991. Springer-Verlag.

[Vög00] Jens Vöge. *Strategiesynthese für unendliche Paritätsspiele und deren Anwendung für das μ-Kalkül-Modelchecking*. PhD thesis, RWTH Aachen, October 2000. To appear.

[Wal96] Igor Walukiewicz. Pushdown processes: Games and model checking. In Thomas A. Henzinger and Rajeev Alur, editors, *Computer Aided Verification, 8th International Conference, CAV'96*, volume 1102 of *LNCS*, pages 62–74. Springer-Verlag, 1996. Full version available through `http://zls.mimuw.edu.pl/~igw`.

[Wal99] Igor Walukiewicz. Personal communication, November 1999.

[Wil99a] Thomas Wilke. Personal communication, July 1999.

[Wil99b] Thomas Wilke. CTL+ is exponentially more succinct than CTL. In C. Pandu Rangan, Venkatesh Raman, and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, 19th Conference, Proceedings*, volume 1738 of *LNCS*, pages 110–121, Chennai, India, December 13–15 1999. Springer-Verlag.

[WN95] Glynn Winskel and Mogens Nielsen. Models for concurrency. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, Semantic Modelling, pages 1–148. Oxford University Press, 1995.

[Zie98] Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.

[ZP96] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.