Counting Logics and Games with Counters

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Simon Robert Leßenich

aus Frechen

Berichter: Universitätsprofessor Dr. Erich Grädel
Professor Dr. Mikołaj Bojańczyk
Universitätsprofessor Dr. Dr. h. c. Dr. h. c. Wolfgang Thomas

Tag der mündlichen Prüfung: 13. Mai 2015

Abstract

In verification and synthesis, properties or models of interest are often quantitative, and many quantitative aspects involve counting. For example, one might be interested in the amount of memory required by a system, how many simultaneous requests a system has to process along a run, or how many infixes belonging to some regular language a word contains. In this thesis, we study two models of infinite games with counters as well as two quantitative counting logics.

The first game model we consider is that of counter parity games. In these quantitative games, a finite set of counters is updated along the edges. Payoffs of finite plays are obtained via the counters, while payoffs of infinite plays are determined via a parity condition. The games are played by a maximizing and a minimizing player, and satisfying the parity condition is good for the maximizing player. We prove that the value of a counter parity game can be computed, and give an algorithm to do so. The key step in the algorithm is to solve the unboundedness problem for the value. This is done with the help of a game with imperfect recall, for which we show that finite-memory strategies suffice for the player suffering from imperfect recall.

In mean counter games, the second class of games studied in this thesis, the payoff of infinite plays is obtained via a mean-payoff condition on a special counter. We prove that the unboundedness problem for the value can be solved in games with only one counter. Furthermore, we show that the exact value can be computed in one-counter single-player games.

The first logic we study is Qμ[#MSO], a counting logic based on the quantitative μ-calculus. This logic is designed specifically for transition systems where the states are labeled with finite relational structures. Counting is introduced to Qμ with the help of counting terms of monadic second-order logic, which are evaluated on the relational structures the states are labeled with. We prove, via a reduction to counter parity games, that the model-checking problem for Qμ[#MSO] is decidable on a generalization of pushdown systems.

We also consider a quantitative counting extension of monadic second-order logic called qcMSO. In this logic, we adapt the classical interpretations of disjunctions as maxima and conjunctions as minima and use counting atoms $|X|$ to count the sizes of sets. We investigate the connections between qcMSO and two other extensions of MSO, namely costMSO and MSO+U. We prove that the qcWMSO-theory of the natural numbers with order is decidable via a reduction to the model-checking problem for an extension of FO+RR on resource-automatic structures. We also provide decomposition algorithms for qcMSO on the natural numbers with order and the infinite binary tree.

Zusammenfassung

Eigenschaften oder Modelle bei der Verifikation und Synthese von Systemen sind oft quantitativ und beinhalten häufig eine Art von Zählen. Fragen können sein, wie groß der Speicherbedarf eines Systems ist, wie viele gleichzeitige Anfragen ein System bearbeiten muss, oder wie viele Infixe eines Wortes zu einer regulären Sprache gehören. Diese Arbeit beschäftigt sich mit zwei Modellen von unendlichen Spielen mit Zählern und zwei quantitativen Zähllogiken.

Die ersten Spiele, die wir betrachten, sind Zählerparitätsspiele. In diesen quantitativen Spielen wird eine endliche Menge von Zählern entlang der Kanten aktualisiert. Die Auszahlungen für endliche Partien hängen von den Zählern ab, während Auszahlungen für unendliche Partien mittels einer Paritätsbedingung bestimmt werden. Das Ziel des einen Spielers ist die Maximierung der Auszahlung, während sein Gegner diese minimieren will. Wir zeigen, dass der Wert von Zählerparitätsspielen effektiv berechnet werden kann. Der wichtigste Schritt im Algorithmus ist dabei das Lösen des Unbeschränktheitsproblems für den Wert. Dies machen wir mit Hilfe eines Spieles mit imperfekter Erinnerung, für das wir zeigen, dass Strategien mit endlichem Speicher für den Spieler mit imperfekter Erinnerung ausreichen.

In Durchschnitts-Zähler-Spielen, der zweiten Klasse von Spielen, die wir betrachten, werden Auszahlungen für unendliche Partien mittels einer Durchschnittsbedingung an einen speziellen Zähler bestimmt. Wir zeigen, dass sich das Unbeschränktheitsproblem des Wertes in Ein-Zähler-Spielen lösen lässt, und dass in Ein-Zähler-Solitärspielen der Wert effektiv berechnet werden kann.

Die erste Logik, die wir betrachten, ist $\mathrm{Q}\mu[\#\mathrm{MSO}]$, eine auf dem quantitativen μ-Kalkül basierende Zähllogik für Transitionssysteme, deren Zustände mit endlichen relationalen Strukturen beschriftet sind. Der Zählaspekt besteht aus der Auswertung von MSO-Zähltermen auf eben diesen Strukturen. Mittels einer Reduktion zu Zählerparitätsspielen beweisen wir, dass das Model-Checking-Problem für diese Zähllogik auf einer Erweiterung von Pushdown-Systemen entscheidbar ist.

Wir führen außerdem eine quantitative Erweiterung namens qcMSO der monadischen Logik zweiter Stufe ein. Bei dieser Logik folgen wir der klassischen Interpretation von Disjunktionen als Maxima und Konjunktionen als Minima, und nutzen Zählatome der Form $|X|$ für das Bestimmen der Größe von Mengen. Wir untersuchen die Beziehungen dieser Logik zu zwei anderen Erweiterungen von MSO, nämlich costMSO und MSO+U. Wir beweisen mittels einer Reduktion zu einem Model-Checking-Problem für eine Erweiterung von FO+RR über Ressource-automatischen Strukturen, dass die qcWMSO-Theorie der natürlichen Zahlen mit Ordnung entscheidbar ist. Zusätzlich präsentieren wir Dekompositionsalgorithmen für qcMSO auf den natürlichen Zahlen mit Ordnung und auf dem unendlichen Binärbaum.

Contents

1 Introduction **1**
- 1.1 Infinite Games on Graphs . . . 2
- 1.2 MSO, the μ-calculus, and Quantitative Extensions . . . 5
- 1.3 Contributions and Results . . . 7
- 1.4 Outline . . . 9
- 1.5 Related Work . . . 10

2 Preliminaries **13**
- 2.1 Basic Definitions and Notations . . . 13
  - 2.1.1 Words and Trees . . . 14
  - 2.1.2 Topological Spaces and Borel Sets . . . 14
- 2.2 Games . . . 16
  - 2.2.1 Arenas and Strategies . . . 16
  - 2.2.2 Several Classes of Games . . . 18
- 2.3 Automata . . . 20
  - 2.3.1 Automata on Words . . . 20
  - 2.3.2 Tree Automata . . . 21
- 2.4 Logics . . . 24
  - 2.4.1 Structures . . . 24
  - 2.4.2 First-order and Monadic Second-Order Logic . . . 24
  - 2.4.3 The Modal and the Quantitative μ-calculus . . . 27

3 Counter Parity Games **29**
- 3.1 Definition . . . 29
- 3.2 Solving Counter Parity Games . . . 31
  - 3.2.1 Marking Counter Parity Games . . . 34
- 3.3 Solving Second-Life Games . . . 40
  - 3.3.1 Recognizing Strategies via Automata . . . 42
  - 3.3.2 Finding Non-Winning Strategies . . . 44
  - 3.3.3 Finite-Memory Strategies for Player 0 . . . 47
- 3.4 The Unboundedness Game . . . 49
  - 3.4.1 Unbounded Values . . . 52

3.4.2 Idempotent Marks and Upper Bounds 52
3.5 Proof of Theorem 3.2.1 57
3.6 Summary 58

4 Mean Counter Games **59**
4.1 Definition and Examples 59
4.2 Solitary Minimizer Games with One Counter 63
4.2.1 Deciding Boundedness of the Value 63
4.2.2 Bounding Counter Values 65
4.2.3 A Reduction to Mean-Payoff Games 67
4.3 Solitary One-Counter Maximizer Games 69
4.3.1 Boundedness is Decidable 70
4.3.2 Solving Solitary Maximizer Games 72
4.4 Two-Player One-Counter Games 82
4.4.1 Bounds for the Value 83
4.5 Affine Mean Counter Games 87
4.6 Summary 89

5 A Counting Logic for Structure Transition Systems **91**
5.1 Structure Transition Systems 91
5.1.1 Logics for STSs 93
5.2 A Counting Logic for STSs 94
5.2.1 Model-Checking Games 95
5.2.2 Examples 96
5.3 Tree-Producing Pushdown Systems 96
5.3.1 Increasing Tree-Rewriting Rules 97
5.3.2 Pushdown Systems for Tree-Rewriting 98
5.3.3 Pushdown Quantitative Parity Games 100
5.4 Model Checking the Counting Logic on TPPDSs 103
5.4.1 Constructing a Finite Game Graph 104
5.4.2 Decomposition and Iterative Evaluation of Terms . . 105
5.4.3 Reducing to Counter Parity Games 113
5.5 Summary 114

6 A Quantitative Counting Variant of MSO **115**
6.1 Syntax and Semantics 115
6.1.1 qcMSO extends MSO 116
6.1.2 Examples 117
6.2 On the Relation to costMSO and the Unbounding Quantifier 118
6.2.1 The Unbounding Quantifier is Expressible in qcMSO 118
6.2.2 Encoding Boundedness and Dominance in qcMSO . . 119
6.3 Deciding qcWMSO on the Natural Linear Order 122

6.3.1 Resource-Automatic Structures and FO+RR 122
6.3.2 Extending FO+RR by Comparisons with ∞ 125
6.3.3 Reducing qcWMSO to FO+RR$^{=\infty}$ 127
6.4 qcMSO and Decidability 128
6.4.1 The Expressive Power of qcMSO on Infinite Words . 128
6.4.2 The Undecidability of qcMSO 130
6.5 Decomposition Theorems 130
6.5.1 Decomposition on the Natural Linear Order 131
6.5.2 Decomposition on the Infinite Binary Tree 140
6.6 Summary 145

7 Summary and Future Work **147**
7.1 Future Work 148

List of Figures **151**

Bibliography **153**

Index **163**

Chapter 1

Introduction

In theoretical computer science, logic and game theory are scientific disciplines of paramount importance. Many central research areas, such as verification and synthesis of programs, have logical and game-theoretical foundations. Whether a system is to be verified against a specification or a system matching a specification is to be constructed automatically, a natural way to encode the specification is to describe it in a formal logic. The actual verification and synthesis tasks are often accomplished with the help of games on graphs. In fact, infinite games on graphs are fundamental tools for modeling and reasoning about reactive systems. For example, the synthesis problem, which was originally formulated by Church [27, 28] for languages, can be modeled naturally via two-player games: The system is represented as a graph or transition system whose vertices or states represent the configurations of the system. These states are partitioned into vertices of the two players, the hostile environment and the controller. Starting with the initial configuration, the owner of the current position may choose a successor configuration, and so on. The (possibly infinite) path obtained by following the chosen configurations corresponds to the run of the system. The synthesis problem of finding a controller such that all runs consistent with the controller meet a given specification amounts to finding a winning strategy for the controller in the game. Of course, the strategy should preferably be realizable, for instance as a finite automaton. (For more information about Church's problem and the developments it entailed, see [84].)

Whether or not the synthesis problem is solvable crucially depends on the systems and specifications at hand. As specifications are commonly given in a formal logic, the expressiveness of the logic needs to be considered, as well as the decidability of two major decision problems: satisfiability and model checking. The former problem asks whether a formula is satisfiable at all, that is, whether there exists some system or structure satisfying it. The latter problem considers the evaluation of a formula on a given structure:

does the structure satisfy the formula or not? This is also the main problem in verification, where one asks whether a given system is correct, for example whether a program works as intended. As a matter of fact, solving the model-checking problem leads back to game theory, as model checking is often done by solving the corresponding model-checking game, exemplifying that game theory and logic are closely related.

Traditionally, the approaches in verification and synthesis are *qualitative*: games are either won or lost, and formulas evaluate to either true or false. In this thesis, we consider *quantitative* aspects of verification and synthesis. On the game-theoretical side, we consider games where players do not win or lose, but try to realize an optimal quantitative outcome or payoff: one player wants to maximize the payoff, while his opponent aims at minimizing it. On the logical side, formulas can be read as queries in addition to Boolean statements. Instead of truth values, we ask how many elements there are that satisfy a property or what the largest value assigned to some variable is. To be more precise, we do not consider arbitrary quantitative aspects, but focus on quantitative elements that involve counting. Hence, we enrich games with a finite set of counters that are updated along the edges, and add counting terms or counting atoms to logics.

In what follows, we first give a brief introduction to infinite games on graphs and logics relevant in the context of this thesis, both with and without counting mechanisms. We then present the results of this thesis, and also give some information on related work. Before doing so, we should mention that there are, of course, many other areas where game theory and logic are of special interest, apart from synthesis and verification: Games are also important in economics, optimization, and related areas in computer science and mathematics, especially the topics of classical game theory with games in strategic form and the investigation of equilibria. Furthermore, infinite games are studied in descriptive set theory, for example Banach-Mazur games and Wadge games. Regarding logic, there is, for example, the whole field of (finite) model theory, where the expressive power and the computational complexity of logics are investigated.

1.1 Infinite Games on Graphs

At first, the objects of interest in game theory were games in strategic form (one-shot games) and games of a finite duration. The theory of infinite games began with the work of Gale and Stewart [48]: In a Gale-Stewart game, two players alternate infinitely often in choosing either 0 or 1. The winning condition is given by a set of infinite strings over 0 and 1, and the first player wins if the play is contained in the given set. One of the fundamental

results obtained by Gale and Stewart was that infinite games are not always determined: it was proved in [48] that there are games where no player has a winning strategy. This motivates investigations about which classes of games are determined, that is, in which games there always is a winning strategy for one of the players.

In [17] as a solution to Church's synthesis problem, Büchi and Landweber showed that if the winning condition is an ω-regular language, the game is determined and finite-memory strategies suffice to win. This result transfers to graph games: In infinite games on graphs, two players move a token along the edges of the graph in such a way that the successor of the current position is chosen by the owner of the position. If the winning condition is ω-regular, the game is determined, and the winner has a winning strategy that uses only finite memory. Further research showed that the complexity of the memory required depends on the winning condition.

In fact, most well-known acceptance conditions for automata on infinite words, such as Büchi, parity, Streett, Rabin, and Muller, were studied directly on graphs: For example, a Büchi condition is a set F of vertices such that Pl. 0 wins if and only if a vertex from F is visited infinitely often. For parity conditions, the graphs are colored. Each vertex is assigned a color or priority from the set of natural numbers and a play is won by Pl. 0 if and only if the minimal priority seen infinitely often is even. It turns out that in both cases positional strategies suffice, that is, whenever a player has a winning strategy, he also has one that does not depend on the history of the play [39, 77]. For Streett-Rabin games, only one player can win positionally, while the other needs finite memory, and for Muller winning conditions, positional strategies do not suffice in general [36, 54].

A very general result on determinacy was proved by Martin [74]: he proved that all games where the winning condition is Borel are determined. Thus, in order to prove that a game is determined, it suffices to place its winning condition in the Borel hierarchy (should this be possible). As common winning conditions that are studied in computer science are Borel, the importance of Martin's theorem is obvious.

In contrast to games considered in classical game theory, as for example studied by von Neumann and Morgenstern [88], the outcomes of ω-regular and Borel games are qualitative. Especially in recent years, however, there has been an increased interest in quantitative games. A class of such games that is highly relevant for this thesis is the class of quantitative parity games. These games were introduced along the study of a quantitative extension of the modal μ-calculus (see below) in [45]. Quantitative parity games extend parity games in such a way that terminal positions are labeled with real numbers. If a terminal is reached, the payoff of the play is the respective label, while a payoff of ∞ is assigned to infinite plays that satisfy the parity

condition. Infinite plays that violate it receive a payoff of $-\infty$. It was proved that quantitative parity games are determined, that is, the players agree on the value of a game [45].

Another, and probably the most well-known class of quantitative games on graphs is that of mean-payoff games. These games are played on graphs where edges are labeled with integers. The payoff is the limit average of the labels seen along the play. Mean-payoff games are played by a minimizing and a maximizing player, and it was proved by Ehrenfeucht and Mycielski that both players have optimal positional strategies [37]. Mean-payoff games are furthermore related to parity games in the sense that every parity game can be reduced to a mean-payoff game (see for example [60]). Recently, mean-payoff objectives have been extended in several ways. They were combined with parity conditions [23], or studied on multiple dimensions [21].

As both the sum of the edge labels seen so far and the number of steps have to be known in mean-payoff games in order to compute the current average, it comes natural to introduce two counters. Whenever an edge is taken, the counter for the sum is updated by adding the edge label, and the other counter is increased by 1. Accordingly, the objective for the players is to maximize and minimize, respectively, the limit of the sequence obtained by dividing the sum counter by the step counter after every move.

When we speak of games with counters in a more general way, we mean games on graphs of infinite duration where an additional (finite) set of counters is maintained and updated along the edges. The outcome of a play, which is usually—but not always—quantitative, is based in some way on these counters. While this at first has an appearance similar to that of counter machines or (1)-counter automata, we focus on games where the availability of edges does not depend on counter values. Thus, regardless of whether a counter is 0 or larger, all edges are always enabled. Still, there may be edges which are immediately losing, for example in energy games (see [18]). Nevertheless, this approach avoids the undecidability results for counter machines (see for example [76]).

In this thesis, we study two games with additional counters, one based on quantitative parity games, and the other on mean-payoff games. In addition to the above conditions on games with counters, we also restrict counter update functions. We do not allow decrements of counters, as we model counter evaluations by natural numbers. Accordingly, to decrease counters, they have to be set to some smaller number, indicated by the update function. For the quantitative counter games that we study, there are two major algorithmic problems. The unboundedness problem is the decision problem of whether the value of the game is unbounded, thus infinite. The second problem is a computational one, namely to compute the exact value of the game.

The setting of these models is similar to other game models that have been studied before. Games with counters appear in the investigation of B- and S-automata, for example in cost games [32], and in the investigation of cost monadic logic [87]. In the examination of games with a model of resources, such as consumption games [14], resource reachability games [71] and ωB-games [22], counters play an important role. Furthermore, variants of mean-payoff and energy games (see for example [18, 19, 21, 23]) can be modeled with counters.

Before we turn to the field of logic, let us briefly mention games without perfect information, as we use a variant of such games to solve an unboundedness problem. Games of imperfect or incomplete information differ from the games described above in a crucial way: players (or sometimes only one player) lack exact information of where the token currently resides in the graph. There are several variants that come to mind. It might be that certain vertices may be indistinguishable, but it might also be that a player has information that is actually false. Games of these sorts have been studied extensively (see for example [20, 70, 80]). Another way information can be influenced is via limiting or modifying knowledge of the history. For example, when a player uses a finite-memory strategy, he has limited recall, as the amount of information about the current history is limited by the size of the memory. In this work, we consider an even stronger form of imperfect recall, where a player forgets special parts of the play and has to continue as if something else was played. While imperfect recall has been studied in classical game theory for finite games in extensive form, there is—to the knowledge of the author—little work on infinite games on graphs with imperfect recall.

1.2 MSO, the μ-calculus, and Quantitative Extensions

In this thesis, we study two (quantitative) counting logics, one based on the quantitative extension Qμ of the modal μ-calculus Lμ, and one based on monadic second-order logic MSO. Both logics are of high interest in verification and synthesis. The modal μ-calculus, the extension of standard modal logic by operators for least and greatest fixed points, is a logic designed specifically for transition systems. There are several convincing reasons to study Lμ. First of all, Lμ is a logic of high expressive power, it subsumes many logics used in the context of verification, such as LTL, CTL, CTL*, and PDL. Secondly, there is a tight connection to parity games: on the one hand, parity games are the model-checking games for Lμ, see for example [40], and on the other hand, the winning region of a parity game is definable in Lμ

[91]. As a third reason, it was proved that Lμ is the bisimulation-invariant fragment of MSO [59].

The quantitative μ-calculus, introduced in [45], adapts the semantics of Lμ to the quantitative case. It is evaluated over quantitative transition systems, where predicates are functions assigning values to states. The semantics follows the traditional interpretation of disjunction as maximum and conjunction as minimum. Accordingly, the quantifiers correspond to infima and suprema. In contrast to Lμ, the domain of truth values of Qμ in its general form is the set of real numbers together with positive and negative ∞. While it is yet unknown whether there is a connection to an appropriate quantitative extension of MSO as for Lμ, it was proved in [45] that the model-checking games of Qμ are quantitative parity games.

Formulas of Qμ evaluate to numbers, and the interpretation of these numbers depends on the quantitative predicates. For example, Qμ has been studied on linear hybrid systems [47]. In this thesis, we focus on counting properties instead and replace the quantitative predicates by counting terms. Canonical examples of counting queries include how many infixes of a certain form occur on the tape of a Turing machine or what the maximal size of the stack in a pushdown system is. One should note that our approach is purely quantitative. Accordingly, the focus differs from that of logics with counting quantifiers studied in finite model theory, where fixed-point logic with counting and first-order logic with counting have been studied thoroughly. (For a survey on fixed-point point logic with counting, see for example [35].)

The second logic we introduce is a quantitative extension of monadic second-order logic. MSO is the extension of first-order logic by quantifiers for sets, and is important especially in the context of automata theory. In fact, MSO on finite words captures precisely the class of regular languages [15, 38, 86], and similar results hold for infinite words [16] and infinite trees [79]. Furthermore, there are, for the classes of structures listed above, effective translations between the logic and corresponding automata models.

We consider a quantitative extension of MSO following the path taken for Qμ: we keep the common interpretation of the operators, but consider quantitative evaluations. The basic quantitative evaluations are introduced as operators to count the size of sets. Thus, when evaluated on infinite words, for example, one might ask for the length of the longest prefix belonging to some regular language, or simply how many times a certain letter occurs. Our extension is in some way similar to two other extensions of MSO that focus on the sizes of sets and which have received a lot of attention in recent years: costMSO and MSO+U.

The first one, MSO+U, is the extension of MSO with the unbounding quantifier. A formula $UX.\varphi(X)$ evaluates to true if there is no bound on

the size of finite sets that satisfy φ. It turns out that this logic is of a higher expressive power than MSO, that is, it is a proper extension. For example, it was proved in [57] that MSO+U-definable languages of infinite words reach arbitrarily high in the projective hierarchy, while MSO-definable languages are on a low level of the Borel hierarchy. Furthermore, MSO+U was recently proved to be undecidable on the natural numbers with linear order [8], which subsumes previous results on the undecidability of MSO+U on the infinite binary tree under certain set-theoretical assumptions [7]. On the other hand, it was proved that the weak variant of MSO+U, where quantifiers range only over finite sets, is decidable on infinite words [5] and infinite trees [10].

The logic costMSO is a quantitative extension introduced during the study of regular cost functions. Syntactically, new atoms $|X| < N$ are added that may only appear positively in the formula, where N is a global fixed numerical variable. An interpretation of a costMSO-formula formally consists of a structure and a number n, with $|X| < N$ evaluating to true if the size of the interpretation of X is below n. As the numerical parameter is the object of interest, one considers evaluations of formulas on structures as queries. The evaluation $[\![\varphi]\!]^{\mathfrak{A}}$ of a formula φ on a structure $\mathfrak{A}$ is the smallest n such that $(\mathfrak{A}, n) \vDash \varphi$. The two major decision problems when evaluating costMSO are boundedness and dominance. A formula φ is bounded on a class $\mathcal{C}$ of structures if there exists a finite bound $n_{\mathcal{C}}$ such that $[\![\varphi]\!]^{\mathfrak{A}} < n_{\mathcal{C}}$ for all $\mathfrak{A} \in \mathcal{C}$. A formula φ is dominated by ψ if on all classes of structures where ψ is bounded, φ is bounded as well. Common classes of structures considered for these problems are words and trees, and it was proved that boundedness and dominance are decidable on finite words [29] and finite trees [32]. For the weak variant costWMSO they are decidable on the infinite binary tree [87].

1.3 Contributions and Results

In this thesis, we introduce two models of games with counters and two counting logics. In the following, we give a brief introduction to each, and mention the results we obtain.

Counter parity games form a class of finitely representable infinite quantitative parity games. They are played by a maximizing player, called Maximizer, and a minimizing one, named Minimizer. There are k counters, represented as a k-dimensional vector of natural numbers. The arena of a counter parity game is a finite graph whose vertices are labeled with priorities as in parity games, and whose edges are labeled with affine k-dimensional functions over the natural numbers. Additionally, the terminal vertices are each labeled with a counter index and a sign (+ or −). In a play, a token is moved along the edges, and the vector of counter values is updated using the affine

functions the edges are labeled with. If the play is infinite, the payoff is ∞ if the parity condition is satisfied, and $-\infty$ otherwise. Should a terminal vertex be reached, the payoff is the current evaluation of the counter indicated by the index at the terminal, either positively or negatively depending on the sign.

It follows from results on quantitative parity games that counter parity games are determined, hence the value of a game always exist. Our main result on counter parity games states that this value can be computed, and we provide an algorithm with a 6Exptime complexity. The main task in this algorithm is to solve the unboundedness problem of whether the value is ∞ or bounded by some computable constant. To do so, we introduce an abstraction from the quantitative counter updates to construct a game with imperfect information and imperfect recall and an ω-regular winning condition. The imperfect recall in this game is used to combine the conditions on finite and infinite plays into a single condition on infinite plays. We prove that the player suffering from imperfect recall can win with a finite-memory strategy if he wins at all, using alternating tree automata. We compute a bound on the size of the memory strategy and apply this bound to obtain an upper bound on the value of the counter parity game. As the roles of the players are dual, we can use the same construction to also obtain a lower bound. With both bounds given, solving a counter parity game reduces to solving a finite, albeit larger, quantitative parity game.

The second game model we study is also played by Maximizer and Minimizer on a finite arena with k counters, but in mean counter games the only allowed counter updates are increments, resets and checks. Furthermore, an additional register is maintained that stores the value that was checked last. The objective is based on this register; it is the limit inferior of the sequence of averages of prefixes. Thus, mean counter games form a subclass of finitely representable mean-payoff games on infinite arenas. However, it turns out that positional strategies do not suffice anymore. There are games where Maximizer does not have optimal strategies, and even if he does, the strategies may require infinite memory. For Minimizer, at least finite-memory strategies are necessary to play optimally.

We first study mean-counter games with only one counter and only one player. We prove that in both cases, Maximizer and Minimizer solitary games, the unboundedness problem can be solved. Furthermore, the exact value can be computed. For Minimizer, this is achieved by showing that register values in optimal plays can be bounded. For Maximizer, we investigate the shape of (infinite-memory) optimal strategies and identify a sufficient pattern. In the end, reductions to mean-payoff games on finite graphs are used to compute the value. In two-player games with one counter, we prove that boundedness is still decidable. At last, we introduce mean counter games with

affine counter updates and show that the memory requirement of Minimizer cannot be based solely on the size of the arena or the number of counters.

The logic Qμ[#MSO] is a counting logic based on the quantitative μ-calculus. It is designed specifically for structure transition systems, that is, for transition systems whose states are labeled with finite relational structures. Instead of quantitative predicates, we use counting terms of monadic second-order logic as atomic formulas. These atoms are evaluated on the relational structures, while the μ-calculus is used to speak about temporal properties of the transition system. We prove that the model-checking problem for Qμ[#MSO] is decidable on a generalization of pushdown systems where the stack is used to store a special kind of tree-rewriting rules. We do so by proving that MSO counting terms are compatible with the decomposition technique, and introduce counters for an iterative evaluation of terms along applications of tree-rewriting rules. Applying this, we reduce the model-checking problem to solving a finite counter parity games.

The second logic we discuss is a quantitative extension of MSO with a focus similar to that of MSO+U and costMSO. We add an atomic operator $|X|$ to measure the size of a set, and base the semantics on the standard semantics for monadic second-order logic: disjunction is a maximum, conjunction a minimum, and the quantifiers correspond to infima and suprema. The new logic qcMSO generalizes MSO, MSO+U and costMSO, and allows furthermore to encode the boundedness and dominance problems of costMSO. On the other hand, undecidability results for MSO+U on the natural numbers with order and the infinite binary tree transfer to qcMSO, and so do results on the complexity of MSO+U-definable languages of infinite words.

Nevertheless, we prove that the qcWMSO-theory of the natural numbers with order is decidable. To do so, we add comparisons with ∞ to first-order logic with resource relations. We prove that FO+RR$^{=\infty}$ is still decidable on resource-automatic structures, extending results on FO+RR. We give a suitable resource-automatic presentation of the finite subsets of the natural numbers along with the relations and atomic operators from qcWMSO, so that deciding the qcWMSO-theory of $(\omega, <)$ reduces to solving the model-checking problem for FO+RR$^{=\infty}$ on this structure. Furthermore, we present decomposition theorems for qcMSO both on the infinite linear order on the natural numbers and on the infinite binary tree, along with decomposition algorithms.

1.4 Outline

We fix the notation for the later chapters and recall relevant definitions and results regarding games, automata theory, and logic in Chapter 2.

Counter parity games are discussed in Chapter 3. Section 3.2.1 is devoted to the abstraction used to define the ω-regular condition of the unboundedness game. In Section 3.3, the class of games with imperfect recall is introduced and studied, while the computation of the value is presented in Section 3.4 and Section 3.5.

In Chapter 4, we define mean counter games, and begin by giving examples that provide insights about the complexity of optimal strategies. Sections 4.2, 4.3, and 4.4 present the results on one-counter mean counter games, while Section 4.5 features affine mean counter games.

Chapter 5 is devoted to the counting variant of the quantitative μ-calculus and structure transition systems. We introduce structure transition systems and the format of suitable logics in Section 5.1, while Section 5.2 presents the counting μ-calculus. Tree-producing pushdown systems are defined in Section 5.3. In Section 5.4, we prove that model-checking the counting logic on these pushdown systems is decidable.

The quantitative counting variant of monadic second-order logic is discussed in Chapter 6. We first give syntax and semantics of this logic. The connections to monadic second-order logic with the unbounding quantifier and to cost monadic second-order logic are described in Section 6.2. We then prove, in Section 6.3, that the theory of the weak variant qcWMSO of the natural numbers with the usual linear order is reducible to a model-checking problem on resource-automatic structures, which we prove to be decidable. We briefly discuss the expressive power of the full variant, that is, with set quantifiers for arbitrary sets, on infinite words in Section 6.4 and conclude that the qcMSO-theory of the natural numbers with order is undecidable. Section 6.5 presents decomposition algorithms, with Section 6.5.1 focusing on the natural numbers and Section 6.5.2 on the infinite binary tree.

1.5 Related Work

The following paragraphs comprise an overview of related work, sorted by chapter.

Counter parity games Counter parity games form a subclass of infinite quantitative parity games [45] and subsume counter-reset games, as discussed in [47] and the conference version of said article [46].

In recent years, several other classes of games on graphs that involve counters have been studied, among them cost games [32, 87], consumption games [14], resource reachability games [71] and ωB-games [22]. Counting also plays a role in finitary (parity) games [24], cost-parity and cost-Streett

games [43], (generalized) energy games [18, 21], energy parity games [19], and mean-payoff parity games [23].

While games on graphs with various notions of imperfect and incomplete information have been studied in computer science, the author is not aware of work involving imperfect recall in the variant used in second-life games.

Mean counter games Mean counter games are counter games where the payoff is determined by a mean-payoff condition, and form a subclass of mean-payoff games [37, 93] on infinite arenas. As mean counter games involve counters, they are related to other games that do so, which were already mentioned in the previous paragraph.

While not much is known about infinite mean-payoff games in general, mean-payoff objectives for pushdown games have recently been studied [26]. As parts of the results presented in Chapter 4 can be viewed as a variant of finding cycles for a modified notion of a cycle, mean-payoff games with partial observation could be mentioned, as results obtained there also involve the construction of a more involved kind of cycle [58]. Although mean counter games involve taking the mean of only one designated counter, multi-dimensional or generalized mean-payoff games [25] can be modeled as mean-payoff games with several counters instead of just one. Furthermore, similar average conditions have also been studied for hybrid automata [12].

The counting μ-calculus for structure transition systems Our counting μ-calculus is based on the quantitative μ-calculus [44, 45], and can be used to express, for example, boundedness problems. As we prove that it is decidable over a generalization of pushdown systems, the results are related to work on unboundedness questions on pushdown systems [11, 52], proving some of the results obtained there in a different way. As our logic subsumes LTL and Lμ with MSO-definable predicates, results on the decidability of these on pushdown systems [42, 69] are generalized. Furthermore, we use and adapt results on eliminating push-operations from pushdown games [82, 89, 90].

Quantitative counting monadic second-order logic The quantitative counting variant of monadic second-order logic is based on the standard semantics, and introduces quantitative aspects at the atomic level with a counting operator to count the size of a set. Since the focus is on the sizes of sets, it is related to MSO+U [4–10, 57] and costMSO [29–32, 87]. The proof technique for the decidability result of the weak variant on the natural numbers with order is based on results for resource-automatic structures and a first-order logic with resource relations designed for these [72]. The

results on decomposition are based on the composition technique [83] and the decomposition algorithm from [50]. There is also work on another quantitative extension, called QMSO, where quantitativity is modeled using an additional semiring structure [66]. Counting in the qualitative setting has been considered using modulo-counting quantifiers [33, 34, 51].

Acknowledgments

I would like to thank my supervisor Erich Grädel and my co-examiners Mikołaj Bojańczyk and Wolfgang Thomas.

I am grateful to my collaborators and co-authors: The results from Chapter 3 were obtained together with Dietmar Berwanger and Łukasz Kaiser and appeared previously in [2, 3]. Chapter 4 began as a joint work with Dietmar Berwanger and Marcus Gelderie. The research question of Chapter 5 was brought to my attention by Łukasz Kaiser, and we worked on it together. The results appeared in [63]. The results from Chapter 6 are part of a joint work with Łukasz Kaiser, Martin Lang and Christof Löding.

I thank my colleagues at the research groups *Mathematische Grundlagen der Informatik* and *Informatik 7*, especially Benedikt, Faried, Kostas, Martin, Sandra, Sarah, Svenja, and Wied.

Last, but not least, I want to thank my family and friends.

Chapter 2

Preliminaries

2.1 Basic Definitions and Notations

We first fix some notation and basic definitions such as words, trees and alphabets. Furthermore, we recall some basic notions from topology.

Given a set x, we write $\mathcal{P}(x)$ for the *powerset* of x, that is, the set containing all subsets of x (including x itself). We say that a set x is *countable* if there is an injection from x to the set of natural numbers, and *uncountable* otherwise.

Throughout this work, we use different notations for the set of natural numbers interchangeably. We write ω or $\mathbb{N}$ for the set of all natural numbers, and often view a number d as the set $[d] = \{0, \dots, d-1\}$ of smaller natural numbers, with $[0] = \varnothing$. The *ordinals* are the extension of this principle via transfinite induction, with $\omega = \bigcup_{\alpha<\omega} \alpha$ being the first infinite ordinal, $\omega + 1 = \omega \cup \{\omega\}$ being its successor, and so on. We write ω_1 for the first uncountable ordinal.

Given some set x, we may use x as the index set of some vector from $\mathbb{N}^x$. This reflects the interpretation of x^y as the set of functions from y to x.

The basic class of structures used in this thesis is that of graphs: A *graph* $G = (V, E)$ consists of a set V of vertices and a binary relation $E \subseteq V \times V$ describing the edges. We say that a graph G is *undirected* if $(v, w) \in E$ entails $(w, v) \in E$, and *directed* otherwise. Given some vertex $v \in V$, we write $\mathrm{rank}(v) = |\{w \in V \mid (v, w) \in E\}|$ for the *rank* or *degree* of v. We denote the set of all maximal paths from a given vertex v, that is, paths that are either infinite or end in a terminal vertex, by $\mathrm{Paths}(G, v)$. For the set of finite, not necessarily maximal, paths from a given vertex v, we write $\mathrm{FinPaths}(G, v)$. If $v \in V$, then $vE = \{w \in V \mid (v, w) \in E\}$ is the set of successors of v.

Although not the main focus of this work, we give complexities of procedures in some places. To render this as readable as possible, we use an

informal version of the $\mathcal{O}$-notation: Given two functions $f, g\colon \mathbb{N} \to \mathbb{N}$, we write $f = \mathcal{O}(g)$ if $f(n) \leq c \cdot g(n)$ for some constant $c \in \mathbb{N}$. When considering exponentials of larger height, we also use the notation $n\mathrm{EXP}(f)$ to indicate

$$n\mathrm{EXP}(f) := \underbrace{2^{\cdot^{\cdot^{2^{\mathcal{O}(q(f))}}}}}_{n \text{ times}}, \text{ where } q \text{ is some polynomial.}$$

Note that in this case, we not only allow multiplication with constants, but the use of arbitrary, but fixed, polynomials. This allows us to, for example, write $\mathrm{EXP}(f)$ for $f(n)2^{f(n)}$, as we can use $q\colon x \mapsto x^2$. In some places, we also write $2^{\mathcal{O}(f)}$ instead of $\mathrm{EXP}(f)$.

2.1.1 Words and Trees

An *alphabet* Σ is a (countable) set of symbols where each symbol is equipped with a rank. For word alphabets, this rank is always 1, for binary trees it is 2, while for other trees it may be arbitrary. We also view alphabets of rank 1 as unranked alphabets, and allow products of ranked with unranked alphabets, where the rank of an element (a, b) in the product is determined by the rank of the letter from the ranked alphabet.

A *finite word* w is a function $w\colon [d] \to \Sigma$ for some natural number d and some word alphabet Σ. We often write $w = w_0 \dots w_{d-1}$ as an ordered sequence of the respective letters, and write Σ^* for the set of all finite words over the alphabet Σ, that is, $\Sigma^* := \bigcup_{d \in \mathbb{N}} \{w \mid w\colon [d] \to \Sigma\}$.

An *infinite word* w is the extension of finite words to ω, thus a function $w\colon \omega \to \Sigma$. Analogously, we write Σ^ω for the set of infinite words over Σ.

A *tree* is a connected cycle-free directed graph with a unique root (i.e., a vertex without incoming edges) and an ordering on the successors of every vertex such that every vertex except for the root has a unique predecessor. A tree is of degree r if every vertex has degree r. As a convention, for a tree with maximal degree r, we refer to the vertices as elements from $[r]^*$: The root is ϵ, while a vertex $v = r_1 \dots r_n$ of degree k has successors $v0, \dots, v(k-1)$.

A *labeled tree* is a tree equipped with a function from its vertex set to a ranked alphabet such that every vertex v is mapped to a symbol of rank $\mathrm{rank}(v)$. If r is the root of t, has rank n and is labeled with Q, we often write $t = Q(t_1, \dots, t_n)$, where t_i is the i-th subtree.

2.1.2 Topological Spaces and Borel Sets

The topology we mainly deal with in this thesis is the natural topology on the infinite words over some alphabet: Let Σ be some unranked alphabet. Then $T = (\Sigma^\omega, \mathcal{O})$ is a topological space, where $\mathcal{O} = \{x\Sigma^\omega \mid x \subseteq \Sigma^*\}$ is

the collection of all *open* sets. For a set $x \subseteq \Sigma^\omega$, we write $x^{\mathcal{C}}$ or $\overline{x}$ for the *complement* $x^{\mathcal{C}} = \overline{x} = \{w \in \Sigma^\omega \mid w \notin x\}$ of x (with respect to Σ^ω). The *closed* sets are the complements of the open sets.

Definition 2.1.1. The *Borel hierarchy* consists of levels $\Sigma^0_\alpha, \Pi^0_\alpha$, $1 \leq \alpha < \omega_1$, and is defined inductively.

$$\begin{aligned}
\Sigma^0_1 &= \mathcal{O} \\
\Pi^0_1 &= \{x \mid x^{\mathcal{C}} \in \mathcal{O}\} \\
\Sigma^0_\alpha &= \left\{ \bigcup_{n\in\omega} x_n \mid \text{each } x_n \text{ is in } \Pi^0_{\beta_n} \text{ for some } \beta_n < \alpha \right\},\ \alpha > 1 \\
\Pi^0_\alpha &= \left\{ \bigcap_{n\in\omega} x_n \mid \text{each } x_n \text{ is in } \Sigma^0_{\beta_n} \text{ for some } \beta_n < \alpha \right\},\ \alpha > 1
\end{aligned}$$

The *Borel sets* are all elements of $\mathfrak{B} := \bigcup_{\alpha\in\omega_1} \Sigma^0_\alpha$, that is, the sets that are on some level of the Borel hierarchy. If we explicitly want to specify the alphabet on which the topology—and thus the Borel sets—are built, we write $\mathfrak{B}(\Sigma)$.

Note that level Π^0_α contains precisely the complements of the sets of level Σ^0_α, and that the levels are linearly ordered by the subset relation. For more information about Borel sets and the Borel hierarchy, see for example [64].

2.2 Games

2.2.1 Arenas and Strategies

An *arena* is a directed graph $G = (V, V_0, V_1, E, \lambda, \Omega, \tau)$, where V_0, V_1 form a partition of the vertex set V and $E \subseteq V \times V$ is the set of edges. The function $\lambda\colon E \to \mathcal{E}$ labels the edges with elements from some set $\mathcal{E}$, while $\Omega\colon V \to \omega$ assigns priorities to the vertices. If the set $T = \{v \in V \mid vE = \varnothing\}$ of terminal vertices is non-empty, then $\tau\colon T \to \mathcal{F}$ labels the terminal vertices by labels from some set $\mathcal{F}$.

To model objectives of players in games, we equip arenas with payoff functions which determine outcomes of plays, and use different payoff functions to distinguish qualitative and quantitative games. A *payoff function* $\mathrm{pay}\colon \mathrm{Paths}(G) \to X$ for an arena G is a function mapping infinite paths and finite paths ending in terminals to some linearly ordered set X such that every subset of X has a supremum and an infimum in X. Throughout this thesis, X will always be a set of numbers. Note that pay will generally depend on the labels λ, Ω and τ, and that we will omit labeling functions from the arena if they do not influence the objective.

Definition 2.2.1. A game $\mathcal{G} = (G, \mathrm{pay})$ consists of an arena G and a payoff function $\mathrm{pay}\colon \mathrm{Paths}(G) \to X$.

A game $\mathcal{G}$, starting in some vertex v_0, is played as follows, as long as the current vertex v_i is not a terminal: If the current vertex v_i is contained in V_0, then Pl. 0 chooses a successor $v_{i+1} \in v_iE$. Otherwise, if $v_i \in V_1$, it is Pl. 1's turn to choose $v_{i+1} \in v_iE$. In this way, the play $\alpha = v_0v_1v_2\ldots$ either reaches a terminal vertex, or is an infinite path. We refer to the set of plays of a game $\mathcal{G}$ as $\mathrm{Plays}(\mathcal{G})$ or $\mathrm{Plays}(G)$ (as they do not depend on the objective). To give meaning to games, we use the convention that the objective of Pl. 0 is always to maximize $\mathrm{pay}(\alpha)$, while Pl. 1 aims at minimizing $\mathrm{pay}(\alpha)$.

Strategies and Determinacy

Definition 2.2.2. A *strategy* σ_i of Pl. i is a function $\sigma_i\colon V^*V_i \to V$ such that $\sigma_i(v_0\ldots v_n) \in v_nE$ for all $v_0\ldots v_n \in \mathrm{FinPaths}(G)$. We refer to the set of strategies of Pl. i as Σ_i.

We say that a play $\alpha = v_0v_1\ldots$ is consistent with a strategy σ_i of Pl. i if $\sigma_i(v_0\ldots v_j) = v_{j+1}$ for all j such that $v_j \in V_i$.

Determinacy, winning, and optimal strategies Given strategies $\sigma_0 \in \Sigma_0, \sigma_1 \in \Sigma_1$ for the two players and an initial vertex v_0, there is a unique play $\alpha_{\sigma_0,\sigma_1,v_0}$ starting in v_0 that is consistent with both σ_0 and σ_1.

Definition 2.2.3. We say that a game $\mathcal{G} = (G, \mathrm{pay})$ is *determined* from vertex v_0 if

$$\sup_{\sigma_0 \in \Sigma_0} \inf_{\sigma_1 \in \Sigma_1} \mathrm{pay}(\alpha_{\sigma_0,\sigma_1,v_0}) = \inf_{\sigma_1 \in \Sigma_1} \sup_{\sigma_0 \in \Sigma_0} \mathrm{pay}(\alpha_{\sigma_0,\sigma_1,v_0}) =: \mathrm{val}\mathcal{G}(v_0).$$

In this case, we call $\mathrm{val}\mathcal{G}(v_0)$ the value of $\mathcal{G}$ from initial vertex v_0.

With the above definition of the value of a game, it is also meaningful to speak about the *value of a strategy*: This corresponds to the value if the respective player of the strategy declares his unalterable intention to use this strategy. Thus, for $\sigma_0 \in \Sigma_0$,

$$\mathrm{val}_{\sigma_0}\mathcal{G}(v_0) = \inf_{\sigma_1 \in \Sigma_1} \mathrm{pay}(\alpha_{\sigma_0,\sigma_1,v_0}),$$

and analogously using the supremum for strategies $\sigma_1 \in \Sigma_1$ of Pl. 1.

If the codomain of the payoff function of a game $\mathcal{G}$ is the set $\{0, 1\}$, we say that the game is a *win-lose game*. If such a game is determined, then either Pl. 0 has a strategy to achieve an outcome of 1, or Pl. 1 can enforce an outcome of 0. Such a strategy is called *winning*. Instead of (G, pay) we write (G, Win) for these games, where $\mathrm{Win} = \{\alpha \in \mathrm{Plays}(G) \mid \mathrm{pay}(\alpha) = 1\}$ is the set of plays that are won by Pl. 0.

If the codomain is a larger set, then a strategy of Pl. 0 that guarantees an outcome of at least the value of the game, or a strategy of Pl. 1 that guarantees an outcome of at most the value, respectively, is called *optimal*.

Classes of simple strategies Strategies, in general, are complex objects, as they map arbitrary finite paths ending in vertices of the strategy's player to next moves. Especially with respect to implementations and applications of strategies, it is crucial to consider simpler classes of strategies. We present here two important examples of simple strategies, namely positional ones and strategies using only finite memory.

If the next move according to a strategy σ depends only on the current vertex, we call the strategy *positional*, or *memoryless*. Formally, this means that $\sigma(\pi v) = \sigma(\pi' v)$ for all finite paths $\pi v, \pi' v$ ending in v. Note that we usually omit the superfluous parts and represent the strategy $\sigma\colon V_i \to V$ as a function defined only on the player's vertices.

We say that a strategy σ is *finite-memory* if σ can be described completely by two functions $\mathrm{up}\colon V \times M \to M$ and $f_M\colon M \times V \to V$ and an element $m_0 \in M$ where M is finite such that $\sigma(v_0 \dots v_n) = f_M(\mathrm{up}^*(v_0 \dots v_n), v_n)$ for the recursive function $\mathrm{up}^*\colon V^* \to M$ defined by

$$\mathrm{up}^*(v_0 \dots v_n) = \begin{cases} \mathrm{up}(v_0, m_0), & n = 0 \\ \mathrm{up}(v_n, \mathrm{up}^*(v_0 \dots v_{n-1})), & n > 0. \end{cases}$$

Intuitively, the function up is used to update the memory set M, while f_M maps vertices and elements from M to next moves. Note that we often describe finite memory by finite automata.

If there are winning or optimal strategies and the respective player also has a positional optimal or winning strategy, then the game is *positionally determined*. If he does not have a positional optimal strategy, but has a finite-memory one, the game is *determined via finite-memory strategies*.

2.2.2 Several Classes of Games

In the following, we introduce some classes of games that are referred to in later chapters.

Borel Games

A win-lose game $\mathcal{G} = (G, \mathrm{Win})$ is said to be *Borel* if the set Win of plays won by Pl. 0 is a Borel set with respect to the natural topology on the plays of G.

The major reasons why Borel games play an important role in computer science are that many applications lead to Borel games and that these games are known to be determined. That is, whenever the set of plays won by Pl. 0 is Borel, then one of the two players has a winning strategy.

Theorem 2.2.4 ([74])**.** Let $\mathcal{G}$ be a Borel game. Then $\mathcal{G}$ is determined.

Parity Games

Parity games are a subclass of Borel games that have many applications, especially in the fields of logic and automata theory. In such games, the vertex set is classically labeled with numbers (or *colors*) from some finite set, that is, $\Omega(V) \subseteq [d]$ for some $d \in \mathbb{N}$. To obtain the set Win, one considers the set $\mathrm{Inf}(\Omega(\alpha))$ of colors that occur infinitely often, that is,

$$\mathrm{Inf}(\Omega(\alpha)) := \{c \in [d] \mid \Omega(\alpha[i]) = c \text{ for infinitely many } i\}.$$

An infinite play α in a parity game is won by Pl. 0 if and only if the minimal color $\min(\mathrm{Inf}(\Omega(\alpha)))$ seen infinitely often is even. If there are terminal positions in G, then τ is used to assign a winner to every terminal vertex.

The most important result about parity games is that positional strategies suffice.

Theorem 2.2.5 ([39, 77])**.** Let $\mathcal{G}$ be a parity game with a finite number of colors. Then $\mathcal{G}$ is positionally determined.

Furthermore, despite it being unknown whether parity games can be solved in Ptime, it is conjectured that they are not Np-complete.

Theorem 2.2.6 ([40])**.** Deciding the winner of a parity game is in NP ∩ coNP.

Quantitative Parity Games

While classical parity games are win-lose games, quantitative parity games allow more outcomes of plays. Therefore, in addition to the vertex labeling $\Omega\colon V \to [d]$, terminals are labeled by $\tau\colon T \to \mathbb{Z}$. The payoff function of a quantitative parity game (QPG) is as follows:

$$\mathrm{pay}(\alpha) = \begin{cases} \tau(v_n), & \alpha = v_0 \dots v_n \text{ is a finite play} \\ \infty, & \min(\mathrm{Inf}(\Omega(\alpha))) \text{ is even} \\ -\infty, & \min(\mathrm{Inf}(\Omega(\alpha))) \text{ is odd} \end{cases}$$

Theorem 2.2.7 ([44, 45])**.** Let $\mathcal{G}$ be a quantitative parity game. Then $\mathcal{G}$ is determined.

Generally, quantitative parity games also have discount factors and the outcomes are nonnegative real numbers. However, for our purposes we restrict the focus to integers, and use $-\infty$ as the lowest payoff instead of 0.

Mean-Payoff Games

Mean-payoff games, introduced in [37], are quantitative games where the objectives of the players are to maximize and minimize, respectively, the average weights along the edges taken in a play. We usually assume that the arena is finite and has no terminals. Accordingly, we omit τ, and also omit Ω, while $\lambda\colon E \to \mathbb{Z}$ maps edges to integers. Given an infinite play $\alpha = v_0 e_1 v_1 e_2 \cdots \in (V \times E)^\omega$, the payoff is defined as

$$\mathrm{pay}(\alpha) = \liminf_{n\to\infty} \frac{\sum_{i=1}^{n} \lambda(e_n)}{n}.$$

Theorem 2.2.8 ([37])**.** Finite mean-payoff games are positionally determined.

Theorem 2.2.9 ([93])**.** Solving mean-payoff games is in NP ∩ coNP.

Note that in [37], the payoff for the minimizing player is defined via $\limsup$, while that for the maximizing player is defined via $\liminf$. However, it follows from the positional determinacy, that—regarding the value—these coincide, as for finite cyclic sequences of weights, the limit exists. Since we later introduce games with a $\liminf$-objective, we define mean-payoff games in this variant here.

2.3 Automata

Throughout this thesis we make use of several results and techniques from the field of automata theory. Therefore, we recall the definitions of the respective classes of automata here, and mention the results relevant for later proofs.

2.3.1 Automata on Words

Automata on Finite Words

A *deterministic finite automaton* (DFA) $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ consists of a finite set Q of states, a finite alphabet Σ, a transition function $\delta\colon Q \times \Sigma \to Q$, an initial state $q_0 \in Q$ and a set $F \subseteq Q$ of final states.

Given an automaton $\mathcal{A}$ and a word $w = a_0 \dots a_{n-1} \in \Sigma^*$, the *run* of $\mathcal{A}$ on w is the sequence $\rho_{\mathcal{A}}(w) = q_0 q_1 \dots q_n$ of states where $q_i = \delta(q_{i-1}, a_{i-1})$. Note that there is a unique run of $\mathcal{A}$ on w. We say that a run is *accepting* if $q_n \in F$. The *language* of $\mathcal{A}$ is the set of all accepted words, that is,

$$L(\mathcal{A}) = \{w \mid \rho_{\mathcal{A}}(w) \text{ is accepting}\}.$$

A *nondeterministic finite automaton* (NFA) $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ is similar to a deterministic one, except that $\delta\colon Q \times \Sigma \to \mathcal{P}(Q)$ maps states and letters to sets of states, and there is not only a single initial state, but a set of initial states. A run on a word $w = a_0 \dots a_{n-1}$, which no longer needs to be unique, nor has to exist, is a sequence $q_0 \dots q_n$ such that $q_0 \in Q_0$ and $q_i \in \delta(q_{i-1}, a_{i-1})$. The language accepted by the automaton is the set of words for which there exists at least one accepting run.

As a convention, we refer to the *size* of an automaton $\mathcal{A}$ by $|\mathcal{A}| := |Q|$, that is, the size of an automaton is the number of states.

We remark that the class of languages accepted by deterministic or nondeterministic finite automata is exactly the class of *regular* languages, and that the class has all common closure properties, for example closure under intersection, union, complement and projection. Furthermore, many important decision problems for languages accepted by finite automata, such as emptiness and inclusion, are decidable, and it is always possible to determinize a nondeterministic finite automaton (which goes along with an increase in size). For more information, we refer to [56].

Automata on Infinite Words

When considering automata on infinite words, the notions of determinism, nondeterminism and of a run remain the same, except that now the run is an infinite sequence of states that respects the transition function. However, the acceptance condition is different from the case of finite words.

For *Büchi* automata, there is still a set F of final states, but now a run is accepting if infinitely many final states occur. Note that while nondeterministic Büchi automata capture the class of all ω-regular languages, this is not true for deterministic Büchi automata.

For this work, the most important class of automata on infinite words is the class of *parity automata*: A parity automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Omega)$ has the same components as a finite or Büchi automaton, but instead of a set of final states, there is a coloring function $\Omega\colon Q \to [d]$ for some $d \in \mathbb{N}$. A run $\rho = q_0 q_1 \dots$ is accepting if the minimal color c such that $\Omega(q_i) = c$ for infinitely many i is even, that is, if $\mathrm{Inf}(\Omega(\rho))$ is even. As above, a deterministic parity automaton accepts a word if the unique run is accepting, while a nondeterministic parity automaton accepts a word if there is at least one accepting run. Furthermore, in contrast to Büchi automata, nondeterministic and deterministic parity automata are equally expressive, and a language is accepted by a parity automaton if and only if it is ω-regular. Note that the language of an automaton is defined, as for NFAs and DFAs, as the set of words—in this case infinite words—accepted by the automaton.

As for regular languages, ω-regular languages are closed under intersection, union and complement, and emptiness and inclusion of languages accepted by given automata is decidable. For further information on ω-regular languages, we refer to [53].

2.3.2 Tree Automata

As we only use automata on infinite trees in later chapters, we do not mention automata on finite trees here. We work with two different classes of tree automata, namely nondeterministic parity tree automata and alternating parity tree automata.

Nondeterministic parity tree automata A *nondeterministic parity tree automaton* $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Omega)$ consists of a finite set Q of states, a finite ranked alphabet Σ, an initial state q_0, a coloring function $\Omega\colon Q \to [d]$ for some $d \in \mathbb{N}$ and a transition function δ, where for a symbol $a \in \Sigma$ of rank k,

$$\delta(q, a) \subseteq \{\{(0, q^0), (1, q^1), \dots, (k-1, q^{k-1})\} \mid q^i \in Q \text{ for all i}\}.$$

In words, δ maps q and a to a set of sets of directions and states in such a way that a set in $\delta(q, a)$ provides exactly one state for every successor vertex.

A run of $\mathcal{A}$ on a tree t is a labeling $\rho\colon t \to Q$ such that for all $v \in t$ with label a of rank k, the set

$$\{(i, \rho(vi)) \mid i < k\} \in \delta(\rho(v), a)$$

of states at the successors together with the respective direction is contained in the result of the transition function. We identify $\rho(t)$ with the induced Q-labeled version of t. A run is accepting if every infinite path in $\rho(t)$ satisfies the parity condition, that is, for every infinite path α in $\rho(t)$, $\mathrm{Inf}(\Omega(\alpha))$ is even.

As for word languages, regular tree-languages, that is, the languages of trees accepted by nondeterministic parity tree automata, are closed under union, intersection and complement, where the famous complementation result is due to Rabin [79]. We remark that, given an automaton $\mathcal{A}$ with n states, m transitions and d colors, the emptiness problem can be decided in time $\mathcal{O}(mn^d)$. For further information, we refer to [53].

Alternating parity tree automata An *alternating parity tree automaton* $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Omega)$ consists of a finite set Q of states, a ranked alphabet Σ with maximal rank r, an initial state q_0, a coloring function $\Omega\colon Q \to [d]$ for some $d \in \mathbb{N}$, and a transition function $\delta\colon Q \times \Sigma \to \mathcal{B}^+([r] \times Q)$, where $\mathcal{B}^+([r] \times Q)$ denotes the set of positive Boolean combinations of elements from $[r] \times Q$. In other words, a state and a letter are mapped to a positive Boolean formula of ranks and states. Note that we always assume that if a letter is of rank k, then in the formula only ranks of less than k appear. We also allow transitions to go to true and false.

A run of $\mathcal{A}$ on a Σ-labeled tree t is a $t \times Q$-labeled tree $\rho(t)$ such that the following conditions are met:

- If a vertex $v \in \rho(t)$ is labeled with (w, q), $w \in t$ has rank k, and v' is a (w', q')-labeled successor of v, then $w' = wi$ for some $i < k$.
- If a vertex $v \in \rho(t)$ is labeled with (w, q), $w \in t$ is labeled with $a \in \Sigma$, and $S = \{(wi_1, q_1), \ldots, (wi_n, q_n)\}$ is the set of labels of successors of v, then the interpretation

$$\mathcal{I}\colon [r] \times Q \to \{0, 1\}, (i, q) \mapsto \begin{cases} 1, & (wi, q) \in S \\ 0, & \text{otherwise} \end{cases}$$

  is a model of $\delta(q, a)$.
- If a vertex $v \in \rho(t)$ is labeled with (w, q), w is labeled with a in t, and $\delta(w, a) \in \{\mathsf{true}, \mathsf{false}\}$, then v is a terminal.

Note that runs of alternating tree automata do not preserve the structure of the input tree, as was the case for nondeterministic parity tree automata. A run is accepting if for every infinite path $\alpha = \alpha_0\alpha_1 \ldots$ in $\rho(t)$ and its corresponding sequence $\rho(\alpha) = (v_0, q_0)(v_1, q_1) \ldots$ of labels, the sequence $q_0q_1 \ldots$ satisfies

the parity condition with respect to Ω, and if no terminal vertex with label (w, q) occurs such that w in t is labeled with a and $\delta(q, a) = \mathsf{false}$. A tree t is accepted if there is an accepting run of $\mathcal{A}$ on t.

Intersection and complementation of alternating tree automata In Section 3.3, we often combine alternating tree automata, that is, we consider intersection and complement automata. Thus, we prove here two lemmas which show that these automata can be constructed efficiently, and with only constant increase in size.

Lemma 2.3.1. Let $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0^1, \Omega_1), \mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_0^2, \Omega_2)$ be two alternating tree automata. Then an intersection automaton $\mathcal{A}_1 \cap \mathcal{A}_2$ for the language $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ can be constructed with $|\mathcal{A}_1| + |\mathcal{A}_2| + 1$ states.

Proof. We define $\mathcal{A}_1 \cap \mathcal{A}_2 = (Q', \Sigma, \delta', q_0', \Omega')$ with $Q' := Q_1 \uplus Q_2 \uplus \{q_0'\}$, where q_0' is a new initial state, coloring $\Omega' = \Omega_1 \cup \Omega_2 \cup \{q_0' \mapsto 0\}$ and transition function $\delta' = \delta_1 \cup \delta_2 \cup \{(q_0', a) \mapsto \delta_1(q_0^1, a) \wedge \delta_2(q_0^2, a) \mid a \in \Sigma\}$.

In words, $\mathcal{A}_1 \cap \mathcal{A}_2$ accepts a tree t if both $\mathcal{A}_1$ and $\mathcal{A}_2$ accept the tree, and this is achieved by simulating both with the help of a new initial state. Note that the new initial state is required as possibly the two automata return to their initial states, and if one of these were used, then the intersection automaton would split again. □

Lemma 2.3.2. Let $\mathcal{A}$ be an alternating tree automaton. Then a complement automaton $\overline{\mathcal{A}}$ for the language of trees over the same alphabet not accepted by $\mathcal{A}$ can be constructed with size $|\mathcal{A}|$.

Proof. $\overline{\mathcal{A}}$ can be obtained by simply exchanging $\wedge$ and $\vee$ in the transition relation, and by changing the parity of the colors, for example by setting $\overline{\Omega}(q) = \Omega(q) + 1$. □

2.4 Logics

Here, we review logics like FO, MSO, Lμ, and Qμ which are used in later proofs and definitions, in order to make notational conventions explicit.

2.4.1 Structures

A *signature* $\zeta = \{R_0, R_1, \ldots, f_0, f_1, \ldots\}$ is a (possibly infinite) set of relation symbols R_i and function symbols f_i, whose respective arity we denote by $\mathrm{ar}(R_i)$ and $\mathrm{ar}(f_i)$.

Definition 2.4.1. A ζ-structure $\mathfrak{A} = (A, (R^{\mathfrak{A}})_{R\in\zeta}, (f^{\mathfrak{A}})_{f\in\zeta})$ for some signature ζ consists of a set A (called the universe), interpretations $R_i^{\mathfrak{A}} \subseteq A^{\mathrm{ar}(R_i)}$ of the relation symbols from ζ, and interpretations $f_i^{\mathfrak{A}}\colon A^{\mathrm{ar}(f_i)} \to A$ of the function symbols.

If ζ does not contain function symbols, then we call corresponding ζ-structures *relational*.

2.4.2 First-order and Monadic Second-Order Logic

In the following, we denote first-order variables, that is, variables that range over elements of the structure, by small letters x, y, z, and monadic second-order variables (which range over sets of elements) by capital letters X, Y, Z.

Given a signature ζ and a set of first-order variables $\mathcal{V}$, *terms* are defined inductively, where every variable is a term, and for a given function symbol $f \in \zeta$ of arity $\mathrm{ar}(f) = n$ together with n terms $e_0, \ldots, e_{n-1}$, the expression $fe_0 \ldots e_{n-1}$ is also a term.

Definition 2.4.2. Formulas of *first-order logic* $\mathrm{FO}(\zeta)$ over a signature ζ, are defined inductively:

- If e, e' are two ζ-terms, then $e = e' \in \mathrm{FO}(\zeta)$.
- If $R \in \zeta$ is a relation of arity n, and $e_0, \ldots, e_{n-1}$ are ζ-terms, then $Re_0 \ldots e_{n-1} \in \mathrm{FO}(\zeta)$.
- If $\varphi, \varphi' \in \mathrm{FO}(\zeta)$, then $\varphi \vee \varphi' \in \mathrm{FO}(\zeta)$, $\varphi \wedge \varphi' \in \mathrm{FO}(\zeta)$, $\varphi \to \varphi' \in \mathrm{FO}(\zeta)$ and $\neg\varphi \in \mathrm{FO}(\zeta)$.
- If $\varphi \in \mathrm{FO}(\zeta)$, then $\forall x\varphi \in \mathrm{FO}(\zeta)$ and $\exists x\varphi \in \mathrm{FO}(\zeta)$.

The first two kinds of formulas are called *atomic*. Furthermore, we say that a variable x occurs free in φ if it appears outside the scope of a quantifier $\exists$ or $\forall$, and write $\mathrm{free}(\varphi)$ for the set of free variables in φ.

Given a formula $\varphi \in \mathrm{FO}(\zeta)$, a ζ-structure $\mathfrak{A}$ and an evaluation $\beta\colon \mathcal{V} \to A$ of the variables in φ, the semantics $[\![\cdot]\!]^{\mathfrak{A},\beta}\colon \mathrm{FO}(\zeta) \to \{0,1\}$ is given by the following evaluations, where we write $\beta(e)$ for the element from A obtained by evaluating e with respect to the $f_i^{\mathfrak{A}}$ when replacing every variable x by $\beta(x)$.

- $[\![e = e']\!]^{\mathfrak{A},\beta} = 1$ if and only if $\beta(e) = \beta(e')$.
- $[\![Re_0 \dots e_{n-1}]\!]^{\mathfrak{A},\beta} = 1$ if and only if $(\beta(e_0), \dots, \beta(e_{n-1})) \in R^{\mathfrak{A}}$.
- $[\![\varphi \vee \varphi']\!]^{\mathfrak{A},\beta} = \max([\![\varphi]\!]^{\mathfrak{A},\beta}, [\![\varphi']\!]^{\mathfrak{A},\beta})$.
- $[\![\varphi \wedge \varphi']\!]^{\mathfrak{A},\beta} = \min([\![\varphi]\!]^{\mathfrak{A},\beta}, [\![\varphi']\!]^{\mathfrak{A},\beta})$.
- $[\![\neg\varphi]\!]^{\mathfrak{A},\beta} = 1 - [\![\varphi]\!]^{\mathfrak{A},\beta}$.
- $[\![\varphi \to \varphi']\!]^{\mathfrak{A},\beta} = \max(1 - [\![\varphi]\!]^{\mathfrak{A},\beta}, [\![\varphi']\!]^{\mathfrak{A},\beta})$.
- $[\![\forall x\varphi]\!]^{\mathfrak{A},\beta} = \inf_{a\in A}[\![\varphi]\!]^{\mathfrak{A},\beta[x\mapsto a]}$, $[\![\exists x\varphi]\!]^{\mathfrak{A},\beta} = \sup_{a\in A}[\![\varphi]\!]^{\mathfrak{A},\beta[x\mapsto a]}$.

We write $\mathfrak{A}, \beta \vDash \varphi$ if the evaluation of φ is 1, and often omit β if clear from the context.

Formulas of *monadic second-order logic*, MSO, are defined similarly, only that now also atomic formulas of the form Xe are allowed, where X is a second-order variable of arity 1 and e is a term, and quantifier formulas $\exists X\varphi$ and $\forall X\varphi$. Accordingly, β is now of the form $\beta\colon \mathcal{V} \to A \cup \mathcal{P}(A)$. The semantics is enriched by the three new rules:

- $[\![Xe]\!]^{\mathfrak{A},\beta} = 1$ if and only if $\beta(e) \in \beta(X)$.
- $[\![\exists X\varphi]\!]^{\mathfrak{A},\beta} = \sup_{A'\subseteq A}[\![\varphi]\!]^{\mathfrak{A},\beta[X\mapsto A']}$.
- $[\![\forall X\varphi]\!]^{\mathfrak{A},\beta} = \inf_{A'\subseteq A}[\![\varphi]\!]^{\mathfrak{A},\beta[X\mapsto A']}$.

We sometimes consider a variant of MSO without first-order variables. If we do so, we add new atomic relations $X \in Y$ and $X \neq \varnothing$, where the latter has the obvious semantics, and $X \in Y$ holds if and only if $\beta(X) = \{a\}$ is a singleton set and $a \in \beta(Y)$.

For any of the above logics, the *quantifier rank* of a formula is once more defined inductively: For an atomic formula φ, the quantifier rank $\mathrm{qr}(\varphi) := 0$ is zero. For negation, we set $\mathrm{qr}(\neg\varphi) := \mathrm{qr}(\varphi)$. For the binary Boolean combinations $\vee, \wedge$ and $\to$, the maximum of the quantifier ranks is considered, for example, $\mathrm{qr}(\varphi \wedge \varphi') := \max(\mathrm{qr}(\varphi), \mathrm{qr}(\varphi'))$. Finally, for quantifier subformulas $\exists x\varphi, \exists X\varphi, \forall x\varphi$, and $\forall X\varphi$, the quantifier rank is increased by one, that is, is $1 + \mathrm{qr}(\varphi)$.

Types and Decomposition

As we later use both the notions of types (for MSO as well as for FO) and the (de)composition technique, we recall the main features of these.

Types The idea of types is to characterize structures up to a certain equivalence, and to obtain formulas for which one knows by construction that only one out of a finite set can hold, and all others must be false. In the following, we use $\mathcal{L}$ to denote any of FO and MSO.

Formally, an *m-type* of $\mathcal{L}$ for a given $m \in \mathbb{N}$ and a number $n \in \mathbb{N}$ of variables is a maximal satisfiable set $T_{m,n}$ of formulas of $\mathcal{L}$ with quantifier rank at most m and free variables among $x_0, \dots, x_{n-1}$. Given a structure $\mathfrak{A}$ and elements $\overline{a} = (a_0, \dots, a_{n-1})$, the m-type of $\overline{a}$ in $\mathfrak{A}$ is the set of formulas φ with $\mathrm{free}(\varphi) \subseteq \{x_0, \dots, x_{n-1}\}$ and quantifier rank at most m such that $\mathfrak{A}, \beta : x_i \mapsto a_i \vDash \varphi$.

Lemma 2.4.3 ([55])**.** Given a signature ζ and a logic $\mathcal{L} \in \{\mathrm{FO}, \mathrm{MSO}\}$ as well as $m, n \in \mathbb{N}$, one can compute a finite set $H_{m,n}$ of formulas with quantifier rank m and free variables $x_0, \dots, x_{n-1}$ such that the following hold:

1. For every ζ-structure $\mathfrak{A}$ and elements $a_0, \dots, a_{n-1}$, there exists a unique $\varphi \in H_{m,n}$ such that $\mathfrak{A}, \beta : x_i \mapsto a_i \vDash \varphi$.

2. For every $\varphi, \varphi' \in H_{m,n}$ with $\varphi \neq \varphi'$ it holds that $\varphi \wedge \varphi'$ is unsatisfiable.

3. Given $\varphi \in H_{m,n}$ and another formula $\psi \in \mathcal{L}$ with $\mathrm{qr}(\psi) \leq m$, either $\varphi \vDash \psi$ or $\varphi \vDash \neg\psi$, and it is computable which of the two holds.

Decomposition The (de)composition technique was shown a successful decidability tool in [83]. We consider, in this thesis, a simplified version to decompose formulas along trees, following [50]. The main idea of the decomposition technique is similar to the tree automata introduced in the previous section, in that a formula to be evaluated on a tree is decomposed into a formula to be evaluated on the single vertex graph consisting only of the root, and a single formula for every subtree to be evaluated there. For technical reasons, it is however necessary to not decompose into a single such tuple of formulas, but into a set, with the property that the formula being true in a tree is equivalent to the existence of a tuple such that every formula in the tuple is true in the respective structure.

Formally, given a tuple $\overline{x} = (x_0, \dots, x_{n-1})$ of variables, and a rank l of a tree, we write $[\overline{x}]_l$ for a partition $(\overline{x}^0, \dots, \overline{x}^l)$ of $\overline{x}$ into $l+1$ disjoint tuples, and denote by $P(\overline{x}, l)$ the set of all such partitions.

Theorem 2.4.4 ([50, 83])**.** Let $t = Q(t_1, \dots, t_l)$ be a tree, and let $\varphi(\overline{x})$ be an MSO-formula whose free variables are among the first-order variables $\overline{x} = (x_0, \dots, x_{n-1})$. Then one can compute a finite set $D(\varphi(\overline{x}))$ of tuples $(\varphi_0(\overline{x}^0), \dots, \varphi_l(\overline{x}^l))$, where $(\overline{x}^0, \dots, \overline{x}^l) \in P(\overline{x}, l)$, of formulas whose quantifier ranks do not exceed that of φ such that for all tuples $\overline{a}$ of vertices in the tree, $t \vDash \varphi(\overline{a})$ if and only if there exists a tuple $\overline{\varphi} \in D(\varphi(\overline{x}))$ such that $Q \vDash \varphi_0(\overline{a}^0)$ and $t_i \vDash \varphi_i(\overline{a}^i)$ for all $1 \leq i \leq l$.

2.4.3 The Modal and the Quantitative μ-calculus

So far we recalled FO and MSO. In this section, we proceed by briefly introducing the modal μ-calculus Lμ as well as its quantitative variant Qμ. As usual for modal logics, these logics are evaluated over Kripke structures.

Definition 2.4.5. A *Kripke structure* $\mathcal{K} = (V, (E_a)_{a \in \Sigma}, (P_i)_{i \in I})$ for some alphabet Σ and some index set I is a graph with Σ-labeled edges whose vertices are labeled by the monadic predicates P_i.

Syntactically, formulas of Lμ and Qμ are identical, and they are built according to the following grammar, where P is a monadic predicate, X is a monadic second-order variable, and $a \in \Sigma$:

$$\varphi ::= P \mid X \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box_a \varphi \mid \Diamond_a \varphi \mid \mu X.\varphi \mid \nu X.\varphi,$$

such that for subformulas $\mu X.\varphi$ and $\nu X.\varphi$, X appears positively in φ (that is, under an even number of negations).

To speak about evaluations, let us first fix, given a Kripke structure $\mathcal{K}$, the set $\mathcal{F}_\mathcal{K} = \{0,1\}^V$ of functions mapping vertices to truth values. This will be used to evaluate second-order variables, and it can easily be adapted later to fit Qμ.

Definition 2.4.6. The semantics of Lμ, given a Kripke structure $\mathcal{K}$ and an evaluation $\beta \colon \mathcal{V} \to \mathcal{F}_\mathcal{K}$ of the second-order variables X, is a function $[\![\cdot]\!]^{\mathcal{K},\beta} \colon V \to \{0,1\}$ that is defined recursively:

1. $[\![P]\!]^{\mathcal{K},\beta} \colon v \mapsto \begin{cases} 1, & v \in P^\mathcal{K} \\ 0, & \text{otherwise.} \end{cases}$
2. $[\![X]\!]^{\mathcal{K},\beta} \colon v \mapsto (\beta(X))(v)$.
3. $[\![\neg\varphi]\!]^{\mathcal{K},\beta}(v) = 1 - [\![\varphi]\!]^{\mathcal{K},\beta}(v)$.
4. $[\![\varphi \wedge \psi]\!]^{\mathcal{K},\beta} = \min([\![\varphi]\!]^{\mathcal{K},\beta}, [\![\psi]\!]^{\mathcal{K},\beta})$.
5. $[\![\varphi \vee \psi]\!]^{\mathcal{K},\beta} = \max([\![\varphi]\!]^{\mathcal{K},\beta}, [\![\psi]\!]^{\mathcal{K},\beta})$.

6. $[\![\Box_a \varphi]\!]^{\mathcal{K},\beta}(v) = \inf_{v' \in vE_a} [\![\varphi]\!]^{\mathcal{K},\beta}(v')$.
7. $[\![\Diamond_a \varphi]\!]^{\mathcal{K},\beta}(v) = \sup_{v' \in vE_a} [\![\varphi]\!]^{\mathcal{K},\beta}(v')$.
8. $[\![\mu X.\varphi]\!]^{\mathcal{K},\beta}$ is the least fixed point of $f \mapsto [\![\varphi]\!]^{\mathcal{K},\beta[X \mapsto f]}$.
9. $[\![\nu X.\psi]\!]^{\mathcal{K},\beta}$ is the greatest fixed point of $f \mapsto [\![\varphi]\!]^{\mathcal{K},\beta[X \mapsto f]}$.

Note that the least and greatest fixed points always exist due to the Knaster-Tarski fixed point theorem, as the functions $V \to \{0,1\}$ combined with the order $\leq$ where $f \leq g$ if $f(v) \leq g(v)$ for all $v \in V$ form a complete lattice, and the fixed-point operators are monotone.

A more accessible characterization of the semantics of the modal μ-calculus can be obtained via parity games: Indeed, it turns out that parity games are precisely the model-checking games for Lμ. The connection is especially tight in this case, as it is also possible to define the winning regions of parity games with a bounded number d of colors via a formula of Lμ using d alternations of least and greatest fixed points. For more details on this connection and on the modal μ-calculus in general, we refer to [13].

The semantics of the quantitative μ-calculus is defined similarly to that of Lμ, with a few differences: First of all, Qμ is evaluated on quantitative Kripke structures, which means that the monadic predicates are replaced with functions $P_i \colon V \to \mathbb{Z}^\infty$, that is, functions from the vertex set to the integers together with ∞ and $-\infty$. Accordingly, $\mathcal{F}_\mathcal{K}$ is replaced with the set of functions $V \to \mathbb{Z}^\infty$. When modifying the semantics of negation to $-[\![\varphi]\!]^{\mathcal{K},\beta}$ and changing the codomain of $[\![\cdot]\!]^{\mathfrak{A},\beta}$ to $\mathbb{Z}^\infty$, the semantics of Qμ is given.

Again, there is a close connection to quantitative parity games. For a more detailed overview, see [45] and Chapter 5. (Note that in general, Qμ also allows discounts on the edges, which is omitted here for the sake of simplicity.)

Chapter 3

Counter Parity Games

In this chapter we introduce a class of games which combine a parity objective for infinite plays with a quantitative objective modeled via a finite set of counters for finite plays. Such games are related to other models of games with counters, such as for example cost games (see, e.g., [32, 87]), but differ from these in that counters are used only for finite plays where the parity condition cannot be applied. The main motivation to study counter parity games comes from the field of quantitative logics, especially the quantitative μ-calculus, as is argued later.

The results of this chapter are joint work with Dietmar Berwanger and Łukasz Kaiser and appeared previously in [2, 3], although there they are presented using a different form of marks.

3.1 Definition

Counter parity games (CPGs) are parity games where quantitative outcomes of finite plays depend on a vector of counters. These counters are updated along the edges, while vertices are colored to determine the outcomes of infinite plays. In the following, we consider games with k counters, where $k \geq 1$ is a natural number. Thus, a counter evaluation can be represented as an element of $\mathbb{N}^k$, and counter updates correspond to affine mappings of such vectors. This leads to the following definition.

Definition 3.1.1. A *counter parity game* (CPG) $\mathcal{G} = (G, \mathrm{pay})$ with k counters is played on a finite arena $G = (V, V_0, V_1, E, \lambda, \Omega, \tau)$, where

- $\Omega\colon V \to [d]$ labels the vertices with colors,
- $\lambda\colon E \to \mathcal{E}$ with $\mathcal{E} := \{f\colon \mathbb{N}^k \to \mathbb{N}^k, c \mapsto Ac + b \mid A \in \mathbb{N}^{k\times k}, b \in \mathbb{N}^k\}$ assigns an affine mapping over k-dimensional vectors of naturals to each edge,

- and $\tau: T \to \{1, -1\} \times [k]$ assigns a sign and a counter index to every terminal position.

The payoff function of a CPG assigns ∞ to infinite plays that satisfy the parity condition, and $-\infty$ to infinite plays where the minimal color seen infinitely often is odd. For a finite play $\pi = v_0 \dots v_n$ with $\tau(v_n) = (s, i)$, the payoff is $s \cdot c_i^\pi$, where c_i^π is the i-th entry of

$$c^\pi := \lambda(v_{n-1}, v_n)(\lambda(v_{n-2}, v_{n-1})(\dots \lambda(v_0, v_1)(0^k))).$$

Intuitively, this means that the game starts with counter evaluation 0^k and is played like a quantitative parity game, updating the counters on an edge from v to w from c to $\lambda(v, w)(c)$. As usual, Pl. 0 wants to maximize the payoff, while Pl. 1 tries to minimize it. To avoid confusion regarding the roles of the players, we refer to Pl. 0 as Maximizer, and to Pl. 1 as Minimizer.

Remark 3.1.2. Counter parity games are a subclass of infinite quantitative parity games. In fact, when considering the product of the arena of a CPG with $\mathbb{N}^k$ in such a way that $((v, c), (w, c'))$ is an edge if and only if $(v, w) \in E$ and $\lambda(v, w)(c) = c'$, $\Omega(v, c) = \Omega(v)$ and $\tau(v, c) = s \cdot c_i$ if $\tau(v) = (s, i)$, then the product is a quantitative parity game where there is an obvious one-to-one-mapping between plays in both games, and corresponding plays yield the same payoffs.

As quantitative parity games, even on infinite arenas, are determined, we obtain determinacy as in the following corollary.

Corollary 3.1.3. Counter parity games are determined. For every CPG $\mathcal{G}$ and every vertex v, $\mathrm{val}_{\mathcal{G}}(v)$ exists.

However, despite proving the existence of the value, the reduction to quantitative parity games does not yield a procedure to compute the value. We provide such an algorithm in the next sections that uses a different approach, namely a reduction to a win-lose game with imperfect recall, to compute a finite quantitative parity game with the same value. Before we do so, let us consider an example of a counter parity game.

Example 3.1.4. Consider the game depicted in Figure 3.1, where vertices of Maximizer are depicted as circles, while square vertices belong to Minimizer.

If the game starts at the middle vertex of Minimizer, then he may choose any number of repetitions of the right loop before moving the token to the vertex of Maximizer. Accordingly, the counter evaluation there is $(10, n)$ for some $n \in \mathbb{N}$. At his turn, Maximizer may now choose to take the negative value of one of the two counters as payoff, or return to Minimizer's vertex.

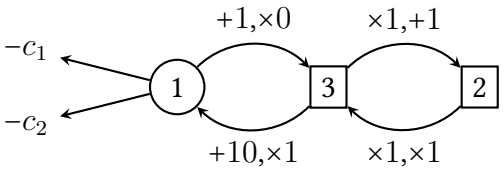


Figure 3.1: An example of a counter parity game

As the first counter is never decreased, but increases with every return to Minimizer's vertex, while Minimizer can set the second counter to a value of his choice, it follows that the maximal outcome of finite plays that Minimizer allows is -10. As taking the right loop infinitely often is bad for Minimizer, he will move to the left loop eventually. As taking the loop is bad for Maximizer, Maximizer has to go to a terminal to avoid a payoff of $-\infty$. Accordingly, the value of the game is -10. ⊣

Applications Besides the theoretical interest in finitely representable infinite quantitative parity games, counter parity games have been applied to solve model-checking problems for variants of the quantitative μ-calculus.

In fact, the solvability result below improved the complexity of model-checking the quantitative μ-calculus on initialized linear hybrid systems from nonelementary to elementary complexity, as shown in [47].

Furthermore, the decidability result for the counting logic described in Chapter 5 on a generalization of pushdown systems is proved via a reduction from the infinite model-checking game to a finite counter parity game (see Chapter 5 and [63]).

3.2 Solving Counter Parity Games

In the current and the following sections, we prove the main result about counter parity games, namely that their values can be computed.

Theorem 3.2.1. Let $\mathcal{G}$ be a counter parity game, and let v be some initial vertex. It is decidable in 4Exptime whether $\mathrm{val}\mathcal{G}(v) = \infty$. The exact value can be computed in 6Exptime. If the number of counters is fixed, the complexities drop to 2Exptime and 4Exptime, respectively.

Corollary 3.2.2. Values of counter parity games can be computed effectively.

As the proof of the theorem requires several steps, it is split across several sections. The central idea is to construct an over-approximation of the set of strategies of Minimizer that guarantee a bounded payoff. This is done

by first abstracting from actual counter update functions to *marks*, which describe how counters change over time with respect to the old evaluation. Using these, we construct an ω-regular game where Minimizer suffers from both incomplete information and incomplete recall, but where his winning strategies correspond to the over-approximation mentioned above. Using alternating tree automata, we then show that these games can indeed be solved, and how to extract a finite-memory winning strategy for Minimizer, provided that he can win. This is summarized in the following main technical lemma.

Lemma 3.2.3. Let $\mathcal{G}$ be a counter parity game with k counters, and let v_0 be the designated initial vertex. One can compute a constant $\mathfrak{m} \in 2\mathrm{EXP}(k)$ and a regular set Σ of strategies of Minimizer which is recognized by a nondeterministic parity tree automaton of size $\mathcal{O}(2\mathrm{EXP}((|V| + k\mathfrak{m})^3))$ with polynomially many colors, such that:

- for every strategy $\sigma \notin \Sigma$, $\mathrm{val}_\sigma \mathcal{G}(v_0) = \infty$, and
- for every strategy $\sigma \in \Sigma$ realizable with a finite memory M,

$$\mathrm{val}_\sigma \mathcal{G}(v_0) < 2\mathrm{EXP}(\mathfrak{m} \cdot |M|^2 \cdot (|V| \cdot 2^k)^2).$$

In the course of the proof, we construct a parity tree automaton that recognizes a regular set of strategies. To this end, observe that a strategy corresponds to the subset (or tree) of plays that are consistent with the strategy. The set of plays of a game $\mathcal{G}$ when starting in vertex v_0, however, is the set of non-extendable paths in the *unfolding* of G from v_0, that is, all infinite paths and the finite paths that end in terminals. Formally, the unfolding of an arena G from v_0 is the tree $T(G, v_0)$ with vertex set $\mathrm{FinPaths}(G, v_0)$, root v_0 and an edge from (πv) to $(\pi v w)$ whenever (v, w) is an edge in G. The partition of the vertex set and the interpretations of Ω, λ, τ translate to $T(G, v_0)$ by using their respective evaluation on v in G for vertices πv in $T(G, v_0)$. Note that we often implicitly view $T(G, v_0)$ as a V-labeled tree, where a vertex πv is labeled with v.

If we label $T(G, v_0)$ using the alphabet $\{\neg S, S\}$ in such a way that the vertices labeled with S and the vertices labeled with $\neg S$ form a partition of the vertex set and that a vertex π is labeled with S if and only if it occurs as a prefix of a play consistent with a given strategy, the plays consistent with the strategy correspond to the non-extendable S-labeled paths. Given a strategy σ, we write $T_\sigma(G, v_0)$ for the tree $T(G, v_0)$ with these additional labels. Note that every S-labeled vertex belonging to the owner of the strategy in $T(G, v_0)$ has exactly one S-labeled successor, while every S-labeled vertex of the opponent has only S-labeled successors.

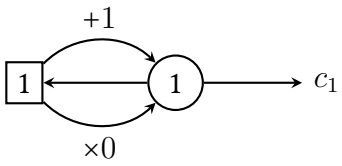


Figure 3.2: Over-approximations are necessary for boundedness

Definition 3.2.4. We say that a set Σ of strategies is *regular* if $\{T_\sigma(G, v_0) \mid \sigma \in \Sigma\}$ is a regular tree-language.

Before beginning with the actual proof, we observe in the next example why an over-approximation of the set of strategies bounding the value is the best possible option when we are interested in regular sets of strategies.

Example 3.2.5. Consider the counter parity game with only one counter depicted in Figure 3.2. In this game, Minimizer can choose, in each of his turns, to either reset the counter to 0, or to increment it by 1. Maximizer can only choose between returning to Minimizer's vertex or taking the current value of the counter. As all priorities are odd, he eventually has to go to the terminal vertex. It is also obvious that the value of the game is 0, as Minimizer can choose to reset the counter in every move.

If, on the other hand, we are interested in *all* strategies of Minimizer that guarantee that the payoff of a consistent play is bounded, that is, is less than ∞, these are exactly the strategies that alternate between the $+1$- and the $\times 0$-edge, but never take the $+1$-edge more than B times in a row, for some fixed constant B. However, the language

$$\{a^{n_1}b^+a^{n_2}b^+a^{n_3}\ldots \mid \text{ there ex. } B \in \mathbb{N} : n_i < B \text{ for all } i\}$$

is not regular.

A respective over-approximation of the set of strategies that bound the value would be the set of strategies that take the $\times 0$-edge infinitely often. Of course there are strategies in the set which do not bound the value, but every finite-memory strategy in the set bounds the payoff by the size of M. ⊣

Proof outline The main result of this chapter that the values of counter parity games can be computed is proved as follows: In Section 3.2.1, we introduce marks to abstract from counter update functions. Marks are functions mapping knowledge about the evolution of counters to the respective knowledge after a counter update has been applied. Before these marks are put to use, we introduce second-life games in Section 3.3 and prove, using automata techniques, that they can be solved and that finite memory suffices for one

of the players. We continue with the marks and construct an unboundedness game. This second-life game allows to decide whether the value of a counter parity game is infinite. If it is finite, the construction gives an upper bound (Section 3.4). The above steps are combined in Section 3.5 to a proof of the main theorem.

3.2.1 Marking Counter Parity Games

The first step in the proof is to replace counter update functions by what we call marks. The main idea behind these marks, and the corresponding marked counter parity games, is to describe the changes of counters during a play. More precisely, a mark (of a single counter update, or also a sequence of counter updates) provides information about which counter was reset to 0, which counter was overwritten by some other counter, and if a new counter value is larger than a set of old counter values, whether these old counter values actually contributed to the respective new counter value. Note that in [2, 3], marks where represented as functions on bit vectors, while we choose a graph-based representation here.

Definition 3.2.6. Let k be the number of counters, and let $C_n = [k] \times \{n\}$ and $C_o = [k] \times \{o\}$ be two times the set of counter indices, once with n for "new" and once with o for "old" as a second component. A *mark-graph* for k counters is a directed graph $h = (C, R_{\simeq}, R_{\succ})$ with vertex-set $\{\bot\} \cup C_n \cup C_o$ and two edge-relations $R_{\simeq} \subseteq C_n \times (C_o \cup \{\bot\})$ and $R_{\succ} \subseteq C_n \times C_o$ satisfying the following properties:

- For all v, $vR_{\simeq} = \varnothing$ or $vR_{\succ} = \varnothing$.
- For all v, $|vR_{\simeq}| \leq 1$.

Let $\mathcal{H}_k$ be the set of all mark-graphs for k counters.

A *mark* m for k counters is a function $m\colon \mathcal{H}_k \to \mathcal{H}_k$.

To simplify notation afterwards, we write $(i, \simeq, j) \in h$ or $(i, \succ, j) \in h$, meaning that $((i, n), (j, o))$ is contained in the respective edge set, and similarly $(i, \simeq, \bot)$.

Marks and mark-graphs provide information about relations between the old counter values and the counter values after the application of a counter update function. However, mark-graphs provide this information for only a subset of the domain of a function, thus several mark-graphs need to be combined to fully characterize a function. Before we make precise what we mean by this, we introduce canonical functions for mark-graphs.

Let $h \in \mathcal{H}_k$ be a mark-graph. Then, we define the function $\mathcal{I}_h \colon \mathbb{N}^k \to \mathbb{N}^k, c \mapsto A_h c + b_h$ in the following way.

$$(A_h)_{i,j} = \begin{cases} 1, & (i, \succ, j) \in h \text{ or } (i, \simeq, j) \in h \\ 0, & \text{otherwise} \end{cases}$$

$$(b_h)_i = \begin{cases} 0, & \text{there exists some } j \in C_o \cup \{\bot\} \text{ such that } (i, \simeq, j) \in h \\ 1, & \text{otherwise} \end{cases}$$

Lemma 3.2.7. Let $h \in \mathcal{H}_k$ be a mark-graph and let $c \in \mathbb{N}^k$ be an arbitrary vector of counters. Then, the following implications hold:

(i) If $(i, \simeq, \bot) \in h$, then $\mathcal{I}_h(c)_i = 0$.

(ii) If $(i, \simeq, j) \in h$, then $\mathcal{I}_h(c)_i = c_j$.

(iii) If $(i, \succ, j) \in h$, then $\mathcal{I}_h(c)_i > c_j$.

(iv) If there is no edge from i to j in h, then $\mathcal{I}_h(c)_i = \mathcal{I}_h(c')_i$ for all $c' \in \mathbb{N}^k$ where $c_l = c'_l$ for all $l \neq j$.

Proof. If $(i, \simeq, \bot) \in h$, then it follows from the properties of mark-graphs that (i, n) does not have any other successors. Accordingly, the i-th row in A_h only contains 0s, and furthermore, $(b_h)_i = 0$. Thus, $\mathcal{I}_h(c)_i = 0$.

If $(i, \simeq, j) \in h$, then again there are no other outgoing edges. Thus, $(b_h)_i = 0$ and $(A_h)_{i,l} = 1$ if and only if $l = j$. It follows that $\mathcal{I}_h(c)_i = c_j$.

In case $(i, \succ, j) \in h$, then there is a 1 in the j-th column of row i. Thus, $\mathcal{I}_h(c)_i$ is at least c_j. As $(b_h)_i = 1$, it is also strictly larger.

If there is no edge from i to j, then the j-th entry of c does not contribute to $\mathcal{I}_h(c)_i$, which concludes the proof. □

Note that the above properties also entail that $\mathcal{I}_h(c)_i = 1$ if (i, n) does not have any successors in h, as the respective entry in b_h is 1.

Definition 3.2.8. Let $m \colon \mathcal{H}_k \to \mathcal{H}_k$ be a mark and let $f \colon \mathbb{N}^k \to \mathbb{N}^k$ be a counter update function. We say that m *marks* f, if for all mark-graphs h and all counter vectors c, the following implications hold:

1. If $(i, \simeq, \bot) \in m(h)$, then $f(\mathcal{I}_h(c))_i = 0$.
2. If $(i, \simeq, j) \in m(h)$, then $f(\mathcal{I}_h(c))_i = c_j$.
3. If $(i, \succ, j) \in m(h)$, then $f(\mathcal{I}_h(c))_i > c_j$.
4. If there is no edge from i to j in $m(h)$, then $f(\mathcal{I}_h(c))_i = f(\mathcal{I}_h(c'))_i$ for all $c' \in \mathbb{N}^k$ where $c_l = c'_l$ for all $l \neq j$.

Again, the above entails that f sets counter i to a constant if for all h the mark-graph $m(h)$ does not have outgoing edges from (i, n). Notice further that a mark can be used not only do describe changes of counter relations which are known from previous applications of functions, but can also be applied directly to vectors: For a vector $c \in \mathbb{N}^k$, we define the *nonzero mark-graph* or *nonzero pattern* $c^{>0}$ as the mark-graph that contains the edge $(i, \simeq, \bot)$ if $c_i = 0$, and $(i, \simeq, i)$ otherwise. It is straightforward to see that $\mathcal{I}_{c^{>0}}(c) = c$, and that all nonzero entries in $A_{c^{>0}}$ are required for this.

So far we defined what mark-graphs and marks are, and what it means for a mark m to mark a function f. However, we did not discuss whether marks exist. In fact, in general there are functions for which marks do not exist: Consider the unary function $f\colon \mathbb{N} \to \mathbb{N}, c \mapsto c^2$, and consider some number $n \neq 0$. Then, $f(n) \neq n$ if and only if $n \neq 1$. However, the mark-graph $h := h_{1^{>0}}$ is the same as $h_{2^{>0}}$, thus both contain only the edge $(0, \simeq, 0)$. It is also easy to see that either $(0, \simeq, 0)$ or $(0, >, 0)$ must be contained in $m(h)$. But then either $1 > 1$ or $4 = 2$ would have to be true for m to mark f, a contradiction.

On the other hand, it turns out that all affine functions over the natural numbers admit marks, and that these marks can be constructed effectively.

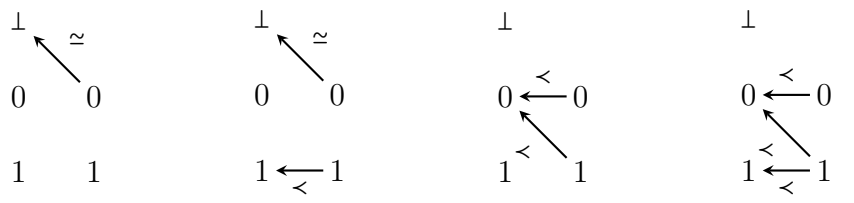
Lemma 3.2.9. Let $f\colon \mathbb{N}^k \to \mathbb{N}^k, c \mapsto Ac + b$ be an affine function over the naturals. Then there exists a mark m that marks f. Furthermore, a mark m for f can be constructed effectively.

Proof. We begin by constructing m for nonzero patterns, that is, for mark-graphs where the only edges are of the form $(i, \simeq, i)$ or $(i, \simeq, \bot)$.

Let thus a nonzero pattern h be given. Consider now, for all i, the set $D_i = \{j \mid (j, \simeq, j) \in h, A_{i,j} \neq 0\}$. Accordingly, the value of $f(c)_i$ for a vector c with nonzero pattern h depends only on the counters with indices in D_i. Thus, if $D_i = \varnothing$ and $b_i = 0$, then $m(h)$ contains the edge $(i, \simeq, \bot)$. If $D_i = \varnothing$ and $b_i > 0$, then no edges from i are added to $m(h)$. If $D_i = \{j\}$, $A_{i,j} = 1$ and $b_1 = 0$, then we add $(i, \simeq, j)$ to $m(h)$, and otherwise we add $(i, >, j)$ to $m(h)$ for all $j \in D_i$. The correctness of $m(h)$ obtained this way is immediate.

To define $m(h)$ for a mark-graph h that is not a nonzero pattern, we consider the nonzero pattern $h^{>0}$ induced by h, namely which contains $(i, \simeq, \bot)$ if h contains this edge, and $(i, \simeq, i)$ in all other cases. We then consider $h' = m(h^{>0})$, which is already defined by the above. To obtain the outgoing edges from i in $m(h)$, we consider first the outgoing edges from i in h'. If there is no outgoing edge from i in h', then there will also be no outgoing edge from i in $m(h)$, and if there is an edge $(i, \simeq, \bot) \in h'$, then we add $(i, \simeq, \bot)$ to $m(h)$.

If there is an edge $(i, \simeq, j)$ in h', we consider the edges from j in h. If there is no outgoing edge from j in h, then again we have no outgoing edge



(a) $m(h_0)$ (b) $m(h_1)$ (c) $m(h_2)$ (d) $m(h_3)$

Figure 3.3: Parts of the mark for the function from Example 3.2.10

from i in $m(h)$, and if there is an edge $(j, \simeq, \bot) \in h$, then we add $(i, \simeq, \bot)$ to $m(h)$. If there is an edge $(j, \simeq, l) \in h$, we add $(i, \simeq, l)$ to $m(h)$, and otherwise we add $(i, \succ, l)$ to $m(h)$ for every l such that $(j, \succ, l) \in h$.

Last, for every j such that $(i, \succ, j) \in h'$, we add $(i, \succ, l)$ for every $l \neq \bot$ such that $(j, \succ, l) \in h$ or $(j, \simeq, l) \in h$.

Note that the above corresponds to the sequential composition of the two mark-graphs when identifying (i, n) in h with (i, o) in h', and identifying all $\bot$-vertices. Once more, the correctness follows from the construction. □

Example 3.2.10. Consider the following counter update function for two counters:

$$f\colon c \mapsto \begin{pmatrix} 3 & 0 \\ 1 & 1 \end{pmatrix} \cdot c + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Let h_0, h_1, h_2, h_3 be the nonzero mark-graphs for the respective vectors $(0,0), (0,1), (1,0)$ and $(1,1)$. f is marked by the mark depicted in Figure 3.3. ⊣

It also follows from the definition that composition of marks is exactly functional composition:

Corollary 3.2.11. Given two counter update functions $f_1, f_2\colon \mathbb{N}^k \to \mathbb{N}^k$ with respective marks m_1, m_2, the mark $m_2 \circ m_1$ is a mark for $f_2 \circ f_1$.

Furthermore, it follows from the definitions that, given a counter vector c and a sequence $f_1, \ldots, f_n$ of functions, the relations between c and $f_n(\ldots(f_1(c)))$ are characterized by $m_n(\ldots(m_1(c^{>0})))$. As there are only finitely many marks and function composition is associative, this induces a finite semigroup structure. In particular, given a sequence of marks $m_1, \ldots, m_n$ and an initial nonzero pattern $c^{>0}$, one can decide regular properties of sequences of marks using finite automata. For example, the language of sequences of marks $m_1 \ldots m_n$ such that after the sequential application of functions marked by the m_j a certain counter c_i has grown is regular: it is the language of words $m_1 \ldots m_n$ where $(i, \succ, i) \in m_n(\ldots(m_1(c^{>0})))$.

Note that until now, we considered marks only as maps from mark-graphs to mark-graphs. From now on, however, we only consider marks for which there actually is a function that is marked by that mark. In particular, this rules out marks where the behavior on non-nonzero mark-graphs differs from the behavior on the respective induced nonzero mark-graphs. Recall that the induced nonzero mark-graph $h^{>0}$ for a mark-graph h is defined analogously as for vectors: If $(i, \simeq, \perp) \in h$, then $(i, \simeq, \perp) \in h^{>0}$. Otherwise, $(i, \simeq, i) \in h^{>0}$. Clearly, $\mathcal{I}_{h^{>0}}(\mathcal{I}_h(c)) = \mathcal{I}_h(c)$ for all c.

Using the existence of corresponding functions, we state and prove some basic properties of marks, which will be used later in combination with the above language properties.

Remark 3.2.12. Let m, m' be two marks for which there are functions, let h be a mark-graph, and $j \in [k]$.

(i) If $(l, \triangle, j) \notin h$ for all $\triangle \in \{\simeq, >\}$ and all l, then $(i, >, j) \notin m(h)$ for all i.

(ii) If $(l, \triangle, j) \notin m(h)$ for all $\triangle \in \{\simeq, >\}$ and all l, then $(i, >, j) \notin m' \circ m(h)$ for all i.

(iii) If $(l, \simeq, j) \notin h$ for all l, then $(i, \simeq, j) \notin m(h)$ for all i.

(iv) If $(l, \simeq, j) \notin m(h)$ for all l, then $(i, \simeq, j) \notin m' \circ m(h)$ for all i.

Proof. (ii) & (iv) follow from (i) & (iii), respectively, as $m(h)$ is a mark-graph.

(i) If j has no incoming edges in h, then all entries of $\mathcal{I}_h(c)$ are unchanged when changing c_j. Accordingly, choose some arbitrary $c \in \mathbb{N}^k$ without zero entries, and let f be a function marked by m. Consider c' with $c'_j = (\max_i f(\mathcal{I}_h(c))_i) + 1$ and $c'_i = c_i$ if $i \neq j$. As $\mathcal{I}_h(c) = \mathcal{I}_h(c')$, $f(\mathcal{I}_h(c)) = f(\mathcal{I}_h(c'))$, but $c'_j > f(\mathcal{I}_h(c'))_i$ for all i. It follows that $(i, >, j) \notin m(h)$.

(iii) Suppose that indeed $(l, \simeq, j) \notin h$. Consider $c \in \mathbb{N}$ where $c_i = 2^{2^i}$, and note that for every subset $J \subseteq [k]$ with $\{j\} \neq J$, we have $\sum_{i \in J} c_i \neq c_j$, and also $\neq c_j + 1$. Accordingly, $\mathcal{I}_h(c)_l \neq c_j$, for all l (as $(l, \simeq, j) \notin h$). Let now f be a function marked by m, which exists by assumption. If for all i, $f(\mathcal{I}_h(c))_i \neq c_j$, there is nothing to show. So assume that $f(\mathcal{I}_h(c))_i = c_j$. To prove that $(i, \simeq, j) \notin m(h)$, it suffices to show that there exists a counter vector c' such that $f(\mathcal{I}_h(c'))_i \neq c'_j$. Thereto, consider the nonzero pattern $h^{>0}$ of $\mathcal{I}_h(c)$ and the mark-graph $m(h^{>0})$, and recall that $\mathcal{I}_{h^{>0}}(\mathcal{I}_h(c)) = \mathcal{I}_h(c)$. For vertex i, there are several cases of possible outgoing transitions:

If $(i, \simeq, \perp) \in m(h^{>0})$, then $f(\mathcal{I}_h(c))_i = f(\mathcal{I}_{h^{>0}}(\mathcal{I}_h(c)))_i = 0 \neq c_j$, a contradiction. If $(i, \simeq, l) \in m(h^{>0})$, then $f(\mathcal{I}_h(c))_i = f(\mathcal{I}_{h^{>0}}(\mathcal{I}_h(c)))_i =$

$\mathcal{I}_h(c)_l \neq c_j$, again a contradiction. Otherwise, let I be the set with $l \in I$ if and only if $(i, >, l) \in m(h^{>0})$. We distinguish two cases, namely where $I = \varnothing$ and where I is nonempty.

If $I = \varnothing$, then $f(\mathcal{I}_h(c))_i = d$ for some constant d, as $f(\mathcal{I}_{h^{>0}}(c'')) = d$ for all c''. Furthermore, if we set $c'_j = c_j + 1$, then $\mathcal{I}_h(c')$ has the same nonzero pattern as $\mathcal{I}_h(c)$, namely $h^{>0}$. Accordingly, $f(\mathcal{I}_h(c'))_i = d \neq c'_j$, and $(i, \simeq, j) \notin m(h)$.

If $I = \{l_1, \ldots, l_n\}$, then $c_j = f(\mathcal{I}_h(c))_i = f(\mathcal{I}_{h^{>0}}(\mathcal{I}_h(c)))_i > \mathcal{I}_h(c)_{l_s}$ for all $1 \leq s \leq n$. Let $J = \{l' \mid (l, \triangle, l') \in h, l \in I, \triangle \in \{\simeq, >\}\}$. If $J \neq \varnothing$, fix $c' = 1^k$. Then, c' has the same nonzero pattern as c, and also $\mathcal{I}_h(c')$ has the same nonzero pattern as $\mathcal{I}_h(c)$. Accordingly, $f(\mathcal{I}_h(c'))_i > \mathcal{I}_h(c')_{l_s}$ for all $1 \leq s \leq n$. As $J \neq \varnothing$, $f(\mathcal{I}_h(c'))_i > 1 = c'_j$. It follows that $(i, \simeq, j) \notin m(h)$.

If $J = \varnothing$, then there is a constant $d = c_j$ such that $f(\mathcal{I}_h(c''))_i = d$ for all c'' with the same nonzero pattern as c. As $c' = (d+1)^k$ has the same nonzero pattern as c, we have $f(\mathcal{I}_h(c'))_i = d < c'_j$. Once more, it follows that $(i, \simeq, j) \notin m(h)$.

□

Before we continue with the definition of marked counter parity games, we compute the number of marks depending on the number of counters. Note that a mark-graph contains at most k^2 edges, as for every counter there may be a single edge to $\perp$ or some other counter labeled with $\simeq$, or there may be at most k edges labeled with $>$. Accordingly, there are at most 2^{k^2} many mark-graphs, and consequently at most

$$\mathfrak{m} := (2^{k^2})^{2^{k^2}} = 2^{k^2 2^{k^2}} = 2^{2^{2\log k + k^2}}$$

many marks, thus the number of marks is at most 2-fold exponential in k.

As outlined before, the idea is to reduce the information from counter update functions to marks. Thus, we define marked counter parity games as extensions of counter parity games where the edge labeling also contains the respective mark. However, the other parts of the game are only adapted to ignore the marks, thus payoffs and counter updates are as before.

Definition 3.2.13. Let $\mathcal{G} = (V, V_0, V_1, E, \lambda, \Omega, \tau, \mathrm{pay})$ be a counter parity game. Given a function $f \in \mathcal{E}$, let m_f be the mark for f as constructed in Lemma 3.2.9. Then the marked counter parity game $\mathcal{G}^m$ is obtained from $\mathcal{G}$ by the following steps:

1. $V^m = V \times \{h \in \mathcal{H} \mid h \text{ is a nonzero mark graph}\}$. (Note that there are 2^k nonzero mark graphs.)

2. $E^m = \{((v,h),(w,h')) \mid e = (v,w) \in E, m_{\lambda(e)}(h)$ has nonzero mark-graph $h'\}$.

3. $\lambda^m(((v,h),(w,h'))) = (\lambda(e), m_{\lambda(e)})$.

4. pay is adapted so that it operates only on the first components of λ^m.

Marks are used later to check for unboundedness, which is done via a game with imperfect recall. Before we give the construction of the unboundedness game, we introduce the model of games with imperfect recall that we use in general and discuss the memory required for winning strategies of Pl. 0. To make explicit that we consider a different game with (potentially) different players, we call the players in second-life games Pl. 0 and Pl. 1 again.

3.3 Solving Second-Life Games

In this section we introduce second-life games and prove that finite-memory strategies suffice for Pl. 0 if the winning condition is ω-regular. Second-life games are constructed from other games where the winning condition is formulated in terms of the edge labels and the vertex labels, and they are designed specifically to provide a certain kind of both imperfect information and imperfect recall. The central idea in the construction is that a player may have to be able to win against two different conditions simultaneously, without actually knowing what the opponent is aiming for. More specifically, the opponent may choose to play for an infinite duration, but he may also choose a terminal, with the intuition that if there are an infinite number of "good" finite plays for the opponent, then this is equivalent to one infinite winning play. Formally, this is modeled via call-return-sequences, where the opponent may choose to enter a call-component which is left only via terminal vertices, and which returns to the position it was called from. However, the player suffering from imperfect recall neither notices that a call occurs, nor is he allowed to remember the occurrence of a return.

Note that in the definition below, $C(a)$ does not indicate a function call, but just denotes a new symbol indicating that a is combined with a call.

Definition 3.3.1. Let $G = (V, V_0, V_1, E, \lambda, \Omega, \tau)$ be an arena with edge labels (now called *actions*) from $\mathcal{E}$ and terminal vertices T. Let $\mathcal{E}_\parallel = \mathcal{E} \cup \{C(a) \mid a \in \mathcal{E}\} \cup \{R\}$. Given a winning condition $W \subseteq \mathcal{E}_\parallel^* \cup \mathcal{E}_\parallel^\omega$, the *second-life game* $\mathcal{S}(G, W)$ is played on the arena G_S where the set of vertices is $V_S = V \cup V^2$,

$(V_0)_S = V_0 \cup (V_0 \times V) \cup (T \times V)$, the set of edges is

$$\begin{aligned} E_S :=& E \cup \{((u,v),(w,v)) \mid (u,w) \in E\} \\ & \cup \{(u,(v,v)) \mid u \in V_1, (u,v) \in E\} \\ & \cup \{((t,v),v) \mid t \in T\}, \end{aligned}$$

the vertices are labeled by Ω_S with $\Omega_S(v) = \Omega(v)$ and $\Omega_S(v,w) = \Omega(v)$, and the edge-labeling λ_S is as follows:

$$\begin{aligned} \lambda_S(v,w) :=& \lambda(v,w) \\ \lambda_S(u,(v,v)) :=& C(\lambda(u,v)) \\ \lambda_S((u,v),(w,v)) :=& \lambda(u,w) \\ \lambda_S((t,v),v) :=& R. \end{aligned}$$

As usual, the players move a token along the edges, and Pl. 0 wins if the word of the edge labels of the resulting play is contained in W. However, the main restriction regarding second-life games lies in the fact that strategies, and thus moves, of Pl. 0 are subject to imperfect information and recall. More specifically, we say that a *C-R-sequence* is a finite path in G_S starting with an edge with label $C(a)$ and ending with an edge with label R, such that in between no other C- or R-labeled edges occur. Given a finite path π in G_S, we denote by $\widehat{\pi}$ the path obtained from π by replacing all C-R-sequences in such a way that $u(v,v)\dots(t,v)v$ is replaced by uv. Note that $\widehat{\pi}$ is also a path in G_S, and that the only positions not contained in V are possibly at the end, if another call occurred that has not yet been followed by a return. We now write $P(\widehat{\pi})$ for the path obtained from $\widehat{\pi}$ by projecting every vertex (u,v) in $\widehat{\pi}$ to u, which yields again a path in G_S.

A strategy σ_0 of Pl. 0 is *allowed in a second-life game*, if for all π ending in a vertex in V_{0S} it holds that

$$\sigma_0(\pi) = \sigma_0(P(\widehat{\pi})).$$

In words, Pl. 0 has to play in such a way that his moves are the same even if all C-R-sequences had not occurred and he had not seen the last C-action. Accordingly, Pl. 0 never knows if the game is currently at a vertex in V or in some V^2-component.

Remark 3.3.2. There is a direct correspondence between strategies of Pl. 0 on the arena G and allowed strategies in second-life games $\mathcal{S}(G,W)$: As Pl. 0 cannot distinguish the different copies of G, his strategies, after a $C(a)$-labeled edge, behave exactly as after the matching a-labeled edge. After a return, which would normally be noticeable by strategies, the strategy is restricted in such a way that it must act as if the sequence had not been

seen. Accordingly, this yields that the next move for a prefix containing C-R-sequences and possibly one more C equals the next move of the restriction of the strategy to plays purely in G. For this reason, we always identify a strategy of Pl. 0 that is allowed with its corresponding strategy on G.

If a play α is such that the main copy (that is, V) is visited infinitely often, then we call the path $\widehat{\alpha}$, obtained as above by removing all C-R-sequences, the *main part* of α.

In the remainder of this section we prove, using tree automata, that finite memory suffices for Pl. 0 in second-life games with ω-regular winning conditions. To do so, we prove that the set of allowed winning strategies of Pl. 0 in a second-life game with ω-regular winning condition is regular, that is, can be recognized by a parity tree automaton. In the next section we then use this to construct, starting with a marked counter parity game, a second-life game that can be used to test whether the value of a counter parity game is bounded.

Theorem 3.3.3. Let G be an arena and let W be an ω-regular winning condition. Pl. 0 has a winning strategy in $\mathcal{S}(G, W)$ if and only if he has a finite-memory winning strategy.

3.3.1 Recognizing Strategies via Automata

We prove Theorem 3.3.3 by constructing a tree automaton that recognizes the winning strategies of Pl. 0. To do so, we first prove some auxiliary lemmas about the existence of certain alternating tree automata that are combined in later proofs. Therefore, recall that intersection and complement automata can be constructed with constant increase in size, as explained in Section 2.3. In the following, we furthermore always assume implicitly that the set of successors of a vertex is ordered, and view the vertex set as a ranked alphabet where the rank of a symbol corresponds to the number of successors of this vertex.

Lemma 3.3.4. Let G be an arena with initial vertex v_0. There exists an alternating tree automaton $\mathcal{A}_{G,v_0}$ of size $|V|$ that accepts a V-labeled tree t if and only if $t = T(G, v_0)$.

Proof. We set $Q = V$, and

$$\delta(v, w) = \begin{cases} \bigwedge_i (i, v_i), & v = w, vE = \{v_0, \dots v_{m-1}\}, \\ \mathsf{true}, & vE = \varnothing, \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

With initial state v_0 and the implicit ordering of the successors, $\mathcal{A}$ accepts exactly $T(G, v_0)$. □

Lemma 3.3.5. Let G be an arena with initial vertex v_0. There exists an alternating tree automaton $\mathcal{A}_0$ of size $|V| + 3$ that accepts a $\Sigma = V \times \{\neg S, S\}$-labeled tree t if and only if t encodes a strategy of Pl. 0.

Proof. We first construct an automaton that checks whether the S-part of the labeling of t is consistent: Let thus $\mathcal{A}_S = (Q_S, \Sigma, \delta_S, q_0^S, \Omega_S)$ be an automaton with $Q_S = \{S, \neg S\}$, $q_0^S = S$, $\Omega(Q_S) = \{0\}$ and

$$\delta_S(\neg S, (v, T)) = \bigwedge_{i<\mathrm{rank}(v)} (i, \neg S) \qquad \text{for } T \in \{S, \neg S\}$$

$$\delta_S(S, (v, \neg S)) = \mathsf{false}$$

$$\delta_S(S, (v, S)) = \begin{cases} \bigwedge_{i<\mathrm{rank}(v)} (i, S), & v \in V_1 \\ \bigvee_{i<\mathrm{rank}(v)} \left((i, S) \wedge \bigwedge_{j \neq i} (j, \neg S) \right), & v \in V_0. \end{cases}$$

We now set $\mathcal{A}_0 := \mathcal{A}_S \cap \mathcal{A}_{G,v_0}$, which is an automaton with the desired properties. □

Lemma 3.3.6. Let G be an arena with initial vertex v_0, and let W be an ω-regular winning condition. Let furthermore $\mathcal{W} = (Q, \Gamma, \delta, q_0, \Omega)$ be a deterministic parity automaton with $L(\mathcal{W}) = \overline{W}$, the complement of W. Then there exists an alternating tree automaton $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, v_0^1, \Omega_1)$ of size $|Q| + |V| + 4$ that accepts a $V \times S$-labeled tree t if and only if $t = T(G, v_0)$ encodes a strategy of Pl. 0 for which there is a consistent play won by Pl. 1.

Proof. By assumption, $\mathcal{W}$ accepts exactly the action sequences of plays won by Pl. 1. The idea is to construct an alternating tree automaton which simulates $\mathcal{W}$ on a path in t that is completely labeled with S. If such an automaton is intersected with $\mathcal{A}_0$ of the previous lemma, it only remains to show that the size limit is met.

Thus define $\mathcal{A}' = (Q', \Sigma, \delta', q_0', \Omega')$ as follows: We copy the states from $\mathcal{W}$, that is, set $Q' = Q$, and accordingly $q_0' = q_0$, $\Omega'(q) = \Omega(q)$. For the transition relation, as soon as a $\neg S$-labeled vertex is reached, the automaton rejects, thus $\delta'(q, (v, \neg S)) = \mathsf{false}$ for all q, v. On S-states, the automaton nondeterministically chooses a successor vertex on which to continue, and updates the state of $\mathcal{W}$:

$$\delta'(q, (v, S)) = \bigvee_{i<\mathrm{rank}(v)} (i, \delta(q, \lambda(v, v_i))),$$

where v_i always denotes the i-th successor of v. Accordingly, an accepting run of $\mathcal{A}'$ on a tree encodes an S-labeled path whose action sequence corresponds to a win of Pl. 1.

As $|\mathcal{A}'| = |Q|$ and $|\mathcal{A}_0| = |V| + 3$, it follows that the desired automaton $\mathcal{A}_1 = \mathcal{A}' \cap \mathcal{A}_0$ has size $|Q| + |V| + 4$. □

3.3.2 Finding Non-Winning Strategies

In this section, we construct automata that recognize strategy trees specifically for second-life games. The main lemma corresponds to the last lemma in the previous section, but now we have to ensure that the strategies are valid and that the call-return-conditions are met. Thus, as after a return the game jumps back to the old position in the tree, which alternating tree automata are not allowed to do, we have to simulate the essential properties of the call-return-sequence directly. Alternation then allows to guess and verify these.

The first lemma captures this intuition, as it provides an automaton which tests if it is possible to return to a certain state with an R-action in such a way that a given color is the minimal color seen until this return occurs.

Lemma 3.3.7. Let G be an arena, let $\mathcal{A} = (Q, \mathcal{E}_\parallel, \delta, q_0, \Omega)$ be a deterministic parity automaton over the alphabet $\mathcal{E}_\parallel$, and let $q_1, q_2 \in Q$, $c \in \Omega(Q)$. There exists an alternating tree automaton $\mathcal{A}^{\textcircled{c}}_{q_1,q_2,c}$ of size $2|Q|$ that accepts a $V \times \{\neg S, S\}$-labeled tree t if and only if there exists a path π labeled with S from the root to a terminal w satisfying the following constraint:

- If $\mathcal{A}$, starting with state q_1, is run on the sequence of actions of π, then it ends at the terminal w in state a q, $\delta(q, R) = q_2$, and the minimum of the color of q_2 and the minimal color seen along the run is c.

Proof. As a state space of $\mathcal{A}^{\textcircled{c}}_{q_1,q_2,c} = (Q', \Sigma, \delta', q_0', \Omega')$, we use two copies of Q, namely $Q' := Q \cup \{q^c \mid q \in Q\}$, and we use the states q^c to indicate that the automaton is in state q and the color c has already been seen, while q means that the automaton is in state q and c has not been seen so far. As we want to simulate a run of $\mathcal{A}$ from state q_1, we set $q_0' := q_1$.

Note that there are some special cases regarding the transitions: If $\Omega(q_2) < c$, then $\mathcal{A}^{\textcircled{c}}_{q_1,q_2,c}$ will always reject, thus $\delta'(q', a) = \mathsf{false}$ for all $q' \in Q', a \in V \times \{S, \neg S\}$. Furthermore, if $\Omega(q_2) = c$, then all transitions which do not go to true or false lead to states from $\{q^c \mid q \in Q\}$, and are analogous to the transitions in the case where $\Omega(q_2) > c$.

The transitions for the case where $\Omega(q_2) > c$ are then as follows, where once more v_i denotes the i-th successor of v, and $a_i^v := \lambda(v, v_i)$.

$$\delta'(q,(v,\neg S)) = \mathsf{false}$$

$$\delta'(q,(v,S)) = \begin{cases} \mathsf{false}, & \Omega(q) < c \\ \mathsf{true}, & \Omega(q) = c, \delta(q,R) = q_2, v \in T \\ \mathsf{false}, & v \in T, \Omega(q) > c \text{ or } \delta(q,R) \neq q_2 \\ \bigvee_{i<\mathrm{rank}(v)} (i, \delta(q,a_i^v)^c), & \Omega(q) = c, v \notin T \\ \bigvee_{i<\mathrm{rank}(v)} (i, \delta(q,a_i^v)), & \Omega(q) > c \end{cases}$$

$$\delta'(q^c,(v,\neg S)) = \mathsf{false}$$

$$\delta'(q^c,(v,S)) = \begin{cases} \mathsf{false}, & \Omega(q) < c \\ \mathsf{true}, & \delta(q,R) = q_2, v \in T \\ \mathsf{false}, & \delta(q,R) \neq q_2, v \in T \\ \bigvee_{i<\mathrm{rank}(v)} (i, \delta(q,a_i^v)^c), & \Omega(q) \geq c \end{cases}$$

Note that if the automaton finds a valid path π, then the transition will be true eventually. Thus, we set $\Omega(q') = 1$ for all $q' \in Q'$. □

Using these automata, which essentially test and simulate C-R-sequences, we now construct an automaton that tests for a given strategy labeling of $T(G, v_0)$ whether Pl. 1 can win in the corresponding second-life game $\mathcal{S}(G, W)$ from v_0, provided that Pl. 0 uses the strategy given by the labeling.

Lemma 3.3.8. Let G be an arena, $v_0 \in V$ an initial vertex, W an ω-regular winning condition over $\mathcal{E}_\parallel$, and let $\mathcal{W} = (Q^{\mathcal{W}}, \mathcal{E}_\parallel, \delta^{\mathcal{W}}, q_0^{\mathcal{W}}, \Omega^{\mathcal{W}})$ be a deterministic parity automaton that recognizes $\overline{W}$.

One can construct an alternating tree automaton $\mathcal{B}$ of size $\mathcal{O}(|\mathcal{W}|^4)$ that accepts a $V \times \{S, \neg S\}$-labeled tree $t = T(G, v_0)$ representing a strategy of Pl. 0 if and only if Pl. 1 can win against the strategy in $\mathcal{S}(G, W)$.

Proof. First of all, we construct from $\mathcal{W}$—using Lemma 3.3.7—the automata $\mathcal{A}^{\textcircled{c}}_{q_1,q_2,c}$ for all $q_1, q_2 \in Q^{\mathcal{W}}$ and $c \in \Omega^{\mathcal{W}}(Q^{\mathcal{W}})$. Let $Q_{q_1,q_2,c}$ denote the respective state sets.

The idea for $\mathcal{B}$ is to simulate the run of $\mathcal{W}$ on the main part of the play of the second-life game, and whenever a C-action occurs, guess the return state and the minimal color, and call the respective automaton $\mathcal{A}^{\textcircled{c}}$ to verify the correctness using alternation. In other words, $\mathcal{B}$ splits and checks both the C-R-sequence and the continuing play in the main part.

Thus, let us first formally construct $\mathcal{B} = (Q, \Sigma, \delta, q_0, \Omega)$, using the above check automata, and then prove its correctness.

$$Q := (Q^{\mathcal{W}} \times (Q^{\mathcal{W}} \cup \{\bot\}) \times \Omega^{\mathcal{W}}(Q^{\mathcal{W}})) \cup \bigcup_{\substack{q_1, q_2 \in Q^{\mathcal{W}}, \\ c \in \Omega^{\mathcal{W}}(Q^{\mathcal{W}})}} Q_{q_1, q_2, c}$$

Here, the first set of triples (q, q', c) is used to simulate $\mathcal{W}$ on the main part, that is, we use the actual state and have two entries to guess state and priority after returns. As initial state we use $(q_0^{\mathcal{W}}, \bot, \Omega^{\mathcal{W}}(q_0^{\mathcal{W}}))$.

As the previous automata, $\mathcal{B}$ rejects upon seeing an $\neg S$-labeled vertex:

$$\delta(p, (v, \neg S)) := \mathsf{false} \qquad \text{for all } p \in Q.$$

Within the state sets of the check automata, the respective transitions are copied. Note that such an automaton accepts only via true-transitions.

$$\delta(p, l) := \delta^{\textcircled{c}}_{q_1, q_2, c}(p, l) \qquad \text{for all } p \in Q_{q_1, q_2, c} \text{ and labels } l.$$

As Pl. 0 does not have any choice when his strategy is fixed, the automaton just updates the respective state of $\mathcal{W}$ if it is in a state where the second component is $\bot$:

$$\delta((q, \bot, c), (v, S)) := \bigvee_{i < \mathrm{rank}(v)} (i, (\delta^{\mathcal{W}}(q, \lambda(v, v_i)), \bot, c)) \qquad \text{for } v \in V_0.$$

On Pl. 1-vertices, there are two possible ways to continue: Either just by choosing an S-successor, or by doing so with a C-action. In the latter case, the respective return state q_2 and the minimal color from the current up until this position are guessed and stored in the second component.

$$\begin{aligned}\delta((q, \bot, c), (v, S)) := &\bigvee_{i < \mathrm{rank}(v)} (i, (\delta^{\mathcal{W}}(q, \lambda(v, v_i)), \bot, c)) \\ &\vee \bigvee_{i < \mathrm{rank}(v)} \bigvee_{\substack{q_2 \in Q^{\mathcal{W}}, \\ c' \in \Omega^{\mathcal{W}}(Q^{\mathcal{W}})}} (i, (\delta^{\mathcal{W}}(q, C(\lambda(v, v_i))), q_2, c')).\end{aligned}$$

Last, we have to define the transitions for states where the second component is not $\bot$. In this case, the automaton splits and starts the check automaton with the first component and continues with $\mathcal{B}$ in the normal manner with the second component:

$$\delta((q, q', c), (v, S)) := \delta((q', \bot, c), (v, S)) \wedge \delta^{\textcircled{c}}_{q, q', c}(q, (v, S)).$$

Regarding the coloring Ω, for the state sets $Q_{q_1, q_2, c}$ of the check automata we copy the respective coloring from these, that is, set every color to 1. For states where the second component is $\bot$, we copy the coloring from $\mathcal{W}$,

$$\Omega((q, \bot, c)) := \Omega^{\mathcal{W}}(q),$$

and use the third component otherwise:

$$\Omega(q, q', c) := c.$$

Note that, by construction, the size restriction required in the lemma is met:

$$|Q| = |\mathcal{W}| \cdot (|\mathcal{W}| + 1) \cdot |\Omega^{\mathcal{W}}(Q^{\mathcal{W}})| + 2|\mathcal{W}| \cdot |\mathcal{W}|^2 \cdot |\Omega^{\mathcal{W}}(Q^{\mathcal{W}})| = \mathcal{O}(|\mathcal{W}|^4).$$

It remains to prove that $\mathcal{B}$ accepts if and only if Pl. 1 can win. Note that we do not require $\mathcal{B}$ to check whether t is a correctly labeled unfolding of G and the strategy labeling is correct.

For the first direction, assume that ρ is an accepting run of $\mathcal{B}$. Consider the play in $\mathcal{S}(G, W)$ where Pl. 1 follows the run of the automaton in the sense that if no C-action occurs, he copies the respective action from the accepting run. If a C occurs, he plays the C-action as in the run, and then follows the accepting run of $\mathcal{A}^{\textcircled{c}}$. At the point of the return, he continues from the former position using the other branch, that is, the $\delta((q', \bot, c), (v, S))$-transition. As the states (q, q', c) are labeled with the lowest color seen in the C-R-sequence, and all other priorities are copied, it follows that the minimal priority seen infinitely often in a run of $\mathcal{W}$ on the play corresponds to the minimal color seen infinitely often in the accepting run, and therefore, $\mathcal{W}$ accepts the play.

Assume now that Pl. 1 has a strategy σ_1 with which he can win in the second-life game $\mathcal{S}(G, W)$ against the strategy σ_0 of Pl. 0 encoded by the tree. Let $\alpha_{\sigma_0,\sigma_1}$ be the unique consistent play. We claim that $\alpha_{\sigma_0,\sigma_1}$ induces an accepting run of $\mathcal{B}$: At positions where the labeling (v, S) is such that $v \in V_0$, the run continues to the successor given by σ_0. At positions in V_1, if σ_1 does not state to play a C-action, the run also copies the respective action. Otherwise, we compute the target state in $\alpha_{\sigma_0,\sigma_1}$ after the return and the minimal color seen in between, and thus are able to correctly guess q_2 and c' such that the C-R-sequence is accepted by $\mathcal{A}^{\textcircled{c}}$ and the run otherwise continues according to the play after the return. As again the minimal priorities occurring infinitely often in this run and in the run of $\mathcal{W}$ on $\alpha_{\sigma_0,\sigma_1}$ coincide, $\mathcal{B}$ accepts. □

3.3.3 Finite-Memory Strategies for Player 0

We now combine the results from the previous sections to prove Theorem 3.3.3. We therefore first prove a technical theorem about the set of winning strategies of Pl. 0, and show how the theorem follows using standard constructions.

In the technical theorem, we show that the set of winning strategies of Pl. 0 can be recognized using nondeterministic parity tree automata. To eliminate the alternation, we use the following theorem:

Theorem 3.3.9 ([78])**.** Given an alternating parity tree automaton $\mathcal{A}$, one can effectively construct an equivalent nondeterministic parity tree automaton of size at most $2^{\mathcal{O}(m|\mathcal{A}|\log(m|\mathcal{A}|))}$, where m is the number of colors of $\mathcal{A}$.

In fact, the above theorem is originally formulated for *Streett-acceptance conditions*: A Streett-condition is a set $S = \{(A_1, B_1), \dots, (A_n, B_n)\}$ of pairs of sets of colors, and an automaton accepts if for every infinite path α and every $1 \leq i \leq n$, $\mathrm{Inf}(\alpha) \cap A_i \neq \varnothing$ or $\mathrm{Inf}(\alpha) \cap B_i = \varnothing$. Note that every parity condition can be written as a Streett-condition: If $[d]$ is the set of colors, then an equivalent Streett-condition is given by

$$\{(\{0\}, \{1\}), (\{0, 2\}, \{3\}), (\{0, 2, 4\}, \{5\}), \dots\}.$$

The original version of the above theorem in[78] states that the bound holds if the A_i-components of the Streett-pairs form a chain. Note further that the m inside the $\log$-term can be omitted: It is at most $|\mathcal{A}|$, and $\log|\mathcal{A}|^2 = 2\log|\mathcal{A}| = \mathcal{O}(\log|\mathcal{A}|)$.

Theorem 3.3.10. Given an arena G and an ω-regular winning condition W whose complement is recognized by a deterministic parity automaton $\mathcal{W}$, the set of winning strategies of Pl. 0 in $\mathcal{S}(G, W)$ can be recognized by a nondeterministic parity tree automaton of size at most exponential in $\mathcal{W}$ and G.

Proof. Let $\mathcal{B}$ be the alternating tree automaton from Lemma 3.3.8, and let $\mathcal{A}_S$ be that from Lemma 3.3.5. We set $\mathcal{B}^* := \overline{\mathcal{B}} \cap \mathcal{A}_S$. Then $|\mathcal{B}^*| = \mathcal{O}(|\mathcal{W}|^4 + |V| + 3)$. By Theorem 3.3.9, we obtain an equivalent nondeterministic parity tree automaton $\mathcal{A}^*$ of size $|\mathcal{A}^*| = 2^{\mathcal{O}(|\mathcal{B}^*|^2 \log|\mathcal{B}^*|)}$. Furthermore, it follows from the properties of $\mathcal{B}$ and $\mathcal{A}_S$ that $\mathcal{A}^*$ accepts a $V \times \{S, \neg S\}$-labeled tree t if and only if $t = T(G, v_0)$ encodes a strategy of Pl. 0 against which Pl. 1 cannot win, thus a winning strategy of Pl. 0. □

With this, we are ready to prove the main theorem of this section:

Theorem 3.3.3. Let G be an arena and let W be an ω-regular winning condition. Pl. 0 has a winning strategy in $\mathcal{S}(G, W)$ if and only if he has a finite-memory winning strategy.

Proof. As the other direction follows trivially, it remains to prove that if Pl. 0 can win, he can also win using a finite-memory strategy. Let thus $\mathcal{A}^*$ be the automaton from the previous theorem. Using standard techniques, we construct an automaton $\mathcal{A}^\dagger$ of size $\mathcal{O}(|\mathcal{A}^*|)$ that guesses the $\{S, \neg S\}$-part of the labeling and then simulates $\mathcal{A}^*$ on the guess. In other words, the automaton $\mathcal{A}^\dagger$ first guesses a strategy, and then verifies that it is winning.

If we build the product $\mathcal{A}^\dagger \times G$, we get a parity game where the verifying player wins if he correctly guesses a strategy labeling of the unfolding that is winning. As mentioned in Section 2.2.2, parity games are positionally determined. Thus, if there is a winning strategy for $\mathcal{S}(G, W)$, then there is a positional winning strategy for Pl. 0 in the parity game, and, by lifting this strategy to G, a winning strategy with memory $\mathcal{A}^\dagger$ for $\mathcal{S}(G, W)$. □

Corollary 3.3.11. If Pl. 0 can win $\mathcal{S}(G, W)$, then he can win with a memory of size $2^{\mathcal{O}(|\mathcal{W}|^9+|V|^3)}$.

Proof.

$$|\mathcal{A}^\dagger| = 2^{\mathcal{O}(|\mathcal{B}^*|^2 \log |\mathcal{B}^*|)} = 2^{\mathcal{O}((|\mathcal{W}|^4+|V|)^2 \log(|\mathcal{W}|^4+|V|))}.$$

Now $|\mathcal{W}|^8 + |\mathcal{W}|^4|V| + |V|^2 = \mathcal{O}(|\mathcal{W}|^8 + |V|^2)$, as whenever $|V| \geq |\mathcal{W}|^4$, then the summand in the middle is at most $|V|^2$, and otherwise it is less than $|\mathcal{W}|^8$, so we can omit $|\mathcal{W}|^4|V|$ if we double the rest. Thus, we get

$$|\mathcal{A}^\dagger| = 2^{\mathcal{O}(|\mathcal{W}|^8 \log(|\mathcal{W}|^4+|V|)+|V|^2 \log(|\mathcal{W}|^4+|V|))}.$$

Furthermore, for the first summand it holds that $|\mathcal{W}|^8 \log(|\mathcal{W}|^4 + |V|) = \mathcal{O}(|\mathcal{W}|^9 + |V|^3)$, as whenever $|V| \geq |\mathcal{W}|^4$, then the term is at most $|V|^3$, and otherwise at most $|\mathcal{W}|^9$. If in the latter summand, $|V| \geq |\mathcal{W}|^4$, then we get $|V|^3$. Otherwise, we get $|\mathcal{W}|^8 \log(2|\mathcal{W}|^4) = \mathcal{O}(|\mathcal{W}|^9)$. □

3.4 The Unboundedness Game

In Section 3.2.1, we introduced marks to abstract from actual counter update functions. Here, we exploit that these marks form a finite semigroup to construct a second-life game with an ω-regular winning condition over marks that is won by the minimizing player if and only if the value of the (marked) counter parity game is bounded from above. The actual bound on the value is computed using the size of the finite memory required for Pl. 0 to win the second-life game.

Note that we will use imperfect recall for Minimizer of the counter parity game. Thus, in the second-life game, Minimizer takes the role of Pl. 0. In other words, Maximizer may play C-actions, which go by unnoticed by Minimizer.

The idea in the construction of the second-life game, which we call the *unboundedness game*, is the following: In a counter parity game, there are two ways by which Maximizer can achieve a value of ∞. If the play is infinite and the parity condition is met, the play gives a payoff of ∞ directly. However, if Maximizer has an infinite sequence $\sigma_0^0, \sigma_0^1, \sigma_0^2, \ldots$ of strategies such that the value guaranteed by σ_0^i is strictly larger than the value guaranteed by σ_0^j for

$j < i$, then the supremum over the values guaranteed by these strategies, and thus the value of the game, is also ∞. To combine this in a single play, we construct a second-life game where in the main part the parity condition is considered, but where Maximizer may choose to go into a copy of the game to demonstrate that he can reach a terminal vertex which provides a high payoff. After this, the game returns to the old position, where he may pursue the parity condition again, or further increase the counters. Note that we use the fact that both the game graph and the number of counters are finite, thus, if there is such a sequence, there must be a counter that can be increased higher and higher and which can be used to generate payoffs at terminals.

Definition 3.4.1. Let G be a counter parity game, and let G^m be the corresponding marked game. The unboundedness game $\mathcal{U}(G)$ is the second-life game $\mathcal{S}(G^{m\prime}, W)$, where $G^{m\prime}$ is obtained from G^m by exchanging the players (i.e., $V_0' = V_1, V_1' = V_0$), and W is the winning condition described below.

Note that for reasons of readability, we refrain from writing $G^{m\prime}$ and write G^m instead, thus we exchange the players implicitly. Instead of describing W, we describe its complement $\overline{W}$, as this meets the intuition and we also need bounds on the size of an automaton recognizing $\overline{W}$ later. A play α in $\mathcal{U}(G) = \mathcal{S}(G, W)$ is won by Maximizer if and only if α contains infinitely many vertices of the main copy (and, accordingly, no terminal vertex inside the main copy is reached), and (i) or (ii) holds:

(i) the main part satisfies the parity condition, that is, $\min \mathrm{Inf}(\Omega(\widehat{\alpha}))$ is even,

(ii) for some counter index $j \in [k]$, and some position $i \geq 0$ in α, the following is repeated ad infinitum: Counter c_j is increased in the main part, then a C-labeled action occurs, and the C-R-sequence ends with a return from a terminal such that a payoff of at least the value of c_j at the time of the call would be granted in G. (For an illustration, see Figure 3.4.)

Formally, concerning (ii), given a counter index j, let $c_j{\nearrow}$ be the language of finite sequences $m_1 \dots m_n$ of marks such that $(j, >, j) \in m_n \circ \dots \circ m_1(c^{>0})$. Let then $\bot^{>j}$ be the language of marks with C and R and vertices where w belongs to $\bot^{>j}$ if and only if $w = C(m_1)m_2 \dots m_n R$ such that if v is the vertex from where R returns to the main copy, $\tau(v) = (1, d)$, then $(d, >, j) \in m_n \circ \dots \circ m_1(c^{>0})$. Thus, the language of (ii) can be obtained by concatenating $\mathcal{E}_{\|}^*$ with $\bigcup_{j\in[k]}(c_j{\nearrow}\ \bot^{>j})^\omega$, which is ω-regular. (Note that, in fact, we use the first symbol $C(m_1)$ of some $\bot^{>j}$ as the first symbol of the subsequent, next $c_j{\nearrow}$, that is, let the respective sequence start with this particular m_1. This does not affect the fact that the language is ω-regular.)

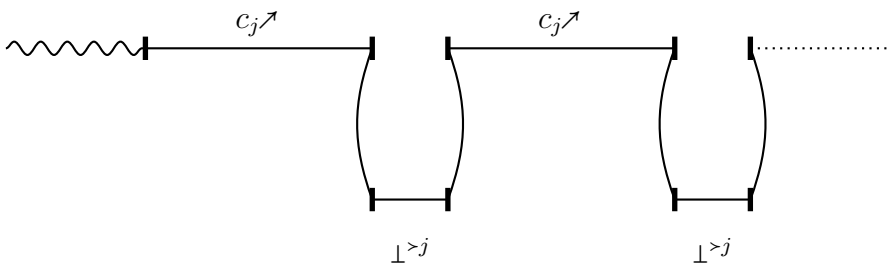


Figure 3.4: An illustration of Part (ii) of the winning condition of $\mathcal{U}(G^m)$

As parity conditions are ω-regular, the property that the main part is visited infinitely often is also ω-regular, and as ω-regular properties are closed under Boolean operations, $\overline{W}$ and also W are ω-regular.

Remark 3.4.2. To estimate the size of an automaton recognizing $\overline{W}$, note that $\mathcal{O}(\mathfrak{m})$ states suffice for a deterministic finite automaton to recognize $c_j\nearrow$ for a given j, and the same holds for $\perp^{>j}$. By guessing the right moment to switch the automata, $c_j\nearrow\ \perp^{>j}$ can thus be recognized nondeterministically with $\mathcal{O}(\mathfrak{m})$ many states, and $(c_j\nearrow\ \perp^{>j})^\omega$ can be checked with the same number of states using Büchi acceptance. By adding a new initial state and taking a copy of the above automaton for every $j \in [k]$, we get an automaton of size $\mathcal{O}(k\mathfrak{m})$ for the language $\mathcal{E}_\parallel^* \cdot \bigcup_{j\in[k]}(c_j\nearrow\ \perp^{>j})^\omega$. Note that the automaton waits in the initial state until j is guessed at some point, and then verifies the respective ω-language.

As there are at most $|V|$ many colors for the parity condition, we get an automaton of size $\mathcal{O}(|V| + k\mathfrak{m})$ for the language of (i) or (ii). Accordingly, we obtain a deterministic parity automaton for $\overline{W}$ with

$$2^{\mathcal{O}(|V|(|V|+k\mathfrak{m})\log(|V|+k\mathfrak{m}))} = 2^{\mathcal{O}((|V|+k\mathfrak{m})^3)}$$

many states.

We conclude by Corollary 3.3.11 that Minimizer can win with a memory of size

$$K_0 := 2^{\mathcal{O}\left(\left(2^{\mathcal{O}\left((|V|+k\mathfrak{m})^3\right)}\right)^9+|V|^3\right)} = 2^{2^{\mathcal{O}\left((|V|+k\mathfrak{m})^3\right)}} = 2\mathrm{EXP}((|V|+k\mathfrak{m})^3),$$

if he can win at all. Furthermore, the set Σ of winning strategies is recognizable by a nondeterministic parity tree automaton of the same size.

3.4.1 Unbounded Values

We constructed the set Σ of winning strategies of Minimizer in $\mathcal{U}(G)$ above, and gave a bound on the size of the memory. To prove Lemma 3.2.3, it remains to show that for every strategy $\sigma \notin \Sigma$ of Minimizer in $\mathcal{U}(G)$ it holds that $\mathrm{val}_\sigma G^m(v_0) = \infty$, and for every strategy $\sigma \in \Sigma$ with memory M it holds that $\mathrm{val}_\sigma G^m(v_0) < 2\mathrm{EXP}(\mathfrak{m} \cdot |M|^2 \cdot (|V| \cdot 2^k)^2)$.

Lemma 3.4.3. Let σ_1 be a strategy of Minimizer in $\mathcal{U}(G)$ from v_0 which is not winning. Then, $\mathrm{val}_{\sigma_1} G^m(v_0) = \infty$.

Proof. First of all, note that $\mathcal{U}(G)$ might not be determined. Thus, if σ_1 is not winning in $\mathcal{U}(G)$, there exists at least one play $\alpha(\sigma_1)$ won by Maximizer in $\mathcal{U}(G)$. As G^m is determined, it suffices to show that, given an arbitrary natural number N, Maximizer can force a payoff of at least N against σ_1 in G^m. This can be achieved as follows: While the current action in $\alpha(\sigma_1)$ is not a C-labeled action, Maximizer copies the respective action. When an action $C(a)$ occurs, Maximizer just plays a, and then continues as after the respective R-action in $\alpha(\sigma_1)$. If $\alpha(\sigma_1)$ is won by Maximizer because of (i) from the winning condition, Maximizer continues as above. Otherwise, there will eventually be a C-labeled action such that the chosen counter j exceeds N, and accordingly, also the payoff at the terminal from where the R-action is taken exceeds N. In this case, Maximizer plays a, and then continues as in the C-R-sequence. In both cases, the payoff obtained is at least N, thus $\mathrm{val}_{\sigma_1} G^m(v_0) = \infty$. □

3.4.2 Idempotent Marks and Upper Bounds

We now turn to proving the second part of Lemma 3.2.3, which is to show that a winning strategy for Minimizer with memory M in $\mathcal{U}(G)$ induces an upper bound for the value of G^m. To prove this, we first need the notion of idempotent marks, and some properties of these.

Definition 3.4.4. Let m be a mark. We call m *idempotent* if $m \circ m = m$.

Note that the marks for a finite number of counters form a finite semigroup, and also a finite monoid ($\mathrm{id}: h \mapsto h$ is an identity element). Thus, idempotent marks exist. Furthermore, we prove using Ramsey's theorem that whenever a strategy with memory is played, for every initial play prefix longer than a certain constant, there exist two consecutive play infixes from the same vertex and memory state which are both labeled with the same idempotent mark.

Lemma 3.4.5. Let σ be a strategy of Minimizer for G^m that uses a finite memory M. There exists a number $\mathcal{R}_M \in \mathbb{N}$, such that, for every initial prefix $\pi = v_0 v_1 \cdots \in (V^m)^N$, $N > \mathcal{R}_M$, of a play consistent with σ of length at least $\mathcal{R}_M$, there exist positions $i_1 < i_2 < i_3$ for which the following properties hold, where q_i denotes the memory state at vertex v_i:

1. $v_{i_1} = v_{i_2} = v_{i_3}$.
2. $q_{i_1} = q_{i_2} = q_{i_3}$.
3. Let $m_{1,2}$ be the mark for $\pi[i_1 \dots i_2]$, $m_{2,3}$ for $\pi[i_2 \dots i_3]$ and $m_{1,3}$ for $\pi[i_1 \dots i_3]$. Then, $m_{1,2} = m_{2,3} = m_{1,3}$ is idempotent.

Furthermore, $\mathcal{R}_M = \mathrm{EXP}(\mathfrak{m} \cdot |M|^2 \cdot (|V| \cdot 2^k)^2)$.

Proof. Let π be such an initial play prefix, and let the q_i be as above. This induces a clique over $\{0, \dots, N-1\}$, where the edge (i, j), $i < j$, is labeled with the 5-tuple $(v_i, q_i, v_j, q_j, m(\pi[i \dots j]))$. Accordingly, there are

$$l = \mathfrak{m} \cdot |M|^2 \cdot |V^m|^2 = \mathfrak{m} \cdot |M|^2 \cdot (|V| \cdot 2^k)^2$$

many colors. By Ramsey's theorem, there exists a number $R(3, \dots, 3)$—with l occurrences of 3—such that for every complete graph with at least $R(3, \dots, 3)$ vertices, there exists a monochromatic triangle (i_1, i_2, i_3). This proves the existence of the claimed positions. By [92], $R(3, \dots, 3) \leq \mathcal{O}(l!)$, thus $\mathcal{R}_M = R(3, \dots, 3) = \mathrm{EXP}(\mathfrak{m} \cdot |M|^2 \cdot (|V| \cdot 2^k)^2)$. □

Lemma 3.4.6. Let σ be a strategy of Minimizer for G^m that uses a finite memory M. There exists a number $K_M \in \mathbb{N}$, such that every play with a payoff of at least K_M contains two consecutive infixes starting and ending in the same vertex with the same memory state and which have the same mark that is furthermore idempotent. K_M can be chosen as $\mathrm{EXP}(\mathcal{R}_M)$.

Proof. Let $f_1 : c \mapsto A_1 \cdot c + b_1$, $f_2 : c \mapsto A_2 \cdot c + b_2, \dots$ be the finitely many counter update functions appearing in G (or G^m). Let $a \in \mathbb{N}$ be maximal such that a occurs in some A_i or b_i, and let $c_0 := (a)^k \in \mathbb{N}^k$ be the vector that has a as entry in every row. Accordingly, when starting with c_0, the maximal counter value after one application of a counter update function is at most $a \cdot a \cdot k + a \leq a^2(k+1)$. After i applications, this generalizes to $a^{i+1}(k+1)^i$. Set

$$K_M := a^{\mathcal{R}_M + 1}(k+1)^{\mathcal{R}_M} + 1 = \mathrm{EXP}(\mathcal{R}_M),$$

from which it follows that every play with payoff $\geq K_M$ has a length of at least $\mathcal{R}_M$. By Lemma 3.4.5, the lemma is proved. □

Lemma 3.4.7. Let m be an idempotent mark, and let h be a nonzero mark-graph such that the nonzero mark-graph of $m(h)$ is again h. Let m' be an arbitrary mark, and let $i < k$.

If for all j with $(i, \triangle, j) \in m' \circ m(h)$, $\triangle \in \{\simeq, >\}$, it holds that $(j, >, j) \notin m(h)$, then for all j such that $(i, \triangle, j) \in m' \circ m(h)$, $\triangle \in \{\simeq, >\}$, we have $(j, \simeq, j) \in m(h)$.

Proof. Assume that for all j such that $(i, \triangle, j) \in m' \circ m(h)$, $\triangle \in \{\simeq, >\}$ it holds that $(j, >, j) \notin m(h)$.

Assume first that $(i, \simeq, j) \in m' \circ m(h)$. It follows from Remark 3.2.12 (iv) that there exists some l such that $(l, \simeq, j) \in m(h)$. We now distinguish two cases: If $l = j$, there is nothing to prove. In the other case, $l \neq j$. As m is idempotent, it follows that $(l, \simeq, j) \in m(m(h)) = m(h)$. Accordingly, there exists some l' such that $(l', \simeq, j) \in m(h)$, and $(l, \simeq, l') \in m(h)$, as $m(h)$ and h have the same nonzero pattern. But then $l' = j$ must hold, thus $(j, \simeq, j) \in m(h)$.

Assume now that $(i, >, j) \in m' \circ m(h)$. By Remark 3.2.12 (ii), there are two immediate cases: There is some l such that $(l, \simeq, j) \in m(h)$ or there is some l such that $(l, >, j) \in m(h)$, and i depends on l in m' (i.e., $(i, \triangle, l) \in m'(m^{>0}(h))$). If the former holds, the claim follows as in the case where $(i, \simeq, j) \in m' \circ m(h)$. Assume thus that the former does not hold, but the latter does. By the precondition, it follows that $l \neq j$. If $(j, \simeq, j) \in m(h)$, we are done, so also assume that $(j, \simeq, j) \notin m(h)$. By the precondition, if $(l, >, l) \in m(h)$, then $(i, >, l) \in m' \circ m(h)$, a contradiction. As $(j, >, j) \notin m(h)$ and $(j, \simeq, j) \notin m(h)$ and m is idempotent, it follows that $(l, >, j) \in m(h) = m(m(h))$ is only possible if there exists some l_2 such that $(l, >, l_2) \in m(h)$, and $(l_2, >, j) \in m(h)$. (The case where $(l_2, \simeq, j) \in m(h)$ was ruled out above, and $(l, \simeq, l_2) \in m(h)$ is impossible since $(l, >, j) \in m(h)$.) If $(l_2, >, l_2) \in m(h)$, then $(i, >, l_2) \in m' \circ m(h)$, a contradiction. So there must exist an l_3 such that $(l_2, >, l_3) \in m(h)$ and $(l_3, >, j) \in m(h)$. Analogously, $(l_3, >, l_3) \notin m(h)$, thus there must exist an l_4, and so on. As k is finite,there must be some l^* such that $(l^*, >, l^*) \in m(h)$ and $(i, >, l^*) \in m' \circ m(h)$, a contradiction. Accordingly, it follows that $(j, \simeq, j) \in m(h)$ must be true. □

In the remainder of this section, we prove that whenever Minimizer has a winning strategy with memory M in $\mathcal{U}(G)$, then this strategy ensures a payoff of less than K_M in G^m. Note that by the constraints on the information available to the strategy, strategies for Minimizer in $\mathcal{U}(G)$ correspond to strategies of Minimizer in G^m. To do so, we first separately prove another lemma to be used in the actual proof.

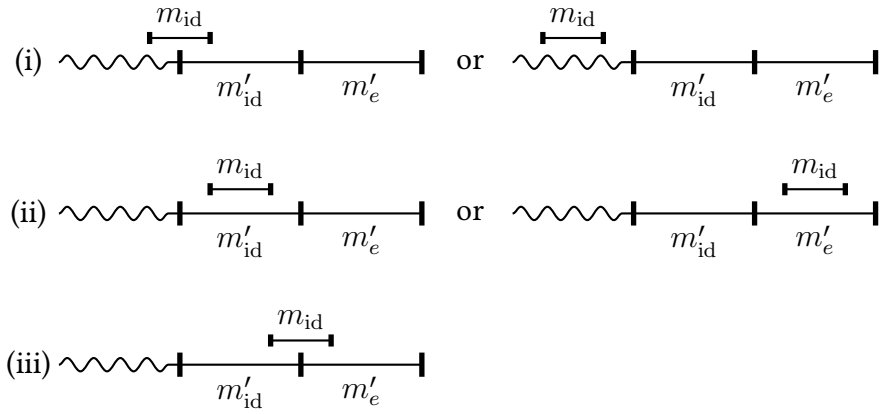


Figure 3.5: Possible locations of m_{id} in suffixes

Lemma 3.4.8. Let σ be a winning strategy of Minimizer with memory M in $\mathcal{U}(G)$. Every finite play ρ in G^m with $\mathrm{pay}(\rho) \geq K_M$ that is consistent with σ has a suffix π satisfying the following properties:

(i) π starts at $\pi_0 = (v, h)$ with memory state s.

(ii) For some $n > 0$, $\pi_n = (v, h)$ and the memory state is again s.

(iii) π ends in $\pi_m = (t, h')$ where $\tau(t) = (1, i)$.

(iv) m_{id}, the mark for $\pi_0 \dots \pi_n$, is idempotent.

(v) There is some j such that $(i, \simeq, j) \in m_e \circ m_{\mathrm{id}}(h)$ or $(i, \succ, j) \in m_e \circ m_{\mathrm{id}}(h)$, where m_e is the mark for $\pi_n \dots \pi_m$, and $(j, \succ, j) \in m_{\mathrm{id}}(h)$.

Proof. Assume towards a contradiction that ρ is a shortest play as required for which no such suffix exists. By Lemma 3.4.6, there exists at least one suffix π' and indices $p < q < r$ such that $\pi'_p = \pi'_q = \pi'_r = (v, h)$, all three with memory state s for some $s \in M$, and where m_{id} is an idempotent mark that is the mark for $\pi'_p \dots \pi'_q$ and $\pi'_q \dots \pi'_r$. Now let π' be the last such suffix, and let m_e be the mark from π'_r to the last vertex (t, h'). Let further i be such that $\tau(t) = (1, i)$. (Note that τ must assign a positive index to t, as otherwise the payoff would be negative.)

Let now $\widehat{\pi} = \pi'_0 \dots \pi'_q \pi'_{r+1} \dots$ be the path obtained by removing the second repetition of the idempotent mark. Note that $\widehat{\rho}$, obtained by replacing π' with $\widehat{\pi}$, is still a play consistent with σ, and the marks and memory states are still as before. It now suffices to prove that $\mathrm{pay}(\widehat{\rho}) \geq K_M$ and that $\widehat{\rho}$ does not end in a suffix π with properties (i) to (v) either, as ρ was chosen to be of minimal length and $\widehat{\rho}$ is shorter.

By choice of π', for all j such that $(i, \simeq, j) \in m_e \circ m_{\mathrm{id}}(h)$ or $(i, \succ, j) \in m_e \circ m_{\mathrm{id}}(h)$ it holds that $(j, \succ, j) \notin m_{\mathrm{id}}(h)$. Accordingly, by Lemma 3.4.7, for all such j we have $(j, \simeq, j) \in m_{\mathrm{id}}(h)$. As m_{id} is idempotent, the counter values of these counters c_j are identical at positions p, r, q, and thus, the payoff of $\widehat{\rho}$ equals the payoff of ρ. (As the payoff of ρ depends only on these values at positions q, while the payoff of $\widehat{\rho}$ depends only on the values at position p, which are the same.)

It remains to show that no suffix π exists in $\widehat{\rho}$ that satisfies properties (i) to (v). So assume that there is such a suffix π, such that $\pi_0 = (v', h'')$ with memory state s', $\pi_n = (v', h'')$ with the same memory state s', and marks m'_{id}, m'_e such that for some j where $(i, \triangle, j) \in m'_e \circ m'_{\mathrm{id}}(h'')$ it holds that $(j, \succ, j) \in m'_{\mathrm{id}}(h'')$. There are several cases, for which we demonstrate that they lead to contradictions when reinserting the m_{id}-infix. The cases are illustrated in Figure 3.5.

(i) The occurrence of m_{id} in $\widehat{\rho}$ is contained in the part of $\widehat{\rho}$ that comes before π_n, that is, the end of m_{id} lies before π_0 or between π_0 and π_n. If the second occurrence of m_{id} is reintroduced, we can simply consider the shifted suffix π, starting at the respective position for the second instead of the first occurrence of m_{id}, thus ρ contains a suffix π with properties (i) to (v), which is a contradiction.

(ii) The occurrence of m_{id} is either contained completely in m'_{id} or in m'_e. But as m_{id} is idempotent, reintroducing the second occurrence does not modify m'_{id} or m'_e, the respective suffix π would simply be longer. Again, we obtain a contradiction to the assumption that ρ does not contain such a suffix.

(iii) It could also happen that the start of m_{id} is within $\pi_0 \dots \pi_n$, and the end is after π_n. In this case, reintroducing the second occurrence leads to a possibly different mark m''_e, while m'_{id} is as before. But as $m_{\mathrm{id}} \circ m_{\mathrm{id}} = m_{\mathrm{id}}$, it follows that $m''_e \circ m'_{\mathrm{id}} = m'_e \circ m'_{\mathrm{id}}$. Accordingly, $(i, \triangle, j) \in m''_e \circ m'_{\mathrm{id}}(h'')$ for some $(j, \succ, j) \in m'_{\mathrm{id}}(h'')$. Once more we arrived at a contradiction, as ρ contains a suffix of the form π.

It follows that $\widehat{\rho}$ does not contain a suffix π with properties (i) to (v) either, in contradiction to ρ being the shortest such path. □

Lemma 3.4.9. Let $\sigma_1 \in \Sigma$ be a winning strategy of Minimizer in $\mathcal{U}(G)$ that uses a memory M. Then, $\mathrm{val}_{\sigma_1} G^m(v_0) < 2\mathrm{EXP}(\mathfrak{m} \cdot |M|^2 \cdot (|V| \cdot 2^k)^2)$.

Proof. Consider a play α consistent with σ_1 in G, and assume that it has a payoff larger than or equal to K_M from Lemma 3.4.6. Let $\widehat{\alpha}$ be the corresponding play in $\mathcal{U}(G)$, where moves of Maximizer are copied in such a

way that no C-labeled actions occur. If α is infinite, then $\widehat{\alpha}$ is also infinite. As σ_1 is winning in $\mathcal{U}(G)$, $\widehat{\alpha}$ does not satisfy the parity condition. As the colors in both plays are identical, the payoff of α must be $-\infty$, a contradiction. It follows that α is finite. By Lemma 3.4.8, α contains a cycle for which Maximizer has control of how many times it is repeated. Furthermore, every repetition of the cycle increases the payoff. But this induces a winning play for Maximizer in $\mathcal{U}(G)$ against σ_1: Maximizer can increase counter j, take a C-action to get the payoff via m_e, return and repeat. This play is still consistent with σ_1, as due to the restriction on the strategies the m_e-part is consistent and goes unnoticed, and the m_{id}-part induces a cycle within the memory structure. It follows that the payoff must be below K_M, thus at most $2\mathrm{EXP}(\mathfrak{m} \cdot |M|^2 \cdot (|V| \cdot 2^k)^2)$. □

3.5 Proof of Theorem 3.2.1

In the last sections, we introduced and proved all ingredients necessary for the main theorem, which is restated below.

Theorem 3.2.1. Let $\mathcal{G}$ be a counter parity game, and let v be some initial vertex. It is decidable in 4Exptime whether $\mathrm{val}\mathcal{G}(v) = \infty$. The exact value can be computed in 6Exptime. If the number of counters is fixed, the complexities drop to 2Exptime and 4Exptime, respectively.

Proof. According to Lemma 3.2.3, the value of a counter parity game G is ∞ if and only if Minimizer does not have a winning strategy with finite memory in $\mathcal{U}(G)$. By Corollary 3.3.11, if Minimizer can win, then also with a finite memory of size $K_0 = 2\mathrm{EXP}((|V| + k\mathfrak{m})^3)$ (see Section 3.4). Stated differently, checking whether the value is ∞ amounts to checking emptiness of a nondeterministic parity tree automaton of size $2\mathrm{EXP}((|V| + k\mathfrak{m})^3)$ with $|V|$ colors, which can be done in $2\mathrm{EXP}((|V| + k\mathfrak{m})^3)$, as $|V| < \mathrm{EXP}(|V|)$. Recall that $\mathfrak{m} = 2^{2^{2\log k + k^2}}$. Thus, checking whether the value is ∞ can be done in time 4-fold exponential in k, and 2-fold exponential in $|V|$.

Analogously, if the value is not ∞, then it is bounded from above by

$$\begin{aligned} K^+ &= \mathrm{EXP}(\mathcal{R}_{K_0}) = 2\mathrm{EXP}(\mathfrak{m}|K_0|^2(|V|2^k)^2) \\ &= 2\mathrm{EXP}(\mathfrak{m} \cdot 2\mathrm{EXP}((|V| + k\mathfrak{m})^3) \cdot (|V|2^k)^2) \\ &= 4\mathrm{EXP}((|V| + k\mathfrak{m})^3). \end{aligned}$$

In words, the upper bound is 4-fold exponential in $|V|$ and 6-fold in k. By using the same construction with players and signs switched, we either obtain that the value is $-\infty$, or a lower bound K^- which is also 4- respectively 6-fold exponential. Let $K = \max(K^+, K^-)$. It follows that the value of the

counter parity game is bounded from below by $-K$ and from above by K. Accordingly, counter values that exceed K can be ignored, as they can only be decreased by being set to a smaller constant.

By hardcoding the counter values in the positions, we construct an equivalent quantitative parity game of size $\mathcal{O}(K^k|V|)$ with at most $|V|$ colors. It follows from [45] that such a game can be solved in time $\mathcal{O}((K^k|V|)^{|V|})$. As K is 6-fold exponential in k and 4-fold exponential in $|V|$, it follows that the complexity is overall $\mathcal{O}(K)$, thus in 6Exptime and in 4Exptime if k is fixed. □

3.6 Summary

In this chapter we presented counter parity games, which are a special case of infinite quantitative parity games and which are useful in the field of quantitative logics. In fact, they have been used to model-check both the quantitative μ-calculus on initialized linear hybrid systems as well as a counting variant of this μ-calculus on a class of pushdown systems. We reduced the problem of solving such games to solving a game with both imperfect information and imperfect recall, for which we proved that finite-memory strategies suffice for one player. The overall proof yields an algorithm which is in 4Exptime for a fixed number of counters, and in 6Exptime otherwise.

Note that the aforementioned application to solve a counting variant of Qμ is discussed in Chapter 5.

Chapter 4

Mean Counter Games

In the previous chapter, we discussed counter parity games, a class of quantitative games where the quantitative aspect is modeled via a finite set of counters. However, in these games the payoffs of infinite plays, thus of models of infinite behavior, depend only on the qualitative parity condition, while the counters are only used to give payoffs to finite plays. In the current chapter, we discuss a variant where the payoff of infinite plays also depends on counter values. As it is known that parity games can be modeled as mean-payoff games, and solving these is also in NP ∩ coNP, our approach is based on the mean-payoff idea: Instead, however, of adding weights to the edges, we consider the average value of a designated register. This register cannot be accessed directly, but counter values can be copied there, which allows a manipulation of counters to be of effect only several steps later.

Part of the work in this chapter has been done jointly with Dietmar Berwanger and Marcus Gelderie, in particular the ideas to remove cycles and to search for a convenient notion of a pattern were developed together.

4.1 Definition and Examples

In contrast to counter parity games, we use a simpler class of counter manipulations for mean counter games: as in many models of automata with counters (e.g., B- and S-automata, see Chapter 6.3), counters may be increased, reset, or checked.

Definition 4.1.1. A *mean counter game* (MCG) $\mathcal{G} = (G, \mathrm{pay})$ with k counters is played on a finite arena $G = (V, V_0, V_1, E, \lambda)$ without terminal positions, where $\lambda\colon E \to \{ⓘ, ⓒ, ⓡ, ⓝ\}^k$ labels edges with k counter update actions, such that $\lambda(e)$ contains at most one ⓒ for every $e \in E$.

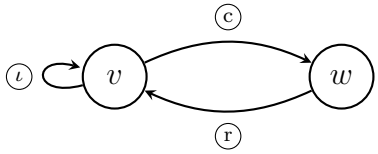


Figure 4.1: An example of a mean counter game

Given an infinite path $\alpha = v_0 v_1 \ldots$, this path induces a sequence of counter evaluations $\mathfrak{c} = c^0 c^1 \cdots \in (\mathbb{N}^k)^\omega$, where $c^0 = 0^k$, and

$$c_j^{i+1} := \begin{cases} c_j^i + 1, & \lambda(v_i, v_{i+1})_j = \text{ι}, \\ 0, & \lambda(v_i, v_{i+1})_j = \text{r}. \\ c_j^i, & \lambda(v_i, v_{i+1})_j \in \{\text{c}, \text{n}\}. \end{cases}$$

Based on $\mathfrak{c}$, we define the sequence $\rho = r_0 r_1 \cdots \in \mathbb{N}^\omega$ of *register values* by

$$r_0 := 0, \qquad r_{i+1} := \begin{cases} r_i, & \text{c} \notin \lambda(v_i, v_i + 1), \\ c_j^i, & \lambda(v_i, v_{i+1})_j = \text{c}. \end{cases}$$

The payoff function pay computes the limit inferior of the averages of prefixes of this sequence:

$$\mathrm{pay}(\alpha) := \liminf_{n \to \infty} \frac{\sum_{i=0}^{n} r_i}{n+1}.$$

As before, we call Pl. 0 Maximizer and Pl. 1 Minimizer.

Example 4.1.2. Before we discuss determinacy of MCGs, we consider a brief example. Consider the arena for one counter depicted in Figure 4.1. Assume first that all vertices belong to Minimizer. In this case, it is optimal to never choose the ι-loop. Thus, all counters always stay 0, and the sequence ρ of register values is also constantly 0. Accordingly, the value is 0.

Assume now that all positions belong to Maximizer. Clearly, never taking the ι-edge is not reasonable. On the other hand, always taking it is neither, as no value is checked in this case. We further remark that it is of no use for Maximizer to take the cr-loop twice in a row. Thus consider a play that is of the form $(\text{ι}^n \text{cr})^\omega$ for some $1 < n \in \mathbb{N}$. From some point onward, the sequence ρ of register values will also be constant, but will always be n. As this can be achieved for every n, it follows that the value of the game is ∞. In the present case, there are also optimal strategies, but they require infinite memory: for example, Maximizer could increase the number of ι-cycle iterations every time he reaches v again. ⊣

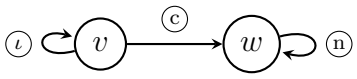


Figure 4.2: Maximizer does not have an optimal strategy

Nevertheless, optimal strategies for Maximizer need not exist in general. One can easily construct a single player mean counter game played only by Maximizer where no strategy is optimal: There are two vertices, v and w, such that v has a self-loop along which the sole counter is increased, and an edge to w labeled with ⓒ. The vertex w only has an ⓝ-labeled self-loop. This game, depicted in Figure 4.2, can be viewed as Maximizer choosing an arbitrary natural number which is then awarded to him as payoff. Of course, the supremum over all his strategies is unbounded, hence the value is ∞. But every fixed strategy yields only a finite payoff: 0 if w is not reached, and some $n \in \mathbb{N}$ otherwise.

Despite the fact that optimal strategies may not exist, mean counter games are always determined, which follows from Martin's theorem about Borel determinacy [74].

Theorem 4.1.3. Let $\mathcal{G}$ be a mean counter game. $\mathcal{G}$ is determined.

Proof. We implicitly consider $\mathcal{G}$ from some fixed initial vertex v. Payoffs of mean counter games are always elements of $\mathbb{R}^{\geq 0} \cup \{\infty\}$. Fix thus an arbitrary threshold value $\delta \in \mathbb{R}^{\geq 0} \cup \{\infty\}$. We prove that for every such δ, the win-lose game where Maximizer wins a play α if and only if $\mathrm{pay}(\alpha) \geq \delta$ is a Borel game, and hence determined. Since the rationals are dense in $\mathbb{Q}$, this means that ρ satisfies

$$\forall(\epsilon \in \mathbb{Q}^{>0})\exists(n_0 \in \mathbb{N})\forall(n > n_0)\left(\frac{1}{n}\sum_{i=0}^{n-1} r_i > \delta - \epsilon\right).$$

Fixing ϵ, n_0 and n, the set of paths where the average of the first n values of the respective ρ exceeds $\delta - \epsilon$ is an open set $x \cdot V^\omega \cap \mathrm{Paths}(G)$. (In fact, it is a *clopen* set, as the complement is also open by the same argument.) It follows that the set of paths such that Maximizer wins is a Borel set, and contained in Π_4^0 as a countable intersection of a countable union of a countable intersection of clopen sets. By [74], the win-lose game is determined for every threshold value δ.

Furthermore, if σ_0 is a winning strategy for Maximizer for threshold value δ, then it is also winning for every $\delta' < \delta$. Similarly, a strategy σ_1 for Minimizer winning for threshold value δ is also winning for all $\delta' > \delta$. Let

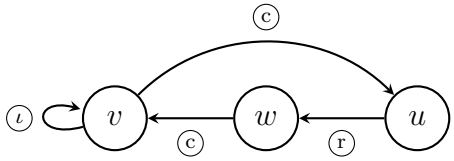


Figure 4.3: Maximizer needs infinite memory despite the value being finite

now S_0 be the set of all δ such that Maximizer has a winning strategy, and let S_1 be the analogous set for Minimizer. Clearly, $\sup S_0 = \inf S_1 = \mathrm{val}\mathcal{G}$. □

We already argued that Maximizer may not have optimal strategies, and that even if he does, he may require infinite memory. Note that even if the value of a game is finite, he may not have finite-memory strategies, as shown in the following example.

Example 4.1.4. Consider the single player Maximizer game in Figure 4.3. In this game, Maximizer can increase the counter while seeing register value 0, and then see the large counter for two steps until it is reset and the process starts all over again. Clearly, choosing larger counter values is advantageous for Maximizer; if he always chooses value N, the payoff is

$$\frac{2N}{N+3} < 2.$$

If the sequence of the respective N is monotonically increasing in every pass through v (which requires infinite memory), then the $\liminf$ of the average value of ρ is indeed 2. ⊣

As the register values are always positive, and discrete, it seems that the situation for Minimizer is different. Still, in contrast to classical mean payoff games, positional strategies do not suffice:

Example 4.1.5. Consider a game where Maximizer can reach vertex v via either setting counter c_0 to 2 or counter c_1 to 3, such that the other counter remains 0. Minimizer may then, at v, either choose c_0 or c_1 as the payoff of the play. Of course, this cannot be done positionally, as information about which counter is 0 is required. ⊣

We further remark that if the value is ∞, then every strategy of Minimizer is optimal. The findings above are summarized as follows.

Remark 4.1.6. 1. Maximizer may not have optimal strategies.

2. If Maximizer has optimal strategies, they may require infinite memory.
3. Optimal strategies of Minimizer may require memory.

We conjecture further that finite memory is sufficient for Minimizer, and that Maximizer has optimal strategies whenever the value is less than ∞. In the following, we focus on subclasses of mean counter games and prove that these can be solved. We study solitary one-counter games, both for Minimizer and Maximizer, which can also be viewed as special kinds of automata. We then prove that boundedness of the value of two-player one-counter games can be decided. Furthermore, we introduce a generalized version of mean counter games using affine functions (as in counter parity games), and discuss some properties of these games.

4.2 Solitary Minimizer Games with One Counter

The first subclass of mean counter games we investigate are games with only one counter where only Minimizer has any choices of next moves. Formally, we thus assume that $V_0 = \varnothing$. These games can be viewed as a class of ω-word automata where the counter is updated and checked. A run is eventually mapped to its payoff, that is, the payoff of the corresponding play, and a word is mapped to the infimum of the payoffs over all runs. Generally, this would mean adding additional labels from some alphabet Σ to the edges; for reasons of readability, we omit this here. A solitary game corresponds to a pathfinder game, thus it can either be viewed as a nondeterministic automaton, or, as an emptiness game. Consequently, solving the solitary minimizer game is equivalent to computing the minimum over all values of words.

The section is organized in two parts, where we first show that one can decide—using a simple criterion—if the value of a game is finite. We then prove, via a reduction to mean payoff games, that the value—if finite—can be computed.

4.2.1 Deciding Boundedness of the Value

To determine if a one-counter mean counter game where only Minimizer has any choices to make has a finite value, it suffices to investigate the cycles in the arena. In fact, as soon as Minimizer can reach a cycle that—repeated infinitely often—yields a finite value, a bound on the value is found. On the other hand, if no such cycle exists, then the value is unbounded.

A cycle is a finite path $\gamma = v_1 \dots v_n$ in the arena such that $(v_i, v_{i+1}) \in E$ for all i, and $(v_n, v_1) \in E$. A cycle is *simple*, if additionally $v_i \neq v_j$ for all $i \neq j$.

We say that a cycle γ is *good*, if at least one of the following three conditions is satisfied:

1. γ does not contain a $\textcircled{c}$-labeled edge.
2. γ contains an $\textcircled{r}$-edge.
3. γ does not contain an $\textcircled{\iota}$-edge.

Accordingly, a cycle is *bad* if none of these hold, that is, if γ contains both $\textcircled{c}$ and $\textcircled{\iota}$, but no $\textcircled{r}$.

Remark 4.2.1. If there exists a good cycle, then there also exists a good simple cycle: Otherwise, let $\gamma = v_1 \cdots v_i(v_{i+1} \dots v_{i+m})v_i v_{i+m+2} \dots v_n$ be a good cycle of minimal length, which thus cannot be a simple cycle. By assumption, $v_i v_{i+1} \dots v_{i+m}$ is bad. In order for γ to be good, $v_1 \dots v_i v_{i+m+2} \dots v_n$ contains an $\textcircled{r}$-labeled edge, and is thus also a good cycle, a contradiction. Conversely, it follows that if all simple cycles are bad, then so are all cycles.

Lemma 4.2.2. Let $\mathcal{G}$ be a one-counter Minimizer mean counter game, and let $v_0 \in V$. Then, $\mathrm{val}\mathcal{G}(v_0)$ is finite if and only if there exists a good simple cycle reachable from v_0.

Proof. If γ is a good simple cycle, reachable from v_0 via some path π, then consider the play $\gamma = \pi\gamma^\omega$. If γ does not contain a $\textcircled{c}$, then the value of α is the register value after π, which is finite as π is finite. If it does not contain $\textcircled{\iota}$, then the value is bounded by the counter value after π, which is again finite. If it contains $\textcircled{r}$, then the value is bounded by the number of $\textcircled{\iota}$-edges in γ or by the register value after π.

On the other hand, if there is no reachable simple cycle that is good, then all reachable simple cycles are bad. Thus, all cycles are bad. As the arena is finite, every play must contain some vertex v that appears infinitely often. Thus, decompose a play α into the cycles from v to v. Accordingly, the play is of the form $\alpha = \pi\gamma_1\gamma_2 \dots$ for cycles γ_i and some finite prefix. As all γ_i are bad, from some point onward, $\textcircled{c}$ and $\textcircled{\iota}$ are seen again and again, but no $\textcircled{r}$. Thus, the value is ∞. □

Corollary 4.2.3. An upper bound on the value of a one-counter Minimizer MCG can be computed. Furthermore, if the bound is finite, then it is linear in the size of the arena.

Proof. Via a depth-first search, all simple cycles reachable from the initial vertex can be computed. Using this, the bound is the minimum of the values of the respective initial paths followed by an infinite number of cycle iterations. □

4.2.2 Bounding Counter Values

By the above, it is possible to test whether the value of a one-counter Minimizer MCG is finite, and to compute an upper bound. However, it may still happen that the actual optimal value cannot be realized by repeating a simple good cycle infinitely often, but that more involved strategies are required. We now prove that, whenever the value is finite and cannot be obtained by repeating a simple cycle infinitely often, it can still be realized with the counters not exceeding a given bound B. To do so, we consider arbitrary plays and prove that in the case of counter values that are too large, we can find a play with bounded counters which has a payoff that is possibly lower, but not greater.

Let thus $\mathcal{G}$ be a game with a finite value, and let P be the bound on the value obtained as above. Let further α be an arbitrary play in $\mathcal{G}$ such that $\mathrm{pay}(\alpha) \leq P$. We want to prove that either α is of the form $\pi \cdot \gamma^\omega$ for some simple cycle γ, or that a payoff not larger than $\mathrm{pay}(\alpha)$ can also be obtained with counters not exceeding a certain bound. To ensure that we find a sufficient number of cycles in the play, we set the bound B to $B := |G|^{|G|}$. If now α is of the form $\pi \cdot \gamma^\omega$, or if all counter values (and thus, all register values) in α are less than B, there is nothing to show. So assume that there are counters which reach or exceed B and that α is not an infinite repetition of a simple cycle.

Let b be the index of the first position where the counter reaches B, that is, where $c^b = B$. Let a be the last position before b such that $\lambda(v_{a-1}, v_a) = \textcircled{r}$, or $a = 0$ if no such position exists. In the following, we consider cycles in $\alpha[a \dots b]$, where we classify two special kinds of cycles:

A cycle γ in $\alpha[a \dots b]$ is of type (i) if it contains increments, but no checks. A cycle γ is of type (ii) if it contains both increment and check, and both ρ and $\mathfrak{c}$ are strictly larger than $\mathrm{pay}(\alpha)$ at the start of the cycle.

We remark that if the register value is at most $\mathrm{pay}(\alpha)$ at the start of a cycle γ of type (i), we are done, as in this case we can repeat γ infinitely often, and ensure at most the same payoff. Thus, we assume from now on that all cycles of type (i) in $\alpha[a \dots b]$ start with a register value of more than $\mathrm{pay}(\alpha)$. Let further $\gamma(i)$ denote the index of the i-th position of the cycle in α. We prove next that cycles of type (i) and of type (ii) can be removed from the play without increasing the payoff.

Lemma 4.2.4.

1. Let $\gamma = g_0 \dots g_{n-1}$ be a cycle of type (i) in $\alpha[a \dots b]$ (i.e., $\alpha[\gamma(0)] = \alpha[\gamma(0) + n]$), and let α' be obtained from α by removing γ. Then, $\mathrm{pay}(\alpha') \leq \mathrm{pay}(\alpha)$.

2. Let γ be a cycle of type (ii) in $\alpha[a \dots b]$, and let α' be obtained from α by removing γ. Then, $\mathrm{pay}(\alpha') \leq \mathrm{pay}(\alpha)$.

Proof. We first prove Part 1. By assumption, it holds for the register values that $r_j > \mathrm{pay}(\alpha)$ for all $\gamma(0) \leq j \leq \gamma(n)$. Note that r_j is furthermore constant along γ, say $r_j = s$, as there are no checks.

Given a position j in α', let j^{-1} denote the respective corresponding position in α, that is, $j^{-1} = j$ if $j < \gamma(0)$, and $j + n$ otherwise. Obviously, if $j^{-1} = j$, then $\mathrm{avg}(\alpha', j) := (\sum_{i=0}^{j} r'_i)/(j+1) = \mathrm{avg}(\alpha, j^{-1})$. If $j \geq \gamma(0)$, we can relate the average in α' to that in α in the following way:

$$\mathrm{avg}(\alpha', j) = \frac{\sum_{i=0}^{j} r'_i}{j+1} = \frac{\sum_{i=0}^{j^{-1}} r_i - n \cdot s}{j + n + 1 - n}.$$

By assumption, $s > \mathrm{pay}(\alpha) + \epsilon$ for some $\epsilon > 0$ as $s > \mathrm{pay}(\alpha)$ and s is a natural number, and $\mathrm{avg}(\alpha, i) < \mathrm{pay}(\alpha) + \epsilon$ for infinitely many $i > b$. Let K be the set of these positions, and let K' be the set of positions in α' such that $\{j^{-1} \mid j \in K'\} = K$. We claim that $\mathrm{avg}(\alpha', j) \leq \mathrm{avg}(\alpha, j^{-1})$ for all $j \in K'$: This holds, as $(x - x')/(y - y') \leq x/y$ if and only if $x'/y' > x/y$ for natural numbers x, y, x', y', and $ns/n = s > \mathrm{pay}(\alpha) + \epsilon$. It follows that $\mathrm{pay}(\alpha') \leq \mathrm{pay}(\alpha)$.

Part (ii) follows analogously, as γ as an infix of $\alpha[a \dots b]$ does not contain resets, and thus, later counter and register values are possibly smaller, but not larger. □

Lemma 4.2.5. If there are no cycles of type (i) in $\alpha[a \dots b]$, then there are cycles of type (ii).

Proof. First of all, recall from the choice of a and b as positions after a reset and a check of a value of at least B, respectively, that $b - a > B$, and that at $\alpha[a]$, the counter is 0, while at $\alpha[b]$ it reaches B. Accordingly, and as there are no resets in $\alpha[a \dots b]$, there have to be cycles with increments (B is exponential in the size of the arena). Thus, if there are no cycles of type (i), that is, with increment, but without check, there have to be cycles with both increment and check. As $\mathrm{pay}(\alpha)$ is linear in the size of the arena, there must be such a cycle where both the counter and the register already exceed $\mathrm{pay}(\alpha)$. This is a cycle of type (ii). □

It follows from the above lemmas that any arbitrary finite number of cycles of type (i) or (ii) can be removed without increasing the payoff, and thus, the first occurrence of a counter value that reaches B can be postponed. Furthermore, since no cycles which yield a payoff $< \mathrm{pay}(\alpha)$ can be found in α, there are infinitely many checks and resets and infinitely often the register is below B. We now consider the limit α^∞, where cycles of type (i) and (ii) are removed until no counter reaches B. Note that this limit always exists, and is a proper infinite path in the arena, as infinitely many positions with register and counter values below B occur in α. However, we still have to prove that the payoff still does not exceed $\mathrm{pay}(\alpha)$.

To do this, we prove that in plays α, the average at positions i with $r_i > \mathrm{pay}(\alpha)$, provided that it is less than r_i, cannot be smaller than at position $i-1$.

Lemma 4.2.6. Let α be a play, and let i be such that $\mathrm{avg}(\alpha, i) < r_i$. Then, $\mathrm{avg}(\alpha, i) \geq \mathrm{avg}(\alpha, i-1)$.

Proof. Note that the average at position i can be written as

$$\mathrm{avg}(\alpha, i) = \frac{\left(\sum_{j=0}^{i-1} r_j\right) + r_i}{i-1+1+1}, \qquad \text{where } \mathrm{avg}(\alpha, i-1) = \frac{\sum_{j=0}^{i-1} r_j}{i-1+1}.$$

Note that also $\mathrm{avg}(\alpha, i-1) < r_i$, as otherwise $\mathrm{avg}(\alpha, i) \geq r_i$. Thus, as $r_i/1 > \mathrm{avg}(\alpha, i)$, it follows, as in Lemma 4.2.4, that $\mathrm{avg}(\alpha, i) \geq \mathrm{avg}(\alpha, i-1)$. □

Now, as in all cycles that are removed to obtain α^∞, it holds that the register values exceed $\mathrm{pay}(\alpha)$, the positions where the average is ϵ-close to $\mathrm{pay}(\alpha)$ are preserved, and, by Lemma 4.2.4, have an average in α^∞ that is at most the respective average in α. It follows that $\mathrm{pay}(\alpha^\infty) \leq \mathrm{pay}(\alpha)$.

Lemma 4.2.7. Given a one-counter Minimizer MCG with finite value, the optimal value can be realized by repeating a simple cycle infinitely often, or by playing such that no counter reaches B.

4.2.3 A Reduction to Mean-Payoff Games

Combining the results about good simple cycles and about the bound on counters, we prove that one-counter Minimizer mean counter games can be reduced to Minimizer mean-payoff games on finite graphs. This allows us to conclude that finite-memory strategies suffice in such MCGs.

The main idea is to explicitly store counter and register values in the game arena, up to the bound B. Additionally, we add sink states for good simple cycles. Note, however, that we only need to consider sink states for those simple cycles which contain no check, as whenever a check is contained, then either also a reset, or no increment, so the counter is always bounded by B. For technical reasons we also add a sink state where Minimizer immediately loses, which is modeled by a payoff of $B+1$ that cannot be optimal.

The mean-payoff game $\mathcal{G}'$ is played on the arena $G' = (V', V_0', V_1', E', \lambda')$, where the components are defined as follows:

- Let $\gamma_1, \ldots, \gamma_n$ denote all good simple cycles that do not contain a check. The vertex set V' is

$$V' := (V \times [B] \times [B]) \cup \{B\} \cup \bigcup_{i=1}^{n} (\{\gamma_i\} \times [B]).$$

  As in $\mathcal{G}$, we set $V_0' := \varnothing$.

- Given $(n, m) \in [B]^2$, we define the following abbreviations:

$$\textcircled{\iota}(n,m) := (n+1, m), \qquad \textcircled{c}(n,m) := (n,n),$$
$$\textcircled{r}(n,m) := (0, m), \qquad \textcircled{n}(n,m) := (n,m).$$

  The edges in G' follow the structure of G:

  For $(v, w) \in E$ with $\lambda(v, w) = \textcircled{\iota}$, we have

$$\{((v,n,m),(w,\textcircled{\iota}(n,m))) \mid n,m \in [B-1]\} \subseteq E',$$

  and

$$\{((v,B-1,m),B) \mid m \in [B-1]\} \subseteq E'.$$

  For $(v, w) \in E$ where $\lambda(v, w) \neq \textcircled{\iota}$, all edges have counters lower than B, so:

$$\{((v,n,m),(w,\lambda(v,w)(n,m))) \mid n,m \in [B]\} \subseteq E'.$$

  To model the simple cycles, we have edges $((v, n, m), (\gamma_i, m)) \in E'$ whenever v is an element of γ_i.

  Last, all vertices (γ_i, n) have self-loops, and so does B.

- As for the edge labeling, outgoing edges of vertices (v, n, m) have edge-labeling m, while $\lambda(B, B) = B + 1$, and $\lambda((\gamma_i, m), (\gamma_i, m)) = m$.

Plays in $\mathcal{G}'$ can be of two forms: they can be contained completely in $V \times [B]^2$, or they can reach a sink state, either of the form (γ_i, m) for some i and some m, or B. For plays α' in $V \times [B]^2$, there is a one-to-one-correspondence to plays in $\mathcal{G}$: Let α be the projection of α' to the V-components. It is immediate that α is a play in $\mathcal{G}$. Furthermore, it is easy to see that the payoffs are identical: the sequence of edge weights in α' corresponds exactly to the sequence of register values ρ.

For plays α' that reach (γ_i, m) and stay there, it is, by the same argument, possible to reach the simple cycle γ_i with the same register value in $\mathcal{G}$. As γ_i does not contain checks, playing it infinitely often yields the same payoff as the self-loop at (γ_i, m).

Lemma 4.2.8. $\mathrm{val}\mathcal{G}(v_0) = \mathrm{val}\mathcal{G}'(v_0, 0, 0)$.

Proof. As the optimal value can be reached by either keeping counters lower than B or by a simple good cycle that does not contain a check, optimal plays in $\mathcal{G}$ can be simulated in $\mathcal{G}'$. The only plays in $\mathcal{G}'$ that cannot be directly translated as explained above are those reaching B. But as such plays cannot be the only optimal ones according to Lemma 4.2.7, the lemma follows. □

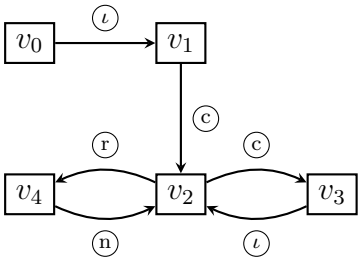


Figure 4.4: Minimizer needs finite memory in one-counter games

As mentioned in Chapter 2.2.2, mean-payoff games on finite arenas are positionally determined. As one only needs to store a finite amount of information to simulate $\mathcal{G}'$ in $\mathcal{G}$, it follows that finite-memory strategies suffice. This holds also for infinite values, as whenever the value is infinite, every strategy of Minimizer is optimal, so in particular every finite-memory strategy.

Corollary 4.2.9. In one-counter Minimizer MCGs, Minimizer can play optimally using finite-memory strategies.

The following example shows that finite memory is really required, and positional strategies do not suffice in general.

Example 4.2.10. Let $\mathcal{G}$ be the game from Figure 4.4. By alternating between v_4 and v_3, Minimizer can achieve a payoff of 0. However, $\mathrm{pay}(v_0v_1(v_2v_3)^\omega) = \infty$ and $\mathrm{pay}(v_0v_1(v_2v_4)^\omega) = 1$. ⊣

4.3 Solitary One-Counter Maximizer Games

Next, we consider the subclass of MCGs dual to the one discussed above. Thus, we consider games where Minimizer has no choices. Again, we first prove that one can decide if the value is bounded or not, and that one can also compute a bound. We then investigate what classes of strategies are required. It already follows from above examples that infinite memory is required, but we prove that the form of infinite memory that suffices is still comparatively simple; we prove that Maximizer can play optimally by repeating a fixed pattern more and more often. As for solitary Minimizer games, the player can be viewed as a pathfinder, and thus the game corresponds to a nondeterministic automaton where the maximum value over all words is computed, or the respective supremum.

4.3.1 Boundedness is Decidable

To show that the value of a solitary Maximizer MCG is finite, we introduce so-called ∞-arenas: these consist of two connected cycles using which Maximizer can achieve arbitrarily high payoffs. This means that Maximizer has a sequence of strategies on the ∞-arena such that the supremum over the payoffs realized by these strategies is ∞. In the subsequent definition, let $A = \{ⓘ, ⓒ, ⓡ, ⓝ\}$ be the set of counter update operations.

Definition 4.3.1. An ∞-arena is a tuple $(\pi_1, \pi_2, \pi_3, \pi_4)$ of finite paths such that π_1, π_4 are cycles (i.e., $(\mathrm{last}(\pi_1), \mathrm{first}(\pi_1)) \in E$ and $(\mathrm{last}(\pi_4), \mathrm{first}(\pi_4)) \in E$), $\pi_1\pi_2\pi_3\pi_4$ is a path, and the following conditions hold:

- $\lambda(\pi_1\pi_1) \in (A \setminus \{ⓡ\})^* ⓘ (A \setminus \{ⓡ\})^*$, i.e., on the cycle π_1, at least one increment occurs, but no reset,
- and $\lambda(\mathrm{last}(\pi_1)\pi_2) \in (A \setminus \{ⓡ\})^* ⓒ$, i.e., π_2 ends with a check an no reset is seen before,
- and either

$$\lambda(\mathrm{last}(\pi_2)\pi_3\mathrm{first}(\pi_4)) \in (A \setminus \{ⓡ\})^*$$
$$\text{and } \lambda(\pi_4\pi_4) \in (A \setminus \{ⓒ\})^* + (A \setminus \{ⓡ\})^*$$

  or

$$\lambda(\mathrm{last}(\pi_2)\pi_3\mathrm{first}(\pi_4)) \in (A \setminus \{ⓡ\})^* ⓡ (A \setminus \{ⓒ\})^*$$
$$\text{and } \lambda(\pi_4\pi_4) \in (A \setminus \{ⓒ\})^*.$$

  This entails that no check occurs after a reset in π_3 and π_4.

Note that in the first condition on π_1, the cycle is considered twice to take into account the edge from $\mathrm{last}(\pi_1)$ to $\mathrm{first}(\pi_1)$.

Lemma 4.3.2. If there exists a reachable ∞-arena, then the value of the game is infinite.

Proof. As it is a single player game where only Maximizer has any choices, the value of the game corresponds to the supremum of the payoffs over all strategies of Maximizer. Consider the sequence $(\sigma_i)_{i \in \omega}$ of strategies, where the unique play consistent with σ_i is $\alpha_i := \pi_1^{i+1}\pi_2\pi_3\pi_4^\omega$. As π_1 contains at least one increment, and no reset occurs until the check in π_2, and later either no check occurs, or even larger values are checked, it follows that $\mathrm{pay}(\alpha_i) \geq i$. Accordingly, the supremum over all i of the payoffs of α_i is ∞, and so is the value. □

While one direction is straightforward, the converse direction also holds, but the proof is more technical.

Lemma 4.3.3. If there exists no reachable ∞-arena, then the value is bounded by $3|V|$.

Proof. If there are no cycles with increment but without reset, then all counters are bounded by $|V|$, and so are all register values. As no register can reach $|V|$, the payoff of every play is at most $|V| - 1$, and so the value is also less than $|V|$.

However, larger counter values may occur, but when they are checked, they cannot stay in the register for arbitrary amounts of time, as this would entail the existence of an ∞-arena. This means that there may be cycles with increments and without reset, but such cycles cannot lead into a cycle with check and without reset. In fact, whenever a counter value of at least $N := |V|$ is checked, after a small number of steps, a value of less than N is checked:

Claim. Let α be a play with register values of at least N. Let i be a position such that $r_i \geq N$, and $r_{i-1} < N$. Then there exists a $j \leq 2N$ such that $r_{i+j} < N$.

> *Proof.* Consider $\alpha_i \dots \alpha_{i+N}$. There exist some $n < m \leq N$ such that $\alpha_{i+n} = \alpha_{i+m}$. As the precondition of the lemma states that there is no reachable ∞-arena, we can distinguish two cases: either there is a reset between α_i and α_{i+n} and a check on the cycle $\alpha_{i+n} \dots \alpha_{i+m}$, or there is a reset on the cycle (in fact also a check, but this may come before the reset). Note that if none of the two above were true, then the cycle could be repeated infinitely often, securing a payoff of at least N. As to achieve a register value of at least N, a cycle containing increment but no reset is required, an ∞-arena is found. In the case where a reset occurs between α_i and α_{i+n} and a check between α_{i+n} and α_{i+m}, it follows immediately that after m steps, a value of less than N has been checked. Otherwise, there exist n', m' with $m \leq n' < m' \leq m + N$ such that $\alpha_{i+n'} = \alpha_{i+m'}$. As above, the cycle $\alpha_{i+n'} \dots \alpha_{i+m'}$ contains a check, as otherwise an ∞-arena were reachable. If a counter value of less than N is checked, there is nothing left to show. Thus, assume that a larger value is checked. But then a cycle must exist between the reset and the check, which in turn must already contain a check of a smaller value, because there are no reachable ∞-arenas. This concludes the proof of the claim. □

We use the claim to prove that every play where register values reach and exceed N has a payoff of at most $3N$. For such a play α, we consider the decomposition $\alpha = \vartheta_0\pi_0\vartheta_1\pi_1 \dots$ where the π_i are those segments where the register values are at least N, while in the ϑ_i-segments they are always below N. By the above claim, we know that the length of every π_i is bounded

by $2N$. We consider two cases: If there is a bound h on the length of the ϑ_i, then register values are bound from above by $h+2N$, as at the beginning of a ϑ_i, they are bound by N, thus at the end by $N+h$, and after N steps in π_i, a reset must have occurred (because there are no ∞-arenas). It is furthermore immediate that the average of the register values along a ϑ_i is bound by $N-1$. Hence,

$$\begin{aligned}\mathrm{pay}(\alpha) = \liminf_{n\to\infty} \frac{\sum_{i=0}^{n} r_i}{n+1} &\leq \frac{(N-1)h + 2N(h+2N)}{h+2N} \\ &= 2N + \frac{h(N-1)}{h+2N} \leq 3N.\end{aligned}$$

If there is no such bound h, then

$$\begin{aligned}\mathrm{pay}(\alpha) &\leq \liminf_{n\to\infty} \frac{(N-1)n + 2N(2N+n)}{n+2N} \\ &\leq \liminf_{n\to\infty} \frac{(N-1)n + 2Nn + 4N^2}{n} \leq 3N.\end{aligned}$$

□

Corollary 4.3.4. The value of a one-counter solitary Maximizer MCG is either infinite or bounded by $3|V|$. Furthermore, this is effectively decidable.

4.3.2 Solving Solitary Maximizer Games

With the above, it is decidable whether the value is ∞ or finite. We now prove that in the case of a finite value, the exact value can be computed. Hence, we assume for the remainder of this section that the value of the mean counter game is finite. If the arena is such that there exists a bound on the counter values that may occur (or on the register values), then the problem immediately reduces to solving a mean-payoff game on a finite arena. However, solving a game where register values may be arbitrarily large requires a more technical construction. To this end, we introduce patterns, which informally correspond to a component to increase the counter, and one to check it afterwards. We prove that, given a simple pattern, one can compute the payoff of playing optimally using this pattern, and then prove that simple patterns suffice to ensure the optimal payoff. This allows us to compute the value with the help of reductions to mean-payoff games.

4.3.2.1 Introducing Patterns

As said above, a pattern consists of a component to increase the counter, and one to check it. As arbitrarily large counters are intended to be realizable by patterns, the first component will be an increment cycle. Note that it follows from the above result on boundedness that large values in the register are overwritten after a bounded number of steps.

Definition 4.3.5. A *pattern* $p = (\gamma, t)$ consists of a simple cycle $\gamma = v_0 \dots v_{n-1}$ (where $(v_{n-1}, v_0) \in E$) on which there is at least one increment, but no reset, and a finite path t, called *tail*, starting in v_0, which contains no cycle more than once, in which a check occurs before a reset, and which ends with the first check after a reset. Formally, $\lambda(t) \in \{ⓘ, ⓝ\}^* ⓒ (A \setminus \{ⓡ\})^* ⓡ (A \setminus \{ⓒ\})^* ⓒ$.

As by assumption the value of the game is finite, it immediately follows for every pattern that γ does not contain a check. For a pattern to be usable in a strategy, it needs to be possible to return to γ from t. Given two (not necessarily different) patterns p_1, p_2, a *connector* ξ is a finite path from $\mathrm{last}(t_1)$ to $\mathrm{first}(\gamma_2)$, such that every cycle occurs at most once. If $p_1 = p_2 = p$ for a connector ξ, we say that ξ is a connector on p.

Given a pattern p and a connector ξ on p, we are interested in *pattern strategies*: These can be described by a function $f\colon \mathbb{N} \to \mathbb{N}$, where $f(i)$ is the number of iterations of γ in the i-th pass. Thus, p, ξ and f yield a unique play, and we write $\mathrm{pay}(p, \xi, f)$ for its payoff:

$$\mathrm{pay}(p, \xi, f) := \mathrm{pay}(\gamma^{f(0)} t \xi \gamma^{f(1)} t \xi \gamma^{f(2)} \dots).$$

For a fixed pattern p and a connector ξ, we say the value of this pattern with the connector is the supremum of the payoffs over all pattern strategies:

$$\mathrm{val} p, \xi := \sup_{f:\mathbb{N}\to\mathbb{N}} \mathrm{pay}(p, \xi, f).$$

Towards computing this value, note that both counter and register values at the end of t do not depend on the number of iterations of γ. Thus, the sequence of register values along ξ can be computed, as it is the same for every pass through the connector, and so can the counter values. Let C_ξ, R_ξ denote the counter and register values when entering γ. Thus, the average of the register values along the pattern when γ is iterated n times amounts to

$$\mathrm{avg}_\xi(p, n) = \frac{n|\gamma| R_\xi + \mathrm{acc}_t(C_\xi + n I_\gamma, R_\xi)}{n|\gamma| + |t|},$$

where I_γ is the number of increments in γ, and $\mathrm{acc}_t(C, R)$ is the sum of register values along t when the initial counter value is C and the register value R. The second summand acc_t is obviously computable, and can be written as $\Theta_t + n' n I_\gamma$ for some constants Θ_t, n' obtainable from t and C_ξ. (Θ_t incorporates how many steps C_ξ contributes to the register and how many steps R_ξ remains in the register before the first check, and also the respective effects of the increments in t are contained. n' similarly is the number of steps for which the increments in γ affect the register.)

Note that $\mathrm{avg}_\xi(p,n)$ is monotone in n, as the sign of its first derivative depends only on $|\gamma||t|R_\xi - |\gamma|\Theta_t + n'I_\gamma|t|$, and is thus independent of n. Accordingly, for increasing n the averages tend to a limit $A_p = \lim_{n\to\infty} \mathrm{avg}_\xi(p,n)$. (As the value of the game is finite, the limit exists.) Let A_ξ denote the average of register values along the connector (which is independent of the number of cycle iterations). If $A_p > A_\xi$, then we prove that $\mathrm{val}p,\xi = \mathrm{pay}(p,\xi,f)$ for some strictly increasing f, and as the connector becomes more and more insignificant, $\mathrm{val}p,\xi = A_p$. If $A_p \leq A_\xi$, then either repeating the cycle only once or completely omitting it is optimal, as repeating the cycle more often would increase the weight of A_p in the weighted sum of A_p and A_ξ that determines the payoff.

As a first step, we compute the limit A_p. As argued above, the sum of register values along a pass of p with n iterations of γ is

$$\mathfrak{S}_p(n) := n|\gamma|R_\xi + \Theta_t + n'nI_\gamma.$$

Thus, the limit A_p is

$$A_p = \lim_{n\to\infty} \mathrm{avg}_\xi(p,n) = \lim_{n\to\infty} \frac{\mathfrak{S}_p(n)}{n|\gamma| + |t|} = R_\xi + \frac{n'I_\gamma}{|\gamma|}.$$

The payoff for pattern functions We now analyze the payoff of pattern functions $f\colon i \mapsto f(i)$ (which we assume to be strictly monotonically increasing), under the assumptions that the function $\mathrm{avg}_\xi(p,n)$ is monotonically increasing. (Otherwise, a constant number of 0 or 1 iterations is optimal. Note that this can be checked via the numerator of the first derivative given above.) Furthermore, for the same reasons we assume that A_p is larger than the average along the connector ξ, as otherwise using the pattern should be avoided. In the analysis, we prove that the payoff for pattern strategies with these assumption is optimal for polynomial f, hence we can fix the identity function as an optimal pattern function.

To simplify the arguments, we omit the 0-fold iteration, and let α start at position 1 with counter value C_ξ and register value R_ξ. Thus, we consider the play

$$\alpha = \pi_1\xi\pi_2\xi\pi_3\xi\ldots,$$

where $\pi_i = \gamma^{f(i)}t$. Accordingly, the π_i are of increasing length. Given an index j of α, let S_j denote the highest i such that π_i is completely contained in $\alpha[1\ldots j]$. For some index j, there are S_j complete π_i segments, and $P_j = S_j - 1$ or $P_j = S_j$ occurrences of ξ in $\alpha[1\ldots j]$. Let $\mathfrak{S}_\xi$ denote the sum of register values along a single pass of ξ, and let, given j, m_j denote the length of

the last non-completed segment minus one (so $\alpha[j - m_j \dots j]$ is an initial segment of ξ or of π_{S_j+1}). We can thus write the payoff as follows.

$$\begin{aligned}\mathrm{pay}(p,\xi,f) &= \liminf_{j\to\infty} \mathrm{avg}(\alpha[1\dots j]) \\ &= \liminf_{j\to\infty} \frac{\sum_{i=1}^{S_j}(|\pi_i|\mathrm{avg}_\xi(p,f(i))) + P_j\mathfrak{S}_\xi + \sum_{i=j-m_j}^{j} r_i}{j}\end{aligned}$$

As $A_p \geq \mathrm{avg}_\xi(p,n)$ for all n, and $A_p > A_\xi$ for the average A_ξ along ξ, it follows that $\mathrm{pay}(p,\xi,f) \leq A_p$.

We argue next that ξ can be neglected: First of all, if $\lim_{j\to\infty} \frac{P_j\mathfrak{S}}{j}$ exists, then

$$\mathrm{pay}(p,\xi,f) = \liminf_{j\to\infty} \frac{\sum_{i=1}^{S_j}|\pi_i|\mathrm{avg}_\xi(p,f(i)) + \sum_{i=j-m_j}^{j} r_i}{j} + \lim_{j\to\infty} \frac{P_j\mathfrak{S}_\xi}{j}.$$

As f is strictly monotonically increasing, the π_i are longer and longer. Thus, j grows much faster than P_j. In fact, if $f = \mathrm{id}$, then $j \geq \sum_{i=1}^{S_j}|\pi_i| \geq \sum_{i=1}^{S_j} i \geq (S_j^2 + S_j)/2 \geq (P_j^2 + P_j)/2$, and otherwise, j is even larger compared to P_j. It follows that $\lim_{j\to\infty} \frac{P_j\mathfrak{S}_\xi}{j} \leq \mathfrak{S}_\xi \lim_{j\to\infty} \frac{2P_j}{P_j^2+P_j}$ goes to 0. Furthermore, if $\alpha[j - m_j \dots j]$ is an initial segment of ξ, then $P_j = S_j - 1$, and the limit still goes to 0. Accordingly, for the $\liminf$ we can assume $\alpha[j - m_j \dots j]$ to be an initial segment of π_{S_j+1}.

It remains to consider

$$L := \liminf_{j\to\infty} \frac{\sum_{i=1}^{S_j}|\pi_i|\mathrm{avg}_\xi(p,f(i)) + \sum_{i=j-m_j}^{j} r_i}{j},$$

where we can safely assume that $\alpha[j - m_j \dots j]$ is an initial segment of π_{S_j+1}. From the definition of patterns, it thus follows that the lowest current average is reached just before the first check in t (as a large value is checked then and the register remains large until the end of t). It also follows that the register is constantly R_ξ until then. Let T be the index in t after which the first check occurs. Thus,

$$L = \liminf_{j\to\infty} \frac{\sum_{i=1}^{S_j}|\pi_i|\mathrm{avg}_\xi(p,f(i)) + (f(S_j+1)|\gamma| + T)R_\xi}{j}.$$

As before, since $\lim_{j\to\infty} \frac{TR_\xi}{j} = 0$, T can be omitted. By the above assumptions, we know that $\mathrm{avg}_\xi(p,n)$ is larger R_ξ. This already hints towards the issue that choosing functions f that grow too fast might affect the payoff negatively.

As we assumed that $\mathrm{avg}_\xi(p,n)$ is monotonically increasing in n, it follows that $\mathrm{avg}_\xi(p,n) = A_p - \epsilon_n$, where $\lim_{n\to\infty} \epsilon_n = 0$. Using this, the above can be rewritten to

$$\begin{aligned} L &= \liminf_{j\to\infty} \frac{\sum_{i=1}^{S_j} |\pi_i| A_p - \sum_{i=1}^{S_j} \epsilon_{f(i)} + f(S_j+1)|\gamma| R_\xi}{j} \\ &= \liminf_{j\to\infty} \frac{\sum_{i=1}^{S_j} |\pi_i| A_p + f(S_j+1)|\gamma| R_\xi}{j}. \end{aligned}$$

By definition of patterns, $|\pi_i| = f(i)|\gamma| + |t|$. Furthermore, the path up to positions j before the check in t consists of segments π_i for all $i \leq S_j$, as well as S_j occurrences of ξ and the first $f(S_j+1)|\gamma| + T$ steps of π_{S_j+1}. Hence,

$$L = \liminf_{j\to\infty} \frac{\sum_{i=1}^{S_j} (f(i)|\gamma| + |t|) A_p + f(S_j+1)|\gamma| R_\xi}{\sum_{i=1}^{S_j} (f(i)|\gamma| + |t|) + S_j|\xi| + f(S_j+1)|\gamma| + T}.$$

It remains to analyze when the weighted sum tends to A_p, and when to R_ξ. Therefore, note that in the denominator, the summand $f(S_j+1)|\gamma|$ can be added to the first sum $\sum_{i=1}^{S_j} f(i)|\gamma|$. Now, $L = A_p$ if $\lim_{j\to\infty} \frac{f(S_j)}{\sum_{i=1}^{S_j} f(i)} = 0$, as in this case, the weight of A_p tends to 1, while that of R_ξ tends to 0. For polynomials, this holds: $\sum_{i=1}^{S_j} i^k$ is a polynomial of degree $k+1$. For exponentials a^i, however, the limit is $\frac{1}{a}$, thus $R_\xi < L < A_p$. If f grows very fast, the limit may also be R_ξ. Thus, polynomials are optimal for the purpose of maximizing the limit average, and as all polynomials yield the same average, we can use $\mathrm{id}\colon n \mapsto n$ as the canonical pattern function.

Corollary 4.3.6. Let p be a pattern and let ξ be a connector such that $\mathrm{avg}_\xi(p,n)$ is monotonically increasing and $A_p > A_\xi$ holds. Let $f\colon \mathbb{N} \to \mathbb{N}$ be a strictly monotonically increasing polynomial. Then,

$$\mathrm{pay}(p,\xi,f) = \mathrm{pay}(p,\xi,\mathrm{id}) = A_p = R_\xi + \frac{n' I_\gamma}{|\gamma|} = \mathrm{val} p,\xi.$$

Corollary 4.3.7. Let p be a pattern and let ξ be a connector such that $\mathrm{avg}_\xi(p,n)$ is monotonically increasing and $A_p > A_\xi$ holds. Then $\mathrm{val} p,\xi = \mathrm{pay}(p,\xi,\mathrm{id})$ can be computed.

4.3.2.2 Computing the Value

So far, we introduced patterns and showed that either the identity function is an optimal pattern function (if the average function for the pattern is monotone and $A_p > A_\xi$ holds, of which both are decidable), or the constants 0 or 1 are. In the following, we extend the arena by storing exact values for

counter and register up to a bound of $3N$. For this arena, we compute the graph of maximal strongly connected components, which is acyclic. We then argue that we can compute the values for the strongly connected components separately, and prove that one of these is the value of the game (which is finite by assumption).

Let $\widetilde{G}$ be the following graph:

$$
\begin{aligned}
\widetilde{V} :=& V \times ([3N] \cup \{\top\})^2 \\
\widetilde{E} :=& \{((v,n,m),(w,a(n,m))) \mid (v,w) \in E, \textcircled{\iota} \neq a = \lambda(v,w)\} \\
&\cup \{((v,n,m),(w,a(n,m))) \mid (v,w) \in E, \textcircled{\iota} = \lambda(v,w), n < 3N-1\} \\
&\cup \{((v,3N-1,m),(w,\top,m)) \mid (v,w) \in E, \lambda(v,w) = \textcircled{\iota}\} \\
&\cup \{((v,\top,m),(w,\top,m)) \mid (v,w) \in E, \lambda(v,w) \notin \{\textcircled{\mathrm{r}},\textcircled{\mathrm{c}}\}\} \\
&\cup \{((v,\top,m),(w,0,m)) \mid (v,w) \in E, \lambda(v,w) = \textcircled{\mathrm{r}}\} \\
&\cup \{((v,\top,m),(w,\top,\top)) \mid (v,w) \in E, \lambda(v,w) = \textcircled{\mathrm{c}}\} \\
&\cup \{((v,\top,\top),(w,\top,\top)) \mid (v,w) \in E, \lambda(v,w) \neq \textcircled{\mathrm{r}}\} \\
&\cup \{((v,\top,\top),(w,0,\top)) \mid (v,w) \in E, \lambda(v,w) = \textcircled{\mathrm{r}}\} \\
&\cup \{((v,n,\top),(w,n+1,\top)) \mid (v,w) \in E, \lambda(v,w) = \textcircled{\iota}, n+1 < N\} \\
&\cup \{((v,n,\top),(w,0,\top)) \mid (v,w) \in E, \lambda(v,w) = \textcircled{\mathrm{r}}\} \\
&\cup \{((v,n,\top),(w,n,\top)) \mid (v,w) \in E, \lambda(v,w) = \textcircled{\mathrm{n}}\} \\
&\cup \{((v,n,\top),(w,n,n)) \mid (v,w) \in E, \lambda(v,w) = \textcircled{\mathrm{c}}\}.
\end{aligned}
$$

Note that increment edges from $(v, N-1, \top)$ need not be defined, as positions of this form with outgoing increment edges are unreachable. This follows from the non-existence of ∞-arenas. In fact, there are no cycles within $(V \times ([3N] \cup \{\top\}) \times \{\top\}, \widetilde{E})$. Furthermore, there is a one-to-one correspondence between infinite paths in G and in $\widetilde{G}$, by storing the exact values of counters and registers up to $3N$.

Let $\mathfrak{C}(G)$ be the acyclic graph whose vertex set is the set $\mathrm{SCC}(\widetilde{G})$ of strongly connected components in $\widetilde{G}$, and where (U, U') is an edge if and only if there is an edge from some $u \in U$ to some $u' \in U'$ in $\widetilde{G}$.

Take an arbitrary play α in G, and consider the corresponding path $\widetilde{\alpha}$ in $\widetilde{G}$ where counter and register values are stored up to $3N$. As there are no ∞-arenas in G, either from some point onward there are no further checks, or the counter is reset infinitely often. In either case, there are infinitely many positions i in $\widetilde{\alpha}$ such that $\widetilde{\alpha}[i] = (v, c, r)$ with $c \in [3N] \cup \{\top\}, r \in [3N]$. Accordingly, there exists a strongly connected component U in $\widetilde{G}$ such that $\widetilde{\alpha}$ contains infinitely many configurations that belong to U. This also means that once a configuration in U is reached, no configurations in other strongly connected components of $\widetilde{G}$ are met anymore.

In the following, given a strongly connected component U, we compute an optimal value for this component in such a way that any given play that meets U infinitely often has a payoff of at most this value, and prove that there is indeed a play with this value.

Values for strongly connected components Fix thus a strongly connected component U, and let $\widetilde{G}_U$ be the induced subgraph of $\widetilde{G}$. Let $\mathcal{I}$ be the set of simple cycles in G on which at least one increment occurs, but no reset. Let $\mathcal{T}$ be the set of possible pattern tails, that is, of finite paths on which every cycle appears at most once and which contain a check followed by a later reset and end after the subsequent check. Let $\mathcal{P} \subseteq \mathcal{I} \times \mathcal{T}$ be the set of patterns. Given a pattern $p \in \mathcal{P}$, we denote by $O(p) = (c_p, r_p) \in \mathbb{N}^2$ the counter and register values at the end of t, and write $\mathrm{val}p(r)$ for the value of the pattern using a fixed connector that reaches p with register value r. Note that $r < N$ can safely be assumed, as otherwise an ∞-arena would exist.

Let $\mathcal{P}_U \subseteq P \times [N]^2$ be the set of patterns along with counter and register values such that for every $(p, c, r) \in \mathcal{P}_U$ we have that $(\mathrm{first}(\gamma), c, r) \in \widetilde{V}_U$ and $(\mathrm{last}(t), O(p)) \in \widetilde{V}_U$. We remark that all positions $(v, \top, \top)$ and $(v, n, \top)$ appear also on some pattern tail in $\mathcal{P}_U$.

We construct a mean-payoff game $\widehat{G}_U$ as follows:

- The vertex set extends $\widetilde{V}_U$ by vertices $(p, c, r, 0), \ldots, (p, c, r, |t|-1)$, but positions $(v, \top, \top)$ and $(v, n, \top)$ for all n are removed.

- All edges from $\widetilde{E}_U \cap (\widehat{V}_U \times \widehat{V}_U)$ exist also in $\widehat{G}_U$. Furthermore, there are edges from (p, c, r, i) to $(p, c, r, i + 1)$ for all $(p, c, r) \in \mathcal{P}_U$ and $i < |t| - 1$, and there is a self-loop on $(p, c, r, 0)$. To connect the two kinds of vertices, there are edges from (v, n, m) to $(p, c, r, 0)$ if $((v, n, m), (\mathrm{first}(\gamma), c, r)) \in \widetilde{E}_U$, and from $(p, c, r, |t| - 1)$ to (v, n, m) if $((\mathrm{last}(t), O(p)), (v, n, m)) \in \widetilde{E}_U$.

- Concerning the weights, outgoing edges of vertices (v, n, m) receive weight m, that is, the respective register value at position v. The self-loop on vertices $(p, c, r, 0)$ receives weight $\mathrm{val}p(r)$, while the edges (p, c, r, i) to $(p, c, r, i + 1)$ up to the position of the first check receive weight r. The edge following the check receives the weight $c + Q_t$, where Q_t is obtained from Θ_t by subtracting the weights of the previous r-edges and the weights of the subsequent edges to $(p, c, r, |t| - 1)$, which receive weight c. Informally, Q_t is the sum over the effects of the increments along t. The outgoing edge of $(p, c, r, |t| - 1)$ receives weight $O(p)_2$, the register value according to $O(p)$.

Lemma 4.3.8. Let α be a play in G that induces infinitely many configurations that belong to the strongly connected component U. Then there exists a play $\widehat{\alpha}$ in $\widehat{G}_u$ such that $\mathrm{pay}(\alpha) \leq \mathrm{pay}(\widehat{\alpha})$.

Proof. To simplify arguments, we identify α with its extension in $(V \times \mathbb{N} \times \mathbb{N})^\omega$, where the second component stores the current counter value and the third component stores the register value. Without loss of generality, we can remove the prefix of α up the point where the first configuration in U is reached, an let the play start there with the respective counter and register values. This is possible as the mean counter condition on the extension is equivalent to a mean-payoff condition, and these are prefix-independent.

Let B be the set of indices of positions in α reached via a check, and let D be the positions reached via a reset. We define $\widehat{\alpha}$ in a blockwise manner, starting with index $i = 0$.

In case that $\alpha[i \dots]$ does not contain register values of at least $3N$, then all subsequent positions have equivalents in U. Thus,

$$\widehat{\alpha}[j] := \begin{cases} (v, \top, r), & \alpha[j] = (v, c, r), c \geq 3N \\ \alpha[j], & \text{otherwise,} \end{cases} \quad \text{for all } j \geq i.$$

If this is not the case, let $k := \min\{d \in D \mid d > i\}$ be the position of the next reset. If in $\alpha[i \dots k]$ all registers are below $3N$, the block is translated analogously to the above, and we set $i = k$. Otherwise, let $j' \in B$ be the last position in $\alpha[i \dots k]$ where a counter of less than N is checked, or set $j' = i$ if no such position exists. The first part then translates to $\widehat{\alpha}[i \dots j'] = \alpha[i \dots j']$. Note furthermore that $\alpha[k]$ necessarily lies on a pattern tail: To be more precise, there exist k' with $j' < k' < k$ and $m > k$ such that k' is the position of the first check after j' and m is the position of the first check after k, and $\alpha[k' \dots m]$ is the suffix of at least one pattern tail. Now, $\alpha[j' \dots k']$ contains at least one increment cycle, as at j', a value of less than N is checked, and at k' a value of at least $3N$. Furthermore, all register values up to position k' equal the register value at j', which is less than N. (This follows from the assumption that there are no ∞-arenas.)

For every pattern p such that $\alpha[k' \dots m]$ is a suffix of t and which is reachable from $\alpha[j']$ without check or reset in U, and every cycle-free path π from $\alpha[j']$ to $\mathrm{first}(p)$ without check or reset (again in U), construct the path $P(p, \pi) = \pi(p, c_\pi, r, 0)^n (p, c_\pi, r, 1) \dots (p, c_\pi, r, |t| - 1)$ where n is such that $|P(p, \pi)| = |\alpha[j' \dots m]|$, c_π is the counter value after π when starting with the actual counter value from $\alpha[j']$, and r is the register value at $\alpha[j']$. Now choose such a pair (p, π) with maximal value $\mathrm{val}p(R)$ where $c_\pi + \frac{nI_\gamma}{|\gamma|} + I' \geq \widehat{c}$ for the counter value $\widehat{c}$ at position $\alpha[k']$ and the number I' of increments along t until the position corresponding to $\alpha[k']$ is reached. (The existence

of such a pair is proved below.) Then,

$$\widehat{\alpha}[j] := \begin{cases} \pi[j-j'], & j-j' < |\pi| \\ (p, c_\pi, r, 0), & |\pi| \leq j-j' \leq |\pi|+n, \text{ for all } j' < j \leq m. \\ (p, c_\pi, r, j-j'-|\pi|-n), & j-j' > |\pi|+n, \end{cases}$$

Intuitively, $\widehat{\alpha}$ takes a short path to a pattern component that matches the tail, stays in the cycle until the tail begins, and takes the tail. Afterwards, set $i = m$, noting that $(p, c_\pi, r, |t|-1)$ has the same successors as the respective vertex in $V \times [3N]^2$, which corresponds to the respective vertex in α.

Remark 4.3.9. Let ϑ be a path without (r) and (c) on which at least $N+1$ increments occur. Then there exists a simple cycle γ with only (l) and (n) and two acyclic paths π, π', which do not contain (r) and (c), such that $\pi\gamma\pi'$ is a path with $\mathrm{first}(\pi) = \mathrm{first}(\vartheta)$, $\mathrm{last}(\pi') = \mathrm{last}(\vartheta)$ and

$$I_\pi + I_{\pi'} + \left(|\vartheta| - (|\pi| + |\pi'|)\right) \cdot \frac{I_\gamma}{|\gamma|} \geq I_\vartheta,$$

where I denotes the number of increments in the respective paths. Intuitively, γ is a simple cycle with a better increment-length-ratio.

Proof. We first construct a set $\mathcal{C}$ of simple cycles in ϑ by the following procedure: Starting with $\mathcal{C} = \varnothing$, take a simple cycle in ϑ, add it to $\mathcal{C}$ and remove it from ϑ, until no more simple cycles can be found.

For each $\gamma \in \mathcal{C}$, compute $\nu(\gamma) := \frac{I_\gamma}{|\gamma|}$, and let γ^* be such that $\nu(\gamma^*)$ is maximal. By construction, γ^* can be found in ϑ (possibly after removing some other cycles). Let ϑ_1 be the prefix of ϑ leading to γ^*, and let ϑ_2 be the suffix after γ^*. Now, remove all cycles from ϑ_1 and ϑ_2 as in the above procedure, to obtain π_1 and π_2, respectively. Thus, it follows that $\pi_1\gamma^*\pi_2$ is a path with the same first and last vertex as ϑ. To see that the requirement on the increments hold, we rewrite the number of increments in ϑ:

$$\begin{aligned} I_\vartheta &= I_\pi + I_{\pi'} + I_{\gamma^*} + \sum_{\gamma \in \mathcal{C}, \gamma \neq \gamma^*} I_\gamma \\ &\leq I_\pi + I_{\pi'} + |\gamma^*| \cdot \nu(\gamma^*) + \sum_{\gamma \in \mathcal{C}, \gamma \neq \gamma^*} |\gamma| \nu(\gamma^*) \\ &\leq I_\pi + I_{\pi'} + \left(|\vartheta| - (|\pi| + |\pi'|)\right) \cdot \nu(\gamma^*). \end{aligned}$$

□

It remains to prove that the payoff of $\widehat{\alpha}$ is indeed at least as large as the payoff of α. As the payoff in mean-payoff games is built along the edges, the weights in $\widehat{\alpha}$ are seen upon leaving a vertex, while in α they are seen at the vertex. Let

w_i denote the sequence of weights in $\widehat{\alpha}$, while r_i is the sequence of register values in α. We prove the following inequality:

$$\sum_{i=0}^{j} r_i \leq \sum_{i=0}^{j} w_i \qquad \text{for all } j.$$

We prove the equality inductively, along the blocks $\widehat{\alpha}$ was defined with. As $r_0 = w_0$, the claim holds for $j = 0$. Assume it holds up to $j-1$. Let $k > j$ be the position of the next reset in α. If $\alpha[j \dots k]$ does not see register values of at least $3N$ or if there is no such k, then $r_i = w_i$ for $j \leq i \leq k$, respectively $i \geq j$.

Otherwise, let j' be as in the translation such that in $\alpha[j' \dots k]$ no value of less than N is checked. Thus, $r_i = w_i$ for $j \leq i \leq j'$. Similarly, along the part corresponding to π in $\widehat{\alpha}$, $r_i = w_i$. Following this, for the number n of cycle iterations in the pattern component, $r_i < w_i$ by a constant $lI_\gamma/|\gamma|$. After this, until position k', we again have $r_i = w_i$, so it follows that $\sum_{i=0}^{k'-1} r_i + nlI_\gamma/|\gamma| \leq \sum_{i=0}^{k'-1} w_i$, and analogously for all $j < k' - 1$. At position k', the register in α contains $\widehat{c}$, while in $\widehat{\alpha}$ the weight is $c_\pi + Q_t$, and afterwards c_π. By construction, $\sum_{i=k'}^{m} r_i \leq (m - k' + 1)c_\pi + Q_t + nlI_\gamma/|\gamma|$. As the sum of the w_i is already larger by Q_t and $nlI_\gamma/|\gamma|$, and c_π is seen from k' to m, the inequality holds up to m. □

Corollary 4.3.10. For every strongly connected component U of $\widetilde{G}$, let Plays_U be the set of plays in G which remain in U from some point onwards.

$$\mathrm{val}\widehat{G}_U \geq \sup_{\alpha \in \mathrm{Plays}_U} \mathrm{pay}(\alpha).$$

From strongly connected components to G So far, we constructed mean-payoff games for every strongly connected component of $\widetilde{G}$ in such a way that for every play in G there exists a strongly connected component where the mean-payoff game has a larger value. It follows that $\mathrm{val}G$ is bounded by the maximal value of the $\mathrm{val}\widehat{G}_U$. While we can translate plays from G to plays in a respective $\widehat{G}_U$ without decreasing the payoff, the converse does not hold in general. However, when restricting ourselves to positional strategies, a translation from plays in $\widehat{G}_U$ to G is possible.

Lemma 4.3.11. Let U be a strongly connected component of $\widetilde{G}$ reachable from the initial position $(v, 0, 0)$. Let σ be a positional strategy in $\widehat{G}_U$. Then there exists a strategy $\widehat{\sigma}$ in G such that $\mathrm{val}(\sigma) = \mathrm{val}(\widehat{\sigma})$.

Proof. First of all, $\widehat{\sigma}$ lifts a shortest path in $\widetilde{G}$ from $(v, 0, 0)$ to the initial position of $\widehat{G}_U$ to G, which is possible by assumption. In the following, we describe $\widehat{\sigma}$ as if it started there.

Let α be the unique play consistent with σ. We distinguish several cases. If α never enters a pattern component, then all register values are below $3N$. By copying the moves according to σ, we obtain a strategy $\widehat{\sigma}$ such that the unique consistent play precisely corresponds to α. Note that this requires memory of size at most $(3(N+1))^2 + 1$.

If pattern components are entered infinitely often, then the cycles in the first vertices of the components are never taken (σ is positional). Hence, by playing the same moves as in the tail and the remainder of α, we obtain a strategy $\widehat{\sigma}$ with the same payoff: As σ is positional, the play is eventually periodic, and so is the respective sequence of weights. Accordingly, the sequence of register values according to $\widehat{\sigma}$ is also periodic. For parts of α in $V \times [3N]^2$, the sequences coincide. For weights along tails of pattern components, this is not true: There, the weight after the position corresponding to the first check is larger than the respective register, and the other way for the remainder of the tail. However, by construction we have that the sum of the weights equals the sum of the registers, as the larger weight exceeds the respective register exactly by the difference of the sums of the remainder. As the play is eventually periodic, the payoffs are identical. Once more, $\widehat{G}_U$ can serve as a memory structure for $\widehat{\sigma}$.

In the last case, α ends in the cycle of some pattern component. But in this case, $\mathrm{pay}(\alpha) = \mathrm{val}p(r)$ (for some given p and r). Hence, the optimal pattern strategy for p with register r is optimal in G. Note that this is an infinite memory strategy, but it can be represented by $\widehat{G}_U$ enriched with a counter to determine the number of cycle repetitions. □

As mean-payoff games are positionally determined, the existence of optimal strategies in solitary Maximizer one-counter games follows: Choose U such that $\widehat{G}_U$ has maximal value. Take an optimal positional strategy σ for $\widehat{G}_U$, and obtain $\widehat{\sigma}$. As the maximal value of $\widehat{G}_U$ is at least the value of G, and the values of the strategies coincide, $\mathrm{val}\widehat{\sigma} = \mathrm{val}G$.

Corollary 4.3.12. Maximizer has optimal strategies in solitary one-counter games. Furthermore, $\mathrm{val}G(v)$ can be computed effectively.

4.4 Two-Player One-Counter Games

After having discussed the single player cases of one-counter MCGs above, we now turn to mean counter games with one-counter where both players have nontrivial moves.

4.4.1 Bounds for the Value

Using an approach similar to the one for solitary Maximizer games, we prove that it is decidable whether or not the value of a game from a given starting position is finite. Therefore, we first give a criterion to determine a set of positions from which Maximizer can achieve arbitrarily high payoffs, and argue later that these are indeed all positions for which the value is ∞.

4.4.1.1 Positions with an Unbounded Value

As we are considering two-player games now, ∞-arenas do not suffice anymore: Minimizer could simply try to leave the arena. To overcome this problem, we construct automata to determine, via simple games, positions which correspond to starting positions of ∞-arenas.

To simplify reasoning about these automata and games, we first construct the arena $\check{G}$ on the incidence graph of the arena $G = (V, V_0, V_1, E, \lambda)$: This arena $\check{G} = (V \cup E, V_0, (V \cup E) \setminus V_0, E', \Omega)$ has as vertex set the set V enriched by an additional vertex for every edge $e \in E$. The partitioning is adapted such that all new edge vertices belong to Minimizer. The new edge relation E' is lifted in such a way that $(v, e) \in E'$ if and only if $e = (v, w)$ for some w, and $(e, w) \in E'$ if and only if $e = (v, w)$ for some v. The former edge labeling is now used to label the edge vertices, while the remaining vertices receive some dummy symbol as label. Note that edge vertices have unique incoming and outgoing edges, thus paths (and also strategies) translate directly between G and $\check{G}$. As above, let $A = \{\textcircled{\iota}, \textcircled{c}, \textcircled{r}, \textcircled{n}\}$ be the set of edge labels.

We first determine the set $S \subseteq \check{V}$ of positions from where Maximizer can ensure that the current register value is not decreased in the future, assuming that the counter value is currently not smaller than the register value. Therefore, let $\mathcal{A}_S = (Q_S, A, \delta_S, q_S, F)$ be the following automaton: $Q_S = \{q_S, q_{\textcircled{r}}, q_f\}$, $F = \{q_S, q_{\textcircled{r}}\}$, and

$$\delta(q_S, a) = \begin{cases} q_S, & a \neq \textcircled{r} \\ q_{\textcircled{r}}, & a = \textcircled{r} \end{cases}, \quad \delta(q_{\textcircled{r}}, a) = \begin{cases} q_{\textcircled{r}}, & a \neq \textcircled{c} \\ q_f, & a = \textcircled{c} \end{cases}, \quad \delta(q_f, a) = q_f.$$

Thus, $\mathcal{A}_S$ is an automaton with a safety condition that states that whenever a reset occurs, no check may occur later. Taking the product of $\check{G}$ and $\mathcal{A}_S$ with the winning condition from $\mathcal{A}_S$, the set S is the set of positions from which Maximizer (i.e., Pl. 0) wins, which is obviously computable.

Let $S_{\textcircled{c}} \subseteq S \cap E$ be the positions e in S for which $\Omega(e) = \lambda(e) = \textcircled{c}$. As a next step, we are interested in the positions L from which Maximizer can enforce the play to reach $S_{\textcircled{c}}$ without a previous reset. These can be computed using a simple attractor construction:

Let Y be a set of vertices, let $\sigma \in \{0,1\}$, and let Λ be a set of labels (of either vertices or edges). The (Λ,σ)-attractor $\mathrm{Attr}^{\Lambda}_{\sigma}(Y)$ of Y is the set of positions from where $Pl.\ \sigma$ has a positional strategy to reach a position in Y seeing only labels from Λ along the way.

Note that attractors can be computed using a least fixed-point induction, or an automaton with a reachability condition. L is now precisely the set $L = \mathrm{Attr}^{A\setminus\{\text{(r)}\}}_{\text{Maximizer}}(S_{\text{(c)}})$, where we ignore the dummy labels on V-vertices.

At last, we define a game with a Büchi winning condition to determine the positions from where Maximizer can guarantee that the counter is increased arbitrarily high, then checked and no smaller value is ever checked again. We do so once more by constructing an appropriate automaton, and taking the product with the arena afterwards. Let $\mathcal{A}^L_I = (Q^L_I, V \cup E, \delta^L_I, q_{\neg\text{(}\iota\text{)}}, F)$ with $Q^L_I = \{q_{\neg\text{(}\iota\text{)}}, q_{\text{(}\iota\text{)}}, q_f\}$, $F = \{q_{\text{(}\iota\text{)}}\}$,

$$\delta^L_I(q_{\neg\text{(}\iota\text{)}}, a) = \delta^L_I(q_{\text{(}\iota\text{)}}, a) = \begin{cases} q_f, & a \notin L \\ q_f, & a \in L, \Omega(a) = \text{(r)} \\ q_{\text{(}\iota\text{)}}, & a \in L, \Omega(a) = \text{(}\iota\text{)} \\ q_{\neg\text{(}\iota\text{)}}, & \text{otherwise} \end{cases}$$

and $\delta^L_I(q_f, a) = q_f$. Let W' be the winning region of Maximizer in the game on the product of $\mathcal{A}^L_I$ and $\check{G}$, where Maximizer takes the role of Pl. 0. Let now $W = \mathrm{Attr}_{\text{Maximizer}}(W')$ be the unrestricted attractor (i.e., ignoring intermediate labels) of W'.

Lemma 4.4.1. For every $v \in W \cap V$ it holds that $\mathrm{val}G(v) = \infty$.

Proof. For every $v \in W \cap V$, Maximizer has a (positional) attractor strategy to reach W'. From every $w \in W' \cap V$, Maximizer wins the respective Büchi game, that is, he has a strategy to remain within L that sees increment edges infinitely often, but no reset. As the play remains in L, Maximizer can choose—at any time—to reach a position with a previous check and no intermediate reset which is furthermore contained in S (as L is the attractor of $S_{\text{(c)}}$ without reset, and edge positions in $\check{G}$ have unique successors). From this position, however, he can ensure that either no future reset occurs, or after the reset no more checks are played. As at the position after the check, counter and register coincide, this means that the register is never again smaller than the value at this respective position.

In summary, for every given $n \in \mathbb{N}$, Maximizer can play a strategy σ_n that reaches W', follows the Büchi strategy until at least n increments have occurred, then switches to the attractor strategy for $S_{\text{(c)}}$, and chooses the

safety strategy afterwards. It follows that

$$\mathrm{val}G(v) \geq \sup_{n\in\mathbb{N}} \mathrm{val}_{\sigma_n} G(v) \geq \sup_{n\in\mathbb{N}} n = \infty.$$

□

In other words, at positions in W, for every $n \in \mathbb{N}$ Maximizer has a strategy to increase the counter to at least n and keep this number in the register (or possibly larger values). As a next step, we use this to show that the value from positions not in W is in fact bounded.

4.4.1.2 Positions with a Bounded Value

Assume that a position is not in W'. Thus, there exists some $n^* \in \mathbb{N}$ such that for all $n \geq n^*$, Maximizer has no strategy to keep values of at least n in the register from some point onward. As the games are determined, Minimizer has a strategy to ensure that whenever a value larger than n^* is checked, a value of less than n^* is checked later. Clearly, as the only way to check a smaller value is via a reset after the previous check, and enforcing a check after such a reset is an attractor condition, Minimizer has a strategy to assure that a smaller value is checked after at most $2N := 2|V|$ steps, and that this value is less than N. In total, there exists some n^* such that Minimizer can play in such way that whenever a value of at least n^* is checked, after at most $2N$ steps the register value is again below N. We prove next a lemma that abstractly states that all plays satisfying this condition yield a bounded payoff.

Lemma 4.4.2. Let $n^*, N \in \mathbb{N}$ with $n^* \geq N$ be two numbers. There exists a constant $B_{n^*} \in \mathbb{N}$ such that for all sequences $(a_i)_{i\in\mathbb{N}}$ of natural numbers that satisfy the conditions 1.-3.,

1. $a_0 = 0$.
2. If $a_i < a_j$ where $i < j$ and $a_{i-1} < a_i$ as well as $a_k = a_i$ for all $i < k < j$, then $a_j \leq a_i + (j - i)$.
3. If $a_i \geq n^*$ and $a_{i-1} < n^*$, then there exists a $j \leq 2N$ such that $a_{i+j} < N$.

it holds that $\liminf_{n\to\infty} \frac{\sum_{i=0}^{n} a_i}{n+1} \leq B_{n^*}$.

Proof. If only finitely many indices i exist with $a_i \geq n^*$, then from some point onward, the sequence is always below n^*, thus the limit average is also at most n^*. Hence we assume that the sequence reaches n^* infinitely often.

Let $(s_i)_{i\in\mathbb{N}}, s_i < s_{i+1}$ be the sequence of indices s such that $a_s \geq n^*$ and $a_{s-1} < n^*$, which is an infinite sequence by 3. and the above. Let $f\colon \mathbb{N} \to \mathbb{N}$

be a function that maps $s \in (s_i)$ to the minimal $t > s$ such that $a_t < n^*$ (thus, $f(s) \leq s + 2N$), and is arbitrary otherwise.

Obviously,

$$\sum_{j=f(s_i)}^{s_{i+1}-1} a_j \leq n^* \cdot ((s_{i+1}) - f(s_i))$$

for all i, and $\sum_{j=0}^{s_0-1} a_j \leq n^* s_0$. Furthermore, we have for all i

$$\begin{aligned}\sum_{j=s_i}^{f(s_i)-1} a_j &\leq (f(s_i) - s_i) \cdot (N + (s_i - f(s_{i-1}))) + \frac{(f(s_i) - s_i)^2}{2} \\ &\leq 2N^2 + 2N(s_i - f(s_{i-1})) + \frac{4N^2}{2} \\ &\leq 4(n^*)^2 + 2n^*(s_i - f(s_{i-1})),\end{aligned}$$

as $n^* \geq N$, $a_{f(s_{i-1})} < N$ and the growth of the sequence respects 2.

Consider now the subsequence from position $f(s_i)$ to position $f(s_{i+1})$. It follows that

$$\begin{aligned}\sum_{i=f(s_i)}^{f(s_{i+1})} a_i &\leq 4(n^*)^2 + 3n^*(s_{i+1} - f(s_i)) \\ &\leq (4(n^*)^2 + 3n^*)(f(s_{i+1}) - f(s_i)).\end{aligned}$$

Accordingly, for all $i \in \mathbb{N}$,

$$\sum_{j=0}^{f(s_i)} a_j \leq f(s_i) \cdot \left(4(n^*)^2 + 3n^*\right).$$

Clearly, $\liminf_{n\to\infty} \frac{\sum_{i=0}^{n} a_i}{n+1} \leq 4(n^*)^2 + 3n^* =: B_{n^*}$. □

Lemma 4.4.3. Let $v \in V \setminus W$. Then $\mathrm{val}G(v) < \infty$.

Proof. As $v \notin W$, there exists an n^* such that Minimizer can ensure that whenever a value of at least n^* is checked, a value of at most N is checked in the subsequent $2N$ steps. Clearly, the sequence of register values of any such play meets the requirements of Lemma 4.4.2, hence the value is bounded. □

Corollary 4.4.4. $\mathrm{val}G(v) = \infty$ if and only if $v \in W$. Furthermore, this can be decided effectively.

In the above result, no information about the size of the possible bound is given. Although we conjecture that possible bounds on the value are rather small, we have not proved this so far. In fact, it seems plausible that

a bound linear in the size of the arena suffices, as whenever counters grow larger, vertices are repeated. It might be that Minimizer could then adapt his strategies for large values to play against smaller values (which are still larger than the size of the arena). However, further insight into the shape of strategies of both Maximizer and Minimizer seems required to prove or disprove the use of such adaptions.

4.5 Affine Mean Counter Games

Thus far, we proved that the boundedness problem for the value can be decided for games with only one counter, and we argued that the exact value can be computed if there is only one player. When it comes to games with more than one counter, the arguments given above do not apply directly. For example, when trying to achieve a low register value after the check of a high one, it does not suffice anymore to play for reset and check on this counter, as it may be impossible, or it may be that in this way, another large counter would take over control of the register immediately. Nevertheless, we conjecture that several of the results presented above could be generalized to the multi-counter case.

Before we conclude the present discussion of mean counter games, we take up the idea of counter updates from the previous chapter. In counter parity games, we allowed arbitrary affine functions as counter updates, and originally, our aim was to do the same for mean counter games.

Definition 4.5.1. An *affine mean counter game* (aMCG) $\mathcal{G} = (G, \mathrm{pay})$ with k counters is played on a finite arena $G = (V, V_0, V_1, E, \lambda)$ without terminal vertices, where

$$\lambda \colon E \to \{c \mapsto Ac + b \mid A \in \mathbb{N}^{k\times k}, b \in \mathbb{N}^k\}$$

labels edges with affine functions over vectors of dimension k.

For an infinite path $\alpha = v_0 v_2 \ldots$, an infinite sequence of counter vectors $\mathfrak{c} = c^0 c^1 \cdots \in (\mathbb{N}^k)^\omega$ is induced, with $c^0 = 0^k$ and $c^{i+1} = \lambda(v_i, v_{i+1})(c^i)$.

In contrast to the mean counter games introduced above, there is no special register, but the payoff depends on the average of the first counter:

$$\mathrm{pay}(\alpha) = \liminf_{n\to\infty} \frac{\sum_{i=0}^{n} c_0^i}{n+1}.$$

While we did not prove decidability results for affine mean counter games that go beyond the results of the previous chapters, we observed that in aMCGs, the sizes of optimal finite-memory strategies for Minimizer (provided

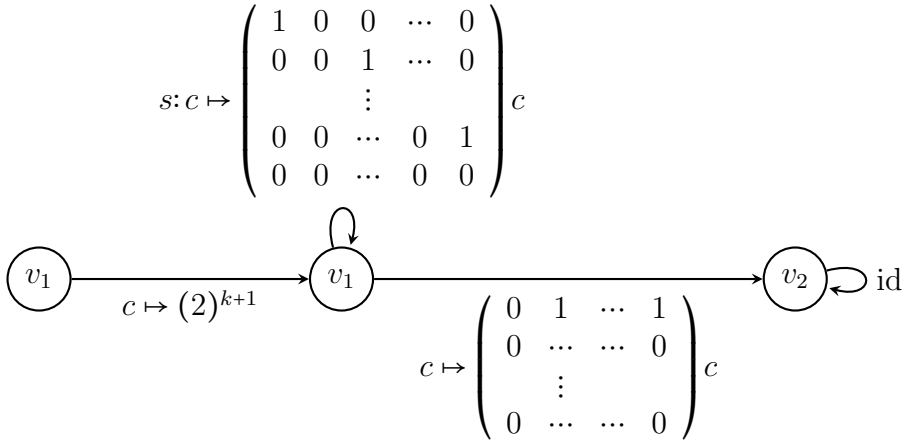


Figure 4.5: An aMCG where the memory depends on the number of counters

they exist) are somewhat independent from both the size of the arena and the number of counters. In fact, we give two examples below: in the first example the size of memory required to play optimally depends only on the number of counters, but not on the size of the arena, while in the second example the converse is true.

Example 4.5.2. Fix some number $k \in \mathbb{N}$. Let $s\colon \mathbb{N}^{k+1} \to \mathbb{N}^{k+1}, c \mapsto Ac$ be the affine function with $A_{i,j} = 1$ if $j = i + 1$ and $i > 0$, $A_{0,0} = 1$, and $A_{i,j} = 0$ otherwise. Let G be the mean counter game with $k + 1$ counters given in Figure 4.5, where all vertices belong to Minimizer. If Minimizer stays in v_1, then the payoff is 2, as c_0 does not change. If Minimizer takes the self-loop in v_1 $j < k$ times and then moves to v_2 the payoff is $2(k - j)$, as the counters 1 to k are shifted, filling with zeros. If Minimizer stays in v_1 for at least k many steps and then moves to v_2, the payoff is 0, which is optimal. Thus, an optimal winning strategy requires a memory of size k, that is, of the number of counters (minus one). Furthermore, the size of memory required does not depend on the size of the arena. ⊣

Example 4.5.3. As the next example, we construct a game that simulates binary counting, and punishes wrong counting. To model binary counting with n bits, we use $k = 2n+1$ counters. Counter c_0 will store the accumulating punishment, while counters $c_1, \cdots, c_n$ will store the current counter value in binary representation, counter c_1 corresponding to the lowest bit. Counters $c_{n+1}, \cdots, c_{2n}$, for technical reasons, store the complement of counters $c_1, \cdots, c_n$, that is, $c_i + n = \overline{c_i}$. The game is played solely by Minimizer on the arena depicted in Figure 4.6, where edges from v_i to v are labeled with the identity function, and the functions e_j are defined as follows:

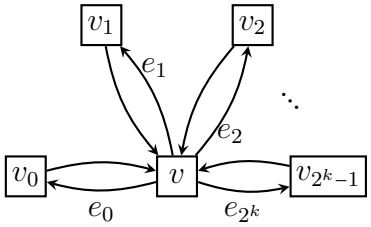


Figure 4.6: An aMCG where the required memory depends on the arena

The function e_j sets counters $c_1, \cdots, c_n$ to the binary representation of j, and $c_{n+1}, \cdots, c_{2n}$ to the according one's complement. Counter c_0 is set to its old value plus the punishment factor p. This number p is obtained by taking the sum of all old counters values for these counters which should be 0 in the binary representation of $j-1$ (and the respective complement's 0-counters). In other words, if in the binary representation of $j-1$ there is a 0 at position i, then c_i adds to the punishment, and otherwise c_{k+i} does.

Minimizer can now achieve a payoff of 0, but in order to do so, he has to take the edges in the right order, that is, if the current counters represent $j-1$, he has to choose the edge to vertex v_j (and cycle from 2^k-1 back to 0). This, of course, requires memory exponential in k, but linear in the size of the arena. ⊣

4.6 Summary

We introduced mean counter games, which are games where the payoff of infinite plays depends on the counters and is modeled as a mean-payoff objective. We showed that Maximizer may not have optimal strategies in these games, and even if he does, they may require infinite memory. For Minimizer, we conjecture that optimal strategies exist. In any case, optimal strategies for Minimizer may require memory, which contrasts the case of classical mean-payoff games.

We continued by studying one-counter variants of the games. For both solitary Minimizer and Maximizer games, we proved that boundedness can be decided by checking simple properties of the graph, and that the actual value can be computed using a reduction to mean-payoff games. In the Minimizer case, this is done by showing that register values occurring in an optimal play need not be arbitrarily large. In the Maximizer case, we introduced patterns to simulate the infinite memory that is potentially required. For the two-player case, we only gave a decision procedure to check for boundedness via automata constructions.

At last, we considered an extension of mean counter games where we allow affine counter update functions as in the case of counter parity games. We gave two examples in which Minimizer requires memory to play optimally, where in one example the size of the memory depends on the number of counters, while in the other it depends on the size of the arena.

Chapter 5

A Counting Logic for Structure Transition Systems

So far, we discussed games where quantitative aspects were modeled via counting. Now, we turn to another area, namely logic, and investigate counting mechanisms there. In particular, we consider two different quantitative logics which have in common that quantitative evaluations are based on counting atoms. The first one is closely related to the game model introduced in Chapter 3, while in the second one, discussed in Chapter 6, the game aspect is more hidden in automata-theoretical proofs. Furthermore, for the latter logic we focus on the specific structures of the natural numbers with order and the infinite binary tree, while we prove that the model-checking problem for the former logic is decidable on a rich class of systems.

Towards this class of systems, we begin by introducing structure transition systems, which form a special class of Kripke structures, and explain which general kinds of logics are suitable for these systems. We then present the aforementioned counting logic, which is a variant of Qμ that allows to count the size of MSO-definable relations, and prove—via a reduction to counter parity games—that the model-checking problem for this logic is decidable on a subclass of structure transition systems based on and generalizing pushdown systems.

The idea to investigate a counting μ-calculus for structure transition systems was proposed to me by Łukasz Kaiser. All results were obtained together, and were published in [63], on which this chapter is based.

5.1 Structure Transition Systems

The central motivation for structure transition systems comes from the area of verification. When modeling computation, it is common to differentiate

between the temporal changes of a system—for example transitions from one control state to another—and the changes in the data. To make this change in data explicit, and to not completely abstract from the data components, we consider transition systems whose states are labeled with finite relational structures (which could, for example, be relational databases). Upon a transition, the change in the data is thus the change from one relational structure to another.

Definition 5.1.1. A *structure transition system* (STS) is a labeled Kripke structure $\mathcal{S} = (S, (T_a)_{a\in\Sigma}, m)$ where $(S, (T_a)_{a\in\Sigma})$ is a directed graph and $m\colon S \to \mathrm{FinStr}(\zeta)$ assigns a finite relational ζ-structure to every state.

Note that the definition is not restrictive regarding possible connections between the relational structures. In fact, the label of a successor state is completely independent from the label of the current state. However, in most instances of structure transition systems (or of systems which can be modeled as such), there is a more restricted evolution of the data component.

Examples of STSs Before we introduce the format of logics for STSs, let us review some important models of computation and how they can be viewed as STSs.

Consider a Turing machine M. Such a machine has a control component, that is, the control states Q, and moves its read/write head along an infinite tape. It is straightforward to view the current tape content as a word, thus a relational structure, with an additional monadic predicate to indicate the current position of the head. The corresponding STS is now the control graph of the Turing machine, where every vertex is labeled with the word structure representing the current tape content. Note that deterministic and nondeterministic Turing machines can both be modeled this way.

To model Petri nets, one can use relational structures of the size matching the number of tokens currently in the net, and predicates P_t that partition the universe and indicate whether a token is in place t. Accordingly, in the structure transition system there is a state for every configuration of the Petri net, and moving along a transition corresponds to firing the respective transition in the net.

The example which we consider in the main part of this chapter is that of pushdown systems. Generally, a pushdown system can be viewed as a finite automaton that has access to a stack, which in turn can be viewed as a finite word. Accordingly, when representing pushdown systems as STSs, one can start with the configuration graph and add finite words as labels to the configurations.

In the above examples, the first two only have one transition relation, while when modeling pushdown systems, different transition relations can be used for the different input symbols.

Note that so far, the additional benefit of the relational structures has been left unclear, as they seem to represent abundant information. The actual purpose of the state labeling for the above systems becomes apparent when introducing the logical format using which we want to reason about STSs. Before we do so, we consider an example where the relational structures play a major role: It is well known that databases can be viewed as relational structures. Using this, a database system can be modeled in such a way that vertices represent states of the system, using the relational structures to represent the current database. The edges are used to reflect the updates of the system, that is, deleting tuples from relations, inserting new ones, or even adding new tables. More generally, one might also use structure transition systems to model systems that evolve using logical transformations, for example relational machines.

5.1.1 Logics for STSs

As we specifically separated temporal from data aspects, it is only natural to reflect this separation in the format of the logics considered. Thus, logics for structure transition systems are twofold, following the approach from [61]: There is a logic $\mathcal{L}_2$ to speak about the relational ζ-structures. Typical candidates in the qualitative setting are FO, MSO and LFP. Then, there is a modal logic $\mathcal{L}_1$ which is used to express temporal properties, and in which formulas of $\mathcal{L}_2$ appear as atoms. Natural candidates here are LTL, CTL, Lμ, or, in the simplest setting, just the single non-atomic operator Reach which states that a position in which the respective atom holds can be reached. Given two such logics, we write $\mathcal{L}_1[\mathcal{L}_2]$ for the composition of the two (where, again, formulas of $\mathcal{L}_2$ appear as $\mathcal{L}_1$-atoms).

Assuming that the semantics for $\mathcal{L}_1$ on Kripke structures and $\mathcal{L}_2$ on ζ-structures are given, the semantics for $\mathcal{L}_1[\mathcal{L}_2]$ is obtained in the straightforward way: When the current subformula to be checked at state s is from $\mathcal{L}_2$, the formula is checked on $m(s)$ using the semantics of $\mathcal{L}_2$. If it is a formula of $\mathcal{L}_1$, one uses the semantics of $\mathcal{L}_1$ on the Kripke structure $(S, (T_a)_{a\in\Sigma})$.

Examples The standard temporal logics LTL, CTL and Lμ are obtained when using only predicates for $\mathcal{L}_2$, together with true and false. In this case, the relational structures consist only of 0-ary relations. LTL, CTL and Lμ are decidable on pushdown systems [69], while LTL is decidable on Petri nets, but Lμ is not [41].

$\mathrm{Reach}[\mathrm{FO}]$ is able to express reachability of states whose structures belong to first-order axiomatizable classes of structures, while $\mathrm{Reach}[\mathrm{MSO}]$ is the same but for MSO-axiomatizable classes, and is decidable on STSs derived from pushdown systems [69] and Petri nets [75].

The qualitative variant of the logic we introduce below is $\mathrm{L}\mu[\mathrm{MSO}]$ (see [61]), where we use the modal μ-calculus to reason about temporal aspects over MSO-definable queries on the relational structures. It follows from [69] that $\mathrm{L}\mu[\mathrm{MSO}]$ is decidable on pushdown systems.

5.2 A Counting Logic for STSs

As mentioned above, we introduce a counting logic whose qualitative analogue is $\mathrm{L}\mu[\mathrm{MSO}]$. In fact, we replace the modal by the quantitative μ-calculus as introduced in Section 2.4.3. Then, instead of using MSO-sentences, we count the size of definable relations via counting terms.

Definition 5.2.1. An *MSO counting term* is of the form $\#_{\overline{x}}\varphi(\overline{x})$, where $\overline{x} = (x_0, \ldots, x_{n-1})$ is a vector of variables and $\varphi(\overline{x})$ is a formula of MSO with the free variables $\mathrm{free}(\varphi) = \overline{x}$. (Note that φ does not contain free second-order variables.) We denote by $\#\mathrm{MSO}$ the set of all counting terms (for a given, but usually omitted, signature ζ).
Given a finite structure $\mathfrak{A}$, a counting term evaluates to the number of satisfying tuples:

$$[\![\#_{(x_0,\ldots,x_{n-1})}\varphi]\!]^{\mathfrak{A}} := |\{(a_0, \ldots, a_{n-1}) \in A^n \mid \mathfrak{A} \vDash \varphi(a_0, \ldots, a_{n-1})\}|.$$

Note that for an MSO sentence φ, we define $[\![\#\varphi]\!]^{\mathfrak{A}}$ to be 1 if $\mathfrak{A} \vDash \varphi$, and 0 otherwise. Although this may appear surprising at first, it is useful later, as it permits to write $\#\varphi$ instead of just φ for guards of other formulas. That is, we avoid the otherwise necessary removal of counting quantifiers in the construction, which makes the proofs more readable.

The counting logic we propose is thus $\mathrm{Q}\mu[\#\mathrm{MSO}]$. Despite its matching the general definition above, we give the syntax again for better readability. Formulas of $\mathrm{Q}\mu[\#\mathrm{MSO}]$ are built according to the following grammar:

$$\psi ::= \#_{\overline{x}}\varphi \mid X \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \Box_a\psi \mid \Diamond_a\psi \mid \mu X.\psi \mid \nu X.\psi,$$

where $\#_{\overline{x}}\varphi \in \#\mathrm{MSO}$ is a counting term. As before for $\mathrm{L}\mu$, we require that for every subformula $\mu X.\vartheta$ or $\nu X.\vartheta$, X appears only positively within ϑ.

When evaluated on an STS, counting terms are evaluated on the relational labelings of states, while the $\mathrm{Q}\mu$-parts of a formula are evaluated on the Kripke structure itself. Recall that $\neg$ translates to the product with -1, while $\vee$ corresponds to $\max$, $\Diamond_a$ to the supremum over a-successors and $\wedge$ and $\Box_a$ accordingly to $\min$ and $\inf$.

Remark 5.2.2. $\mathrm{L}\mu[\mathrm{MSO}] \subseteq \mathrm{Q}\mu[\#\mathrm{MSO}]$, that is, the qualitative combination of $\mathrm{L}\mu$ with MSO can be embedded in the quantitative version with counting terms.

Proof. Note that the semantics of $\mathrm{Q}\mu$ corresponds to that of $\mathrm{L}\mu$ if the values of the quantitative atoms are restricted. Using the convention that true corresponds to ∞ or 1, while false corresponds to $-\infty$ or 0, a formula $\varphi \in \mathrm{L}\mu[\mathrm{MSO}]$ in negation normal form can be translated to $\mathrm{Q}\mu[\#\mathrm{MSO}]$ by replacing every positive MSO formula (which can only be a sentence) φ by $\#\varphi$, and every $\neg\varphi$ by $\#\neg\varphi$. Accordingly, the thus translated formula evaluates to ∞ or 1 using $\mathrm{Q}\mu[\#\mathrm{MSO}]$ semantics if and only if φ evaluates to true. □

5.2.1 Model-Checking Games

It was already mentioned in the preliminaries that there is a close connection between $\mathrm{Q}\mu$ and quantitative parity games. As we make use of this connection in the decidability proof of our counting logic, we recall the definition of model-checking games for $\mathrm{Q}\mu$ here, adapted to $\mathrm{Q}\mu[\#\mathrm{MSO}]$.

Therefore note that from now on we assume that formulas of $\mathrm{Q}\mu[\#\mathrm{MSO}]$ are in *negation normal form* concerning the $\mathrm{Q}\mu$ part: Negation is thus allowed only to occur at counting quantifiers ($\neg\#_{\overline{x}}\varphi$). This is not a proper restriction, as the dual of every operator is part of the syntax. Furthermore, we assume that every fixed point variable is bound only once.

Definition 5.2.3. Let $\psi \in \mathrm{Q}\mu[\#\mathrm{MSO}]$ and let $\mathcal{S}$ be a structure transition system such that the #MSO subformulas of ψ fit the signatures of the relational structures in $\mathcal{S}$. Then $\mathrm{MC}(\psi, \mathcal{S}) = (V, V_0, V_1, E, \Omega, \tau)$ is a quantitative parity game with the following components:

- $V = (S \times \mathrm{Sub}(\psi))$, with $\mathrm{Sub}(\psi)$ denoting the set of $\mathrm{Q}\mu[\#\mathrm{MSO}]$ subformulas of ψ (i.e., using #MSO formulas as atoms).
- Positions of the form $(s, \#_{\overline{x}}\varphi)$ or $(s, \neg\#_{\overline{x}}\varphi)$ are the only terminals, and $\tau(s, \#_{\overline{x}}\varphi) = [\![\#_{\overline{x}}\varphi]\!]^{m(s)}$, while $\tau(s, \neg\#_{\overline{x}}\varphi) = -[\![\#_{\overline{x}}\varphi]\!]^{m(s)}$.
- Positions where the outermost operator in the formula component is a disjunction or a $\Diamond$ belong to Pl. 0, while all other positions are Pl. 1's.
- The edges are as follows:

$$\begin{aligned} E :=& \{((s, \varphi \vartriangle \vartheta), (s, \varphi)), ((s, \varphi \vartriangle \vartheta), (s, \vartheta)) \mid \vartriangle \in \{\vee, \wedge\}\} \\ &\cup \{((s, \vartriangle_a \varphi), (s', \varphi)) \mid (s, s') \in T_a, \vartriangle \in \{\Diamond, \Box\}\} \\ &\cup \{((s, \xi X.\varphi), (s, \varphi)) \mid \xi \in \{\mu, \nu\}\} \\ &\cup \{((s, X), (s, \xi X.\varphi)) \mid \xi \in \{\mu, \nu\}, \xi X.\varphi \text{ binds } X\}. \end{aligned}$$

- Ω depends only on the formula component, and all but positions (s, X) for fixed point variables receive unimportant priorities. The priorities at positions (s, X) are even if X is bound by a ν and odd in the case of μ. Furthermore, the fixed point variable quantified outmost receives the lowest priority, and a variable that is quantified within the scope of another variable receives a higher priority.

In the game, Pl. 0 as usual tries to maximize the payoff, while Pl. 1 tries to minimize it. This corresponds to Verifier and Falsifier in the classical model checking game for Lμ.

For more details, and a proof of correctness of the following theorem, we refer to [45]. Note that we omitted the discounts in the version of the model checking game given above, as they are not needed for our purposes.

Theorem 5.2.4 ([45])**.** Let $\varphi \in \mathrm{Q}\mu[\#\mathrm{MSO}]$ and let $\mathcal{S}$ be a structure transition system. Then $\mathrm{val}(\mathrm{MC}(\varphi, \mathcal{S}), (s, \varphi)) = [\![\varphi]\!]^{\mathcal{S}}(s)$.

5.2.2 Examples

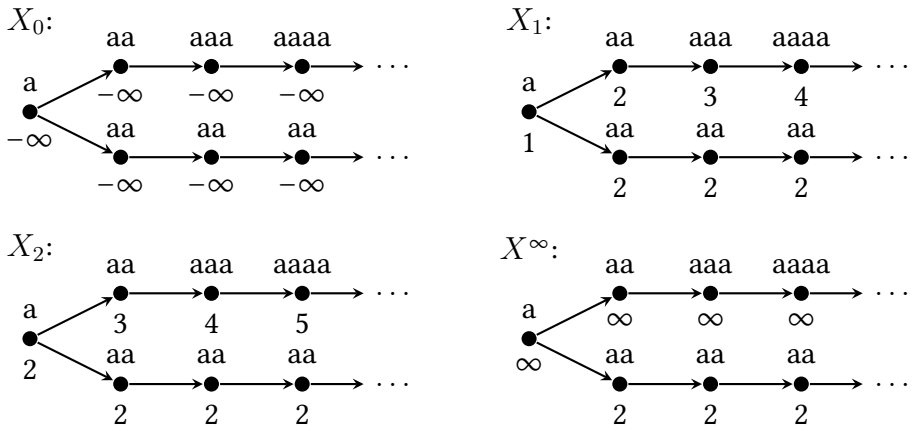
Before we introduce the class of pushdown systems we show that $\mathrm{Q}\mu[\#\mathrm{MSO}]$ is decidable on, let us consider some examples of formulas to understand the expressiveness of the logic.

A classical example of a formula of Lμ is the one expressing reachability. In the quantitative setting, when replacing the monadic target predicate by a counting term, the formula computes the maximum/supremum over the values the counting term reaches on paths: Consider the formula $\mu X.\#_x(Ax) \vee \Diamond X$. When evaluated on a structure transition system whose states are labeled with words (for example a pushdown graph), the formula computes the supremum over the number of A occurring in the words. That is, the formula evaluates to ∞ at a given state s if for every $n \in \mathbb{N}$ there is a state s' reachable from s such that $m(s)$ contains at least n positions labeled with A. For an illustration of the computation of the fixed point, see Figure 5.1.

Conversely, the formula $\nu X.\#_x(Ax) \wedge \Box X$ computes the minimal number of A-occurrences reachable from a state. To see that larger growth rates are also possible, note that if $\#_x(Ax)$ is replaced by $\#_{x,y}(Ax \wedge Ay)$, then instead of the number of occurrences of A, this number squared is considered.

5.3 Tree-Producing Pushdown Systems

So far we only considered structure transition systems in general, introduced a logic for these systems, and gave some examples of both classes of systems



Relational word structures are given above the vertices, while the value of the current evaluation function is given below.

Figure 5.1: An example of a fixed point computation for $\mu X.\#_x(Ax) \vee \Diamond X$

that can be modeled as STSs and formulas. We now introduce a new class of systems: This class consists of pushdown systems whose stack symbols are tree-rewriting rules. The goal is to evaluate the counting logic introduced above on the corresponding configuration graphs in such a way that counting terms are evaluated on the tree that is obtained by the successive application of the tree-rewriting rules currently on the stack.

For these systems, we show that the model-checking games of $\mathrm{Q}\mu[\#\mathrm{MSO}]$ can be reduced to pushdown quantitative parity games without pop-operations, and we further reduce these to counter parity games (as discussed in Chapter 3).

5.3.1 Increasing Tree-Rewriting Rules

The first ingredient needed to define tree-producing pushdown systems are tree-rewriting rules. We consider only rules that are increasing (that is, trees can only grow, but never shrink on applications of rules), that rewrite using trees of a bounded height, and which are applied universally.

Definition 5.3.1. Let Σ be a tree alphabet. An *increasing tree-rewriting rule* (ITRR) r is of the form $r = a \leftarrow t$, where $a \in \Sigma$ and t is a tree of height 1 or 2.

We remark that we view a single vertex as a tree of height 1, and that height 2 means that there are two levels of vertices.

Given a tree t' and an ITRR $r = a \leftarrow t$, the application $r(t')$ of r on t' is the tree obtained from t' by replacing *every* a-labeled leaf of t' with t.

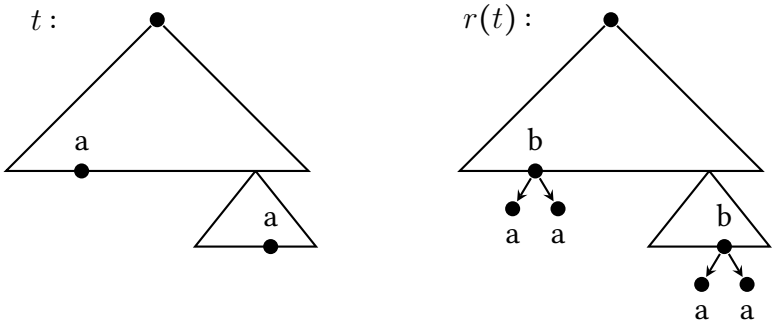


Figure 5.2: An example of an application of an ITRR

Example 5.3.2. As an example, let $r = a \leftarrow b(a, a)$ and let t be the tree on the left of Figure 5.2. ⊣

Note that for a given alphabet Σ, there are only finitely many increasing tree-rewriting rules, and we denote the set of all these rules by $\mathcal{R}_\Sigma$, or simply $\mathcal{R}$ if Σ is clear from the context.

In general, we are interested in sequences of ITRR. Therefore, given an initial tree t and a sequence $r_1 \dots r_n$ of ITRR, we denote by $tr_1 \dots r_n$ the tree $r_n(r_{n-1}(\cdots r_1(t)\cdots))$. In [61], it was shown that for a given formula $\varphi \in$ MSO, the subset $L_t \subseteq \mathcal{R}^*$ of sequences of rules w such that $tw \vDash \varphi$ is a regular language over $\mathcal{R}$. A similar result is shown later in this chapter for MSO counting terms: in fact, we can associate an affine function with every increasing tree-rewriting rule in such a way that the value of the counting term is obtained by the composition of the affine functions, together with an initialization function and an output function (see Section 5.4.2).

5.3.2 Pushdown Systems for Tree-Rewriting

Using the tree-rewriting rules defined above, we can now give the definition of tree-producing pushdown systems. As we are not interested in these systems as word acceptors, input symbols are neglected and we view the systems purely as stack manipulators, without further notions of input or acceptance.

Definition 5.3.3. A *tree-producing pushdown system* (TPPDS) $\mathfrak{T} = (Q, E)$ is a directed graph with labeled edges $E \subseteq Q \times (\mathcal{R} \cup \{\bot\}) \times (\mathcal{R} \cup \{\mathrm{pop}, \epsilon\}) \times Q$.

Note that an edge is only enabled if the current top stack symbol matches the first component of the edge labeling, where $\bot$ denotes the empty stack. In general, provided an initial tree t, a TPPDS $\mathfrak{T}$ induces a structure transition system $\mathcal{S}(\mathfrak{T}) = (S, T, m)$ obtained from the corresponding configuration

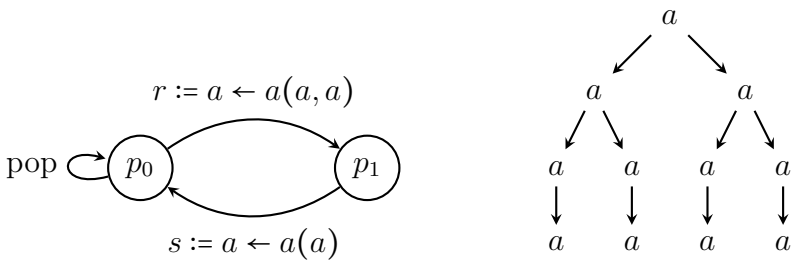


Figure 5.3: An example of a TPPDS and a tree resulting from a run

graph of $\mathfrak{T}$: The states are $S = Q \times \mathcal{R}^*$, $m(q, r_1 \dots r_n) = tr_1 \dots r_n$, and the transitions are as follows:

$$\begin{aligned} T = &\{((q, \epsilon), r', (q', r')) \mid (q, \bot, r', q') \in E\} \\ &\cup \{((q, r_1 \dots r_n), r', (q', r_1 \dots r_n r')) \mid (q, r_n, r', q') \in E\} \\ &\cup \{((q, r_1 \dots r_n r), \mathrm{pop}, (q', r_1 \dots r_n)) \mid (q, r, \mathrm{pop}, q') \in E\} \\ &\cup \{((q, r_1 \dots r_n), \epsilon, (q', r_1 \dots r_n)) \mid (q, r_n, \epsilon, q') \in E\}. \end{aligned}$$

Example 5.3.4. Consider the TPPDS on the left in Figure 5.3. Let $t = a$ be a tree with one vertex, labeled with a. Note that we abuse notation and use the same symbol for rank 1 and 2, and furthermore, we assume that the pop-transition can only be applied if the stack is nonempty. Below, we consider an example run and the respective stack contents.

run	stack
p_0	$\bot$
p_0p_1	r
$p_0p_1p_0$	rs
$p_0p_1p_0p_0$	r
$p_0p_1p_0p_0p_1$	rr
$p_0p_1p_0p_0p_1p_0$	rrs

On the right of Figure 5.3 is a drawing of $trrs$. ⊣

Simulating pushdown systems To see that TPPDSs generalize classical pushdown systems, we now give a construction to obtain, for a pushdown system $\mathfrak{P}$, a TPPDS $\mathfrak{T}_\mathfrak{P}$ such that the corresponding structure transition systems are isomorphic. Therefore, recall that a classical pushdown system (again, without input or acceptance) with stack alphabet Γ can be represented as a directed graph $\mathfrak{P} = (P, \Delta)$, where P is the set of states, and $\Delta \subseteq P \times (\Gamma \cup \{\bot\}) \times (\Gamma \cup \{\mathrm{pop}, \epsilon\}) \times P$, and the configuration graph, viewed as an STS,

is obtained analogously as for TPPDSs, but for m: $m(p, \gamma_1 \dots \gamma_n) = \gamma_1 \dots \gamma_n$. That is, the relational structures are finite words representing the stack contents.

Accordingly, to define a corresponding TPPDS, the rules have to be such that the rules produce the word of the current stack content. As the tree alphabet for the tree-rewriting rules, we thus take $\Gamma \uplus \{e\}$, where e represents the empty word, or analogously the bottom stack symbol, and we copy the state set. The translations of the edges from $\mathfrak{P}$ to $\mathfrak{T}_{\mathfrak{P}}$ are as follows, where the idea is to simulate a letter a by an ITRR $a' \leftarrow a'(a)$ for the respective previous letter a':

$$\begin{aligned}
(p, *, \epsilon, p') \in \Delta &\rightsquigarrow (p, *, \epsilon, p') \in E \\
(p, a, b, p') \in \Delta &\rightsquigarrow (p, * \leftarrow *(a), a \leftarrow a(b), p') \in E \\
(p, a, \mathrm{pop}, p') \in \Delta &\rightsquigarrow (p, * \leftarrow *(a), \mathrm{pop}, p') \in E \\
(p, \bot, a, p') \in \Delta &\rightsquigarrow (p, \bot, e \leftarrow e(a), p') \in E.
\end{aligned}$$

It follows that for a given position $(p, \gamma_1 \dots \gamma_n)$ in $\mathcal{S}(\mathfrak{P})$, the corresponding position in $\mathcal{S}(\mathfrak{T}_{\mathfrak{P}})$ is labeled with $e\gamma_1 \dots \gamma_n$. Thus, a decidability result for TPPDSs entails the respective result for classical pushdown systems.

To see that TPPDSs properly generalize classical pushdown systems in the sense that they induce a larger class of STSs, it suffices to note that the growth rate of the relational structures in the structure transition system can be much larger: in classical pushdown systems, the size can only increase by 1 in each step, while in the case of tree-rewriting, every leaf can be replaced by a (properly branching) tree of height 2.

5.3.3 Pushdown Quantitative Parity Games

Before we consider the model-checking problem for $\mathrm{Q}\mu[\#\mathrm{MSO}]$ on tree-producing pushdown systems, we discuss pushdown quantitative parity games in general.

Definition 5.3.5. Let $\mathcal{G} = (V, V_0, V_1, E, \Omega, \tau)$ be a quantitative parity game. We say that $\mathcal{G}$ is a *pushdown parity game*, if there exists a pushdown system $\mathfrak{P} = (P, \Delta)$ with stack alphabet Γ such that $V = P \times \Gamma^*$, the edges in E are according to Δ, and the partition (V_0, V_1) of V as well as the coloring Ω depend only on the P-entry of a vertex v, but not on the stack content.

In other words, positions in a pushdown parity game can be seen as configurations of a pushdown system, and the edges manipulate the stack accordingly. On this configuration graph, an additional partition of the vertices and a vertex coloring is given, as well as a function assigning values to terminals.

$$
\begin{aligned}
\Delta' :=& \{((p,m,C,l),a,\epsilon,(p',\min(m,\Omega(p')),C,l)) \mid (p,a,\epsilon,p') \in \Delta\} \\
&\cup \{((p,m,C,l),a,\epsilon,(p,H)) \mid (p,a,\mathrm{pop},p') \in \Delta \text{ and} \\
&\qquad\qquad (l=0,\min(m,\Omega(p')) \in C(p')) \\
&\qquad\qquad \text{or } (l=1,\min(m,\Omega(p')) \notin C(p'))\} \\
&\cup \{((p,m,C,l),a,\epsilon,(p,L)) \mid (p,a,\mathrm{pop},p') \in \Delta \text{ and} \\
&\qquad\qquad (l=0,\min(m,\Omega(p')) \notin C(p')) \\
&\qquad\qquad \text{or } (l=1,\min(m,\Omega(p')) \in C(p'))\} \\
&\cup \{((p,m,C,l),a,\epsilon,(p',b,C',l',m,C,l)) \mid (p,a,b,p') \in \Delta, \\
&\qquad\qquad C' \in \mathcal{C}, l'=0 \Leftrightarrow p \in V_0^{\mathcal{G}}\} \\
&\cup \{((p',b,C',l',m,C,l),*,b,(p',\Omega(p'),C',l'))\} \\
&\cup \{((p',b,C',l',m,C,l),*,\epsilon,(p_c,m',C,l)) \mid p_c \in P, n \in C'(p_c) \neq \varnothing, \\
&\qquad\qquad m'=\min(m,n)\}
\end{aligned}
$$

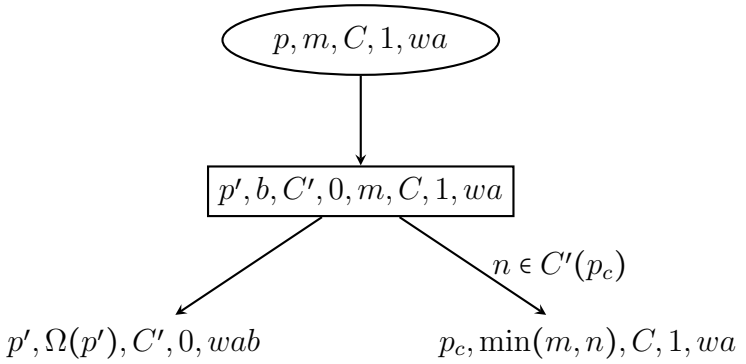
Figure 5.4: Transitions of $\mathfrak{P}'$

In the course of the decidability proof in the next section, we require another important property. We need the pushdown quantitative parity game to be pop-free: A pushdown parity game $\mathcal{G}$ is pop-*free*, if the underlying pushdown process does not contain pop-transitions. For the configuration graph, this means that the word representing the current stack content never decreases in length along an edge. We adapt the technique to achieve this for classical pushdown systems from [89, 90] and from [82] to show that for every pushdown quantitative parity game one can construct an equivalent one which is pop-free.

Theorem 5.3.6. Let $\mathcal{G}$ be a pushdown quantitative parity game and let $\mathfrak{P}$ be the associated pushdown system. One can compute a finite set P' and a pop-free pushdown quantitative parity game $\mathcal{G}'$ with associated pushdown system $\mathfrak{P}' = (P \times P', \Delta')$ such that for all $p \in P$ there exists a computable $p'_{0,p} \in P'$ with

$$\mathrm{val}_{\mathcal{G}}(p,\epsilon) = \mathrm{val}_{\mathcal{G}'}((p,p'_{0,p}),\epsilon).$$

We give the precise construction of the pushdown quantitative parity game $\mathcal{G}'$ and describe the idea. We then also argue why the values of the two games are identical, but omit, for sake of conciseness, a formal proof, which can be achieved in a similar way as in [82, Chapter 5].



p-positions belong to Pl. 0, $(p, a, b, p') \in \Delta$, $C' \in \mathcal{C}$ is chosen by Pl. 0.

Figure 5.5: Illustration of the claim construction

Proof. The idea for $\mathcal{G}'$ is that upon the occurrence of a push-move, the respective player who chose the move claims the possible states (and minimal colors until then) at the position where the symbol just pushed is popped again, if it is popped. The opponent can then choose: he can accept the claim, choosing one of the claimed states and the respective intermediate color, or he can refute the claim, which means that he loses immediately if the respective symbol is popped according to the claim.

Formally, a single *claim* is a function $C\colon P \to \mathcal{P}([d])$, where d is the number of colors, with the intended meaning that a state p is claimed to be reachable via the respective pop while seeing some color $c \in C(p)$ as the minimal color along the way. Let $\mathcal{C}$ be the set of all claims. The additional set P' is comprised of three sets, $P' := \{H, L\} \cup P_0 \cup P_1$. H and L indicate new sink states, while $P_0 := [d] \times \mathcal{C} \times \{0, 1\}$ is used for making claims, and $P_1 := \Gamma \times \mathcal{C} \times \{0, 1\} \times (\{\bot\} \cup P_0)$ represents positions where claims are answered.

By the above, the state set of $\mathfrak{P}'$ is completely determined. For the initial state, we set $p'_{0,p} = (\Omega(p), C_0\colon * \mapsto \varnothing, 0)$. As no pop actions can occur upon an empty stack, this claim is never challenged or taken, but simplifies the construction as rules for the empty stack need not be given separately.

The transitions of $\mathfrak{P}'$ are given in Figure 5.4. At first, ϵ-moves are copied. Then, as pop-moves are eliminated, upon a pop-move, a positive or negative sink state is reached, depending on whether the claim is true and who made it. As an example, if Pl. 0 made the claim, which corresponds to $l = 0$, and the minimal color until p' is indeed in $C(p')$, then there is a move to (p, H), which receives payoff ∞ in the game. For the claims, if at a given position b is pushed, then the respective player choosing this edge is allowed to provide a claim for every possible state with what colors this state is reachable on a

pop. The other player may then accept one of the claimed colors and states, or he may choose to refute the claim. In the latter case, the symbol is pushed and the claim is stored to be verified. If later proven correct or incorrect, states with L or H are reached. Note that upon acceptance of a claim, the old claim remains in place. For an illustration of the claims, see Figure 5.5.

It remains to give the components of the game $\mathcal{G}'$ which do not follow directly from $\mathfrak{P}'$: Positions (p, m, C, l, w) for stack contents w belong to the same player as positions with p in $\mathcal{G}$. Furthermore, Ω' is obtained from Ω by ignoring the P'-parts, thus focusing only on p. (Recall that the partitioning and the coloring do not depend on the actual stack content.) Positions $(p, a, C', l', m, C, l, w)$ belong to Pl. $1 - l'$, and the coloring is chosen so that it is irrelevant (i.e., large enough).

Terminal positions that have matching terminal positions in $\mathcal{G}$ receive the same τ-value. The new terminals (p, H) receive $\tau'(p, H) = \infty$, while for (p, L) we set $\tau'(p, L) = -\infty$.

As already mentioned, we omit a formal proof of correctness for the above construction. The main idea in the proof is to construct, given a strategy in the original game $\mathcal{G}$, the corresponding *truthful* strategy. In this strategy, which can be constructed inductively on the length of the current prefix, when a symbol is pushed, one considers all extensions according to the original strategy and collects the states and respective minimal colors such that there are consistent positions where the pushed symbol is popped. This then works as a claim. On the other hand, when confronted with a claim, one checks whether there is a consistent extension in $\mathcal{G}$ such that the claim holds true. If yes, it is accepted and the play moves to the respective pop-position. Otherwise, the claim is refuted, and one continues according to the original strategy and the translation described. □

5.4 Model Checking the Counting Logic on TPPDSs

In this section, we prove that the model-checking problem for our counting logic is decidable on tree-producing pushdown systems, or, to be more precise, on the structure transition systems induced by those. The proof consists of the following steps: First, we consider the (infinite) model-checking game on the induced structure transition system, and observe that it can be understood as a pushdown quantitative parity game. Using the above result on these games, it is possible to construct an equivalent pop-free pushdown quantitative parity game, and the respective corresponding pop-free pushdown system. Secondly, we prove that counting term evaluation can be done using a finite

number of counters and affine counter update functions: To compute the evaluation of a counting term $\#_{\overline{x}}\varphi$ on a tree $tr_1 \dots r_n$, we can start with a finite vector $\overline{c}$ of counters, and then update these counters along every rule r_i, so that after n counter updates, the evaluation of the term on $tr_1 \dots r_n$ can be read from the counters. At last, we use the pop-free pushdown process and the affine counter updates to construct a finite counter parity game with the same value as the original model-checking game. As counter parity games are solvable effectively, this proves the decidability of the model-checking problem.

The main theorem of this section is thus formulated as follows:

Theorem 5.4.1. Let $\psi \in \mathrm{Q}\mu[\#\mathrm{MSO}]$ be a formula, and let $\mathfrak{T} = (Q, E)$ be a tree-producing pushdown system. For every state $q \in Q$ and every one-vertex tree t, one can compute $[\![\psi]\!]^{\mathcal{S}(\mathfrak{T})}(q, \bot)$ with initial tree t.

5.4.1 Constructing a Finite Game Graph

The first step is to construct a finite game graph with the property that the value of the game is the same as the evaluation of the formula. To do so, we start with the model-checking game—which is infinite—and use that outgoing edges of positions do not depend on the history. Note however that the payoff function of the game thus constructed crucially depends on the history, so that solving the game directly does not look promising. Hence, we further simplify the game later, using the aforementioned iterative evaluation technique for counting terms.

In the following, fix a $\mathrm{Q}\mu[\#\mathrm{MSO}]$-formula ψ and a TPPDS $\mathfrak{T}$, as well as an initial state q and an initial tree t.

Constructing the finite game graph $\widehat{\mathrm{MC}}$ First of all, we assume that ψ is in negation normal form. Note that this applies only to the $\mathrm{Q}\mu$ part of the formula, as counting terms are treated as atoms. We remark that if the formula is not in negation normal form, it can easily be transformed using standard techniques, as duals of all logical operators exist in $\mathrm{Q}\mu$.

Now consider $\mathrm{MC} := \mathrm{MC}(\psi, \mathcal{S}(\mathfrak{T}))$. By Theorem 5.2.4, this is a quantitative parity game such that the value of the game from the initial position $(q, \bot, \psi)$ is the same as the evaluation of the formula. The positions in this game are of the form (q, w, φ), where $q \in Q$, w is the stack content and $\varphi \in \mathrm{Sub}(\psi)$. Thus, MC, when reordering the tuples to (q, φ, w), can be seen as a pushdown quantitative parity game. Accordingly, by Theorem 5.3.6, one can construct an equivalent pop-free pushdown quantitative parity game MC' with pushdown process (Q', Δ') and initial state q'. Now, as no tree-rewriting rules are removed from the stack once pushed, the changes to the

trees are permanent. Thus, when keeping the actual tree on the side, the stack content becomes irrelevant. Towards the finite game, we use this and ignore all stack symbols but the topmost one: We define $\widehat{\mathrm{MC}}$ to be the quantitative parity game with vertex set $\widehat{V} = (Q' \times \{\mathcal{R}, \bot\})$, that is, for positions (q, wr) in MC', we remove the stack content w and just store r. Note that all positions (q, wr) and $(q, w'r)$ in MC' have exactly the same outgoing edges. Thus, the edge relation of $\widehat{\mathrm{MC}}$ is

$$\begin{aligned}\widehat{E} :=& \{((q,r), r', (q', r')) \mid ((q, wr), r', (q', wrr')) \in E'\} \\ &\cup \{((q,r), \epsilon, (q', r)) \mid ((q, wr), \epsilon, (q', wr)) \in E'\}.\end{aligned}$$

Defining $\widehat{\Omega}$ is straightforward, as colors in pushdown quantitative parity games do not depend on the stack, but only on the control state, so Ω' can be lifted. For infinite plays, we use the quantitative parity condition. Given a finite play $\widehat{\pi}$ in $\widehat{\mathrm{MC}}$, the last position is either a counting term, or a negated counting term, or a sink position where the payoff is known to be either ∞ or $-\infty$. In the latter case, we use the respective payoff. In the former cases, the payoff depends on $\widehat{\pi}$: We consider the sequence $r_1 \dots r_n$ of ITRR in $\widehat{\pi}$, and evaluate the respective counting term from the control state on $tr_1 \dots r_n$.

Lemma 5.4.2. Let $\psi \in \mathrm{Q}\mu[\#\mathrm{MSO}]$ and let $\mathfrak{T}$ be a TPPDS. Let $q \in Q$ be an initial state, and t an initial tree. Let $\widehat{\mathrm{MC}}$ be defined as above. Then, for the respective initial state q', $\mathrm{val}_{\widehat{\mathrm{MC}}}(q', \bot) = [\![\psi]\!]^{\mathcal{S}(\mathfrak{T})}(q, \bot)$ with initial tree t.

Proof. The adaptions to MC' preserve the value of MC, thus, by the respective theorems, the value of MC' corresponds to the evaluation of the formula. As there is a one-to-one correspondence between plays in MC' and $\widehat{\mathrm{MC}}$, it suffices to compare the payoffs of corresponding plays: For infinite plays, the parity condition is the same, and so are the colors, so the payoff is identical. For finite plays, the same is true, as MC' is pop-free, and thus the changes to the tree are permanent, and the way the payoff is determined at terminals is the same as in $\widehat{\mathrm{MC}}$. Clearly, $\mathrm{val}_{\widehat{\mathrm{MC}}}(q', \bot) = \mathrm{val}_{\mathrm{MC}'}(q', \bot)$, which proves the lemma. □

The goal is further to simplify $\widehat{\mathrm{MC}}$ via the introduction of counters, so that only a finite list of counters needs to be maintained from which the payoff at terminals can be read. The construction of these counters and their update functions is the subject of the next section.

5.4.2 Decomposition and Iterative Evaluation of Terms

5.4.2.1 Decomposition for Counting Terms

The first step is to provide a suitable general decomposition theorem, adapting Theorem 2.4.4. This theorem states that given a tree and a formula from MSO,

one can compute a finite set of tuples of formulas such that the formula holds in the whole tree if and only if there exists a tuple such that the entries hold in the respective subtrees and the root.

Theorem 5.4.3. Let $t = Q(t_1, \dots, t_l)$ be a tree, and let $\#_{\overline{x}}\varphi(\overline{x})$ be a counting term. For every partition $[\overline{x}]_l$ of the free variables, one can compute a finite set $\Psi_{[\overline{x}]_l}$ of $(l+1)$-tuples of Hintikka formulas $\tau \in H_{\mathrm{qr}(\varphi),\leq|\overline{x}|} = \bigcup_{n\leq|\overline{x}|} H_{\mathrm{qr}(\varphi),n}$ such that the following holds:

$$[\![\#_{\overline{x}}\varphi]\!]^t = \sum_{[\overline{x}]_l \in P(\overline{x},l)} \sum_{(\tau_0,\dots,\tau_l)\in\Psi_{[\overline{x}]_l}} [\![\#_{\overline{x}_0}\tau_0]\!]^Q \cdot [\![\#_{\overline{x}_1}\tau_1]\!]^{t_1} \cdot \dots \cdot [\![\#_{\overline{x}_l}\tau_l]\!]^{t_l}.$$

The theorem follows from Theorem 2.4.4 and Lemma 2.4.3. Any possible assignment of the variables $\overline{x}$ that is to be counted matches exactly one of the partitions from $P(\overline{x}, l)$ in such a way that an element from a subtree is assigned to a variable if and only if the variable is in the corresponding class of the partition. Given a fixed assignment, φ holds in t if there is at least one tuple in the classical decomposition set such that the respective formulas with the respective restricted assignments hold in the subtrees. Thus, the tuples from the classical decomposition set are split into sets for the respective partitions of the variables. To ensure that no assignment is counted twice (e.g., because not one but two tuples are satisfied with it), the original tuples are turned into tuples of Hintikka formulas. We remark that a single tuple of formulas may be turned into several tuples of Hintikka formulas. Now, Lemma 2.4.3 ensures that for any given assignment, at most one of the tuples is satisfied. As satisfiability in the respective subtrees is independent, the numbers of valid assignments in the subtrees need to be multiplied. Also note that for τ_0, there is either no or exactly one valid assignment, as the tree containing only the former root has exactly one element.

5.4.2.2 Counters for Iterative Evaluation

The setting we consider here is such that an initial tree $t = P$ is given, on which a sequence of rules $r_1, \dots, r_n$ is applied, and in the end, a counting term $\#_{\overline{x}}\varphi$ is to be evaluated on $tr_1 \dots r_n$. We prove that one can choose a finite list of counters, initialized by a function I, and that these counters can be updated in such a way that when r_1 is applied to t, the counters are updated, then again on the application of r_2, and so on. In the end, there is an output function E which combines the counters to $[\![\#_{\overline{x}}\varphi]\!]^{tr_1\dots r_n}$. This will be made more precise along the way. Note that we omit rules of the form $P \leftarrow Q$, as they can be eliminated from the TPPDS using standard techniques.

Before we begin with the definition of the counters and the initialization, we fix some notation. Fix thus an initial tree $t = P$ and a sequence of rules

$r_1 \dots r_n \in \mathcal{R}^*$. As a reminder, we write $tr_1 \dots r_n$ for $r_n(\cdots(r_1(t))\cdots)$. We now also use the notation $r_1 \cdots r_k t$ to denote $r_k(\cdots(r_1(t))\cdots)$, especially in contexts like $r_1 \dots r_k t r_{k+1} \dots r_n$ to indicate that the first k rules have already been applied. Given a Hintikka formula $\tau \in H_{\mathrm{qr}(\varphi),\leq|\overline{x}|}$, a symbol Q and an index $i \in \{1, \dots, n\}$, we write $[\![\tau(\overline{y}), Q]\!]^i$ for the number of tuples $\overline{a}$ that satisfy τ in the final subtree $Qr_{i+1} \dots r_n$ of leaves labeled with Q in $r_1 \dots r_i t$. Formally, $[\![\tau, Q]\!]^i := [\![\#_{\overline{y}}\tau]\!]^{Qr_{i+1}\dots r_n}$.

As a further prerequisite, we assume that MSO types of future subtrees $Qr_{i+1} \dots r_n$ for all i, Q are known, or can be computed. (Of course, when later applied to $\widehat{\mathrm{MC}}$, these are unknown in advance, but will be guessed.) Denote by Λ the set of unordered sequences $(\overline{\tau, Q}) = \{(\tau_1(\overline{y_1}), Q_1), \dots, (\tau_k(\overline{y_k}), Q_k)\}$, where Q_i is a symbol, and $(\overline{y_1}, \dots, \overline{y_k})$ is a partition into nonempty sets $\overline{y}_i$ of a subset $\overline{y} \subseteq \overline{x}$ of the free variables, and each $\tau_i \in H_{\mathrm{qr}(\varphi),\leq|\overline{x}|}$ has free variables $\overline{y_i}$ and is a Hintikka formula. This set Λ functions as the index set of the counters and is finite as the alphabet, thus the ranks occurring in the rewriting rules, and the set of Hintikka formulas are finite. Accordingly, the vector c of current counter values is an element $c \in \mathbb{N}^\Lambda$, and we address the entries by $c[(\overline{\tau, Q})]$.

Counters for single applications of rules The first step towards the iterative evaluation of counting terms is to prove that there exists a counter vector that can be used as weights for the evaluations of Hintikka formulas at leaves after the first application of a rule in order to obtain an evaluation for the overall counting term.

Lemma 5.4.4. There exists computable a vector $c \in \mathbb{N}^\Lambda$ such that

$$[\![\#_{\overline{x}}\varphi]\!]^{tr_1\dots r_n} = \sum_{(\overline{\tau,Q})\in\Lambda} c[(\overline{\tau, Q})] \cdot \prod_{(\tau,Q)\in(\overline{\tau,Q})} [\![\tau, Q]\!]^1.$$

Proof. Without loss of generality, let $r_1 = P \leftarrow P'(P_1, \dots, P_l)$. By applying Theorem 5.4.3, we obtain that $[\![\#_{\overline{x}}\varphi]\!]^{tr_1\dots r_n}$ equals

$$\sum_{[\overline{x}]_l \in P(\overline{x},l)} \sum_{\overline{\tau}\in\Psi_{[\overline{x}]_l}} [\![\#_{\overline{x}_0}\tau_0]\!]^{P'} \cdot [\![\#_{\overline{x}_1}\tau_1]\!]^{P_1 r_2\dots r_n} \cdot \dots \cdot [\![\#_{\overline{x}_l}\tau_l]\!]^{P_l r_2\dots r_n}.$$

As by the definition above $[\![\#_{\overline{x}_i}\tau_i]\!]^{P_i r_2\dots r_n} = [\![\tau_i, P_i]\!]^1$, the inner product can be written as $[\![\#_{\overline{x}_0}\tau_0]\!]^{P'} \cdot \prod_{i=1}^{l} [\![\tau_i, P_i]\!]^1$. Now some of the τ_i in an inner product, $i \geq 1$, may be sentences, and thus evaluate either to 0 or to 1. As we assumed that types of the respective future subtrees are known, it is known which evaluate to 1. These can simply be omitted from the product. If there is one which evaluates to 0, then the whole product can be removed from the sum. Consider now the sum after the removal of all such factors which evaluate

to 1 and all such products which evaluate to 0 because of the evaluation of sentences. It may now be the case that there are τ_0 and τ_0' with $\tau_0 \neq \tau_0'$ such that the remainder of the respective products is the same (syntactically). But then, in the overall sum, these can be combined, that is:

$$\begin{aligned}&[\![\#_{\overline{x}_0}\tau_0]\!]^{P'} \cdot \prod_{i=1}^{l'} [\![\tau_i, P_i]\!]^1 + [\![\#_{\overline{x}_0}\tau_0']\!]^{P'} \cdot \prod_{i=1}^{l'} [\![\tau_i, P_i]\!]^1 \\ &\quad = \left([\![\#_{\overline{x}_0}\tau_0]\!]^{P'} + [\![\#_{\overline{x}_0}\tau_0']\!]^{P'}\right) \cdot \prod_{i=1}^{l'} [\![\tau_i, P_i]\!]^1.\end{aligned}$$

Adapting this idea, $c[(\overline{\tau, Q})]$ is set to the sum over all $[\![\#_{\overline{x}_0}\tau_0]\!]^{P'}$ where the remainder of the product corresponds to a product over $(\overline{\tau, Q})$. □

Again, observe that the values of c depend only on the rule r_1 and on which sentences are true in $Pr_2 \dots r_n$ for all P, which can be read from the respective types. After explaining how a counter evaluation can be updated from one step to the next, we explain how the types can be guessed, so that the sequence of rules does not need to be known in advance.

Updating counters In the above lemma, we showed that one can obtain a counter evaluation which allows to simulate the first application of a rule. We now adapt this and prove that, given a counter evaluation which is correct after the first $i-1$ steps, one can compute, using linear functions, a correct evaluation for after i applications of rules, again assuming that types of future subtrees are known.

To make the lemma more readable, we write $\Upsilon := [\![\#_{\overline{x}}\varphi]\!]^{Pr_1 \dots r_n}$ for the overall evaluation of the term, that is, for the target value.

Lemma 5.4.5. Assume that $c \in \mathbb{N}^\Lambda$ is such that

$$\Upsilon = \sum_{(\overline{\tau,Q}) \in \Lambda} c[(\overline{\tau, Q})] \cdot \prod_{(\tau,Q) \in (\overline{\tau,Q})} [\![\tau, Q]\!]^{i-1}.$$

There exists a vector $\widetilde{c} \in \mathbb{N}^\Lambda$ such that

$$\Upsilon = \sum_{(\overline{\tau,Q}) \in \Lambda} \widetilde{c}[(\overline{\tau, Q})] \cdot \prod_{(\tau,Q) \in (\overline{\tau,Q})} [\![\tau, Q]\!]^{i}$$

and, additionally, each $\widetilde{c}[(\overline{\tau, Q})]$ can be obtained effectively as an affine combination of elements from c.

Proof. We first prove that a vector $\widetilde{c}$ indeed exists, and then, in the second part of this proof, show how it can be obtained by linearly combining the entries of c.

Let thus $r_i = R \leftarrow P(P_1, \ldots, P_n)$. Using Lemma 5.4.4 for r_i instead of r_1, it follows that for every τ there is a c_τ such that

$$[\![\tau(\overline{y}), R]\!]^{i-1} = \sum_{(\overline{\tau,Q})\in\Lambda} c_\tau[(\overline{\tau,Q})] \cdot \prod_{(\tau,Q)\in(\overline{\tau,Q})} [\![\tau, Q]\!]^i. \tag{5.1}$$

For other symbols $R' \neq R$, it trivially follows that $[\![\tau, R']\!]^{i-1} = [\![\tau, R']\!]^i$, for every τ, as the respective leaves are left untouched by r_i.

By assumption, Υ equals the c-weighted sum for after $i-1$ steps. This sum can be split into two sums, depending on whether R occurs in a sequence $(\overline{\tau,Q})$ or not:

$$\Upsilon = \underbrace{\sum_{(\overline{\tau,Q})\in\Lambda, R\notin\overline{Q}} c[(\overline{\tau,Q})] \cdot \prod_{(\tau,Q)\in(\overline{\tau,Q})} [\![\tau, Q]\!]^{i-1}}_{S_1} + \underbrace{\sum_{(\overline{\tau,Q})\in\Lambda, R\in\overline{Q}} c[(\overline{\tau,Q})] \cdot \prod_{(\tau,Q)\in(\overline{\tau,Q})} [\![\tau, Q]\!]^{i-1}}_{S_2}.$$

As all factors in the inner product of S_1 are for symbols other than R, when replacing the evaluation $[\![\tau, Q]\!]^{i-1}$ by $[\![\tau, Q]\!]^i$, S_1 is left unchanged. In the sum S_2, some factors of inner products are of the form $[\![\tau, R]\!]^{i-1}$ for some τ. But for these, we know that they can be replaced by the right hand side of Equation 5.1. Thus, S_2 can be rewritten to

$$S_2 = \sum_{\substack{(\overline{\tau,Q})\in\Lambda,\\ R\in\overline{Q}}} c[(\overline{\tau,Q})] \cdot \left(\prod_{\substack{(\tau,Q)\in(\overline{\tau,Q}),\\ R\neq Q}} [\![\tau, Q]\!]^i \right) \cdot \left(\prod_{\substack{(\tau,Q)\in(\overline{\tau,Q})\\ R=Q}} \left(\underbrace{\sum_{(\overline{\tau,Q})'\in\Lambda} c_\tau[(\overline{\tau,Q})'] \cdot \prod_{(\tau,Q)\in(\overline{\tau,Q})'} [\![\tau, Q]\!]^i}_{=[\![\tau,R]\!]^{i-1}} \right) \right).$$

Consider now a fixed summand of the outer sum of the rewritten S_2, and consider the right hand side of Equation 5.1 for some $[\![\tau, R]\!]^{i-1}$, where $\overline{z}$ are the free variables of τ. It follows that $c_\tau[(\overline{\tau,Q})] = 0$ for all $(\overline{\tau,Q})$ where there exists an element which has free variables other than those of $\overline{z}$, hence after removing these, the only remaining summands are for sequences whose free variables are among $\overline{z}$. Furthermore, when considering factors $[\![\tau', Q]\!]^i$ for $Q \neq R$ in S_2—that is, in the first line of the above equation—the free variables of τ' are also different from $\overline{z}$.

Hence, S_2 can be multiplied out in order to get a sum over Λ of products weighted by some $\widetilde{c}$. Altogether, when combining this with S_1, we obtain

$$\Upsilon = \sum_{(\overline{\tau,Q})\in\Lambda} \widetilde{c}[(\overline{\tau,Q})] \cdot \prod_{(\tau,Q)\in(\overline{\tau,Q})} [\![\tau,Q]\!]^i$$

for some $\widetilde{c}$ of which we now show how to obtain it.

Computing $\widetilde{c}$ First of all, as we need to copy old counter values $c[(\overline{\tau,Q})]$ where R does not occur, we define a vector o to store these values, and 0 for those sequences where R occurs, as these are changed to other sequences upon multiplying out.

$$o[(\overline{\tau,Q})] := \begin{cases} c[(\overline{\tau,Q})], & R \notin \overline{Q} \\ 0, & \text{otherwise.} \end{cases}$$

To obtain the actual values of $\widetilde{c}$, we have to consider the sequences $(\overline{\tau,Q})$ obtained when replacing $[\![\tau,R]\!]^{i-1}$-factors. Therefore, for every $\tau \in H_{\mathrm{qr}(\varphi),\leq|\overline{x}|}$, we form a set D_τ of all sequences $(\overline{\tau,Q})$ along with the weights $c_\tau[(\overline{\tau,Q})]$ appearing on the right hand side of Equation 5.1 for $[\![\tau,R]\!]^{i-1}$:

$$D_\tau := \{((\overline{\tau,Q}), c_\tau[(\overline{\tau,Q})]) \mid (\overline{\tau,Q}) \text{ appears in Eq. 5.1 for } [\![\tau,R]\!]^{i-1}\}.$$

To make this more accessible, we combine all D_τ, and filter out all weights c_τ that are 0.

$$D := \{(\tau, (\overline{\tau,Q}), d) \mid \tau \in H_{\mathrm{qr}(\varphi),\leq|\overline{x}|}, ((\overline{\tau,Q}), d) \in D_\tau, d > 0\}.$$

Now, $\widetilde{c}[(\overline{\tau,Q})]$ may consist of its old value—if $R \notin \overline{Q}$—but also those sequences with R need to be considered where, upon multiplying out, a summand for $(\overline{\tau,Q})$ appears. This is reflected as follows: For a subsequence $(\overline{\tau,Q})'$ of $(\overline{\tau,Q})$ and every τ such that this subsequence appears on the right hand side of Equation 5.1 for $[\![\tau,R]\!]^{i-1}$, we add the values from c indexed by $(\overline{\tau,Q})$ when replacing $(\overline{\tau,Q})'$ by (τ,R), and weight these values by $c_\tau[(\overline{\tau,Q})']$. This procedure can be written as a sum over D, which leads to the following computation to obtain $\widetilde{c}[(\overline{\tau,Q})]$:

$$\widetilde{c}[(\overline{\tau,Q})] := o[(\overline{\tau,Q})] + \sum_{(\overline{\tau,Q})'\subseteq(\overline{\tau,Q})} \sum_{(\tau,(\overline{\tau,Q})',d)\in D} d \cdot c[(\overline{\tau,Q}) \setminus (\overline{\tau,Q})' \cup (\tau,R)].$$

We remark that the above is an affine transformation of c, provided that all vectors c_τ are known. These vectors c_τ are obtained by Lemma 5.4.4, using the assumption that types of future subtrees of leaves are given. □

5.4.2.3 Guessing Types of Subtrees

As mentioned, the above construction of the counters in Lemma 5.4.4 and Lemma 5.4.5 crucially depends on knowledge of the types of trees $Qr_i \dots r_n$ for all i and all symbols Q. We now introduce a way to *guess* these types, and thus to simultaneously construct the counters for all possible combinations of types. In the end, when a counting term is to be evaluated, the types which are correct then are known, as leaves are truly leaves. Therefore, we need a way to represent types, and we do so using Hintikka formulas: it is an easy observation that an m-type of a tree t can be represented by the unique sentence $\tau \in H_{m,0}$ such that $t \models \tau$.

Definition 5.4.6. Let Σ be the tree alphabet, and let m be a natural number. A *type function* $L\colon \Sigma \to H_{m,0}$ assigns an m-type to every symbol in Σ.

Under the assumption that Σ is finite, for every given m there are only finitely many type functions, as $H_{m,0}$ is finite. Let $\mathcal{L}$ be the set of these type functions L for $m = \operatorname{qr}(\varphi)$. The *final type function* $L_F \in \mathcal{F}$ is the type function that assigns to every $Q \in \Sigma$ the type of the one-vertex tree Q. This allows us to define compatible sequences of type functions: Let $r_1 \dots r_n$ be a sequence of ITRRs. A sequence $L_0, \dots, L_n$ of type functions is *compatible*, if $L_i(Q)$ is the type of $Qr_{i+1} \dots r_n$ for all Q (and hence $L_n = L_F$). We prove that one can uniformly compute compatible sequences of type functions in a backwards fashion:

Lemma 5.4.7. There exists a computable function $\operatorname{pre}\colon \mathcal{L} \times \mathcal{R} \to \mathcal{L}$ such that for every sequence $r_1 \dots r_n$, the sequence $L_0, \dots, L_n$ with $L_n = L_F$ and $L_{i-1} = \operatorname{pre}(L_i, r_i)$ is compatible.

Proof. First of all, we remark that types can be composed using standard methods: there is a function $\oplus$ such that, when given types $\tau_1, \dots, \tau_n$ of trees $t_1, \dots, t_n$ and a type τ_0 of a one-vertex graph Q, one can compute, using only the types, the type $\oplus(\tau_0, \tau_1, \dots, \tau_n)$ of $Q(t_1, \dots, t_n)$.

Using this, we define pre as follows:

$$\operatorname{pre}(L, r) := R \mapsto \begin{cases} L(R), & R \neq P \\ \oplus(L_F(Q), L(Q_1), \dots, L(Q_n)), & R = P, \end{cases}$$

where $r = P \leftarrow Q(Q_1, \dots, Q_n)$. (Note that $L_F(Q)$ is the type of the one-vertex tree Q.)

It remains to prove that a sequence $L_0, \dots, L_n$ satisfying the properties in the lemma for a given sequence $r_1 \dots r_n$ of ITRRs is compatible. This is done via a backwards induction. If $i = n$, then $L_n(Q) = L_F(Q)$ is the type of the tree Q. Assume thus that $L_i, \dots, L_n$ is compatible for

$r_{i+1} \dots r_n$. Let $r_i = P \leftarrow Q(Q_1, \dots Q_n)$, and $L_{i-1} = \mathrm{pre}(L_i, r_i)$. If $R \neq P$, then $Rr_i \dots r_n = Rr_{i+1} \dots r_n$, and also $L_i(R) = L_{i-1}(R)$ by definition. For P, $Pr_i \dots r_n = Q(Q_1 r_{i+1} \dots r_n, \dots, Q_n r_{i+1} \dots r_n)$. The induction hypothesis states that $L_i(Q)$ is the type of $Qr_{i+1} \dots r_n$. As by definition of pre, $L_{i-1}(P) = \oplus(L_F(Q), L_1(Q_1), \dots, L_n(Q_n))$, compatibility of L_{i-1} follows from the correctness of $\oplus$ and the induction hypothesis. □

Using this, we do not update c according to known types, but maintain a vector c_L for every type function L. This vector c_L will then be updated assuming that the types according to L are correct, and the types are updated according to pre, in the sense that $\widetilde{c_L}[(\overline{\tau, Q})]$ is updated using $c_{\mathrm{pre}(L,r)}$ instead of c. At the end, we choose the counter vector c_{L_F}.

5.4.2.4 Counters and Update Functions

We now combine the defined counters, the counter updates and the guessing of types of subtrees into overall affine counter update functions which allow to evaluate counting terms after a sequence of updates.

Theorem 5.4.8. Let $Q \in \Sigma$ be a symbol, and let $\#_{\overline{x}}\varphi$ be a counting term. One can compute a number $k \in \mathbb{N}$, an initial vector $I^k \in \mathbb{N}^k$, an evaluation matrix $E \in \mathbb{N}^{1 \times k}$ and, for every $r \in \mathcal{R}$, an affine counter update function $\mathrm{up}_r \colon \mathbb{N}^k \to \mathbb{N}^k$ such that for all $r_1 \dots r_n \in \mathcal{R}^*$,

$$[\![\#_{\overline{x}}\varphi]\!]^{Qr_1 \dots r_n} = E \cdot (\mathrm{up}_{r_n} \circ \dots \circ \mathrm{up}_{r_1})(I).$$

Note that E, k and the functions up_r depend only on $\mathrm{qr}(\varphi)$ and the free variables $\overline{x}$.

Proof. Let Λ and $\mathcal{L}$ be as above, and fix an enumeration $L_1, \dots, L_{|\mathcal{L}|}$ of $\mathcal{L}$. Now set $k := |\Lambda| \cdot |\mathcal{L}|$. The first step is to define the initial vector $I_L \in \mathbb{N}^\Lambda$ for every $L \in \mathcal{L}$. Therefore, we compute a set $T = \{\tau_1, \dots, \tau_s\} \subseteq H_{\mathrm{qr}(\varphi), |\overline{x}|}$ such that $\bigvee T = \tau_1 \vee \dots \vee \tau_s \equiv \varphi$. In I_L, for every $L \in \mathcal{L}$, all positions are set to 0 but positions $I_L[(\tau_i, Q)]$ for some i, which are set to 1:

$$I_L[(\overline{\tau, Q})] := \begin{cases} 1, & (\overline{\tau, Q}) = \{(\tau_i, Q)\} \text{ for some } 1 \leq i \leq s \\ 0, & \text{otherwise.} \end{cases}$$

Then for I, we set $I = \langle I_{L_1}, \dots, I_{L_{|\mathcal{L}|}} \rangle \in \mathbb{N}^k$.

Let now $r \in \mathcal{R}$ be an ITRR. We set

$$\mathrm{up}_r(\langle c_1, \dots, c_{|\mathcal{L}|} \rangle) := \langle \widetilde{c}_1, \dots, \widetilde{c}_{|\mathcal{L}|} \rangle,$$

where each $\widetilde{c}$ is obtained as in the proof of Lemma 5.4.5, using $c_{\mathrm{pre}(\mathcal{L},r)}$ instead of c (for the function pre from Lemma 5.4.7). Thus, it follows from the lemmas

that, at the end, c_{L_F} contains the entries of the compatible type sequence. Accordingly,

$$[\![\#_{\overline{x}}\varphi]\!]^{Qr_1\dots r_n} = \sum_{(\overline{\tau,Q})\in\Lambda} c_{L_F}[(\overline{\tau,Q})] \cdot \prod_{(\tau,Q)\in(\overline{\tau,Q})} [\![\#\tau]\!]^Q,$$

and every $[\![\#\tau]\!]^Q$, $\tau \in H_{\mathrm{qr}(\varphi),\leq|\overline{x}|}$, $Q \in \Sigma$, is a constant. Thus, the product $\prod_{(\tau,Q)\in(\overline{\tau,Q})}[\![\#\tau]\!]^Q$ is constant for every $(\overline{\tau,Q})$. This allows us to define $E = \langle E_{L_1}, \dots, E_{L_{|\mathcal{L}|}}\rangle$ by $E_{L_i} = 0^\Lambda$ for $L_i \neq L_F$, and

$$E_{L_F}[(\overline{\tau,Q})] := \prod_{(\tau,Q)\in(\overline{\tau,Q})} [\![\#\tau]\!]^Q. \qquad \square$$

5.4.3 Reducing to Counter Parity Games

To conclude the proof of Theorem 5.4.1, we use the affine counter updates introduced above to transform $\widehat{\mathrm{MC}}$ into a finite counter parity game. As we proved in Chapter 3, such games can be solved effectively, and accordingly, the counting logic $\mathrm{Q}\mu[\#\mathrm{MSO}]$ is decidable over TPPDSs.

Constructing $\widetilde{\mathrm{MC}}$ Let again ψ be the $\mathrm{Q}\mu[\#\mathrm{MSO}]$ formula and let $\mathfrak{T}$ be the TPPDS with initial state q, and let t be the initial tree. Let further $\widehat{\mathrm{MC}}$ be as defined above.

First of all, we fix an enumeration $\#_{\overline{x}_1}\varphi_1, \dots, \#_{\overline{x}_n}\varphi_n$ of the counting terms in $\mathrm{Sub}(\psi)$. Let now k^i, I^i, E^i and up^i_r ($r \in \mathcal{R}$) be the vectors and functions for $\#_{\overline{x}_i}\varphi_i$ according to Theorem 5.4.8. Combine the initial vectors to a new vector $I = \langle I^1, \dots, I^n\rangle$ of dimension $\sum_{i=1}^n k^i$, and compose the update functions accordingly:

$$\mathrm{up}_r(\langle \overline{c}^1, \dots, \overline{c}^n\rangle) := \langle \mathrm{up}^1_r(\overline{c}^1), \dots, \mathrm{up}^n_r(\overline{c}^n)\rangle.$$

As for the evaluation vectors E^i, we extend these to vectors $\widetilde{E}^i$ by inserting 0s at all positions not belonging to the respective positions for k^i. Note that it now holds that

$$[\![\#_{\overline{x}_i}\varphi_i]\!]^{tr_1\dots r_s} = \widetilde{E}^i \cdot (\mathrm{up}_{r_s} \circ \cdots \circ \mathrm{up}_{r_1})(I). \tag{5.2}$$

The counter parity game $\widetilde{\mathrm{MC}}$ is obtained from $\widehat{\mathrm{MC}}$ by replacing every edge label r by the respective affine function up_r, and by updating the terminal function τ such that at terminal positions belonging to a positive occurrence of the counting term $\#_{\overline{x}_i}\varphi_i$, the payoff is $\widetilde{E}^i \cdot c$ for the respective current counter value, while at positions for a negated counting term $\neg\#_{\overline{x}_i}\varphi_i$ it is $-\widetilde{E}^i \cdot c$. (Technically, this is achieved via the introduction of dummy nodes along which the counters are updated according to $\widetilde{E}^i$, and stored in a single counter, which is then the output. For the sake of readability, we skip this.)

Lemma 5.4.9. Let $\psi \in \mathrm{Q}\mu[\#\mathrm{MSO}]$ and let $\mathfrak{T}$ be a TPPDS. Let $q \in Q$ be an initial state, and t an initial tree. Let $\widetilde{MC}$ and $\widetilde{\mathrm{MC}}$ be defined as above. Then, for the respective initial state q', $\mathrm{val}_{\widehat{\mathrm{MC}}}(q', \bot) = \mathrm{val}_{\widetilde{\mathrm{MC}}}(q', \bot)$.

Proof. Note that for corresponding plays, the payoffs in $\widehat{\mathrm{MC}}$ and $\widetilde{\mathrm{MC}}$ coincide due to Equation 5.2. As the correspondence between plays, moves, and strategies is a one-to-one correspondence, this means that values are preserved as well, and the games are equivalent. □

Proving the main theorem Combining the results of the previous sections, the decidability result of $\mathrm{Q}\mu[\#\mathrm{MSO}]$ on TPPDSs follows.

Proof of Theorem 5.4.1. For a given formula ψ and a TPPDS $\mathfrak{T}$ with initial state q and initial tree $t = Q$, we construct the counter parity game $\widetilde{\mathrm{MC}}$. By Lemma 5.4.9 and Lemma 5.4.2, the value of $\widetilde{\mathrm{MC}}$ from the respective starting position coincides with the value of the formula at the initial configuration. By Theorem 3.2.1, the value of $\widetilde{\mathrm{MC}}$ can be computed. □

5.5 Summary

In this chapter we introduced structure transition systems, which model computation in such a way that the data parts of the system are separated from the temporal aspects and made available as relational structures. We then defined the format in which logics for such systems should be given, and defined a counting logic based on counting terms of monadic second-order logic and the quantitative μ-calculus. For the class of tree-producing pushdown systems, which is a class of systems that generalizes the class of pushdown systems in that the stack is used to store rules that rewrite trees, we proved that the counting logic is decidable. This we did by considering the model-checking game and by introducing a way to iteratively evaluate counting terms using counter update functions upon every application of a rewriting rule. At last, we used the decidability result for counter parity games from Chapter 3 to obtain the value.

Chapter 6

A Quantitative Counting Variant of Monadic Second-Order Logic

In the previous chapter, we presented a counting logic for structure transition systems. The logic is based on the quantitative μ-calculus, and extends it with counting terms of monadic second-order logic. In the present chapter, we consider monadic second-order logic directly, and define a quantitative variant where the emphasis is again on a counting mechanism. Although this logic is inspired by other extensions of MSO (for example, costMSO and MSO+U), we follow the approach used for the quantitative μ-calculus, as we keep the standard definitions of the semantics, but change the domain of evaluations from true and false to the natural numbers (with ∞).

This chapter is based on joint work with Łukasz Kaiser, Martin Lang and Christof Löding, and the results have been submitted for publication [62].

6.1 Syntax and Semantics

As in the case of Qμ, formulas of the new extension, which we call qcMSO, are syntactically very close to formulas of traditional MSO. However, as this simplifies the proofs, we define the logic based on the variant MSO_0 of monadic second-order logic where all variables are second-order. Note that a translation to classical MSO with first-order variables and back is possible.

Accordingly, to be able to simulate first-order variables, we not only have relations and the usual operators, but also have a binary relation $\in$ to denote set membership, and a unary construct $= \varnothing$ to test sets for emptiness. The quantitative aspects are introduced using the operator $|\cdot|$, which counts the size of a set. As in this context it is rather unclear what negation should mean, we omit negation, but add constructs to compare evaluations of subformulas to ∞. As we explain later, this allows us to simulate negation on formulas with

$$[\![R\overline{X}]\!]^{\mathfrak{A},\beta} = \begin{cases} \infty, & \beta(\overline{X}) \in R^{\mathfrak{A}} \\ 0, & \text{otherwise} \end{cases} \qquad [\![X \in Y]\!]^{\mathfrak{A},\beta} = \begin{cases} \infty, & \beta(X) = \{a\}, \\ & a \in \beta(Y) \\ 0, & \text{otherwise} \end{cases}$$

$$[\![|X|]\!]^{\mathfrak{A},\beta} = |\beta(X)| \qquad [\![X = \varnothing]\!]^{\mathfrak{A},\beta} = \begin{cases} \infty, & \beta(X) = \varnothing \\ 0, & \beta(X) \neq \varnothing \end{cases}$$

$$[\![\varphi \wedge \psi]\!]^{\mathfrak{A},\beta} = \min([\![\varphi]\!]^{\mathfrak{A},\beta}, [\![\psi]\!]^{\mathfrak{A},\beta}) \qquad [\![\varphi \vee \psi]\!]^{\mathfrak{A},\beta} = \max([\![\varphi]\!]^{\mathfrak{A},\beta}, [\![\psi]\!]^{\mathfrak{A},\beta})$$

$$[\![\varphi = \infty]\!]^{\mathfrak{A},\beta} = \begin{cases} \infty, & [\![\varphi]\!]^{\mathfrak{A},\beta} = \infty \\ 0, & \text{otherwise} \end{cases} \qquad [\![\varphi < \infty]\!]^{\mathfrak{A},\beta} = \begin{cases} \infty, & [\![\varphi]\!]^{\mathfrak{A},\beta} < \infty \\ 0, & \text{otherwise} \end{cases}$$

$$[\![\exists X \varphi]\!]^{\mathfrak{A},\beta} = \sup_{A' \subseteq A} [\![\varphi]\!]^{\mathfrak{A},\beta \cup \{X \mapsto A'\}} \qquad [\![\forall X \varphi]\!]^{\mathfrak{A},\beta} = \inf_{A' \subseteq A} [\![\varphi]\!]^{\mathfrak{A},\beta \cup \{X \mapsto A'\}}$$

Figure 6.1: Semantics for qcMSO

Boolean evaluations. Formally, the formulas of qcMSO are built according to the following grammar:

$$\begin{aligned} \psi ::= & R\overline{X} \mid X \in X \mid |X| \mid X = \varnothing \mid \\ & \psi \wedge \psi \mid \psi \vee \psi \mid \psi = \infty \mid \psi < \infty \mid \\ & \exists X \psi \mid \forall X \psi. \end{aligned}$$

Following the extension approach already used for Qμ, we define the semantics in a way similar to the classical MSO case. However, as we focus explicitly on counting—keeping questions like boundedness in mind—and treat negation differently, we use $\mathbb{N} \cup \{\infty\}$ as the domain for evaluating formulas. (Note that ∞ is needed to properly define suprema.)

Thus, given a relational structure $\mathfrak{A} = (A, R_1, \ldots, R_n)$ and an evaluation β of the second-order variables, the semantics is given in Figure 6.1.

On several occasions, especially regarding decidability, we do not consider the full logic qcMSO, but its weak variant qcWMSO: As in the case of WMSO, quantification in qcWMSO is restricted to finite sets. Formally, in the semantics of $\exists$ and $\forall$, $A' \subseteq A$ is changed to $A' \subseteq A, |A| < \infty$.

6.1.1 qcMSO extends MSO

One advantage of the above way of defining the semantics is that qcMSO directly extends MSO. The main insight here is that when avoiding the use

of $|X|$ and identifying true with ∞ and false with 0, then the evaluations translate.

Remark 6.1.1. Let $\varphi \in$ MSO be a classical MSO formula (without first-order variables), and let $\mathfrak{A}$ be a structure over the same signature. Let $\widetilde{\varphi}$ be the qcMSO formula where every subformula $\neg\psi$ is replaced by $\widetilde{\psi} < \infty$. If $\mathfrak{A} \vDash \varphi$ with classical MSO semantics, then $[\![\widetilde{\varphi}]\!]^{\mathfrak{A}} = \infty$ with qcMSO semantics, and if $\mathfrak{A} \nvDash \varphi$, then $[\![\widetilde{\varphi}]\!]^{\mathfrak{A}} = 0$.

Proof. The claim follows from a simple induction on the structure of formulas. For the atomic cases, qcMSO evaluation exactly evaluates to ∞ if the MSO formula would evaluate to true, and to 0 otherwise. As $\min, \max, \sup$ and $\inf$ preserve the property that the evaluation is from $\{0, \infty\}$, and $< \infty$ thus exactly corresponds to negation, the remark follows. □

6.1.2 Examples

In general, it turns out that formulas can be written almost as if they were pure MSO formulas, and often have the intended evaluation in qcMSO. As a first example, we construct a formula that evaluates to (the least upper bound of) the number of elements that are in a relation R. This can, for example, be thought of as the number of positions in a word labeled with a letter a. As only sizes of sets can be counted, the idea is to collect all elements from R in a set X, and then to take the supremum over all such sets X. This could be represented as $\sup_X\{|X| \mid X \subseteq R\}$. The subset-relation can be expressed classically as $\forall y(Xy \to Ry)$. Combining this, a correct formula in qcMSO would be

$$\exists X(|X| \wedge \forall Y((Y \in X < \infty) \vee RY)).$$

Note that $Y \in X < \infty$ corresponds to $\neg Y \in X \equiv Y \notin X$.

The above thus expresses the supremum over all sets X that satisfy a certain condition (in this case, being a subset of R) of a formula (in this case, $|X|$). This relativization of the quantifier generalizes, and can similarly be done for the infimum, that is, for the quantifier $\forall$.

In qcMSO it is furthermore possible to express that a set is finite, using the simple approach $\mathrm{fin}(X) := |X| < \infty$. This means that one can—regardless of the structure considered—quantify over all finite sets, which will be useful later when embedding MSO+U.

Outline The remainder of this chapter is organized in the following way. We begin by discussing the relationship between qcMSO and other quantitative variants of MSO, more specifically the connection with costMSO and

MSO+U. We then turn to a first decidability problem for qcMSO: we prove, using a reduction to a model-checking problem for an extended version of a logic for resource-automatic structures, that the qcWMSO-theory of the natural linear order $(\omega, <)$ is decidable. The presented techniques, however, cannot be adapted for qcMSO with full second-order quantification, and it furthermore already follows from results on MSO+U that the qcMSO-theory of $(\omega, <)$ is undecidable. After presenting the undecidability results, we conclude this chapter by giving decomposition algorithms for qcMSO on $(\omega, <)$ and the infinite binary tree.

6.2 On the Relation to costMSO and the Unbounding Quantifier

costMSO and MSO+U are extensions of MSO with quantitative aspects that have been studied extensively in the last years [5–10, 30–32, 57, 67, 68, 85, 87]. In most cases, the two logics have been studied separately, and have been compared to different automata models in order to examine the decidability problems on finite and infinite words and finite and infinite trees. As it turns out, MSO+U can be embedded into qcMSO, and several problems discussed for costMSO (such as boundedness and dominance) are expressible in qcMSO. Thus, decidability algorithms for qcMSO provide a uniform approach for several problems from both the fields of costMSO and MSO+U.

6.2.1 The Unbounding Quantifier is Expressible in qcMSO

The logic MSO+U is the extension of MSO with two quantifiers U and B. The first one, U, expresses that there are arbitrarily large (finite) sets satisfying the subsequent formula, whereas B demands that there is a finite upper bound on the size of sets satisfying a formula. Note that we already remarked as an example that finiteness of a set is expressible in qcMSO.

As MSO is subsumed by qcMSO, it suffices to give translations for U and B in order to show that every MSO+U-formula can be translated into an equivalent formula from qcMSO. Given a formula $\varphi = UX.\psi$, where $\widetilde{\psi}$ is a translation for ψ, the following equivalence holds:

$$\mathfrak{A} \vDash UX.\psi \Leftrightarrow [\![(\exists X(|X| \wedge \widetilde{\psi} \wedge \mathrm{fin}(X))) = \infty]\!]^{\mathfrak{A}} = \infty.$$

Similarly, for B we obtain

$$\mathfrak{A} \vDash BX.\psi \Leftrightarrow [\![(\exists X(|X| \wedge \widetilde{\psi} \wedge \mathrm{fin}(X))) < \infty]\!]^{\mathfrak{A}} = \infty.$$

Lemma 6.2.1. For every MSO+U formula φ, one can construct a formula $\widetilde{\varphi} \in$ qcMSO such that for all structures $\mathfrak{A}$:

$$\mathfrak{A} \vDash \varphi \Leftrightarrow [\![\widetilde{\varphi}]\!]^{\mathfrak{A}} = \infty.$$

As finiteness of quantified sets is inherent in qcWMSO, the following corollary is immediate by just omitting $\mathrm{fin}(X)$.

Corollary 6.2.2. WMSO+$U \subseteq$ qcWMSO.

6.2.2 Encoding Boundedness and Dominance in qcMSO

Another quantitative extension of MSO is costMSO. This logic is also quantitative, but in a different sense, as a model of a formula is not solely a structure (commonly a word or tree), but a structure together with an additional numerical parameter. Thus, one is often interested in finding this parameter, which is a task closely related to the theory of cost functions. For our purposes, it suffices to view cost functions as functions that characterize costs of structures in a manner that preserves boundedness. (For a general introduction to cost functions, see for example [30, 31].) Thus, the value of a word is finite if and only if the actual costs are finite, and similarly for languages.

The classical definition of costMSO is such that MSO is extended by a new kind of atom, namely $|X| < N$, where X is a second-order variable, and N is a unique variable for the single external numerical parameter. This atom is only allowed positively in the formula, that is, under an even number of negations. Given a costMSO formula φ, a model is a tuple $(\mathfrak{A}, n)$ consisting of a relational structure $\mathfrak{A}$ and a number $n \in \mathbb{N}$, where $\mathfrak{A}, n \vDash \varphi$ if and only if $\mathfrak{A} \vDash \varphi$ such that an atom $|X| < N$ holds only if $|X| < n$ for the respective interpretation of X. The semantics of a formula evaluated on a structure is the infimum over all such n, with $\inf \varnothing = \infty$.

In [31], an equivalent inductive definition of the semantics of costMSO was given, which is closer to the way the semantics of qcMSO was defined and thus allows an easier translation:

Definition 6.2.3. Let $\varphi, \psi \in$ costMSO be formulas, let $\mathfrak{A}$ be a structure, and let β be an evaluation of the free variables. The costMSO semantics $[\![\varphi]\!]^{\mathfrak{A},\beta}$ is as follows:

$$[\![R\overline{X}]\!]^{\mathfrak{A},\beta} = \begin{cases} 0, & \beta(\overline{X}) \in R^{\mathfrak{A}} \\ \infty, & \text{otherwise} \end{cases} \qquad [\![\neg R\overline{X}]\!]^{\mathfrak{A},\beta} = \begin{cases} \infty, & \beta(\overline{X}) \in R^{\mathfrak{A}} \\ 0, & \text{otherwise} \end{cases}$$

$$[\![|X| < N]\!]^{\mathfrak{A},\beta} = |\beta(X)|$$

$$[\![\varphi \vee \psi]\!]^{\mathfrak{A},\beta} = \min([\![\varphi]\!]^{\mathfrak{A},\beta}, [\![\psi]\!]^{\mathfrak{A},\beta}) \qquad [\![\varphi \wedge \psi]\!]^{\mathfrak{A},\beta} = \max([\![\varphi]\!]^{\mathfrak{A},\beta}, [\![\psi]\!]^{\mathfrak{A},\beta})$$

$$[\![\exists X \varphi]\!]^{\mathfrak{A},\beta} = \inf_{A' \subseteq A} [\![\varphi]\!]^{\mathfrak{A},\beta \cup \{X \mapsto A'\}} \qquad [\![\forall X \varphi]\!]^{\mathfrak{A},\beta} = \sup_{A' \subseteq A} [\![\varphi]\!]^{\mathfrak{A},\beta \cup \{X \mapsto A'\}}$$

Note that we again used the variant of MSO without first-order variables. For better readability, we also omitted the definition of the semantics of $\in$, as it is just as for other relation symbols.

Again, there is a weak variant, where quantification ranges only over finite sets.

As the inductive definition is already close to the definition of the semantics of qcMSO, it is no surprise that indeed, for every formula in costMSO, one can construct an equivalent one in qcMSO.

Lemma 6.2.4. For every $\varphi \in$ costMSO one can effectively construct a formula $\widetilde{\varphi} \in$ qcMSO such that $[\![\varphi]\!] = [\![\widetilde{\varphi}]\!]$.

Proof. The lemma is proved via an induction on the structure of formulas. For atoms $R\overline{X} \in$ costMSO, we set $\widetilde{R\overline{X}} := (R\overline{X} < \infty)$, and analogously $\widetilde{\neg R\overline{X}} := R\overline{X}$. For $|X| < N$, we just omit the N: $\widetilde{|X| < N} := |X|$. As the definition of $\vee$ and $\wedge$ is dual to the one in qcMSO, we swap them: $\widetilde{\varphi \vee \psi} := \widetilde{\varphi} \wedge \widetilde{\psi}$, $\widetilde{\varphi \wedge \psi} := \widetilde{\varphi} \vee \widetilde{\psi}$, and use the same principle for quantifiers: $\widetilde{\exists X \varphi} := \forall X \widetilde{\varphi}$ and $\widetilde{\forall X \varphi} := \exists X \widetilde{\varphi}$. □

6.2.2.1 Boundedness and Dominance

Two major decision problems in the context of costMSO are boundedness and dominance. We say that a costMSO formula φ is *bounded* over a domain $\mathcal{D}$ of structures if $\sup_{\mathfrak{A} \in \mathcal{D}} [\![\varphi]\!]^{\mathfrak{A}} < \infty$. Usual domains here are the set of finite words, or the infinite words or finite/infinite trees.

Dominance is defined similarly, but now does not require a formula to be bounded on the whole domain, but only where another formula is bounded: We say that φ is *dominated* by ψ if for all subsets $\mathcal{L} \subseteq \mathcal{D}$, $\sup_{\mathfrak{A} \in \mathcal{L}} [\![\psi]\!]^{\mathfrak{A}} < \infty$ implies $\sup_{\mathfrak{A} \in \mathcal{L}} [\![\varphi]\!]^{\mathfrak{A}} < \infty$.

Using the above lemma, it turns out that boundedness and dominance problems for costMSO can be reduced to certain model-checking problems for qcMSO.

Lemma 6.2.5.

1. The boundedness problem for costMSO on finite words can be reduced to the model-checking problem for qcWMSO on $(\omega, <)$.
2. The boundedness problem for costMSO on infinite words can be reduced to the model-checking problem for qcMSO on $(\omega, <)$.
3. The boundedness problem for costMSO on finite trees can be reduced to the model-checking problem for qcWMSO on the infinite binary tree.
4. The dominance problem for costMSO on finite words can be reduced to the model-checking problem for qcMSO on the infinite binary tree.

Proof. 1. We consider $\{0,1\}$-labeled finite words. Obviously, such a word w can be represented by two finite disjoint sets $X_w, X_1 \subseteq \mathbb{N}$ where $X_w = \{0,\dots,n-1\}$ for some n and $X_1 \subseteq X_w$, selecting an initial segment of the natural numbers and indicating those positions which are labeled with 1. Let now $\rho(X_w, X_1) \in$ WMSO be a formula that defines these sets in $(\omega, <)$, which clearly exists, and let $\widetilde{\rho}$ be the equivalent formula as defined in Remark 6.1.1. Consider now the formula $\widetilde{\varphi}$ as in Lemma 6.2.4. By relativizing all quantifiers to X_w and replacing P_1X by $X \in X_1$ and P_0X by $(X \in X_1 < \infty)$, we obtain a formula $\varphi'(X_w, X_1) \in$ qcMSO such that $[\![\varphi']\!]^{\omega,<,X_w,X_1} = [\![\varphi]\!]^w$ for all words w and the respective sets X_w, X_1. Boundedness of φ on finite words is then expressed by the following formula on $(\omega, <)$:

$$(\exists X_w \exists X_1 (\widetilde{\rho}(X_w, X_1) \wedge \varphi'(X_w, X_1))) < \infty.$$

2. Using a similar idea, an infinite word can be represented by a single set indicating the positions labeled with 1. (Note that this set may be infinite.) Thus, one simply has to replace P_1X and P_0X as above, and thus obtains a respective formula with one additional quantifier.

3. A finite tree can be encoded in the infinite binary tree by a set X_t defining the universe, and again by a set X_1. To define the universe, one simply has to check that X_t is prefix-closed, which can be done in WMSO. The remaining proof is analogous to (1.).

4. To encode dominance on finite words in the binary tree, note first that each position in the tree can be identified with a word, as the universe of the tree can be seen as $\{0,1\}^*$. It is not hard to define, for an MSO sentence ψ, an MSO formula $\widehat{\psi}(x)$ with one free variable such that for every word w: $w \vDash \psi$ if and only if $\mathfrak{T}_2 \vDash \widehat{\psi}(w)$. This can easily be extended to costMSO such that for every formula in costMSO over words one can define one over trees with a free variable that, when interpreting the free variable with a position w, evaluates to the same number as the original formula when evaluated on w.

   Given two costMSO formulas on finite words φ and ψ, we now compute the respective formulas $\widehat{\varphi}$ and $\widehat{\psi}$ for the infinite binary tree, and then apply Lemma 6.2.4 to these to obtain $\widetilde{\varphi}$ and $\widetilde{\psi}$. Dominance of ψ over φ is then equivalent to the formula

$$\begin{aligned}&\forall X(((\exists Y(Y \in X \wedge \widetilde{\psi})) < \infty) < \infty \vee (\exists Y(Y \in X \wedge \widetilde{\varphi})) < \infty)\\ \equiv &\forall X((\exists Y(Y \in X \wedge \widetilde{\psi})) = \infty \vee (\exists Y(Y \in X \wedge \widetilde{\varphi})) < \infty)\end{aligned}$$

   evaluating to ∞. □

We remark that dominance cannot be reduced directly to qcWMSO and costWMSO, as quantification over all—and especially also all infinite—sets of words is required. Nevertheless, the above results about MSO+U and costMSO allow to conclude that the decision procedure presented in the next chapter provides a uniform approach to problems from (slightly) different fields.

6.3 Deciding qcWMSO on the Natural Linear Order

In the previous sections we showed that costMSO and MSO+U can be embedded into qcMSO, but so far we have not presented any results on decidability. In the present section we provide an automata-theoretical construction—based on a reduction to model-checking problems on resource-automatic structures—to show that the qcWMSO-theory of the natural infinite linear order $(\omega, <)$ is decidable. Note that this also shows once more, using similar techniques, that the boundedness problem for costMSO on finite words is decidable (which was shown in [29, 65]), and that the WMSO+U-theory of $(\omega, <)$ is decidable (as already shown in [5]), and does so uniformly.

The proof is organized as follows: We begin with a brief introduction to resource-automatic structures and recall the logic FO+RR. We then extend this logic by ∞-comparisons and prove that this extension is still decidable on resource-automatic structures. At last, we present a fixed resource-automatic structure and show that the model-checking problem for qcWMSO-sentences on $(\omega, <)$ can be reduced to the model-checking problem for the extension of FO+RR defined before on that fixed resource-automatic structure.

6.3.1 Resource-Automatic Structures and FO+RR

Resource-automatic structures extend the well-studied theory of automatic structures (see for example [1, 81]) by so-called resources: being in a relation now involves a certain cost, and thus it may be expensive to satisfy a given formula. To model this, instead of classical nondeterministic finite automata (see Chapter 2.3), one uses B- and S-automata to define languages that represent the universe and the relations upon it. The introduction to resource-automatic structures is based on [72].

Definition 6.3.1. A *B-/S-automaton* is an NFA $\mathcal{A}_B/\mathcal{A}_S = (Q, \Sigma, \delta, q_0, F)$ that, additionally, is equipped with a finite set Γ of counters. Thus, the transition function is of the form $\delta\colon Q \times \Sigma \to \mathcal{P}(Q \times \mathfrak{C}^\Gamma)$, where $\mathfrak{C}$ is the set

of counter operations. For B-automata, we use $\mathfrak{C}_B = \{\textcircled{ic}, \textcircled{r}, \textcircled{n}\}$, while for S-automata one uses $\mathfrak{C}_S = \{\textcircled{ι}, \textcircled{cr}, \textcircled{r}, \textcircled{n}\}$.

The notion of a run is as for NFAs, with the addition that a Γ-dimensional vector of counters is maintained. Upon each transition with counter operations $\gamma = \gamma_1, \ldots, \gamma_{|\Gamma|}$, the counter vector $c \in \mathbb{N}^\Gamma$ is updated to $\gamma(c)$: If $\gamma_i = \textcircled{r}$, then $\gamma(c)_i = 0$. If $\gamma_i = \textcircled{n}$, then $\gamma(c)_i = c_i$. If $\gamma_i = \textcircled{ι}$, then $\gamma(c)_i = c_i + 1$. The actions $\textcircled{ic}$ and $\textcircled{cr}$ update the counters just as the operations $\textcircled{ι}$ and $\textcircled{r}$, respectively. Additionally, however, they *check* the counter value c_i: This means that the counter value is added to the set $C_i(\rho)$ of checked values for counter i. These sets are used for the semantics, where $R(\mathcal{A}, w)$ is the set of accepting runs of $\mathcal{A}$ on w:

$$\mathcal{A}_B(w) := \inf_{\rho \in R(\mathcal{A}_B, w)} \sup\Big(\bigcup_{i \in \Gamma} C_i(\rho)\Big),$$
$$\mathcal{A}_S(w) := \sup_{\rho \in R(\mathcal{A}_B, w)} \inf\Big(\bigcup_{i \in \Gamma} C_i(\rho)\Big).$$

Here, we make use of the convention that $\inf(\varnothing) = \infty$ and $\sup(\varnothing) = 0$.

To work with these automata later, we use that one can effectively transform B- into S-automata, and vice versa:

Theorem 6.3.2 ([29, 30])**.** 1. Every B-automaton $\mathcal{A}_B$ can be transformed effectively into an equivalent S-automaton $\mathcal{A}_S$.

2. Every S-automaton $\mathcal{A}_S$ can be transformed effectively into an equivalent B-automaton $\mathcal{A}_B$.

Equivalence is understood as equivalent up to a monotone correction function $\alpha\colon \mathbb{N} \cup \{\infty\} \to \mathbb{N} \cup \{\infty\}$ where $\alpha(x) = \infty$ if and only if $x = \infty$: $\mathcal{A}_B$ is equivalent to $\mathcal{A}_S$, if there exists a correction function $\alpha\colon \mathbb{N} \cup \{\infty\} \to \mathbb{N} \cup \{\infty\}$ such that for all words w: $\mathcal{A}_B(w) \leq \alpha(\mathcal{A}_S(w))$ and $\mathcal{A}_S(w) \leq \alpha(\mathcal{A}_B(w))$.

Note that correction functions preserve boundedness: If $\mathcal{A}$ is α-equivalent to $\mathcal{A}'$, then for all $L \subseteq \Sigma^*$: $\{\mathcal{A}(w) \mid w \in L\}$ is bounded if and only if $\{\mathcal{A}'(w) \mid w \in L\}$ is bounded.

6.3.1.1 Resource-Automatic Structures

Resource-automatic structures are (infinite) resource structures that are representable by B- and S-automata. A *resource structure* $\mathfrak{A} = (A, R_1, \ldots, R_n)$ is an extended relational structure, where relations are now functions $R_i\colon A^{\mathrm{ar}(R_i)} \to \mathbb{N} \cup \{\infty\}$. The intended meaning of this is that it may be costly for a tuple to be in a relation, and the cost is ∞ if it is not in the relation at all.

To represent n-ary relations as words, we use n-tuples of letters, that is, the alphabet $(\Sigma \cup \{\square\})^n$, where $\square$ is a padding symbol that may only appear

at the end of words. For example, the tuple $(0110, 01111100)$ is represented as the *convolution*

$$\sigma(0110, 01111100) := \begin{matrix} 0 & 1 & 1 & 0 & \square & \square & \square & \square \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{matrix} \in (\Sigma \cup \{\square\})^2.$$

For a formal definition of such convolutions and of the respective transducers, we refer to [72].

Definition 6.3.3. A resource structure $\mathfrak{A} = (A, R_1, \dots, R_n)$ is *resource-automatic*, if there is a regular language $L \subseteq \Sigma^* = \{0,1\}^*$, a bijection $\pi\colon L \to A$, and B-automata $\mathcal{A}_1, \dots, \mathcal{A}_n$ such that for all i, and all tuples $(a_1, \dots, a_{\mathrm{ar}(R_i)})$:

$$R_i(a_1, \dots, a_{\mathrm{ar}(R_i)}) = \mathcal{A}_i(\sigma(\pi^{-1}(a_1), \dots, \pi^{-1}(a_{\mathrm{ar}(R_i)}))).$$

Example 6.3.4. As an example for a resource-automatic structure, we consider the structure $\mathfrak{F} = (\mathrm{FinPot}(\mathbb{N}), \in, <, = \varnothing, |\cdot|)$ of finite subsets of natural numbers, together with the $\in$- and $<$-relations (technically, the respective complements) and a test for the empty set as well as a relation which evaluates to the size of set. Note that this structure will also be used in the reduction from qcWMSO to the extension of FO+RR.

As the regular language representing the universe, we take the set of all words ending with a 1, and the additional word 0 to represent the empty set, thus $L = \{0,1\}^*1 \cup \{0\}$. As for the bijection, a word $w = w_0 \dots w_{n-1}$ is mapped to the set $\{i \mid w_i = 1\}$.

For the relation $\in$, we want that $a \in^{\mathfrak{F}} b = \infty$ if a is a singleton set which is a subset of b, and $a \in^{\mathfrak{F}} b = 0$ otherwise. This amounts to a B-automaton that can be viewed as an NFA which rejects all words

$$\left\{ \begin{matrix} 0 \\ 0 \end{matrix}, \begin{matrix} 0 \\ 1 \end{matrix} \right\}^* \begin{matrix} 1 \\ 1 \end{matrix} \left\{ \begin{matrix} \square \\ 0 \end{matrix}, \begin{matrix} \square \\ 1 \end{matrix} \right\}^*,$$

and accepts all other words. Similarly, as $a <^{\mathfrak{F}} b = \infty$ if and only if a and b are singletons and the natural order holds on the respective elements, this amounts to an NFA which rejects all words

$$\left(\begin{matrix} 0 \\ 0 \end{matrix} \right)^* \begin{matrix} 1 \\ 0 \end{matrix} \left(\begin{matrix} \square \\ 0 \end{matrix} \right)^* \begin{matrix} \square \\ 1 \end{matrix}$$

and accepts otherwise. The automaton for $= \varnothing$ rejects only 0, and accepts all other words.

For $|\cdot|$, we take a 1-counter B-automaton $\mathcal{A}$, that accepts all words and increases the counter upon seeing a 1. Clearly, $\mathcal{A}(w) = |\{w_i \mid w_i = 1\}|$. ⊣

6.3.1.2 FO+RR

In [72], the authors introduce a logic for resource-automatic structures based on classical first-order logic. As the syntax is precisely that of FO without negation, it suffices to give the definition of the semantics of this logic called FO+RR. The semantics closely follows the intuition that the further away from being in a relation a tuple is, the more costly it is, with ∞ indicating that a tuple is not in the relation at all. This amounts to viewing 0 as true, and ∞ as false.

As stated above, the syntax of FO+RR is precisely as the syntax of FO without negation. The semantics of FO+RR is given below, where $\mathfrak{A}$ is a resource structure, and β is an interpretation of the free variables:

$$[\![Rx_1 \dots x_n]\!]^{\mathfrak{A},\beta} := R^{\mathfrak{A}}(\beta(x_1), \dots, \beta(x_n))$$

$$[\![\varphi \vee \psi]\!]^{\mathfrak{A},\beta} := \min([\![\varphi]\!]^{\mathfrak{A},\beta}, [\![\psi]\!]^{\mathfrak{A},\beta}) \quad [\![\varphi \wedge \psi]\!]^{\mathfrak{A},\beta} := \max([\![\varphi]\!]^{\mathfrak{A},\beta}, [\![\psi]\!]^{\mathfrak{A},\beta})$$
$$[\![\exists x\varphi]\!]^{\mathfrak{A},\beta} := \inf_{a\in A} [\![\varphi]\!]^{\mathfrak{A},\beta\cup\{x\mapsto a\}} \qquad [\![\forall x\varphi]\!]^{\mathfrak{A},\beta} := \sup_{a\in A} [\![\varphi]\!]^{\mathfrak{A},\beta\cup\{x\mapsto a\}}$$

Up to equivalence up to a correction function, the operators of FO+RR can be simulated by B- and S-automata. The key ingredient in the proof of the following theorem, which can be found in [72], is showing that automata for the projections in the convolutions—which correspond to the quantifiers—can indeed be constructed. This corresponds to a combination of the projections from [29, 30] and a kind of ϵ-transition elimination.

Theorem 6.3.5 ([72])**.** Given a resource-automatic structure $\mathfrak{A}$ and the respective automata as well as an FO+RR sentence φ, it is decidable if $[\![\varphi]\!]^{\mathfrak{A}} < \infty$.

6.3.2 Extending FO+RR by Comparisons with ∞

In order to be able to reduce the problem of deciding the qcWMSO-theory of $(\omega, <)$ to a model-checking question of FO+RR on resource-automatic structures, we have to extend FO+RR in order to be able to simulate the comparisons $\varphi = \infty$ and $\varphi < \infty$ of qcMSO. We do so by adding these operators to the syntax and semantics of FO+RR in the obvious way, and by showing that this extension is still decidable on resource-automatic structures via giving an automata construction for the added operators.

The syntax of the extension, which we call FO+RR$^{=\infty}$, is given by the following grammar:

$$\varphi ::= Rx_1 \dots x_n \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x\varphi \mid \forall x\varphi \mid \varphi = \infty \mid \varphi < \infty.$$

The semantics for the operators from FO+RR remains the same. The additional semantics can be found below.

$$[\![\varphi = \infty]\!]^{\mathfrak{A},\beta} := \begin{cases} 0, & [\![\varphi]\!]^{\mathfrak{A},\beta} = \infty \\ \infty, & \text{otherwise} \end{cases} \qquad [\![\varphi < \infty]\!]^{\mathfrak{A},\beta} := \begin{cases} 0, & [\![\varphi]\!]^{\mathfrak{A},\beta} < \infty \\ \infty, & \text{otherwise} \end{cases}$$

In order to lift the decidability result for FO+RR to FO+RR$^{=\infty}$, we have to give a construction that takes an automaton $\mathcal{A}$ and returns an automaton $\mathcal{A}'$ which returns 0 on a word if $\mathcal{A}$ returns ∞—and ∞ otherwise—, and similarly for the negation. As the automata for $= \infty$ need to be Boolean, automata for $< \infty$ can be obtained from these via NFA complementation, so we focus on the former.

Lemma 6.3.6. Let $\mathcal{A}$ be a B-automaton. One can effectively construct a B-automaton $\mathcal{A}_\infty$ such that for all words w:

$$\mathcal{A}_\infty(w) = 0 \iff \mathcal{A}(w) = \infty,$$
$$\mathcal{A}_\infty(w) = \infty \iff \mathcal{A}(w) \neq \infty.$$

Proof. As we do not allow ϵ-transitions in the automata, the counter values that occur along a run of $\mathcal{A}$ on a finite word w are bounded by the length of w. Thus, the infimum over all runs of the maximal counter values that occur can only be ∞ if the set of accepting runs is empty. But this means that $\mathcal{A}$ never reaches a final state. Consider thus the NFA $\mathcal{A}'$ induced by $\mathcal{A}$ when ignoring the counters. Then, $\mathcal{A}_\infty$ has to return 0 if $\mathcal{A}'$ rejects, and ∞ if $\mathcal{A}'$ accepts. Therefore, construct an NFA $\mathcal{A}''$ that accepts the complement of the language of $\mathcal{A}'$. $\mathcal{A}_\infty$ is then precisely $\mathcal{A}''$ viewed as a B-automaton: As there are no counters, the supremum over all checked counter values when reaching a final state is 0, and this happens precisely on these words which are rejected by $\mathcal{A}'$. On the other hand, there are no accepting runs exactly on those words that are accepted by $\mathcal{A}'$. □

Using the above construction, it follows that FO+RR$^{=\infty}$ is decidable on resource-automatic structures: The inductive translation from formulas to corresponding B- and S-automata is extended by the translations for $= \infty$ and $< \infty$ (which is the complement automaton to $= \infty$).

Corollary 6.3.7. Given a resource-automatic structure $\mathfrak{A}$ and the respective automata and a sentence $\varphi \in$ FO+RR$^{=\infty}$, one can effectively decide whether $[\![\varphi]\!]^{\mathfrak{A}} < \infty$.

6.3.3 Reducing qcWMSO to FO+RR$^{=\infty}$

With the above, we are ready to prove that the qcWMSO-theory of $(\omega, <)$ is decidable. The proof is via a reduction to the model-checking problem for FO+RR$^{=\infty}$. The central idea is to reduce the second-order quantification over finite sets to first-order quantification over the powerset structure (restricted to finite sets), as in [73]. As the interpretation is different in qcWMSO and FO+RR$^{=\infty}$, we consider the powerset structure with negated relations, as introduced in Example 6.3.4.

Theorem 6.3.8. Let $\varphi \in$ qcWMSO be a formula, and let $\mathfrak{F}$ be the resource-automatic structure from Example 6.3.4. One can effectively construct a formula $\widehat{\varphi} \in$ FO+RR$^{=\infty}$ such that

$$[\![\varphi]\!]^{(\omega,<),\beta} = [\![\widehat{\varphi}]\!]^{\mathfrak{F},\widehat{\beta}}, \qquad \text{where } \widehat{\beta}(x) = \beta(X).$$

Proof. We describe the translation inductively over the structure of formulas, and simultaneously prove that values are preserved.

Let φ be an atomic formula, thus $\varphi = R\overline{X}$ for some relation R. We set $\widehat{\varphi} := R\overline{x}$, replacing second-order variables X_i by their respective first-order counterparts x_i. By definition of $\mathfrak{F}$, the evaluation is preserved.

Let $\varphi = \varphi_1 \vee \varphi_2$, assuming that φ_1, φ_2 are equivalent to $\widehat{\varphi}_1, \widehat{\varphi}_2$, respectively. As in qcWMSO disjunction amounts to a maximum, we set $\widehat{\varphi} := \widehat{\varphi}_1 \wedge \widehat{\varphi}_2$. Analogously, for $\varphi = \varphi_1 \wedge \varphi_2$, we set $\widehat{\varphi} := \widehat{\varphi}_1 \vee \widehat{\varphi}_2$. The evaluations are preserved, as the interpretation of the operators in FO+RR$^{=\infty}$ is dual to the one in qcWMSO. Following this approach, for $\varphi = \exists X \psi$ we set $\widehat{\varphi} := \forall x \widehat{\psi}$, and for $\varphi = \forall X \psi$ we set $\widehat{\varphi} := \exists X \widehat{\psi}$.

It remains to give the translation for the comparisons with ∞. Note that $[\![\varphi = \infty]\!]^{(\omega,<),\beta} = \infty$ if and only if φ evaluates to ∞. Accordingly, using once more the duality of the operators, for $\varphi = (\psi = \infty)$ we set $\widehat{\varphi} := \widehat{\psi} < \infty$, and analogously set $\widehat{\varphi} := (\widehat{\psi} = \infty)$ for $\varphi = (\psi < \infty)$. □

Thus, it follows from the decidability of the boundedness problem for FO+RR$^{=\infty}$ on resource-automatic structures that the boundedness problem for qcWMSO on $(\omega, <)$ is decidable:

Corollary 6.3.9. Let $\varphi \in$ qcWMSO be a sentence. It is decidable whether $(\omega, <) \models \varphi < \infty$.

It was argued in [72] that given an FO+RR-sentence φ, it is decidable for every $k \in \mathbb{N}$ whether $[\![\varphi]\!]^{\mathfrak{A}} \leq k$, and that this entails that $[\![\varphi]\!]^{\mathfrak{A}}$ is computable if it is known whether the evaluation of φ on $\mathfrak{A}$ is bounded. As the automata $\mathcal{A}_\infty$ for ∞-comparisons are essentially NFAs, we can use these for the respective subformulas in the construction to compute evaluations from [72]. Thus, exact evaluations can also be computed in the case of FO+RR$^{=\infty}$.

Corollary 6.3.10. Let $\varphi \in$ qcWMSO be a sentence. The evaluation $[\![\varphi]\!]^{(\omega,<)}$ can be computed.

6.4 qcMSO and Decidability

Above, we proved that the qcWMSO-theory of $(\omega, <)$ is decidable. This immediately raises the question whether the result extends to full qcMSO, where quantification over all and not just finite sets is allowed. As qcMSO subsumes MSO+U, it follows from results on MSO+U that this is not the case. Before we say more about the undecidability of full qcMSO, we remark that the topological complexity of qcMSO-definable languages on infinite words is very high, which follows from a result from [57] described below.

6.4.1 The Expressive Power of qcMSO on Infinite Words

In contrast to MSO-definable languages, MSO+U-definable languages exhaust all levels of the Borel hierarchy, and go even further. To formulate the subsequent result on the topological complexity of these languages, we first introduce the projective hierarchy and the notion of topological reductions.

The projective hierarchy is the hierarchy obtained from the Borel hierarchy (see Chapter 2.1.2) by projections. Given a set x of infinite words over an alphabet $\Sigma' \times \Sigma$, the *projection* $\pi(x)$ of x is the set

$$\pi(x) := \{w \in \Sigma^\omega \mid \text{there exists } v \in \Sigma'^\omega \text{ s.th. } \sigma(v, w) \in x\}.$$

Commonly, we use $\Sigma' = \omega$.

Definition 6.4.1. The *projective hierarchy* is inductively defined as follows, for ω levels, where Σ is some alphabet.

$$\begin{aligned} \Sigma^1_1(\Sigma) &:= \{\pi(x) \in \Sigma^\omega \mid x \in \mathfrak{B}(\omega \times \Sigma)\} \\ \Pi^1_n(\Sigma) &:= \{x \mid x^{\mathcal{C}} \in \Sigma^1_n\} \\ \Sigma^1_{n+1}(\Sigma) &:= \{\pi(x) \mid x \in \Pi^1_n(\omega \times \Sigma)\} \end{aligned}$$

As for the Borel hierarchy, the levels of the projective hierarchy are contained in higher levels, and the hierarchy properly extends the Borel hierarchy.

To define topological reductions, the notion of a continuous function is necessary: A function $f\colon \Sigma^\omega \to \Sigma'^\omega$ is *continuous* if for every open set $x \in \Sigma'^\omega$, the inverse image $f^{-1}(x) \in \Sigma^\omega$ is open as well. Given two sets x, y, we say that a continuous function f *reduces* x to y if $f^{-1}(y) = x$. If x is reducible to y (that is, a continuous function f that reduces x to y exists) and y belongs to

some level of the Borel or the projective hierarchy, then so does x. Let Σ^i_α be a level of these hierarchies, and let x be a set. Then x is *hard* for Σ^i_α if every $y \in \Sigma^i_\alpha$ is reducible to x, and is *complete* for the level if, additionally, $x \in \Sigma^i_\alpha$.

6.4.1.1 qcMSO Reaches up the Projective Hierarchy

It is known that ω-regular languages are Borel sets of low rank (at most Σ^0_3), and by Büchi's theorem [16], these are exactly the sets definable in classical MSO. However, this does not necessarily hold anymore when quantitative aspects are introduced. It was shown in [57] that for every level in the projective hierarchy there are languages definable in MSO+U that are hard for the respective level. As MSO+U is subsumed by qcMSO, and in particular the model-checking problem for MSO+U on a structure is reducible to the boundedness problem for qcMSO on that structure, it follows that the topological complexity of qcMSO-definable languages also reaches up the whole projective hierarchy.

Theorem 6.4.2 ([57])**.** Let $n \in \omega$. There exist a finite alphabet B_n and a formula $\varphi_n \in$ MSO+U such that the language $L_n = \{w \in B_n^\omega \mid w \vDash \varphi_n\}$ is Σ^1_n-hard.

Corollary 6.4.3. Let $n \in \omega$. There exist a finite alphabet B_n and a formula $\varphi_n \in$ qcMSO such that the set of words $L_n = \{w \in B_n^\omega \mid w \vDash \varphi_n = \infty\}$ is Σ^1_n-hard.

Note also that the nondeterministic ω-counter automata considered so far (such as ωB-, ωS- and ωBS-automata) define languages that are at most in Σ^0_4, and the alternating variants reach only up to Σ^1_2 [57].

Remark 6.4.4. One should note that the result presented above is of a different form compared to the decidability result for qcWMSO. For the latter, we did not construct an equivalent automaton model which captures all definable languages, but rather focused on the theory, that is, only on sentences of the logic. In contrast to that, the former speaks about the topological complexity of definable languages, that is, of the boundedness problem for formulas with free variables.

While we did not study this question for qcWMSO, there is work on WMSO+U on infinite words, showing that WMSO+U is equivalent to deterministic $\max$-automata and that emptiness of these is decidable [5]. As every $\max$-automaton is equivalent to a nondeterministic ωBS-automaton [5], WMSO+U-definable languages are at most in Σ^0_4 [57].

6.4.2 The Undecidability of qcMSO

Towards undecidability results for qcMSO, we look at undecidability results for MSO+U, as these results entail the respective results for qcMSO. The first undecidability result for MSO+U investigated MSO+U on the infinite binary tree $\mathfrak{T}_2 = (\{0,1\}^*, \prec, L, R)$. In [7], it was proved that the MSO+U-theory of $\mathfrak{T}_2$ is undecidable under certain set-theoretical assumptions:

Theorem 6.4.5 ([7]). Assuming ZF + $(V = L)$, whether $\mathfrak{T}_2 \models \varphi$ or not for a given sentence $\varphi \in$ MSO+U is undecidable.

Only very recently, this result was subsumed by the result that MSO+U is already undecidable on the natural numbers with linear order, which can be proved without any further set-theoretical assumptions.

Theorem 6.4.6 ([8]). The MSO+U-theory of $(\omega, <)$ is undecidable.

The main idea in this proof is a reduction from the acceptance problem of 2-counter machines. In this reduction, accepting runs of 2-counter machines are encoded as MSO+U-definable languages of infinite words. The existence of an accepting run can thus be described by an MSO+U-sentence φ, and such a run exists if φ belongs to the MSO+U-theory of $(\omega, <)$.

Corollary 6.4.7. Given a qcMSO-sentence φ, it is undecidable whether $(\omega, <) \models (\varphi = \infty)$. Analogously, whether a qcMSO-sentences evaluates to ∞ on $\mathfrak{T}_2$ is undecidable.

Accordingly, as full qcMSO is undecidable on the natural numbers and thus also on the infinite binary tree while both the weak variants costWMSO and WMSO+U are decidable on $\mathfrak{T}_2$ [10, 87], we turned to the weak variant qcWMSO on the infinite binary tree. In [62], an analogous result to Corollary 6.3.9 can be found: given a qcWMSO-sentence, it is decidable whether $\mathfrak{T}_2 \models \varphi = \infty$.

6.5 Decomposition Theorems

After presenting automata-theoretical approaches above, we now turn to a different approach towards decidability results for qcWMSO. Since we already successfully used a decomposition approach for Qμ[#MSO], we investigate similar decomposition theorems here. As the structure of the natural linear order can be exploited in these theorems, we first focus on a decomposition theorem for qcMSO on $(\omega, <)$, and later present a generalized variant for the infinite binary tree.

6.5.1 Decomposition on the Natural Linear Order

In this section we present a decomposition algorithm for qcMSO on $(\omega, <)$. This algorithm will decompose a formula for $(\omega, <)$ into an equivalent formula for $(\omega \setminus \{0\}, <)$. While this does not look like a classical decomposition technique, the actual decomposition is hidden in the algorithm: As we decompose the linear order into the first element and the rest, and the first element is only a singleton structure, we can directly evaluate the respective formulas there and incorporate them into the remaining formula.

Before we present the algorithm, we consider a new normal form, called tree normal form, which imposes certain requirements on the syntax tree of the formula. Furthermore, for technical reasons we slightly change the syntax of qcMSO, considering the Boolean operators to range over sets of formulas, and adding additional summands to the counting operator. To simplify notation, we do not modify the name of the logic, but from now on assume that formulas of qcMSO are built as follows:

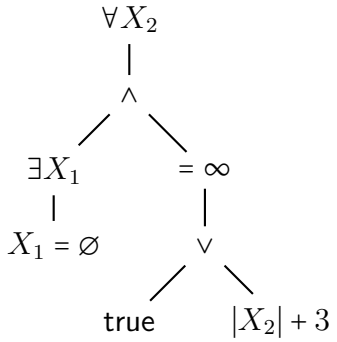
- The atoms are of the form $X < Y, X \in Y, |X| + n$ and $X = \varnothing$, where $n \in \mathbb{N}$. The semantics is as before, with the additional rule $[\![|X|+n]\!]^{\mathfrak{A},\beta} = |\beta(X)| + n$.
- Furthermore, there are two new atoms true and false, with $[\![\mathsf{true}]\!] = \infty$ and $[\![\mathsf{false}]\!] = 0$.
- If Φ is a finite set of formulas, then $\wedge\Phi$ and $\vee\Phi$ are formulas, where the semantics is the minimum, respectively maximum, of the evaluations of the formulas in Φ.
- As in the syntax introduced above, $\varphi = \infty, \varphi < \infty$ as well as $\exists X \varphi, \forall X \varphi$ are formulas, with the semantics as before.

6.5.1.1 Tree Normal Form

Before we give the definition of the normal form and prove that every formula can be transformed effectively into an equivalent formula in tree normal form, we observe some basic equivalences. First of all, comparisons with ∞ distribute over Boolean operators:

$$\begin{aligned}
\wedge\{\varphi_1, \ldots, \varphi_n\} = \infty &\equiv \wedge\{\varphi_1 = \infty, \ldots, \varphi_n = \infty\} \\
\vee\{\varphi_1, \ldots, \varphi_n\} = \infty &\equiv \vee\{\varphi_1 = \infty, \ldots, \varphi_n = \infty\} \\
\wedge\{\varphi_1, \ldots, \varphi_n\} < \infty &\equiv \vee\{\varphi_1 < \infty, \ldots, \varphi_n < \infty\} \\
\vee\{\varphi_1, \ldots, \varphi_n\} < \infty &\equiv \wedge\{\varphi_1 < \infty, \ldots, \varphi_n < \infty\}
\end{aligned}$$

$$\forall X_2 \wedge\{\exists X_1(X_1 = \varnothing),\ (\vee\{\mathsf{true}, |X_2| + 3\}) = \infty\}$$



(a) φ (b) $t(\varphi)$

Figure 6.2: An example of a syntax tree

Additionally, nested occurrences of comparisons with ∞ can be avoided:

$$\begin{aligned}
(\varphi = \infty) = \infty &\equiv \varphi = \infty \\
(\varphi < \infty) = \infty &\equiv \varphi < \infty \\
(\varphi = \infty) < \infty &\equiv \varphi < \infty \\
(\varphi < \infty) < \infty &\equiv \varphi = \infty
\end{aligned}$$

Tree normal form is defined by properties of the syntax tree of a formula. Therefore, consider the following (infinite) alphabet:

$$\begin{aligned}
\Gamma :=& \{X_i < X_j, X_i \in X_j, X_i = \varnothing, |X_i| + n \mid i, j, n \in \mathbb{N}\} && =: A \\
&\cup \{\wedge, \vee\} && =: B \\
&\cup \{\exists X_i, \forall X_i \mid i \in \mathbb{N}\} && =: C \\
&\cup \{= \infty, < \infty\} && =: D
\end{aligned}$$

The syntax tree is a labeled tree over the alphabet Γ where the children of vertices are unordered to reflect that Boolean operators range over sets. The syntax tree $t(\varphi) = (V, E, l\colon V \to \Gamma)$ of a formula φ is constructed in the usual way; for an example see Figure 6.2.

Definition 6.5.1. A formula φ is in *tree normal form* if $t(\varphi)$ satisfies the following properties:

1. If $l(v) = \vee$ and v is not the root, then $l(w) \in C$ for the predecessor w.
2. If $l(v) = \vee$, then for all w such that $(v, w) \in E$, it holds that $l(w) = \wedge$.
3. If $l(v) = \wedge$ and v is not the root, then $l(w) = \vee$ for the predecessor w.

4. If $l(v) = \wedge$, then for all w such that $(v, w) \in E$, it holds that $l(w) \in \Gamma \setminus B$.

5. If $l(v) \in B$, then the subtrees of children w of v are pairwise nonisomorphic.

6. If $l(v) = QX_i \in C$, then the subtree starting at v does not contain a vertex with $l(w) = QX_j$, $Q \in \{\exists, \forall\}$, where $j \geq i$, and for every $j < i$, it contains a vertex with $l(w) = QX_i$, $Q \in \{\exists, \forall\}$. Furthermore, the unique successor w of v is labeled with $l(w) = \vee$.

7. If $l(v) \in D$, then the unique successor w of v is labeled with $l(w) \in A \cup C$.

Thus, a formula is in tree normal form if every Boolean combination is in disjunctive normal form, the quantifiers are ordered, and comparisons with ∞ are pushed as far inside as possible according to the equivalences at the beginning of this section. Furthermore, every subformula is encapsulated in a Boolean combination. The purpose of this definition is to later define an order on the set of formulas, which allows us to compare a formula to its decomposition result.

By renaming quantified variables according to the quantifier rank of the respective subformulas, pushing comparisons with ∞ inside, and computing respective disjunctive normal forms, the following lemma about tree normal forms follows:

Lemma 6.5.2. Every formula $\varphi \in$ qcMSO can be transformed effectively into a formula in tree normal form.

If the set of variables $\{X_0, \ldots, X_n\}$ of variables used in formulas is bounded (which is equivalent to saying that the quantifier rank of the formula is bounded), and only a finite number of summands in atoms $|X| + n$ is allowed, it follows from the above properties that there are only finitely many formulas in tree normal form, as there are only finitely many nonisomorphic allowed syntax trees.

Lemma 6.5.3. Let $q, N \in \mathbb{N}$. The set of formulas in tree normal form with quantifier rank $< q$ and where all atoms $|X| + n$ satisfy $n < N$ is finite.

6.5.1.2 A Partial Order on Formulas

Using the syntax trees and the properties of tree normal forms, we define a partial order on formulas. This order is defined using embeddings on syntax trees:

Definition 6.5.4. Let φ, ψ be two formulas in tree normal form. We say that φ is smaller than ψ, $\varphi \preccurlyeq \psi$, if there exists an embedding $f : t(\varphi) \to t(\psi)$.

As a reminder, an embedding between syntax trees is an injective function f such that v and $f(v)$ receive the same label, and $(v, w) \in E$ if and only if $(f(v), f(w)) \in E$, for all $v, w \in V$. Note that the relation $\preccurlyeq$ corresponds to simulations if the syntax trees are viewed as transition systems.

Lemma 6.5.5. For fixed $q, N \in \mathbb{N}$, $\preccurlyeq$ is a partial order on formulas in tree normal form with quantifier rank $< q$ and where for all atoms $|X| + n$ it holds that $n < N$. Furthermore, every ascending chain has a maximal element.

Proof. Obviously, $\preccurlyeq$ is reflexive. It is also antisymmetric, as syntax trees are finite, thus whenever $\varphi \preccurlyeq \psi$ and $\psi \preccurlyeq \varphi$, then the syntax trees are isomorphic. Transitivity follows using standard function composition.

As the set of formulas with bounded quantifier rank and bounded summands in counting atoms is finite, it follows that every ascending chain has a maximal element. □

6.5.1.3 Decomposition

We now give a decomposition algorithm for formulas in tree normal form which has the additional property that the decomposed formula is larger with respect to the partial order defined above. The setting for the algorithm is as follows: For a formula $\varphi \in$ qcMSO which is to be evaluated on $(\omega, <)$, we compute a formula $D(\varphi)$ to be evaluated on $\omega[1] := (\omega \setminus \{0\}, <) = (\{1, 2, \dots\}, <)$ such that $[\![\varphi]\!]^{\omega} = [\![D(\varphi)]\!]^{\omega[1]}$. For technical reasons, we first give an algorithm Decompose$(\varphi, \mathcal{V})$, which takes as an additional input a subset of the free variables of φ. This set is intended to encode those variables X_i where $0 \in \beta(X_i)$. The algorithm is defined recursively, and is given in Figure 6.3.

Note that the result of the above Decompose algorithm is not necessarily a formula in tree normal form. To achieve this, and to further be able to omit the additional parameter $\mathcal{V}$, we define another algorithm, $D(\varphi)$, which only works properly for formulas in tree normal form. Thereto, note that all formulas are decomposed into a formula of the same kind, but for $X < Y$, which might become a conjunction. Thus, as a consequence of assuming tree normal form in the first place, we know that all Boolean combinations are in disjunctive normal form, and that the result of decomposing a disjunctive normal form is again a DNF.

Algorithm 6.2 ($D(\varphi)$)**.** 1. Compute Decompose$(\varphi, \varnothing)$.

Algorithm 6.1 (Decompose$(\varphi, \mathcal{V})$)**.**

$$
\begin{aligned}
\varphi = \mathsf{true} &: \textit{Return } \mathsf{true}\\
\varphi = \mathsf{false} &: \textit{Return } \mathsf{false}\\
\varphi = X < Y &: \textit{Return } \begin{cases} X < Y, & X, Y \notin \mathcal{V} \\ \wedge\{X = \varnothing, Y \in Y\}, & X \in \mathcal{V}, Y \notin \mathcal{V} \\ \mathsf{false}, & \text{otherwise} \end{cases}\\
\varphi = X \in Y &: \textit{Return } \begin{cases} X \in Y, & X, Y \notin \mathcal{V} \\ X = \varnothing, & X, Y \in \mathcal{V} \\ \mathsf{false}, & X \in \mathcal{V}, Y \notin \mathcal{V} \end{cases}\\
\varphi = X = \varnothing &: \textit{Return } \begin{cases} X = \varnothing, & X \notin \mathcal{V} \\ \mathsf{false}, & X \in \mathcal{V} \end{cases}\\
\varphi = |X| + n &: \textit{Return } \begin{cases} |X| + n, & X \notin \mathcal{V} \\ |X| + n + 1, & X \in \mathcal{V} \end{cases}\\
\varphi = \wedge\Phi &: \textit{Return } \wedge\{\text{Decompose}(\psi, \mathcal{V}) \mid \psi \in \Phi\}\\
\varphi = \vee\Phi &: \textit{Return } \vee\{\text{Decompose}(\psi, \mathcal{V}) \mid \psi \in \Phi\}\\
\varphi = \psi = \infty &: \textit{Return } \text{Decompose}(\psi, \mathcal{V}) = \infty\\
\varphi = \psi < \infty &: \textit{Return } \text{Decompose}(\psi, \mathcal{V}) < \infty\\
\varphi = \exists X\psi &: \textit{Return } \exists X \vee\{\text{Decompose}(\psi, \mathcal{V}), \text{Decompose}(\psi, \mathcal{V} \cup \{X\})\}\\
\varphi = \forall X\psi &: \textit{Return } \forall X \wedge\{\text{Decompose}(\psi, \mathcal{V}), \text{Decompose}(\psi, \mathcal{V} \cup \{X\})\}
\end{aligned}
$$

Figure 6.3: The recursive decomposition algorithm for $(\omega, <)$

2. If there is a comparison with ∞ of a conjunction, push it inside using the equivalences stated above, and then unify the conjunctions, using $\wedge(\Phi \cup \{\wedge\Psi\}) \equiv \wedge(\Phi \cup \Psi)$.

3. As existential quantifiers are followed by a disjunction of two DNFs, combine these using $\vee\{\vee\Phi, \vee\Psi\} \equiv \vee(\Phi \cup \Psi)$.

4. Universal quantifiers are followed by a conjunction of two formulas in DNF, that is, by $\wedge\{\vee\{\wedge\Phi_1, \ldots, \wedge\Phi_n\}, \vee\{\wedge\Psi_1, \ldots, \wedge\Phi_m\}\}$. This is transformed into the equivalent $\vee\{\wedge(\Phi_i \cup \Psi_j) \mid 1 \leq i \leq n, 1 \leq j \leq m\}$.

Lemma 6.5.6. Let φ be a formula in tree normal form. Then $D(\varphi)$ is in tree normal form.

Proof. Note that Decompose does not alter the quantifier structure of formulas. Furthermore, comparisons with ∞ remain in place but for the case where an atom $X < Y$ is decomposed into a conjunction. Thus, pushing inside these comparisons meets the respective requirements of tree normal form.

The only remaining constraint is that all Boolean combinations are in disjunctive normal form. But this is achieved using Steps 3 and 4. Thus, the resulting formula is in tree normal form. □

As the quantifier structure is not altered, it follows immediately that $D(\varphi)$ does not increase the quantifier rank. Furthermore, we remark that the numbers in atoms $|X| + n$ are increased by at most 1.

Theorem 6.5.7. For all formulas φ it holds that

$$[\![\varphi]\!]^{\omega[0]} = [\![D(\varphi)]\!]^{\omega[1]}.$$

(Recall that $\omega[0] = (\omega, <)$ and $\omega[i] = (\{i, i+1, \dots\}, <)$.)

Proof. The theorem follows directly from the definitions of the semantics and of the Decompose-algorithm, as $D(\varphi)$ only applies equivalence transformations to the result of Decompose, which in turn just applies the semantics. □

Remark 6.5.8. Note that the above decomposition theorem easily extends to labeled versions of $(\omega, <)$, that is, to infinite words. However, the subsequent results about the partial order may not hold anymore.

In the following we also use $D(\varphi)$ for formulas with free variables $\mathcal{V}$, and write $D(\varphi, \mathcal{V})$ for this variant where we replace Decompose$(\varphi, \varnothing)$ by Decompose$(\varphi, \mathcal{V})$ in the first line of $D(\varphi)$.

Lemma 6.5.9. Let φ be in tree normal form. Then $\varphi \preccurlyeq D(\varphi, \varnothing)$.

Proof. Towards a proof of the lemma, we first remark that for all atomic formulas ψ with variables $\mathcal{X}$, the decomposition where $\mathcal{V}$ does not contain any variable from $\mathcal{X}$ is the formula again: $\psi =$ Decompose$(\psi, \mathcal{V})$ whenever $\mathcal{X} \cap \mathcal{V} = \varnothing$, in particular for $\mathcal{V} = \varnothing$.

During the decomposition, $\mathcal{V}$ grows only in recursion steps where the current formula is of the form $QX_i\psi$. However, in these cases, Decompose is called twice, once with X_i added to $\mathcal{V}$, and once with the old $\mathcal{V}$. Accordingly, for every subformula, Decompose is called at least once with $\mathcal{V} = \varnothing$, and potentially a few times with $\mathcal{V}$ not containing a variable occurring in the formula.

The embedding f of the syntax tree of φ into that of $D(\varphi)$ can be found following the decomposition along $\mathcal{V} = \varnothing$ and the existence of f is shown by structural induction. For atomic formulas, we have $\varphi = D(\varphi, \varnothing)$, hence f maps the unique vertex to its respective copy in the other tree.

For Boolean operators and comparisons to ∞, the existence of f follows by combining the respective embeddings from the induction hypothesis.

In case $\varphi = \exists X \vee\{\wedge\Phi_1, \ldots, \wedge\Phi_n\}$, we know that the elements of the Φ_i are not Boolean combinations, and we know by the induction hypothesis that $f_i{:}\wedge\Phi_i \preccurlyeq D(\wedge\Phi_i, \varnothing)$ is a valid embedding. However, $D(\varphi, \varnothing) = \exists X \vee\{D(\wedge\Phi_1, \varnothing), \ldots D(\wedge\Phi_n, \varnothing), D(\wedge\Phi_1, \{X\}), \ldots, D(\wedge\Phi_n, \{X\})\}$. Thus, an embedding can be found by combining the mappings f_i to the respective subtrees for $D(\wedge\Phi_i, \varnothing)$, and adding the respective targets for $\exists X$ and $\vee$.

If $\varphi = \forall X \vee\{\wedge\Phi_1, \ldots, \wedge\Phi_n\}$, let f_i be again the respective embedding $\wedge\Phi_i \preccurlyeq D(\wedge\Phi_i, \varnothing)$ from the induction hypothesis. We write $D(\Phi_i, \varnothing)$ for the set of decompositions with $\varnothing$ of elements in Φ_i. It holds that

$$D(\varphi, \varnothing) = \forall X \vee\{\wedge(D(\Phi_i, \varnothing) \cup D(\Phi_j, \{X\})) \mid 1 \leq i, j \leq n\}.$$

When decomposing the f_i into a set of mappings f_i^l such that f_i^l maps the l-th element φ_i^l of $\Phi_i = \{\varphi_i^1, \ldots, \varphi_i^{n_i}\}$ to the respective subtree, we can obtain an embedding of φ into $D(\varphi, \varnothing)$ by choosing a fixed j for every i, say $j = i$, such that f_i^l is used to obtain a mapping from the l-th element φ_i^l of Φ_i to the respective subtree in $D(\Phi_i, \varnothing) \cup D(\Phi_i, \{X\})$, and by adding the respective mappings between the outer quantifier, disjunction and conjunction. □

Corollary 6.5.10. Let φ be a sentence in tree normal form. Then, $\varphi \preccurlyeq D(\varphi)$.

Sequential decomposition As $\omega[1]$ is isomorphic to $\omega[0]$, it follows that the decomposition algorithm can be applied again to $D(\varphi)$ to achieve a formula $D(D(\varphi))$ to be evaluated on $\omega[2]$. In principle, this can be continued ad infinitum. However, even though the quantifier rank does not change, the formulas may keep on growing in size, as the number of atoms is infinite: As the numbers in atoms $|X| + n$ are potentially increased in every application of the decomposition algorithm, the procedure produces wider and wider formulas.

We thus consider a variant of the algorithm $D(\varphi)$ where we index the algorithm with a natural number $N \in \mathbb{N}$: $D_N(\varphi)$ works as $D(\varphi)$, only that after Step 1, every atom $|X| + M$ where $M > N$ is replaced by $|X| + N$. Because also the depth of the syntax tree of a formula in tree normal form is not increased during an application of the D-algorithm, it follows from the usage of Boolean operators only over sets that there are only finitely many formulas obtainable from a formula φ (in tree normal form) by sequential application of D_N. Furthermore, for formulas φ such that N exceeds the maximal number occurring in atoms $|X| + n$ in φ, the semantics of D_N and the order on formulas are analogous to the general decomposition algorithm introduced above.

Lemma 6.5.11. Let φ be in tree normal form, and let $M = \max\{m \mid |X| + m$ appears in $\varphi\}$. For all $N \in \mathbb{N}$, the following hold:

(a) $[\![D_N(\varphi)]\!]^{\omega[1]} \leq [\![\varphi]\!]^{\omega} \leq [\![D_N(\varphi)]\!]^{\omega[1]} + \begin{cases} 0, & N \geq M+1 \\ M+1-N, & N \leq M+1. \end{cases}$

(b) $[\![D^i_N(\varphi)]\!]^{\omega[1]} \leq [\![\varphi]\!]^{\omega} \leq [\![D^i_N(\varphi)]\!]^{\omega[1]} + \begin{cases} 0, & N \geq M+1+i \\ M+1+i-N, & N \leq M+1+i. \end{cases}$

Lemma 6.5.12. Let φ be in tree normal form, and let $M = \max\{m \mid |X| + n$ appears in $\varphi\}$. For all $N \geq M$, $\varphi \preccurlyeq D_N(\varphi)$.

The former lemma follows from the fact that the respective summand adds again what has potentially been subtracted by restricting the numbers in atoms $|X| + n$. The latter lemma can be proved analogously to the case for the unrestricted decomposition, as decomposition with $\mathcal{V} = \varnothing$ does not increase the numbers.

As the order on formulas is preserved and only finitely many formulas are reachable, the following corollary about the existence of a fixed point follows.

Corollary 6.5.13. Let φ be in tree normal form and let N be larger than the maximal number occurring in atoms $|X|+n$. Then, the sequence $(D^i_N(\varphi))_{i\in\omega}$ reaches a fixed point $D^{\infty}_N(\varphi)$.

6.5.1.4 Towards a Decomposition-Based Algorithm for Unboundedness of qcWMSO

In the following we give a partial algorithm towards deciding the boundedness question for qcWMSO on $(\omega, <)$ via decomposition techniques. The main idea is to define a recursive algorithm, which for quantifier subformulas applies decomposition sequentially in order to, at some point, eliminate the quantifier. To this end, we first of all extend the domain of numbers in counting atoms to include ∞, that is, also allow atoms $|X| + \infty$. We then introduce an operation $[X/\varnothing]$ on formulas, which intuitively replaces every occurrence of the set variable X with the empty set. As this violates the syntax, we directly use the semantics to give equivalents with respect to boundedness. The respective equivalences are given below.

$$\varnothing < Y \equiv \mathsf{false} \qquad \varnothing \in Y \equiv \mathsf{false}$$

$$\varnothing = \varnothing \equiv \mathsf{true} \qquad |\varnothing| + n \equiv \begin{cases} \mathsf{true}, & n = \infty \\ \mathsf{false}, & n < \infty \end{cases}$$

Accordingly, $\varphi[X/\varnothing]$ is obtained from φ by replacing every atomic formula containing X by its respective equivalent, and by omitting the quantifier

binding X, as no atom with X appears anymore. (Note that this means that potentially the Boolean combination around the quantifier needs to be transformed into DNF again, if the quantifier is not the outermost symbol in the formula).

The partial unboundedness algorithm $\mathcal{U}$ now takes a sentence, and works recursively along the structure of the sentence. It returns true if the formula evaluates to ∞, and false otherwise. The different cases are given below. Note that the algorithm works in such a way that it ensures that on every recursive call there are no free variables in the formula.

Atomic formulas The only atomic sentences are true and false. Since $[\![\mathsf{true}]\!] = \infty$, and $[\![\mathsf{false}]\!] = 0$, the algorithm is obvious.

Boolean combinations As conjunctions and disjunctions correspond to minima and maxima over finite sets, the algorithm is called recursively on the conjuncts and disjuncts, and the results are combined:

$$\mathcal{U}(\vee\{\varphi_1,\dots,\varphi_n\}) = \begin{cases} \mathsf{true}, & \mathcal{U}(\varphi_i) = \mathsf{true} \text{ for some } i \\ \mathsf{false}, & \text{otherwise} \end{cases}$$

$$\mathcal{U}(\wedge\{\varphi_1,\dots,\varphi_n\}) = \begin{cases} \mathsf{true}, & \mathcal{U}(\varphi_i) = \mathsf{true} \text{ for all } i \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

Comparisons with ∞ These comparisons can easily be obtained by applying the algorithm to the respective subformula that is to be compared:

$$\mathcal{U}(\varphi = \infty) = \mathcal{U}(\varphi)$$
$$\mathcal{U}(\varphi < \infty) = \neg\mathcal{U}(\varphi).$$

Universal quantification To eliminate universal quantifiers, we first observe that the infimum over all finite sets of the value of a formula is actually a minimum over all finite sets. This means that whenever $[\![\forall X\varphi]\!] < \infty$, then there is a finite witness $X^* \subseteq \omega$ such that $[\![\varphi(X^*)]\!] < \infty$. As such a witness is necessarily contained in an initial subset $\{0,\dots,n\}$ of ω, we can also compute the minimum (or infimum) over all subsets of the first $n+1$ numbers, instead of ω. This, however, corresponds to decomposing $n+1$ times, and assuming the set to be empty afterwards. As any atom $|\varnothing| + n$ where $n \in \mathbb{N}$ can never evaluate to ∞, the actual numbers are irrelevant. We claim that we can simplify the computation by using the fixed point.

Lemma 6.5.14. Let $\varphi = \forall X\psi$ be in tree normal form. $[\![\varphi]\!] < \infty$ if and only if $[\![D_0^\infty(\varphi)[X/\varnothing]]\!] < \infty$.

Proof. Assume that $[\![\varphi]\!] < \infty$. Accordingly, there exists a finite set X^* such that $[\![\psi(X^*)]\!] = [\![\forall X\psi]\!] < \infty$, and $X^* \subseteq \{0,\dots,n\}$ for some n. It follows from Theorem 6.5.7, that

$$[\![D^{n+1}(\forall X\psi)]\!] = [\![D^{n+1}(\forall X\psi)[X/\varnothing]]\!] = [\![D^{m}(\forall X\psi)[X/\varnothing]]\!] = [\![\forall X\psi]\!],$$

for all $m \geq n+1$. Let j be such that $D_0^\infty(\forall X\psi) = D_0^j(\forall X\psi)$. By Lemma 6.5.11, $[\![D_0^\infty(\forall X\psi)]\!] \leq [\![D^j(\forall X\psi)]\!]$, and the same holds for $D_0^\infty(\forall X\psi)[X/\varnothing]$ and $D^j(\forall X\psi)[X/\varnothing]$.

Assume on the other hand that $[\![\varphi]\!] = \infty$. Accordingly, for every finite set X^* it holds that $[\![\psi(X^*)]\!] = \infty$. As $[\![D_0^\infty(\varphi)]\!] \leq [\![D^j(\varphi)]\!] = [\![\varphi]\!] \leq [\![D_0^\infty(\varphi)]\!] + j + 1$, it follows that $[\![D_0^\infty(\varphi)[X/\varnothing]]\!] = \infty$. □

Thus, $\mathcal{U}(\forall X\varphi) = \mathcal{U}(D_0^\infty(\varphi)[X/\varnothing])$.

Existential quantification Unfortunately, so far we have not found a solution for eliminating existential quantifiers. The first issue to be overcome in order to use sequential decomposition appears to be finding the right N for D_N. We believe, although, that when considering the sequence $(D_i^\infty(\exists X\varphi)[X/\varnothing])_{i\in\mathbb{N}}$, structural properties can be discovered that allow one to find a single formula that stores the relevant information of the sequence. However, the major problem still remains: When comparing with ∞, a single atom of the form $|X| + n$ is always less than ∞, while a proper supremum of such terms can evaluate to ∞. Thus, one cannot rewrite the atom to $|X| + \infty$ without losing information, respectively introducing unwanted side effects.

6.5.2 Decomposition on the Infinite Binary Tree

In the following, we present an analogous decomposition algorithm for the infinite binary tree $\mathfrak{T}_2 = (\{0,1\}^*, \prec, L, R)$, where $\prec$ denotes the prefix relation, $L = \{0,1\}^*0$ holds exactly at the left successors and R is analogous for right successors. Here, we decompose the tree into the one-vertex structure of the root and the two subtrees. Accordingly, the algorithm will be more complicated, as it will return formulas to be checked in the left subtree, and formulas for the right subtree. To still give a simple algorithm, we use a different normal form here, which ensures that the algorithm yields a Boolean combination of formulas for the respective subtrees.

6.5.2.1 Type Normal Form

The first step towards the desired normal form is to give an algorithm that computes—given a formula—an equivalent formula in *disjunctive normal form.*

Here, DNF means that the outermost Boolean combination is in disjunctive normal form, and that comparisons with ∞ are pushed inside.

Algorithm 6.3 (DNF(φ))**.**

$$
\begin{aligned}
\varphi = \vee\{\varphi_1, \dots, \varphi_n\} &: \textit{Return } \vee\{\wedge\Phi_i^j \mid 1 \le i \le n, 1 \le j \le n_i\},\\
&\qquad \text{where DNF}(\varphi_i) = \vee\{\wedge\Phi_i^1, \dots, \wedge\Phi_i^{n_i}\}\\
\Phi = \wedge\{\varphi_1, \dots, \varphi_n\} &: \textit{Return } \vee\{\wedge(\Phi_1^{j_1} \cup \dots \cup \Phi_n^{j_n}) \mid j_i \in \{1, \dots, n_i\}\},\\
&\qquad \text{where DNF}(\varphi_i) = \vee\{\wedge\Phi_i^1, \dots, \wedge\Phi_i^{n_i}\}\\
\varphi = \psi = \infty &: \textit{Return } \vee\{\wedge\{\vartheta = \infty \mid \vartheta \in \Psi_i\} \mid 1 \le i \le n\},\\
&\qquad \text{where DNF}(\psi) = \vee\{\wedge\Psi_1, \dots, \wedge\Psi_n\}\\
\varphi = \psi < \infty &: \textit{Return } \text{DNF}(\wedge\{\vee\{\vartheta < \infty \mid \vartheta \in \Psi_i\} \mid 1 \le i \le n\}),\\
&\qquad \text{where DNF}(\psi) = \vee\{\wedge\Psi_1, \dots, \wedge\Psi_n\}\\
\text{otherwise} &: \textit{Return } \vee\{\wedge\{\varphi\}\}
\end{aligned}
$$

Note that we can define an algorithm CNF for conjunctive normal form in a similar way.

In the following, we extend the results and definitions for type normal form from [49, 50] to qcMSO. Informally, we say that a formula φ is in *type normal form* (TNF) if it is a disjunctive normal form over *normal* formulas. A formula is normal, if it is atomic, is of the form $\varphi = \infty$ or $\varphi < \infty$ such that φ is normal, or is of the form $QX_i\varphi$, $Q \in \{\exists, \forall\}$, with $\varphi = \vee\{\wedge\Phi_1, \dots, \Phi_n\}$ in disjunctive normal form such that $i = \mathrm{qr}(\varphi)$, X_i appears free in ψ for every $\psi \in \Phi_j$ for all j, and each Φ_j contains only normal formulas. Note that this entails that $QX\varphi(X)$ and $QY\varphi(Y)$ have the same type normal form.

Formally, a formula is in type normal form if $\varphi = \text{TNF}(\varphi)$, for the algorithm given in Figure 6.4. Note that the algorithm does not increase the quantifier rank, and that all transformations made are equivalence preserving. Thus, it follows that $\varphi \equiv \text{TNF}(\varphi)$.

Type normal form with disjoint variable sets Towards the decomposition into formulas for two subtrees, we introduce disjoint variable sets, that is, we label every variable X by either L or R, thus write X^L or X^R. The algorithm for TNF is updated in such a way that the labels are preserved, and we assume that in all atomic formulas either all variables are labeled with L or all are labeled with R. Type normal form guarantees that every formula with variables from disjoint variable sets which do not occur together in atoms is normalized into a Boolean combination of formulas with only L- and formulas with only R-labeled variables.

Lemma 6.5.15. Let φ be a formula such that every variable is labeled with L or with R, and only variables with the same label occur together in atoms. Then, $\text{TNF}(\varphi) = \vee\{\wedge\Phi_1, \dots, \wedge\Phi_n\}$ is a disjunctive normal form such that for

Algorithm 6.4 (TNF(φ)).

$$
\begin{aligned}
\varphi \text{ is atomic} &: \textit{Return } \mathrm{DNF}(\varphi)\\
\varphi = \vee\{\varphi_1,\dots,\varphi_n\} &: \textit{Return } \mathrm{DNF}(\vee\{\mathrm{TNF}(\varphi_1),\dots,\mathrm{TNF}(\varphi_n)\})\\
\varphi = \wedge\{\varphi_1,\dots,\varphi_n\} &: \textit{Return } \mathrm{DNF}(\wedge\{\mathrm{TNF}(\varphi_1),\dots,\mathrm{TNF}(\varphi_n)\})\\
\varphi = \psi = \infty &: \textit{Return } \mathrm{DNF}(\mathrm{TNF}(\psi) = \infty)\\
\varphi = \psi < \infty &: \textit{Return } \mathrm{DNF}(\mathrm{TNF}(\psi) < \infty)\\
\varphi = \exists X\psi &: \textit{Return } \vee\{\wedge(\{\psi_i^j \mid j \notin J_i\}\\
&\qquad \cup \{\exists X_{q_i} \vee\{\wedge\{\psi_i^j[X/X_{q_i}] \mid j \in J_i\}\}\})\\
&\qquad \mid 1 \le i \le n\}\\
&\quad \text{where } \mathrm{TNF}(\psi) = \vee\{\wedge\Phi_1,\dots,\wedge\Phi_n\},\\
&\quad \Phi_i = \{\psi_i^j \mid 1 \le j \le n_i\},\\
&\quad J_i = \{j \mid X \in \mathrm{free}(\psi_i^j)\},\\
&\quad q_i = \max\{\mathrm{qr}(\psi_i^j) \mid j \in J_i\}.\\
\varphi = \forall X\psi &: \textit{Return } \mathrm{DNF}(\wedge\{\vee(\{\psi_i^j \mid j \notin J_i\}\\
&\qquad \cup \{\forall X_{q_i} \vee\{\wedge\{\psi_i^j[X/X_{q_i}]\} \mid j \in J_i\}\})\\
&\qquad \mid 1 \le i \le n\})\\
&\quad \text{where } \mathrm{CNF}(\mathrm{TNF}(\psi)) = \wedge\{\vee\Phi_1,\dots,\wedge\Phi_n\},\\
&\quad \Phi_i = \{\psi_i^j \mid 1 \le j \le n_i\},\\
&\quad J_i = \{j \mid X \in \mathrm{free}(\psi_i^j)\},\\
&\quad q_i = \max\{\mathrm{qr}(\psi_i^j) \mid j \in J_i\}.
\end{aligned}
$$

Figure 6.4: The tree normal form algorithm

every $\psi \in \bigcup_{i=1}^{n} \Phi_n$, either all variables are labeled with L or all variables are labeled with R.

Proof. Assume towards a contradiction that there are formulas in TNF which do not satisfy the above condition. Let φ be such a formula of smallest size. Obviously, φ is of the form $\vee\{\wedge\{\vartheta\}\}$, as the condition is only on single formulas in the conjunctions. By definition of TNF, ϑ is not a Boolean combination. As ϑ was chosen minimal in size, all subformulas of ϑ satisfy the condition of the lemma, and furthermore, they are, by the recursive nature of the algorithm, also in TNF.

Clearly, ϑ cannot be an atom, as no differently labeled variables appear together in atoms. Hence, it is either of the form $\exists X_q^l\psi$, $\forall X_q^l\psi$, $\psi = \infty$ or $\psi < \infty$. If it is of the latter two forms, then $\vee\{\wedge\{\psi\}\}$ is a shorter counterexample, so $\vartheta = Q_q^l X \vee\{\wedge\Psi_1,\dots,\wedge\Psi_n\}$. Each Ψ_i contains only formulas with all variables labeled either with L or with R, as otherwise there would be a smaller counterexample. But X_q^l appears free in every $\psi_j \in \Psi_i$, hence all Ψ_i contain only variables labeled with l, and so does ϑ. □

6.5.2.2 The Decomposition Algorithm

With the above results on type normal form, we are ready to give the actual decomposition algorithm. This algorithm will decompose a formula φ for the infinite binary tree into a Boolean combination of formulas in which all variables are labeled with either L or with R. The intuition here is that the formulas are evaluated on the left, respectively the right, subtree, and the evaluations are then combined according to the Boolean combination. Let thus λ be a mapping that, given a formula where all variables are labeled with the same label, returns the label of the variables. Using this, we define a new semantics, $[\![\cdot]\!]_\alpha$, $\alpha \in \{1, 3\}$, for formulas in type normal form.

If all variables in a formula are labeled with the same label, the semantics is defined just as the normal semantics, but for $|X| + n$, where we set $[\![|X| + n]\!]_\alpha^{\mathfrak{A},\beta} = \alpha^n(|\beta(X)|+n)$. If there are variables that are labeled differently, then, without loss of generality, the formula is of the form $\varphi = \vee\{\wedge\Phi_1, \ldots, \wedge\Phi_n\}$, where in each $\psi \in \bigcup_{i=1}^n \Phi_i$ all variables are labeled with the same label. In this case, we set

$$[\![\varphi]\!]_\alpha = \max_i \left(\min\{[\![\psi]\!]_\alpha^{t_{\lambda(\psi)}} \mid \psi \in \Phi_i\}\right),$$

where t_L is the left subtree of the root of $\mathfrak{T}_2$, and t_R is the right subtree.

The purpose of this new semantics is to add an additional weight factor for atoms $|X| + n$: Whenever such an atom is evaluated in a subtree, it should also incorporate the sizes of X in the root and the other subtree. This will be achieved by taking the maximum of the sizes of X in both subtrees, and by multiplying it by 3.

We define the decomposition algorithm Decompose$(\cdot)$ on the infinite binary tree, for formulas in type normal form, as

$$\text{Decompose}(\varphi) := \text{TNF}(\text{split}(\varphi, \varnothing)),$$

where the algorithm split$(\cdot,\cdot)$ is given in Figure 6.5. Note that due to the type normal form algorithm, it is guaranteed that variables which are labeled differently do not appear together in subformulas below the outermost Boolean combination. Furthermore, we omitted the labels of variables in the case distinction in the algorithm for better readability. However, if we introduce X^L for a labeled variable, we drop the old label and replace it by L, and accordingly for R. Thus, upon splitting a quantified variable, we double the variable, and have one variable for the label L and one for the label R.

Theorem 6.5.16. Let φ be a formula in type normal form. Then,

$$[\![\text{Decompose}(\varphi)]\!]_1 \leq [\![\varphi]\!]^{\mathfrak{T}_2} \leq [\![\text{Decompose}(\varphi)]\!]_3.$$

Algorithm 6.5. $\mathrm{split}(\varphi, \mathcal{V})$

$\varphi = X \in Y$:	if $X, Y \in \mathcal{V}$ *Return* $\wedge\{X^L = \varnothing, X^R = \varnothing\}$
	if $X \in \mathcal{V}, Y \notin \mathcal{V}$ *Return* false
	if $X \notin \mathcal{V}$ *Return* $\vee\{\wedge\{X^L \in Y^L, X^R = \varnothing\}, \wedge\{X^L = \varnothing, X^R \in Y^R\}\}$
$\varphi = LX$:	if $X \in \mathcal{V}, \lambda(X) = L$ *Return* $\wedge\{X^L = \varnothing, X^R = \varnothing\}$
	if $X \in \mathcal{V}, \lambda(X) = R$ *Return* false
	if $X \notin \mathcal{V}$ *Return* $\vee\{\wedge\{LX^L, X^R = \varnothing\}, \wedge\{X^L = \varnothing, LX^R\}\}$
$\varphi = RX$:	analogously to LX
$\varphi = X \prec Y$:	if $X \in \mathcal{V}, Y \notin \mathcal{V}$ *Return* $\vee\{\wedge\{X^L = \varnothing, Y^L \in Y^L, X^R = \varnothing, Y^R = \varnothing\}, \wedge\{X^L = \varnothing, Y^L = \varnothing, X^R = \varnothing, Y^R \in Y^R\}\}$
	if $X, Y \notin \mathcal{V}$ *Return* $\vee\{\wedge\{X^L < Y^L, X^R = \varnothing, Y^R = \varnothing\}, \wedge\{X^L = \varnothing, Y^L = \varnothing, X^R < Y^R\}\}$
	if $Y \in \mathcal{V}$ *Return* false
$\varphi = X = \varnothing$:	if $X \in \mathcal{V}$ *Return* false
	if $X \notin \mathcal{V}$ *Return* $\wedge\{X^L = \varnothing, X^R = \varnothing\}$
$\varphi = \lvert X\rvert + n$:	if $X \in \mathcal{V}$ *Return* $\vee\{\lvert X^L\rvert + n + 1, \lvert X^R\rvert + n + 1\}$
	if $X \notin \mathcal{V}$ *Return* $\vee\{\wedge\{\lvert X^L\rvert + n + 1, X^R = \varnothing < \infty\}, \wedge\{\lvert X^L\rvert + n, X^R = \varnothing\}, \wedge\{X^L = \varnothing < \infty, \lvert X^R\rvert + n + 1\}, \wedge\{X^L = \varnothing, \lvert X^R\rvert + n\}\}$
$\varphi = \vee\{\varphi_1, \ldots, \varphi_n\}$:	*Return* $\vee\{\mathrm{split}(\varphi_1, \mathcal{V}), \ldots, \mathrm{split}(\varphi_n, \mathcal{V})\}$
$\varphi = \wedge\{\varphi_1, \ldots, \varphi_n\}$:	*Return* $\wedge\{\mathrm{split}(\varphi_1, \mathcal{V}), \ldots, \mathrm{split}(\varphi_n, \mathcal{V})\}$
$\varphi = \psi = \infty$:	*Return* $\mathrm{split}(\psi, \mathcal{V}) = \infty$
$\varphi = \psi < \infty$:	*Return* $\mathrm{split}(\psi, \mathcal{V}) < \infty$
$\varphi = \exists X \psi$:	*Return* $\exists X^L \exists X^R \vee\{\mathrm{split}(\psi, \mathcal{V}), \mathrm{split}(\psi, \mathcal{V} \cup \{X\})\}$
$\varphi = \forall X \psi$:	*Return* $\forall X^L \forall X^R \wedge\{\mathrm{split}(\psi, \mathcal{V}), \mathrm{split}(\psi, \mathcal{V} \cup \{X\})\}$

Figure 6.5: The split-procedure

Proof. To prove the theorem, we have to prove correctness for the split-procedure, as applying TNF$(\cdot)$ is equivalence preserving.

For the base cases but $|X| + n$, this follows directly from the definition of the semantics. When evaluating a term $|X| + n$ on the tree, the algorithm incorporates whether the root belongs to X by adding 1 to n. As qcMSO does not allow addition of numbers, we replace the sum of $|X^L|$ and $|X^R|$ by its maximum (thus the disjunction), and add 1 to the maximum if X is nonempty in the other subtree. In the 1-semantics, this evaluates to simply the maximum, potentially plus 1, which is lower than its original value. In the 3-semantics, it evaluates to three times the maximum, which is larger than the sum of $|X^L|$ and $|X^R|$ and 1.

For Boolean operators and comparisons with ∞, correctness follows from the induction hypothesis. For existential quantifiers, we consider the supremum over all sets in the respective subtrees, and consider this for both the root belonging to the set, and not belonging to the set, and analogously for universal quantification. □

Remark 6.5.17. Note that—even though it appears differently—Decompose does not increase the quantifier rank: In fact, the nested doubling of quantifiers is removed along the TNF-transformation.

Because of the missing ability to take sums of atoms $|X| + n$, it is not possible to decompose in such a way that exact values are preserved when considering the binary tree—in contrast to the case of $(\omega, <)$. However, sequential decomposition can still be achieved, due to the definition of the α-semantics with a factor of α^n at counting atoms. Unfortunately, we have not found results complementing the fixed point results for the linear order, thus when trying to obtain analogous algorithms using decomposition, even the universal quantification case is more involved. Still, as there are only finitely many formulas which may result from applying decomposition sequentially while restricting additional summands in counting terms, any decomposition sequence runs into a cycle, which may be used to obtain an analogue of the fixed point $D^\infty(\varphi)$.

6.6 Summary

In this chapter, we introduced a quantitative extension of MSO where the emphasis is on a counting atom. The semantics of this extension is adapted from the classical qualitative semantics. We proved that the logic extends MSO+U and costMSO (using the inductive semantics), and that boundedness and dominance problems for costMSO can be expressed in qcMSO.

We presented an extension of a resource variant of first-order logic, FO+RR, and showed that the model-checking problem for this logic is decidable on resource-automatic structures. Using a reduction to such a model checking problem, we proved that the qcWMSO-theory of $(\omega, <)$ is decidable.

At last, we presented decomposition algorithms for both the natural numbers with order and the infinite binary tree, using respective normal forms for qcMSO.

Chapter 7

Summary and Future Work

In this thesis, we presented results on two models of infinite games with counters and two approaches to logics with counting. We began with counter parity games, where a (quantitative) parity condition is combined with a condition on a finite set of counters. In these games, the counter condition is only used for the payoffs of finite plays. In mean counter games, which we studied subsequently, this is no longer the case, and counters are not only updated, but may also be checked. The payoff is then based on the checked counter values, using a mean-payoff condition.

The first logic we studied, $Q\mu[\#\mathrm{MSO}]$, is based on the quantitative μ-calculus, but replaces quantitative predicates by counting terms of monadic second-order logic. As a variant of the μ-calculus, model-checking this logic amounts to solving a quantitative parity game, and we reduced the model-checking problem on a generalization of pushdown systems to solving counter parity games. We then turned to monadic second-order logic, and presented a purely quantitative extension, where again quantitative evaluations are introduced via counting. In qcMSO, we do not use counting terms, instead an operator to count the size of a set is added.

In Chapter 3, we proved that finite counter parity games can be solved effectively and gave a 6Exptime-algorithm. To do so, we abstracted from affine counter update functions to marks of these functions, and solved the unboundedness problem of the value using second-life games with imperfect information and imperfect recall. We proved that finite-memory strategies suffice for the player lacking perfect recall in second-life games with ω-regular winning conditions, and that a bound on the size of that memory can be computed. We applied this to study the plays which are good for the minimizing player in counter parity games. After using the duality of the game, we thus obtained lower and upper bounds for the value, so that computing the actual value amounts to solving a finite quantitative parity game.

After observing some general facts about memory requirement and the existence of optimal strategies for mean counter games in the beginning of Chapter 4, we turned to studying the one-counter case. We proved for both Minimizer- and Maximizer-solitary games that the unboundedness problem can be solved, and that the actual value can be computed using reductions to finite mean-payoff games. While the reduction is rather straightforward for Minimizer, it involved a study of so-called patterns for Maximizer, and a more complex reduction procedure, where multiple mean-payoff games have to be solved. We also proved that boundedness is still decidable in the two-player one-counter case, and gave some interesting examples for affine mean-counter games.

Chapter 5 began with an introduction to structure transition systems, where states are labeled with finite relational structures. We formulated the syntax and semantics of Qμ[#MSO], and turned to the model-checking problem on tree-producing pushdown systems. In this generalization of pushdown systems, the stack is used to store increasing tree-rewriting rules, and the finite relational structures in the induced structure transition systems are obtained as the results of the iterative applications of the rules on the stack. We proved, using pop-elimination, decomposition and a reduction to counter parity games, that the model-checking problem is decidable.

In the last Chapter 6, we reused the idea of generalizing the semantics of Lμ to that of Qμ, but considered MSO this time. We gave a definition of the counting logic qcMSO and investigated the connection between this logic and two other quantitative extensions of MSO, namely costMSO and MSO+U. We considered the problem of deciding the qcWMSO-theory of $(\omega, <)$, and reduced it to model-checking a new extension FO+RR$^{=\infty}$ of first-order logic with resource relations, FO+RR, on resource-automatic structures, which we proved decidable. After a discussion of the expressive power and decidability of full qcMSO on $(\omega, <)$ and the infinite binary tree, we turned to decomposition algorithms, which we gave both for the tree and the natural numbers.

7.1 Future Work

In the following, we mention some areas of further research, focusing first on continuations of the work presented here. We then name some more general research questions.

Regarding mean counter games, it remains open whether the exact value of a two-player one-counter game can be computed. Although we conjecture this to be possible, so far an algorithm to do so is missing. Furthermore, the decidability problems for the multi-counter case have not been answered,

for solitary as well as two-player games. Again, we conjecture that (partial) solutions can be found by generalizing the techniques used in the one-counter cases. It would also be interesting to further study affine mean counter games, as the format of these games is closer to counter parity games.

Turning to more general questions, there are still many interesting fields of research regarding counter games. While several winning conditions that are based on counting of some sort have been studied in recent years, the connection between the different variants is often unknown. It would be interesting to study counter games in a systematic way, beginning with winning conditions based purely on counters, and later extending the study to combinations with other winning conditions. This could also help to shed light on the power of different counter update functions, for example whether games are more difficult if affine counter updates are used compared to pure increments and resets.

Regarding the counting logics discussed in this thesis, we presented a variant of the μ-calculus and an extension of monadic second-order logic. Although these are similar, the focus of the implementations presented here is different. Still, the question remains what the connection between matching variants of $\mathrm{Q}\mu$ and qcMSO is, especially in view of (quantitative) bisimulation.

List of Figures

3.1 An example of a counter parity game 31
3.2 Over-approximations are necessary for boundedness 33
3.3 Parts of the mark for the function from Example 3.2.10 37
3.4 An illustration of Part (ii) of the winning condition of $\mathcal{U}(G^m)$ . . 51
3.5 Possible locations of m_{id} in suffixes 55

4.1 An example of a mean counter game 60
4.2 Maximizer does not have an optimal strategy 61
4.3 Maximizer needs infinite memory despite the value being finite . 62
4.4 Minimizer needs finite memory in one-counter games 69
4.5 An aMCG where the memory depends on the number of counters 88
4.6 An aMCG where the required memory depends on the arena . . 89

5.1 An example of a fixed point computation for $\mu X.\#_x(Ax) \vee \Diamond X$ 97
5.2 An example of an application of an ITRR 98
5.3 An example of a TPPDS and a tree resulting from a run 99
5.4 Transitions of $\mathfrak{P}'$ 101
5.5 Illustration of the claim construction 102

6.1 Semantics for qcMSO 116
6.2 An example of a syntax tree 132
6.3 The recursive decomposition algorithm for $(\omega, <)$ 135
6.4 The tree normal form algorithm 142
6.5 The split-procedure 144

Bibliography

[1] V. Bárány, E. Grädel, and S. Rubin. Automata-Based Presentations of Infinite Structures. In J. Esparza, C. Michaux, and C. Steinhorn, editors, *Finite and Algorithmic Model Theory*, volume 379 of *London Mathematical Society Lecture Notes*, pages 1–76. Cambridge University Press, Cambridge, 2011.

[2] D. Berwanger, Ł. Kaiser, and S. Leßenich. Imperfect Recall and Counter Games. Research Report LSV-11-20, Laboratoire Spécification et Vérification, ENS Cachan, France, 2011.

[3] D. Berwanger, Ł. Kaiser, and S. Leßenich. Solving counter parity games. In B. Rovan, V. Sassone, and P. Widmayer, editors, *Mathematical Foundations of Computer Science 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 160–171. Springer Berlin Heidelberg, 2012.

[4] M. Bojańczyk. A bounding quantifier. In J. Marcinkowski and A. Tarlecki, editors, *Computer Science Logic*, volume 3210 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin Heidelberg, 2004.

[5] M. Bojańczyk. Weak MSO with the unbounding quantifier. *Theory of Computing Systems*, 48(3):554–576, 2011.

[6] M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science, LICS 2006*, pages 285–296, 2006.

[7] M. Bojańczyk, T. Gogacz, H. Michalewski, and M. Skrzypczak. On the decidability of MSO+U on infinite trees. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Automata, Languages, and Programming*, volume 8573 of *Lecture Notes in Computer Science*, pages 50–61. Springer Berlin Heidelberg, 2014.

[8] M. Bojańczyk, P. Parys, and S. Toruńczyk. The MSO+U theory of $(\mathbb{N}, <)$ is undecidable. *arXiv:1502.04578 [cs.LO]*, 2015.

[9] M. Bojańczyk and S. Toruńczyk. Deterministic Automata and Extensions of Weak MSO. In R. Kannan and K. N. Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 73–84, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[10] M. Bojańczyk and S. Toruńczyk. Weak MSO+U over infinite trees. In C. Dürr and T. Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 648–660, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[11] A.-J. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with unboundedness and regular conditions. In P. K. Pandya and J. Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Science*, pages 88–99. Springer Berlin Heidelberg, 2003.

[12] P. Bouyer, T. Brihaye, M. Jurdziński, R. Lazić, and M. Rutkowski. Average-price and reachability-price games on hybrid automata with strong resets. In F. Cassez and C. Jard, editors, *Formal Modeling and Analysis of Timed Systems*, volume 5215 of *Lecture Notes in Computer Science*, pages 63–77. Springer Berlin Heidelberg, 2008.

[13] J. Bradfield and C. Stirling. Modal mu-calculi. In P. Blackburn, J. Van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 721–756. Elsevier, 2007.

[14] T. Brázdil, K. Chatterjee, A. Kučera, and P. Novotný. Efficient controller synthesis for consumption games with multiple resource types. In P. Madhusudan and S. Seshia, editors, *Computer Aided Verification*, volume 7358 of *Lecture Notes in Computer Science*, pages 23–38. Springer Berlin Heidelberg, 2012.

[15] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[16] J. R. Büchi. On a decision method in restricted second order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Logic, Methodology and Philosophy of Science: Proceedings of the 1960 International Congress*. Stanford University Press, Stanford, California, 1962.

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

[18] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In R. Alur and I. Lee, editors, *Embedded Software*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer Berlin Heidelberg, 2003.

[19] K. Chatterjee and L. Doyen. Energy parity games. *Theoretical Computer Science*, 458:49–60, 2012.

[20] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games with imperfect information. In Z. Ésik, editor, *Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 287–302. Springer Berlin Heidelberg, 2006.

[21] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Generalized Mean-payoff and Energy Games. In K. Lodaya and M. Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 505–516, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[22] K. Chatterjee and N. Fijalkow. Infinite-state games with finitary conditions. In S. Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 181–196, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[23] K. Chatterjee, T. Henzinger, and M. Jurdziński. Mean-payoff parity games. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science, LICS 2005*, pages 178–187, 2005.

[24] K. Chatterjee, T. A. Henzinger, and F. Horn. Finitary winning in ω-regular games. *ACM Transactions on Computational Logic*, 11(1):1–27, 2009.

[25] K. Chatterjee, M. Randour, and J.-F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Informatica*, 51(3–4):129–163, 2014.

[26] K. Chatterjee and Y. Velner. Mean-payoff pushdown games. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012*, pages 195–204, 2012.

[27] A. Church. Application of recursive arithmetic to the problem of circuit synthesis. In *Summaries of talks presented at the Summer Institute of Symbolic Logic in 1957 at Cornell University*, pages 3–50, 1957.

[28] A. Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35. Institut Mittag-Leffler, Djursholm, Sweden, 1962.

[29] T. Colcombet. The theory of stabilisation monoids and regular cost functions. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas, and W. Thomas, editors, *Automata, Languages and Programming*, volume 5556 of *Lecture Notes in Computer Science*, pages 139–150. Springer Berlin Heidelberg, 2009.

[30] T. Colcombet. Fonctions régulières de coût. Habilitation thesis, Université Paris Diderot – Paris 7, 2013.

[31] T. Colcombet. Regular Cost Functions, Part I: Logic and Algebra over Words. *Logical Methods in Computer Science*, 9(3), 2013.

[32] T. Colcombet and C. Löding. Regular cost functions over finite trees. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010*, pages 70–79, 2010.

[33] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[34] B. Courcelle. The monadic second-order logic of graphs X: Linear orderings. *Theoretical Computer Science*, 160(1–2):87–143, 1996.

[35] A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.

[36] S. Dziembowski, M. Jurdziński, and I. Walukiewicz. How much memory is needed to win infinite games? In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, LICS 1997*, pages 99–110, 1997.

[37] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.

[38] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961.

[39] E. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the IEEE 32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, 1991.

[40] E. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of μ-calculus. In C. Courcoubetis, editor, *Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396. Springer Berlin Heidelberg, 1993.

[41] J. Esparza. On the decidability of model checking for several μ-calculi and petri nets. In S. Tison, editor, *Trees in Algebra and Programming – CAAP'94*, volume 787 of *Lecture Notes in Computer Science*, pages 115–129. Springer Berlin Heidelberg, 1994.

[42] J. Esparza, A. Kučera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Information and Computation*, 186(2):355–376, 2003.

[43] N. Fijalkow and M. Zimmermann. Cost-Parity and Cost-Streett Games. In D. D'Souza, T. Kavitha, and J. Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 124–135, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[44] D. Fischer. *The Quantitative μ-Calculus*. PhD thesis, RWTH Aachen University, 2013.

[45] D. Fischer, E. Grädel, and Ł. Kaiser. Model Checking Games for the Quantitative μ-Calculus. *Theory of Computing Systems*, 47(3):696–719, 2010.

[46] D. Fischer and Ł. Kaiser. Model checking the quantitative μ-calculus on linear hybrid systems. In L. Aceto, M. Henzinger, and J. Sgall, editors, *Automata, Languages and Programming*, volume 6756 of *Lecture Notes in Computer Science*, pages 404–415. Springer Berlin Heidelberg, 2011.

[47] D. Fischer and L. Kaiser. Model checking the quantitative μ-calculus on linear hybrid systems. *Logical Methods in Computer Science*, 8(3), 2012.

[48] D. Gale and F. M. Stewart. Infinite games with perfect information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the theory of games, Volume II*, volume 28 of *Annals of Mathematics Studies*, pages 245–266. Princeton University Press, Princeton, 1953.

[49] T. Ganzow. *Definability and Model Checking: The Role of Orders and Compositionality*. PhD thesis, RWTH Aachen University, 2012.

[50] T. Ganzow and Ł. Kaiser. New algorithm for weak monadic second-order logic on inductive structures. In A. Dawar and H. Veith, editors, *Computer Science Logic*, volume 6247 of *Lecture Notes in Computer Science*, pages 366–380. Springer Berlin Heidelberg, 2010.

[51] T. Ganzow and S. Rubin. Order-Invariant MSO is Stronger than Counting MSO in the Finite. In S. Albers and P. Weil, editors, *25th International Symposium on Theoretical Aspects of Computer Science*, volume 1 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 313–324, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[52] H. Gimbert. Parity and exploration games on infinite graphs. In J. Marcinkowski and A. Tarlecki, editors, *Computer Science Logic*, volume 3210 of *Lecture Notes in Computer Science*, pages 56–70. Springer Berlin Heidelberg, 2004.

[53] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*. Number 2500 in Lecture Notes in Compter Science. Springer, 2002.

[54] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 60–65, New York, 1982. ACM.

[55] J. Hintikka. Distributive normal forms in the calculus of predicates. *Acta philosophica fennica*, 6, 1953.

[56] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[57] S. Hummel and M. Skrzypczak. The topological complexity of MSO+U and related automata models. *Fundamenta Informaticae*, 119(1):87–111, 2012.

[58] P. Hunter, G. A. Pérez, and J.-F. Raskin. Mean-payoff games with partial-observation. In J. Ouaknine, I. Potapov, and J. Worrell, editors, *Reachability Problems*, volume 8762 of *Lecture Notes in Computer Science*, pages 163–175. Springer International Publishing, 2014.

[59] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In U. Montanari and V. Sassone, editors, *CONCUR '96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer Berlin Heidelberg, 1996.

[60] M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[61] Ł. Kaiser. Synthesis for structure rewriting systems. In R. Královič and D. Niwiński, editors, *Mathematical Foundations of Computer Science 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 415–426. Springer Berlin Heidelberg, 2009.

[62] Ł. Kaiser, M. Lang, S. Leßenich, and C. Löding. A unified approach to boundedness properties in MSO. Submitted for publication, 2015.

[63] Ł. Kaiser and S. Leßenich. A Counting Logic for Structure Transition Systems. In P. Cégielski and A. Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL*, volume 16 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 366–380, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[64] A. S. Kechris. *Classical Descriptive Set Theory*, volume 156 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995.

[65] D. Kirsten. Distance desert automata and the star height one problem. In I. Walukiewicz, editor, *Foundations of Software Science and Computation Structures*, volume 2987 of *Lecture Notes in Computer Science*, pages 257–272. Springer Berlin Heidelberg, 2004.

[66] S. Kreutzer and C. Riveros. Quantitative monadic second-order logic. In *Proceedings of the 28th Annual IEEE Symposium on Logic in Computer Science, LICS 2013*, pages 113–122. IEEE Computer Society Press, 2013.

[67] D. Kuperberg. *Etude de classes de fonctions de coût régulières*. PhD thesis, Université Paris Diderot – Paris 7, 2012.

[68] D. Kuperberg and M. Vanden Boom. On the expressive power of cost logics over infinite words. In A. Czumaj, K. Mehlhorn, A. Pitts, and R. Wattenhofer, editors, *Automata, Languages, and Programming*, volume 7392 of *Lecture Notes in Computer Science*, pages 287–298. Springer Berlin Heidelberg, 2012.

[69] O. Kupferman and M. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In E. Emerson and A. Sistla, editors, *Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 36–52. Springer Berlin Heidelberg, 2000.

[70] O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science, LICS 2001*, pages 389–398, Washington, DC, USA, 2001. IEEE Computer Society.

[71] M. Lang. Resource reachability games on pushdown graphs. In A. Muscholl, editor, *Foundations of Software Science and Computation Structures*, volume 8412 of *Lecture Notes in Computer Science*, pages 195–209. Springer Berlin Heidelberg, 2014.

[72] M. Lang and C. Löding. Modeling and verification of infinite systems with resources. *Logical Methods in Computer Science*, 9(4:22), 2013.

[73] M. Lang, C. Löding, and A. Manuel. Definability and transformations for cost logics and automatic structures. In E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik, editors, *Mathematical Foundations of Computer Science 2014*, volume 8634 of *Lecture Notes in Computer Science*, pages 390–401. Springer Berlin Heidelberg, 2014.

[74] D. A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

[75] E. W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 238–246, New York, NY, USA, 1981. ACM.

[76] M. L. Minsky. *Computation: Finite and infinite machines*. Prentice-Hall, Englewood Cliffs, USA, 1967.

[77] A. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdańsk, 1991.

[78] D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1–2):69–107, 1995.

[79] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[80] J. H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274–301, 1984.

[81] S. Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14(2):169–209, 2008.

[82] O. Serre. *Contribution à l'étude des jeux sur des graphes de processus à pile*. PhD thesis, Université Paris 7 – Denis Diderot, 2004.

[83] S. Shelah. The monadic theory of order. *Annals of Mathematics*, 102(3):379–419, 1975.

[84] W. Thomas. Facets of synthesis: Revisiting church's problem. In L. de Alfaro, editor, *Foundations of Software Science and Computational Structures*, volume 5504 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin Heidelberg, 2009.

[85] S. Toruńczyk. *Languages of profinite words and the limitedness problem*. PhD thesis, University of Warsaw, 2011.

[86] B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. In *Twelve Papers on Logic and Algebra*, volume 59 of *American Mathematical Society Translations, Series 2*. American Mathematical Society, Providence, 1966. Appeared in Russian in 1961.

[87] M. Vanden Boom. Weak cost monadic logic over infinite trees. In F. Murlak and P. Sankowski, editors, *Mathematical Foundations of Computer Science 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 580–591. Springer Berlin Heidelberg, 2011.

[88] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, 1944.

[89] I. Walukiewicz. Pushdown processes: Games and model checking. In R. Alur and T. A. Henzinger, editors, *Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer Berlin Heidelberg, 1996.

[90] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.

[91] I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275(1–2):311–346, 2002.

[92] H. Wan. Upper bounds for Ramsey numbers R(3, 3, ..., 3) and Schur numbers. *Journal of Graph Theory*, 26(3):119–122, 1997.

[93] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.

Index

ζ-structure, 24
∞-arena, 70

affine mean counter game, 87
alternating tree automaton, 22
arena, 16

B-automaton, 122
Büchi automaton, 20
Borel game, 18
Borel hierarchy, 15

claim, 102
convolution, 124
costMSO, 119
counter parity game, 29
counting term, 94
counting term decomposition, 106

decomposition, 26
determinacy, 17
deterministic finite automaton, 20
disjunctive normal form, 141
dominance, 120

finite word, 14
finite-memory strategy, 17
FO, 24
FO+RR, 125
FO+RR$^{=\infty}$, 125

game, 16
graph, 13

increasing tree-rewriting rule, 97
infinite word, 14

Kripke structure, 27

labeled tree, 14
Lμ, 27
logic for structure transition systems, 93

mark, 34
mark-graph, 34
marked counter parity game, 39
mean counter game, 59
mean-payoff game, 19
model-checking game for Qμ[#MSO], 95
MSO, 25
MSO+U, 118

nondeterministic finite automaton, 20

$\mathcal{O}$-notation, 14
open set, 15

parity automaton, 21
parity game, 18
parity tree automaton, 21
pattern, 73
pattern strategy, 73
payoff, 16
pop-free pushdown parity game, 101
positional strategy, 17
projection, 128
projective hierarchy, 128
pushdown parity game, 100

qcMSO, 115

Qμ, 27
Qμ[#MSO], 94
quantifier rank, 25
quantitative parity game, 19

rank, 13
regular strategy, 33
relational structure, 24
resource structure, 123
resource-automatic structure, 124

S-automaton, 122
second-life game, 40
semantics of qcMSO, 116
signature, 24
size of an automaton, 20
strategy, 16
structure transition system, 92
syntax of qcMSO, 115

term, 24
tree, 14
tree normal form, 132
tree-producing pushdown system, 98
type, 26
type function, 111
type normal form, 141

unboundedness game, 50

value, 17