A Study of Pushdown Games

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

Dipl.-Inform.

Wladimir Fridman

aus Sankt Petersburg

Berichter:
Universitätsprofessor Dr. Dr.h.c. Wolfgang Thomas
Lecturer Dr. Sven Schewe

Tag der mündlichen Prüfung:
29. Januar 2013

Zusammenfassung

Unendliche Zwei-Personen-Spiele sind von wesentlicher Bedeutung in der Informatik, denn sie stellen einen algorithmischen Rahmen für die Untersuchung von nicht-terminierenden reaktiven Systemen bereit. Üblicherweise werden unendlichen Spiele durch eine ω-Sprache spezifiziert, die alle gewinnenden Partien für einen der beiden Spieler enthält, oder durch einen Spielgraphen und eine Gewinnbedingung auf den unendlichen Pfaden durch diesen Graphen. Viele algorithmische Resultate sind bekannt für den Fall von regulären Spezifikationen und für endliche Spielgraphen mit Gewinnbedingungen wie der Muller- oder Paritätsbedingung, die reguläre Problemstellungen erfassen. Die Ergebnisse der vorliegenden Arbeit behandeln eine Klasse von nicht-regulären Spezifikationen und eine Klasse von unendlichen Spielgraphen, nämlich solche, die durch Pushdown-Automaten spezifiziert werden, d.h. wir betrachten kontextfreie Spezifikationen und Pushdown-Spielgraphen mit Muller- oder Paritätsgewinnbedingungen. Wir erweitern zahlreiche zentrale für reguläre Spiele bekannte Resultate auf die Klasse der Pushdown-Spiele. Dabei konzentrieren wir uns auf folgende vier Problemstellungen.

Zunächst analysieren wir, wie das Format einer Gewinnstrategie mit dem Typ der Spezifikation zusammenhängt. Dabei stellen wir für etliche kontextfreie Fälle eine Verbindung zwischen den Formaten der Spezifikationen und den Formaten der entsprechenden Gewinnstrategien her, zeigen aber auch Fälle von kontextfreien Spielen auf, wo diese Übereinstimmung nicht gilt.

Zweitens untersuchen wir Delay-Spiele mit kontextfreien Gewinnbedingungen. In einem Delay-Spiel hat einer der beiden Spieler die Möglichkeit, seine Züge für eine gewisse Zeit hinauszuzögern, damit wird eine Vorausschau auf die Züge des Gegners erzielt. Wir klären, ob der Gewinner eines deterministisch kontextfreien Delay-Spiels berechnet werden kann, und wie groß die erforderliche Vorausschau zum Gewinnen solcher Spiele ist.

Drittens untersuchen wir das Synthese-Problem für verteilte Systeme, welche aus mehreren kooperierenden Komponenten bestehen, die miteinander und mit der Umgebung kommunizieren. Dabei wird die Kommunikationsstruktur durch eine Architektur spezifiziert. Es werden beide Hauptkonzepte studiert, das der globalen und das der lokalen Spezifikationen. Wir geben eine Charakterisierung der entscheidbaren Architekturen für lokale Spezifikationen an, welche regulär oder deterministisch kontextfrei sein können. Außerdem zeigen wir Unentscheidbarkeit des verteilten Synthese-Problems für globale deterministisch kontextfreie Spezifikationen.

Schließlich wird das Problem behandelt, ob der Gewinner eines unendlichen Spiels bereits nach einem endlichen Partiepräfix bestimmt werden kann. Aufbauend auf den Resultaten für den Fall der endlichen Spielgraphen führen wir eine Paritätsspiele endlicher Dauer auf Pushdown-Graphen ein und

zeigen ihre Vollständigkeit zum Lösen von Paritätsspielen auf Pushdown-Graphen. Dies ergibt eine neue Reduktionsmethode, mit welcher der Gewinner eines Pushdown-Spiels durch Lösen eines Erreichbarkeitsspiels auf einem endlichen Spielgraphen bestimmt werden kann.

Abstract

Infinite two-player games are of interest in computer science since they provide an algorithmic framework for the study of reactive nonterminating systems. Usually, an infinite game is specified by an ω-language containing all winning plays for one of the two players or by a game graph and a winning condition on infinite paths through this graph. Many algorithmic results are known for the case of regular specifications and for finite game graphs with winning conditions like parity or Muller conditions capturing regular objectives. The present thesis offers results that also cover a class of nonregular specifications as well as a class of infinite game graphs, namely those specified by pushdown automata, i.e, we consider contextfree specifications and pushdown game graphs with parity or Muller winning conditions. We extend various central results known for regular games to the class of pushdown games. We focus on the following four questions.

Firstly, we analyze how the format of a pushdown winning strategy matches the type of the pushdown game specification. Here, we establish a strong connection between the formats of specifications and formats of corresponding winning strategies for several types of contextfree games, but we also exhibit cases of contextfree games where this correspondence fails.

Secondly, we investigate delay games with contextfree winning conditions. In such a game one of the players has the possibility to postpone his moves for some time, thus obtaining a lookahead on the moves of the opponent. We clarify whether the winner of a deterministic contextfree delay game can be determined effectively as well as what amount of lookahead is necessary to win such games.

Thirdly, we investigate the synthesis problem for distributed systems which consist of several cooperating components communicating with each other and with the environment. A distributed system is specified by an architecture comprising the communication structure of the system. Here, we study both main concepts, that of global and that of local specifications. We offer a complete characterization of the decidable architectures for local specifications, which may be deterministic contextfree or regular. Moreover, we prove that, for global deterministic contextfree specifications, the distributed synthesis problem is undecidable.

Finally, we address the problem whether the winner of an infinite game can be already determined after a finite play prefix. Extending results for the case of finite game graphs, we introduce finite-duration parity pushdown games and establish their completeness for solving parity pushdown games. This yields a new reduction method to determine the winner of a pushdown game by solving a reachability game on a finite game graph.

Contents

1 Introduction **1**
- 1.1 Contribution . . . 3
- 1.2 Organization of the thesis . . . 7

2 Preliminaries **9**
- 2.1 Basic Definitions . . . 9
- 2.2 Automata . . . 10
  - 2.2.1 Finite Automata . . . 10
  - 2.2.2 Pushdown Systems, Pushdown Automata . . . 12
  - 2.2.3 Trees and Tree Automata . . . 17
- 2.3 Infinite Games . . . 20
  - 2.3.1 Gale-Stewart Games . . . 20
  - 2.3.2 Games on Graphs . . . 21
  - 2.3.3 Game Reduction, Game Simulation . . . 23

3 Pushdown Games and Pushdown Winning Strategies **25**
- 3.1 Classes of Contextfree Languages . . . 26
- 3.2 Formats of Winning Conditions and Winning Strategies . . . 35
  - 3.2.1 Solving Pushdown Games using A2TA . . . 38
  - 3.2.2 Proof of Theorems . . . 44
- 3.3 Summary of Results . . . 52

4 Pushdown Delay Games **55**
- 4.1 Games with Delay . . . 57
- 4.2 Decision Problems . . . 59
- 4.3 Lower Bounds on Delays . . . 68
- 4.4 Summary of Results . . . 75

5 Distributed Synthesis with Pushdown Specifications **77**
- 5.1 Preliminaries . . . 78
- 5.2 Architectures . . . 80
- 5.3 Global Specifications . . . 86

5.4 Local Specifications 89
5.4.1 Decidable Cases 89
5.4.2 Undecidable Cases 107
5.5 Characterization 114
5.6 Summary of Results 121

6 Finite-Time Pushdown Games **123**
6.1 Finite-Time Games 124
6.2 Walukiewicz's Reduction 132
6.3 Equivalence: Infinite-Time and Finite-Time 137
6.4 Lower Bounds 141
6.5 Summary of Results 144

7 Conclusion **145**
7.1 Results 145
7.2 Further Research 146

Bibliography **149**

List of Figures

2.1 Finite Borel hierarchy 21

3.1 Classes of visibly and non-visibly pushdown languages 33
3.2 Parity-StDVPA recognizing $L_2 \in \mathsf{VPL}_\omega \setminus \mathsf{DCFL}_\omega$ 34
3.3 Parity-DV1CA recognizing L 50
3.4 Parity blind one-counter game 52

4.1 Part of a play encoding three configurations 63
4.2 A prefix containing three blocks w_0, w_1 and w_2 69
4.3 A play prefix containing violations of the successor relation $\vdash$ indicated by the players 71

5.1 Induced graph $G_{\mathcal{A}}$ of an architecture $\mathcal{A}$ 82
5.2 Hierarchical architectures 84
5.3 Undecidable architecture for global regular specifications . . . 86
5.4 Decidable architectures for global specifications from DCFL_ω . 88
5.5 Two-flanked pipeline with two system processes 105
5.6 Undecidable architectures for local regular specifications . . . 113
5.7 Generic decidable architecture 116

6.1 The pushdown game graph induced by $\mathcal{S}$ 128
6.2 A finite path w, its stair positions and values of the stair-scoring functions 129
6.3 The pushdown game graph induced by $\mathcal{P}$ 131
6.4 Pushdown game (G_2, col_2) 143

Chapter 1

Introduction

For several decades, the theory of finite automata has proved to be a powerful framework for the development of effective methods for program verification and program synthesis. A very active branch of research is concerned with non-terminating reactive systems, i.e., systems which continuously interact with an environment in contrast to programs that transform data and then terminate. Such non-terminating reactive systems occur in numerous contexts, among them operating systems, control systems, and protocol design. A versatile conceptual model of non-terminating reactive systems is the notion of infinite games, played between an antagonistic environment and one or several system controllers, where plays correspond to infinite sequences of actions performed by the players.

The fundamental problem of this theory is Church's Problem [Chu57, Chu63], first proposed in the context of controller synthesis and then transformed to a game theoretical question by McNaughton [McN65]. It asks, for a given regular specification consisting of all correct behaviors of the considered system, to decide whether there is a finite-state controller such that all possible behaviors of the system satisfy the given specification, and furthermore to synthesize such a controller if it exists. In the framework of infinite games, Church's Problem is formulated as a slightly modified version of a Gale-Stewart game [GS53]. So, it is to decide, for a winning condition given in terms of a logical formula, who is the winner of the game, and to provide a winning strategy for the winning player.

The first solution of this problem was offered by the pioneering work of Büchi and Landweber [BL69] who showed that, for specifications formulated in monadic second-order logic of one successor (over the natural numbers), the controller synthesis problem is decidable and finite-state solutions can be constructed effectively. Hence, for regular winning conditions, the winner can be decided and winning strategies can be implemented by finite automata with output. Starting from the Büchi-Landweber result, the research area

of infinite games has undergone a very active and intensive development, in particular the study of the synthesis problem, due to the prospect of being able to construct system controllers automatically from the specifications, rather than to implement and then to verify already built systems.

Many variants and extensions of the basic setting have been explored in a multitude of different directions. For example, various specification formalisms were considered like linear-time temporal logic [MW81, PR89], branching-time temporal logic [EC82, KV97] or the modal μ-calculus [Koz83, KV00b]. Or the case of incomplete information was considered [KV97] where systems have to satisfy specifications which also refer to signals not known to the controller, in contrary to the basic setting, where both agents have the entire knowledge about all actions performed so far. Another extension is to provide the controller with a buffer allowing to store some actions performed by the environment, in this way the controller gains the possibility to postpone his actions for some time, which leads to so-called delay games [HL72, HKT10]. A further natural extension where both scenarios, incomplete information as well as delay, are conceivable is the extension to distributed systems [PR90, KV01, MT01, FS05] which consist of several components which communicate with each other and with the environment via certain communication channels. Hence, the task of the distributed synthesis is to construct several controllers, one for each process, such that every overall system behavior satisfies a given specification. In the framework of infinite games, distributed synthesis corresponds to multi-player games where several cooperating controller players play against the antagonistic environment player [MW03]. The task is then to find a set of winning strategies, one for each controller player, comprising a joint winning strategy for the system. Of course, there are much more extensions like, for instance, stochastic versions of reactive systems [CJH04, Zie04].

A prerequisite for Büchi-Landweber's solution was the transformation of a logically defined game into a graph game, played on a finite game graph, where the winning condition is just concerned with the requirement that certain vertices are visited infinitely often and others only finitely often. These games are known as Muller games over finite graphs. A classical approach for solving Muller games is the reduction to so-called parity games [Zie98] where every vertex of the game graph is assigned a natural number and the winning condition now is concerned with the smallest color visited infinitely often. A central topic is the extension of this game models to cover infinite game graphs which emerge when the specification formalisms for the synthesis problem go beyond regular languages. The essential case of games played on infinite graphs which has been shown to be decidable is the class of parity games played on pushdown graphs [Wal96]. These are configuration graphs of pushdown systems which, roughly speaking, are infinite state systems

obtained from finite state systems by augmenting them with an infinite last-in-first-out data structure, a pushdown stack. Pushdown systems naturally occur in the context of program analysis and compiler construction. Moreover, the importance of this model in formal languages also arises from the equivalence between the corresponding language acceptors, pushdown automata, and contextfree grammars, which yields definability of contextfree languages in terms of pushdown automata.

The technique for solving parity games played on pushdown graphs proposed by Walukiewicz [Wal96] is a reduction to a parity game played on a finite game graph. The core idea is to encode the information stored on the stack by some finite memory structure. For this, a sophisticated finite game graph is defined to simulate the pushdown game which intends one player to make predictions about the future of the play and the opponent checks correctness of this predictions. Another method is due to Kupferman and Vardi [KV00a], it uses an infinite tree that represents all possible stack contents and alternating two-way tree automata operating on this tree to simulate parity pushdown games. These basic methods will be presented in detail and adapted appropriately in chapters 3 and 6 to prove the main results of those chapters.

1.1 Contribution

This thesis investigates several refinements and extensions, which have been considered within the domain of games over finite game graphs, in the context of pushdown games. So, we study the possibility of lifting the central results known for regular games to the class of contextfree games. Let us sketch the particular contributions informally.

Connecting Game Specifications to Winning Strategies

The Büchi-Elgot-Trakhtenbrot result [Büc60, Elg61, Tra61] establishes a connection between monadic second-order logic and finite automata by showing that both formalisms are expressively equivalent and providing effective translations from one formalism to another and vice versa. By this correspondence, with the focus on the formats of game specifications and their solutions, the Büchi-Landweber theorem states that solutions of games with monadic second-order logic specifications are again definable by monadic second order logic. This result has been refined in several works, where a close conceptual connection between the types of the winning conditions on one hand and winning strategies on the other hand has been established. Rabinovich and Thomas exhibited several fragments $\mathcal{L}$ of the monadic second-order logic, among them first order logic over $(\mathbb{N}, <)$ and its extension by

modular counting quantifiers, first order logic over $(\mathbb{N}, S)$ with successor relation S and the quantifier-free first-order logic over $(\mathbb{N}, 0, +1)$, called strictly bounded logic, for which the mentioned correspondence holds [RT07]. I.e., games given by specifications definable in $\mathcal{L}$ are determined with winning strategies which are again definable in $\mathcal{L}$. Selivanov showed an analogous result for a further regular subclass stating that every game defined by an aperiodic regular winning condition is determined by winning strategies which can be realized by aperiodic transducers [Sel07, Sel08]. Chaturvedi et al. study this correspondence in the recent paper [COT11] for subclasses of star-free languages by considering piecewise testable languages and languages from the dot-depth hierarchy [CB71].

We address this question relating game representations and corresponding solutions for several classes of contextfree games. Thereby, we consider Gale-Stewart games with various types of contextfree specifications emerging from the synthesis problem, as well as more general games played on pushdown game graphs defined by several types of pushdown automata. We prove the existence of a strong connection between the formats of specifications and formats of the corresponding winning strategies for plenty of cases. Among them, games played on configuration graphs of realtime pushdown systems, visibly pushdown systems, and one-counter systems with parity winning conditions, as well as with stair parity winning conditions and Gale-Stewart games defined by corresponding pushdown automata. Furthermore, we exhibit some cases for which this correspondence between games and solutions does not hold, namely for visibly and blind one-counter.

The results are partially based on the publication [Fri10].

Delay Games

The generalization of the game model where the strict alternation between the moves of the environment and the controller is modified by allowing the controller to postpone his moves for some time has been considered by Hosch and Landweber in [HL72]. This so called delay games capture systems provided with first-in-first-out buffers to store inputs coming from the environment. Using these buffers, the controller obtains a lookahead on the moves of the environment. Hosch and Landweber showed that it is decidable, whether a regular game can be won by the controller with bounded lookahead, i.e., using a finite buffer. This result was improved by Holtmann et al. [HKT10], who showed that if a regular game is won with arbitrary lookahead then it is also won with a bounded lookahead, which is at most doubly-exponential in the size of the parity automaton recognizing the winning condition.

We study contextfree delay games and give an answer to the questions

stated in [HKT10] whether the winner of a deterministic contextfree delay game can be determined effectively and what amount of lookahead is necessary to win such games. We prove that, for fixed bounded lookahead, determining the winner is decidable. However, the question whether a given player can win a deterministic contextfree delay game with arbitrary lookahead is undecidable. Moreover, we characterize classes of delay functions, for which the problem of determining the winner is decidable, if the lookahead is restricted to the given class of delay functions. Furthermore, we establish non-elementary lower bounds on delay functions by showing that there exist deterministic contextfree delay games such that the controller wins with some lookahead, however, if the lookahead is bounded by an elementary function, i.e., a k-fold exponential for some fixed natural number k, then the environment wins.

The results are partially based on joint work with Christof Löding and Martin Zimmermann published in [FLZ11].

Distributed Synthesis

Distributed systems are specified by architectures consisting of a set of cooperating processes and a set of channels via which the processes communicate with each other and with the environment. Based on the work of Peterson and Reif on multi-player games [PR79], Pnueli and Rosner proved that in general distributed synthesis is undecidable for specifications in linear time temporal logic, however, for a special class of acyclic architectures, called pipelines, where the processes are linearly ordered and the information flows from the environment in direction of the worst informed process, they proved decidability for linear time temporal logic specifications [PR90]. Kupferman and Vardi extended this decidability result to some further classes of architectures which also may contain cycles and to specifications in branching time temporal logic [KV01]. In [FS05], Finkbeiner and Schewe gave a full characterization of decidable architectures for specifications given in linear time, branching time temporal logic as well as in μ-calculus. Moreover, a uniform tree automata based synthesis algorithm for decidable architectures was provided. Madhusudan and Thiagarajan introduced the concept of local specifications [MT01], where the system specification is given by a set of local specifications, one for each system process. For regular local specifications and the class of acyclic architectures, they gave a characterization of all decidable architectures.

We investigate the distributed synthesis problem for contextfree global specifications as well as local specifications which are regular or contextfree. We give a complete characterization of decidable architectures for the case of local specifications which extends the result of [MT01] in two structural as-

pects. First, we consider general architectures where also cycles are allowed, and second, the local specifications may also be deterministic contextfree. We give an effective criterion which concerns the graph structure of the given architecture and the assignment of regular and contextfree local specifications to the individual processes. Moreover, for deterministic contextfree global specifications, we prove undecidability for almost all architectures. Only the corner cases corresponding to the nondistributed setting or the case where the environment does not send information, are decidable.

The results were obtained in collaboration with Bernd Puchala and are partially based on the publication [FP11].

Finite-Time Games

McNaughton introduced a finite-duration variant for Muller games played on finite game graphs using scoring functions which rate finite play prefixes of infinite plays [McN00]. The scoring functions[1] give a condition on termination of plays in a finite-time game, i.e., a play is stopped when some scoring function reaches a given threshold score value. McNaughton proved equivalence of Muller games and corresponding finite-time Muller games with factorial threshold score, i.e., both variants have the same winner. Thus, the winner of a Muller game on a finite game graph can be determined by solving a reachability game over a game graph, which is doubly-exponential in the size of the game graph of the original Muller game. This result was improved by Fearnley and Zimmermann, who showed that the constant threshold score value of three suffices for the equivalence of the corresponding games [FZ10]. Furthermore, a score-based reduction from Muller games to safety games evolved from this result which yields non-deterministic winning strategies called permissive strategies [Zim12, NRZ12]. This extends the work of Bernet et al. [BJW02] on permissive strategies for parity games to Muller games.

We introduce a new finite-duration variant for parity pushdown games which is required since the known results on finite game graphs don't hold for infinite game graphs. We define stair-scoring functions by exploiting the intrinsic structure of the pushdown graphs and prove the equivalence between parity pushdown games and corresponding finite-time versions with threshold stair-score which is exponential in the size of the underlying pushdown system. Moreover, we give an almost matching lower bound on the threshold stair-score such that the equivalence between the corresponding games holds, which is exponential in the cube root of the size of the underlying pushdown system. Our result yields a new reduction method to determine the winner of a pushdown game by solving a reachability game over a finite

[1] The idea of declaring the winner of a play after a finite number of rounds using scores was first posed in [McN93].

game graph.

The results were obtained in collaboration with Martin Zimmermann and are partially based on [FZ12].

1.2 Organization of the thesis

The thesis is structured as follows. In the subsequent Chapter 2 we introduce the basic notions like finite automata, pushdown automata, tree automata, infinite games, strategies and game reductions, and fix our notations used throughout the thesis.

In Chapter 3 we present our results on the connection between contextfree specifications and pushdown winning strategies. The considered classes of contextfree languages are introduced in Section 3.1. We state our main result in Section 3.2. First, the technique of Kupferman and Vardi is recalled in Subsection 3.2.1. Subsequently, we prove our main result in Subsection 3.2.2.

Chapter 4 presents our results on contextfree delay games. After delay games are introduced formally in Section 4.1, we prove our decidability and undecidability results and give a general criterion characterizing the sets of delay functions where decidability is guaranteed in Section 4.2. Chapter 4 is concluded by presenting a lower bound for the lookahead in Section 4.3.

We handle distributed synthesis in Chapter 5. First, we fix our notations and give some definitions in Section 5.1 which are used throughout this chapter. Architectures are introduced in Section 5.2. We prove undecidability for architectures with global deterministic contextfree specifications in Section 5.3. The following Section 5.4 is composed of decidability and undecidability results for special cases of architectures with local specifications, presented in Subsection 5.4.1 and Subsection 5.4.2, respectively. These results are assembled in the subsequent Section 5.5 where a complete characterization of the decidable architectures with regular or deterministic contextfree local specifications is presented.

Chapter 6 is concerned with finite-time games which we introduce formally in Section 6.1. There, we also define the new notion of stair-scoring functions for finite-time games over pushdown game graphs. In Section 6.2, Walukiewicz's construction is recalled and adapted as it is needed in the following Section 6.3 where we prove the equivalence between parity pushdown games and their corresponding finite-duration variants. Section 6.4 presents a lower bound on the threshold stair-score value for which the equivalence between parity pushdown games and their finite-duration variants holds.

Finally, we give a conclusion in Chapter 7 where we point out some open questions for further research.

Acknowledgements

First, I would like to express my deep gratitude to my advisor Wolfgang Thomas who initiated this doctoral project and gave me the opportunity to work on the present topic. I am grateful for his constant support, valuable advice and professional guidance he gave me throughout the last three years. For me it was a great stroke of luck and personal enrichment to have him as my advisor as his versatile personality sets himself apart from the computer science community.

I would like to thank Sven Schewe for his kind and prompt readiness to act as an external reviewer as well as for his cordial and complaisant support in the last months before the examination. Moreover, his excellent PhD thesis was one of the initial works I was suggested to study when I started my research on distributed synthesis. Furthermore, I am grateful to Joost-Pieter Katoen and Thomas Seidl for joining my thesis committee.

It was a pleasure to work with Christof Löding, Bernd Puchala, and Martin Zimmermann, I wish to thank them for our fruitful collaborations.

I would like to thank Martin Lang, Daniel Neider, Roman Rabinovich, and Nora Schaal for proofreading parts of this thesis.

Furthermore, I am also indebted to a couple of my colleagues from the Chair Logic and Theory of Discrete Systems, the Research Group on Mathematical Foundations of Computer Science and from the Chair of Communication and Distributed Systems who provided some diversion from the effort of doing research and from daily routine, especially Christof Löding for the juggling and passing breaks, Jörg Olschewski for innumerable discussions about anything and everything as well as for the short trips we undertook. Furthermore, I would also like to mention Helen Bolke-Hermanns, Namit Chaturvedi, Silke Cormann, Alex Kwiecien, and Alex Spelten.

Finally, I wish to thank my parents, my sister Emma, and my brother Alex for their support and encouragement, as well as my grandparents to whom I wish to dedicate this thesis. I want to offer my special thanks to my wife Nona for her support and patience she gave me in the last years.

Chapter 2

Preliminaries

In this chapter we give the definitions and fix our notations which will be used throughout this work. After introducing some basic definitions in Section 2.1 we present the automata models of finite automata and pushdown automata over finite and infinte words as well as various kinds of tree automata operating on infinite trees in Section 2.2. Section 2.3 is concerned with turn-based infinite two-player games. Gale-Stewart games, games on graphs as well as notions of winning conditions, winning strategies and game reductions are introduced.

2.1 Basic Definitions

The set of non-negative integers is denoted by $\mathbb{N}$. For $n \in \mathbb{N}$, let $\mathrm{Par}(n) = 0$ if n is even, and $\mathrm{Par}(n) = 1$ if n is odd. Furthermore, we denote the set of the first n elements of $\mathbb{N}$ by $[n] = \{0, \ldots, n-1\}$. For a set X, the power set of X is denoted by $\mathcal{P}(X)$ and the cardinally of X is denoted by $|X|$.

A finite nonempty set Σ of symbols or letters is called alphabet. For an alphabet Σ, the set of finite words is denoted by Σ^*. The length of a word $w \in \Sigma^*$ is denoted by $|w|$, and ε denotes the empty word, i.e., the word of length $|\varepsilon| = 0$. For a word $w \in \Sigma^*$ and a letter $a \in \Sigma$, the number of occurrences of a in w is denoted by $|w|_a$. We denote the set of words of length at most n by $\Sigma^{\leq n}$, for $n \in \mathbb{N}$, and for the set of nonempty words $\Sigma^* \setminus \{\varepsilon\}$ we also write Σ^+. The set of infinite words, also called ω-words, is denoted by Σ^ω.

For a word $w \in \Sigma^* \cup \Sigma^\omega$ and $n \in \mathbb{N}$ we write $w(n)$ for the n-th letter of w where the first letter of w is $w(0)$. If w is finite, we denote its last letter by $\mathsf{last}(w)$. The prefix of length n of a word $w \in \Sigma^* \cup \Sigma^\omega$ is denoted by $\mathsf{pref}_n(w)$, i.e., $\mathsf{pref}_0(w) = \varepsilon$ and $\mathsf{pref}_n(w) = w(0) \cdots w(n-1)$, for $0 < n \leq |w|$. For two words $w \in \Sigma^*$ and $w' \in \Sigma^* \cup \Sigma^\omega$, we write $w \sqsubseteq w'$ if w is a prefix of w', and $w \sqsubset w'$ if w is a strict prefix of w', i.e., if in addition $w \neq w'$ holds. For

a word $w \in \Sigma^*$ the reverse of w is denoted by $\mathsf{rev}(w)$, i.e., $\mathsf{rev}(\varepsilon) = \varepsilon$ and $\mathsf{rev}(ua) = a \cdot \mathsf{rev}(u)$, for $u \in \Sigma^*$ and $a \in \Sigma$.

A language over an alphabet Σ is a subset of Σ^* or a subset of Σ^ω. If it is not clear from the context whether a language contains finite or infinite words, we will also refer to languages containing finite word as $*$-languages and to those containing infinite words as ω-languages.

For a word $w \in \Sigma^* \cup \Sigma^\omega$, we define the occurrence set of w by

$$\mathrm{Occ}(w) = \{a \in \Sigma \mid w(n) = a \text{ for some } n \in \mathbb{N}\} \ ,$$

and the infinity set of w by

$$\mathrm{Inf}(w) = \{a \in \Sigma \mid w(n) = a \text{ for infinitely many } n \in \mathbb{N}\} \ .$$

Let $\Sigma_0, \ldots, \Sigma_{n-1}$ be alphabets. For a cartesian product $\Sigma = \prod_{i\in[n]} \Sigma_i = \Sigma_0 \times \cdots \times \Sigma_{n-1}$ let $\mathsf{dim}(\Sigma) = n$. For the cartesian product $\Sigma = \prod_{i\in[n]} \Sigma_i$ and a set $X \subseteq [n]$ of indices, let the cartesian product restricted to alphabets with an index from X be denoted by $\Sigma_X = \prod_{i\in X} \Sigma_i$ (here again, an ascending order of indices is assumed).

The projection operator $\mathrm{Pr}_X \colon \Sigma \to \Sigma_X$ is defined by $\mathrm{Pr}_X(a) = (a_i)_{i\in X}$, for $a = (a_0, \ldots, a_{n-1}) \in \Sigma$. If $X = \{i\}$ is a singleton set we will also write Pr_i instead of $\mathrm{Pr}_{\{i\}}$. Moreover, we will also write Pr_{Σ_X} instead of Pr_X if we do not refer to an explicit ordering of the components of Σ. We extend this definition in a natural way to words and languages. For a word $w \in \Sigma^* \cup \Sigma^\omega$, define $\mathrm{Pr}_X(w) = \mathrm{Pr}_X(w(0))\mathrm{Pr}_X(w(1))\cdots$, and for a language $L \subseteq \Sigma^*$ or $L \subseteq \Sigma^\omega$, define $\mathrm{Pr}_X(L) = \{\mathrm{Pr}_X(w) \mid w \in L\}$.

Moreover, for two ω-words $w \in \Sigma^\omega$ and $w' \in (\Sigma')^\omega$ over the alphabets Σ and Σ', by $w^\frown w' \in (\Sigma \times \Sigma')^\omega$ we denote the ω-word with $(w^\frown w')(i) = \binom{w(i)}{w'(i)}$, for all $i \in \mathbb{N}$.

2.2 Automata

2.2.1 Finite Automata

A (nondeterministic) finite automaton (NFA) $\mathcal{A} = (Q, \Sigma, Q_{\mathrm{in}}, \Delta, F)$ consists of a finite set of states Q with the set of initial states $Q_{\mathrm{in}} \subseteq Q$, an input alphabet Σ, a set of final states $F \subseteq Q$ and a transition relation $\Delta \subseteq Q \times \Sigma \times Q$. An NFA is deterministic (DFA) if $|Q_{\mathrm{in}}| = 1$ and for all $q \in Q$ and all $a \in \Sigma$, $|\{q' \in Q \mid (q, a, q') \in \Delta\}| \leq 1$ is satisfied. In this case, we use a (partial) function $\delta \colon Q \times \Sigma \to Q$ to denote the transition relation Δ and we just write q_{in} to denote the set $Q_{\mathrm{in}} = \{q_{\mathrm{in}}\}$.

A run ρ of $\mathcal{A}$ on a word $w \in \Sigma^*$ is a finite sequence of states $\rho = \rho(0)\cdots\rho(|w|)$ such that $\rho(0) \in Q_{\mathrm{in}}$, and for every $0 \leq i < |w|$, we have

$(\rho(i), w(i), \rho(i+1)) \in \Delta$. If $\mathcal{A}$ is deterministic then there is a unique run on every word. A run ρ is accepting if $\mathsf{last}(\rho) \in F$. A word w is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The language recognized by $\mathcal{A}$ is

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid w \text{ is accepted by } \mathcal{A}\} \ .$$

A language $L \subseteq \Sigma^*$ is called regular if there exists an NFA $\mathcal{A}$ such that $L = L(\mathcal{A})$. We denote the class of all regular languages by REG.

Remark 2.1. For every $L \subseteq \Sigma^*$, the following are equivalent

1. there exists an NFA $\mathcal{A}$ such that $L = L(\mathcal{A})$,
2. there exists a DFA $\mathcal{A}$ such that $L = L(\mathcal{A})$.

A (nondeterministic) finite ω-automaton (ω-NFA) $\mathcal{A} = (Q, \Sigma, Q_{\text{in}}, \Delta, \Omega)$ consists of a finite set of states Q with the set of initial states $Q_{\text{in}} \subseteq Q$, an input alphabet Σ, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$ and an acceptance condition $\Omega \subseteq Q^\omega$. As in the case of NFA, $\mathcal{A}$ is deterministic (ω-DFA) if $|Q_{\text{in}}| = 1$ and $|\{q' \in Q \mid (q, a, q') \in \Delta\}| \leq 1$, for all $q \in Q$ and all $a \in \Sigma$. In this case, we also use a (partial) function $\delta \colon Q \times \Sigma \to Q$ to denote the transition relation Δ and write q_{in} to denote the set $Q_{\text{in}} = \{q_{\text{in}}\}$.

A run ρ of $\mathcal{A}$ on a word $w \in \Sigma^\omega$ is an infinite sequence of states $\rho = \rho(0)\rho(1)\cdots$ such that $\rho(0) \in Q_{\text{in}}$, and for every $i \in \mathbb{N}$, we have $(\rho(i), w(i), \rho(i+1)) \in \Delta$. If $\mathcal{A}$ is deterministic then there is a unique run on every word. A run ρ is accepting if $\rho \in \Omega$. A word w is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The language recognized by $\mathcal{A}$ is

$$L(\mathcal{A}) = \{w \in \Sigma^\omega \mid w \text{ is accepted by } \mathcal{A}\} \ .$$

Let $\mathrm{col} \colon Q \to [n]$ be a priority function, also called coloring function, which assigns to every state in Q a priority, also called color, from $[n]$, for some $n \in \mathbb{N}$. We extend the function col to sequences of states by defining $\mathrm{col}(\rho) = \mathrm{col}(\rho(0))\mathrm{col}(\rho(1))\cdots$. A parity acceptance condition is defined by

$$\Omega_{\text{parity}} = \{\rho \in Q^\omega \mid \min\{\mathrm{Inf}(\mathrm{col}(\rho))\} \text{ is even}\} \ ,$$

i.e., a run ρ of an automaton with a parity acceptance condition, denoted by parity-NFA (parity-DFA), is accepting if the minimal priority seen infinitely often in ρ is even.

Let $\mathcal{F} \subseteq \mathcal{P}(Q)$ be a set of subsets of Q. A Muller acceptance condition is defined by

$$\Omega_{\text{Muller}} = \{\rho \in Q^\omega \mid \mathrm{Inf}(\rho) \in \mathcal{F}\} \ .$$

i.e., a run ρ of an automaton with a Muller acceptance condition, denoted by Muller-NFA (Muller-DFA), is accepting if the set of states seen infinitely often in ρ is contained in $\mathcal{F}$.

A language $L \subseteq \Sigma^\omega$ is called ω-regular if there exists a parity-NFA $\mathcal{A}$ such that $L = L(\mathcal{A})$. The class of all ω-regular languages is denoted by REG_ω.

Remark 2.2. For every $L \subseteq \Sigma^\omega$, the following are equivalent

1. there exists a parity-NFA $\mathcal{A}$ such that $L = L(\mathcal{A})$,
2. there exists a parity-DFA $\mathcal{A}$ such that $L = L(\mathcal{A})$,
3. there exists a Muller-NFA $\mathcal{A}$ such that $L = L(\mathcal{A})$,
4. there exists a Muller-DFA $\mathcal{A}$ such that $L = L(\mathcal{A})$,

We call the parity and Muller acceptance conditions, which depend on the infinity set of a run, strong acceptance conditions. However, we also consider acceptance conditions which depend on the occurrence set of a run which are called weak. A weak-parity acceptance condition is defined by

$$\Omega_{\text{weak-parity}} = \{\rho \in Q^\omega \mid \min\{\mathrm{Occ}(\mathrm{col}(\rho))\} \text{ is even}\} \ ,$$

i.e., a run ρ of an automaton with a weak-parity acceptance condition, denoted by weak-parity-NFA (weak-parity-DFA), is accepting if the minimal priority occurring in ρ is even.

We call a weak-parity acceptance condition E-acceptance condition if $\mathrm{col}(q) \in \{0,1\}$ for every $q \in Q$, and it is called A-acceptance condition if $\mathrm{col}(q) \in \{1,2\}$ for every $q \in Q$. Thus, a run ρ of an automaton with an E-acceptance condition, denoted by E-NFA (E-DFA), is accepting if ρ visits a state with priority 0 at least once, and for an automaton with an A-acceptance condition, denoted by A-NFA (A-DFA), a run is accepting if it never visits a state with priority 1.

In the remainder of this work we will also write col or $\mathcal{F}$ for the acceptance condition instead of Ω.

2.2.2 Pushdown Systems, Pushdown Automata

A pushdown system (PDS) $\mathcal{S} = (Q, \Gamma, \Delta, q_{\text{in}})$ consists of a finite set of states Q with the initial state $q_{\text{in}} \in Q$, a stack alphabet Γ with an initial stack symbol $\bot \notin \Gamma$, and a transition relation

$$\Delta \subseteq Q \times \Gamma_\bot \times Q \times \Gamma_\bot^{\leq 2} \ ,$$

where by $\Gamma_\bot$ we denote the set $\Gamma \cup \{\bot\}$. Moreover, the transition relation Δ satisfies the following requirement. For every transition $\delta = (q, A, p, \alpha) \in \Delta$,

if $A = \bot$ then $\alpha = \gamma\bot$ with $\gamma \in \Gamma^{\leq 1}$, otherwise $\alpha \in \Gamma^{\leq 2}$, i.e., the initial stack symbol $\bot$ can neither be written nor deleted from the stack.

A transition $\delta = (q, A, p, \alpha) \in \Delta$ is called a push-transition if $|\alpha| = 2$, it is called a skip-transition if $|\alpha| = 1$, and δ is a pop-transition if $\alpha = \varepsilon$. We say that $\mathcal{S}$ is deadlock-free if for every $q \in Q$ and every $A \in \Gamma_\bot$ there exist $p \in Q$ and $\alpha \in \Gamma_\bot^{\leq 2}$ such that $(q, A, p, \alpha) \in \Delta$.

A stack content is a word from $\Gamma^*\bot$ where we assume the leftmost symbol to be the top of the stack. A configuration is a pair (q, γ) consisting of a state $q \in Q$ and a stack content $\gamma \in \Gamma^*\bot$. We define the stack height of a configuration (q, γ) by

$$\mathsf{sh}(q, \gamma) = |\gamma| - 1 \ .$$

Moreover, we write $(q, \gamma) \vdash_{\mathcal{S}} (q', \gamma')$ if there exists $(q, \gamma(0), q', \alpha) \in \Delta$ and $\gamma' = \alpha\gamma(1) \cdots \gamma(|\gamma| - 1)$.

We extend the notion of PDS to pushdown machines, pushdown automata and pushdown transducers. A pushdown machine is a pushdown system augmented by an input alphabet. For pushdown automata, input alphabets and acceptance conditions are attached, and for pushdown transducers, input and output alphabets, and output functions are provided.

A (nondeterministic) pushdown machine (PDM) $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}})$ consists of a finite set of states Q with the initial state $q_{\text{in}} \in Q$, an input alphabet Σ, a stack alphabet Γ with the initial stack symbol $\bot \notin \Gamma$, and a transition relation

$$\Delta \subseteq Q \times \Gamma_\bot \times \Sigma_\varepsilon \times Q \times \Gamma_\bot^{\leq 2} \ ,$$

where $\Gamma_\bot = \Gamma \cup \{\bot\}$ and $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. Analogously to PDS, the initial stack symbol $\bot$ marks the bottom of the stack and can neither be written nor deleted from the stack. For a transition $\delta = (q, A, a, p, \alpha)$, we call δ a push-, skip- or pop-transition depending on the length of α as for PDS. Furthermore, δ is an ε-transition if $a = \varepsilon$ and it is called non-ε-transition otherwise. A PDM $\mathcal{M}$ is deterministic (DPDM) if the transition relation Δ satisfies

$$\big|\{(q', \alpha) \mid (q, A, a, q', \alpha) \in \Delta\}\big| + \big|\{(q', \alpha) \mid (q, A, \varepsilon, q', \alpha) \in \Delta\}\big| \leq 1$$

for all $q \in Q$, all $a \in \Sigma$, and all $A \in \Gamma_\bot$. In this case we use a (partial) function $\delta \colon Q \times \Gamma_\bot \times \Sigma_\varepsilon \to Q \times \Gamma_\bot^{\leq 2}$ to denote the transition relation Δ.

For two configurations $(q, \gamma), (q', \gamma') \in Q \times \Gamma^*\bot$, we write $(q, \gamma) \overset{a}{\vdash}_{\mathcal{M}} (q', \gamma')$ if there exists $(q, \gamma(0), a, q', \alpha) \in \Delta$ and $\gamma' = \alpha\gamma(1) \cdots \gamma(|\gamma| - 1)$.

A run ρ of $\mathcal{M}$ on a finite word $w \in \Sigma^*$ is a finite sequence of configurations $\rho = (q_0, \gamma_0) \cdots (q_r, \gamma_r)$ such that

1. $(q_0, \gamma_0) = (q_{\text{in}}, \bot)$, and
2. for every $i \in [r]$, there exists $a_i \in \Sigma_\varepsilon$ such that $(q_i, \gamma_i) \overset{a_i}{\vdash}_{\mathcal{M}} (q_{i+1}, \gamma_{i+1})$ and $a_0 \cdots a_r = w$, and
3. $\{(q, \alpha) \mid (q_r, \gamma_r(0), \varepsilon, q, \alpha) \in \Delta\} = \emptyset$.

The last item of this definition requires that no execution of an ε-transition is possible from the last configuration of a run of $\mathcal{M}$ on a finite word.

A run ρ of $\mathcal{M}$ on an infinite word $w \in \Sigma^\omega$ is defined as an infinite sequence of configurations $\rho = (q_0, \gamma_0)(q_1, \gamma_1) \cdots$ such that

1. $(q_0, \gamma_0) = (q_{\text{in}}, \bot)$, and
2. for every $i \in \mathbb{N}$, there exists $a_i \in \Sigma_\varepsilon$ such that $(q_i, \gamma_i) \overset{a_i}{\vdash}_{\mathcal{M}} (q_{i+1}, \gamma_{i+1})$ and $a_0 a_1 \cdots = w$.

Notice, that if $\mathcal{M}$ is deterministic then, for every word $w \in \Sigma^* \cup \Sigma^\omega$, if there exists a run of $\mathcal{M}$ on w then there is exactly one unique run. We say that a PDM $\mathcal{M}$ has the *continuity property* if for every word $w \in \Sigma^\omega$ there exists a run of $\mathcal{M}$ on w.

A pushdown automaton (PDA) $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, F)$ consists of a PDM $\mathcal{M}^{\mathcal{P}} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}})$ and a set of final states $F \subseteq Q$. A PDA $\mathcal{P}$ is deterministic (DPDA) if $\mathcal{M}^{\mathcal{P}}$ is deterministic. We say that a PDA $\mathcal{P}$ has the continuity property if $\mathcal{M}^{\mathcal{P}}$ has the continuity property.

For two configurations $c, c' \in Q \times \Gamma^* \bot$, we write $c \overset{a}{\vdash}_{\mathcal{P}} c'$ if $c \overset{a}{\vdash}_{\mathcal{M}^{\mathcal{P}}} c'$. A run ρ of $\mathcal{P}$ on a word $w \in \Sigma^*$ is a run of $\mathcal{M}^{\mathcal{P}}$ on w. A run ρ is accepting if $\mathsf{last}(\rho) \in F$. A word w is accepted by $\mathcal{P}$ if there is an accepting run of $\mathcal{P}$ on w. The language recognized by $\mathcal{P}$ is

$$L(\mathcal{P}) = \{w \in \Sigma^* \mid w \text{ is accepted by } \mathcal{A}\} \ .$$

A language $L \subseteq \Sigma^*$ is called (nondeterministic) contextfree if there exists a PDA $\mathcal{P}$ such that $L = L(\mathcal{P})$, and it is called deterministic contextfree if there exists a DPDA $\mathcal{P}$ such that $L = L(\mathcal{P})$. We denote the class of all contextfree languages by CFL and the class of all deterministic contextfree languages by DCFL. It is well-known that $\mathsf{REG} \subsetneq \mathsf{DCFL} \subsetneq \mathsf{CFL}$ [GG66].

Lemma 2.3.

1. *For every PDA $\mathcal{P}$, a PDA $\mathcal{P}'$ can be constructed such that $L(\mathcal{P}) = L(\mathcal{P}')$ and $\mathcal{P}'$ has the continuity property.*
2. *For every DPDA $\mathcal{P}$, a DPDA $\mathcal{P}'$ can be constructed such that $L(\mathcal{P}) = L(\mathcal{P}')$ and $\mathcal{P}'$ has the continuity property.*

The first statement of Lemma 2.3, can easily be established by extending the PDA by just one non-accepting sink state which is reached from every configuration via an ε-transition and where every further computation remains forever. The construction for the deterministic case, the second statement of this lemma, involves elimination of loops consisting solely of ε-transitions (cf. [GG66]).

Now, we define pushdown automata on infinite words. An ω-pushdown automaton (ω-PDA) $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \Omega)$ consists of a PDM $\mathcal{M}^{\mathcal{P}} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}})$ and an acceptance condition $\Omega \subseteq (Q \times \Gamma^* \bot)^\omega$. Again, $\mathcal{P}$ is deterministic (ω-DPDA) if $\mathcal{M}^{\mathcal{P}}$ is deterministic.

A run ρ of $\mathcal{P}$ on a word $w \in \Sigma^\omega$ is a run of $\mathcal{M}^{\mathcal{P}}$ on w. A run ρ is accepting if $\rho \in \Omega$. A word w is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The language recognized by $\mathcal{A}$ is

$$L(\mathcal{A}) = \{w \in \Sigma^\omega \mid w \text{ is accepted by } \mathcal{A}\} \ .$$

Let $\text{col}\colon Q \to [n]$ be a coloring function, for some $n \in \mathbb{N}$. We extend the function col to configurations by defining $\text{col}(q, \gamma) = \text{col}(q)$ for every state $q \in Q$ and every stack content $\gamma \in \Gamma^* \bot$, i.e., the color of a configuration depends only on the state and not on the stack content of the configuration.

Now, we consider the acceptance conditions presented in the previous subsection and define the corresponding ω-pushdown automata. A run ρ of a parity-PDA (parity-DPDA) $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \text{col})$ is accepting if

$$\min\{\text{Inf}(\text{col}(\rho))\} \text{ is even} \ .$$

A run ρ of a Muller-PDA (Muller-DPDA) $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \mathcal{F})$ is accepting if the set of states seen infinitely often in ρ is contained in $\mathcal{F}$, i.e.,

$$\text{Inf}(\text{Pr}_0(\rho)) \in \mathcal{F} \ .$$

A language $L \subseteq \Sigma^\omega$ is called (nondeterministic) ω-contextfree if there exists an parity-PDA $\mathcal{P}$ such that $L = L(\mathcal{P})$, and it is called deterministic ω-contextfree if there exists an parity-DPDA $\mathcal{P}$ such that $L = L(\mathcal{P})$. We denote the class of all ω-contextfree languages by CFL_ω, and the class of all deterministic ω-contextfree languages by DCFL_ω. For these classes the proper inclusion holds as well, $\mathsf{REG}_\omega \subsetneq \mathsf{DCFL}_\omega \subsetneq \mathsf{CFL}_\omega$ [CG78].

Remark 2.4. For every $L \subseteq \Sigma^\omega$, the following are equivalent

1. there exist parity-PDA $\mathcal{P}$ such that $L = L(\mathcal{P})$.
2. there exist Muller-PDA $\mathcal{P}$ such that $L = L(\mathcal{P})$.

Remark 2.5. For every $L \subseteq \Sigma^\omega$, the following are equivalent

1. there exist parity-DPDA $\mathcal{P}$ such that $L = L(\mathcal{P})$.

2. there exist Muller-DPDA $\mathcal{P}$ such that $L = L(\mathcal{P})$.

Lemma 2.6.

1. *For every parity-PDA $\mathcal{P}$, a parity-PDA $\mathcal{P}'$ can be constructed such that $L(\mathcal{P}) = L(\mathcal{P}')$ and $\mathcal{P}'$ has the continuity property.*

2. *For every parity-DPDA $\mathcal{P}$, a parity-DPDA $\mathcal{P}'$ can be constructed such that $L(\mathcal{P}) = L(\mathcal{P}')$ and $\mathcal{P}'$ has the continuity property.*

Analogously to Lemma 2.3, the construction for parity-PDA utilizes nondeterminism and for parity-DPDA the crucial point is again the elimination of loops consisting of ε-transitions (cf. [GG66] and [CG78]). Unless otherwise stipulated in the remainder of this work we assume every PDA and every ω-PDA to have the continuity property.

We continue by defining weak ω-pushdown automata. A run ρ of a weak-parity-PDA (weak-parity-DPDA) $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \text{col})$ is accepting if

$$\min\{\text{Occ}(\text{col}(\rho))\} \text{ is even } .$$

For E-acceptance and A-acceptance conditions we abbreviate the corresponding pushdown automata by E-PDA (E-DPDA) and A-PDA (A-DPDA), respectively.

We complete by defining pushdown transducers, PDM provided by output. A pushdown transducer (PDT) $\mathcal{T} = (Q, \Sigma_I, \Sigma_O, \Gamma, \Delta, q_{\text{in}}, \lambda)$ consists of a PDM $\mathcal{M}^{\mathcal{T}} = (Q, \Sigma_I, \Gamma, \Delta, q_{\text{in}})$, an output alphabet Σ_O and a partial output function $\lambda\colon Q \to \Sigma_O$. A pushdown transducer $\mathcal{T}$ is deterministic (DPDT) if $\mathcal{M}^{\mathcal{T}}$ is deterministic. A PDT $\mathcal{T}$ is a nondeterministic transducer (NFT) if $\Gamma = \emptyset$, i.e., $\mathcal{M}^{\mathcal{T}}$ can be seen as a finite automaton, since it has no access to the stack. An NFT is deterministic (DFT) if $\mathcal{M}^{\mathcal{T}}$ is deterministic. We will omit Γ in the description of NFT and DFT.

A run ρ of $\mathcal{T}$ on a word $w \in (\Sigma_I)^*$ is a run of $\mathcal{M}^{\mathcal{T}}$ on w. A DPDT $\mathcal{T}$ defines a partial function $f_{\mathcal{T}}\colon (\Sigma_I)^* \to \Sigma_O$ as follows. For a word $w \in (\Sigma_I)^*$, let ρ be the unique run of $\mathcal{T}$ on w and $\mathsf{last}(\rho) = (q, \gamma)$, for some state $q \in Q$ and some stack content $\gamma \in \Gamma^*\bot$, then $f_{\mathcal{T}}(w) = \lambda(q)$. We will also use another definition of DPDT where the output function is of the form $\lambda\colon Q \times \Gamma_\bot \to \Sigma_O$. In this case the output symbol not only depends on the state but also on the top stack symbol of the last configuration of the run of the DPDT $\mathcal{T}$, i.e., for a word $w \in (\Sigma_I)^*$, and the unique run ρ of $\mathcal{T}$ on w with $\mathsf{last}(\rho) = (q, \gamma)$, $f_{\mathcal{T}}(w) = \lambda(q, \gamma(0))$. Note, that the definitions are equivalent, as the current top stack symbol can be stored in the current state.

2.2.3 Trees and Tree Automata

For a set X, an X-tree is a prefix closed set $T \subseteq X^*$, i.e., for $w \in X^*$ and $x \in X$ if $wx \in T$ then also $w \in T$. The elements of T are called nodes and the node ε is called root of T. For a node $wx \in T$ with $w \in X^*$ and $x \in X$, wx is called child or successor of w, and w is the parent or predecessor of wx. If $T = X^*$ then it is called full infinite X-tree.

For an alphabet Σ, a Σ-labeled X-tree is a pair (T, t) where T is an X-tree and $t\colon T \to \Sigma$ is a function assigning to each node from T a symbol from Σ. We call a labeled tree (T, t) full if T is full. By X_Σ we denote the set of all full infinite Σ-labeled X-trees. To simplify our notation we will sometimes write t instead of (X^*, t) for a tree in X_Σ.

For a tree $t \in X_\Sigma$ and a set Y we define the Y-widening of t as a Σ-labeled $(X \times Y)$-tree $\mathsf{wide}_Y(t) = t'$ such that $t'(z) = t(\mathrm{Pr}_X(z))$, for every node $z \in (X \times Y)^*$. Furthermore, we say that a tree $t \in X_\Sigma$ is regular if it is generated by a finite transducer, i.e., if there is a DFT $\mathcal{T} = (Q, X, \Sigma, \delta, q_{\text{in}}, \lambda)$ such that for every node $w \in X^*$, $f_\mathcal{T}(w) = t(w)$.

For a finite set S, we denote the set of all positive Boolean formulas over propositional variables from S (where the formulas true and false are also allowed) by $\mathcal{B}^+(S)$. For Boolean formulas we assume, that $\wedge$ has precedence over $\vee$. Notice that with this definition, every formula from $\mathcal{B}^+(S)$ is in disjunctive normal form (DNF). We denote such formulas φ in DNF also as sets $\varphi = \{\psi_1, \ldots, \psi_k\}$ of conjuncts where we denote the conjuncts ψ_i also as sets $\psi_i \subseteq S$ of propositional variables. A set $S' \subseteq S$ satisfies a formula $\varphi \in \mathcal{B}^+(S)$ if and only if φ is true when assigned true to all elements in S' and false to all elements in $S \setminus S'$.

To be able to navigate through a tree we define sets of directions. For a finite set X, let $\mathsf{Dir} = \{\downarrow_x |\ x \in X\}$, $\mathsf{Dir}_\uparrow = \mathsf{Dir} \cup \{\uparrow\}$, $\mathsf{Dir}_\circlearrowleft = \mathsf{Dir} \cup \{\circlearrowleft\}$, and $\mathsf{Dir}_{\uparrow,\circlearrowleft} = \mathsf{Dir} \cup \{\uparrow, \circlearrowleft\}$. For all $w \in X^*$ and $x \in X$, we define $w{\cdot}\circlearrowleft = w$, $w{\cdot}\downarrow_x = wx$, and $wx{\cdot}\uparrow = w$.

Let X be a finite set. An alternating parity two-way tree automaton (parity-A2TA) $\mathcal{A} = (Q, \Sigma, \delta, q_{\text{in}}, \mathrm{col})$ consists of a finite set of states Q with the initial state q_{in}, a labeling alphabet Σ, a coloring function $\mathrm{col}\colon Q \to [n]$, for some $n \in \mathbb{N}$, and a transition function $\delta\colon Q \times \Sigma \to \mathcal{B}^+(\mathsf{Dir}_{\uparrow,\circlearrowleft} \times Q)$.

A run of $\mathcal{A}$ on a full infinite Σ-labeled X-tree $t \in X_\Sigma$ is a not necessarily full $(Q \times X^*)$-labeled $\mathbb{N}$-tree (T, ρ) such that the following conditions hold.

1. $\varepsilon \in T$ and $\rho(\varepsilon) = (q_{\text{in}}, \varepsilon)$.

2. If $y \in T$ with $\rho(y) = (q, w)$ and $\delta(q, t(w)) = \varphi$ then there is a conjunct $\psi = \{(d_0, q_0), \ldots, (d_{k-1}, q_{k-1})\} \subseteq \mathsf{Dir}_{\uparrow,\circlearrowleft} \times Q$ in φ such that the set of successors of y in T is precisely $\{y \cdot i \,|\, i \in [k]\}$ and $\rho(y \cdot i) = (q_i, w \cdot d_i)$.

A run (T, ρ) is accepting if all its paths satisfy the parity condition col, i.e., for each path π through T (starting in ε) $\min\{\mathrm{Inf}(\mathrm{col}(\rho(\pi)))\}$ is even, where $\rho(\pi)$ is the infinite sequence of labelings of π and col is extended to labelings $(q, w) \in Q \times X^*$ such that $\mathrm{col}(q, w) = \mathrm{col}(q)$ for every $q \in Q$ and $w \in X^*$.

A tree $t \in X_\Sigma$ is accepted by $\mathcal{A}$ if there is an accepting run (T, ρ) of $\mathcal{A}$ on t. The tree language recognized by $\mathcal{A}$ is

$$L(\mathcal{A}) = \{t \in X_\Sigma \mid t \text{ is accepted by } \mathcal{A}\} \ .$$

We call a parity-A2TA $\mathcal{A}$ an alternating parity one-way tree automaton or just alternating parity tree automaton, denoted by parity-A1TA or parity-ATA, if the transition function δ does not use the directions $\uparrow$ and $\circlearrowleft$, i.e., it is of the form $\delta\colon Q \times \Sigma \to \mathcal{B}^+(\mathsf{Dir} \times Q)$. We call a parity-ATA $\mathcal{A}$ over Σ-labeled an alternating parity finite automaton, denoted by parity-AFA if $|X| = 1$, in this case we omit the Dir component in the transition function, i.e., for parity-AFA the transition function is of the form $\delta\colon Q \times \Sigma \to \mathcal{B}^+(Q)$.

Let $X = \{x_0, \ldots, x_k\}$ for some $k \in \mathbb{N}$. A parity-ATA is called nondeterministic, denoted by parity-NTA or parity-N1TA, if the transition function is of the following form. For all $q \in Q$ and all $a \in \Sigma$, $\delta(q, a)$ has the form

$$\bigvee_{j=0}^{n} (\downarrow_{x_0}, q_0^j) \wedge \ldots \wedge (\downarrow_{x_k}, q_k^j) \quad \text{for some } n \in \mathbb{N}.$$

Notice that for every parity-ATA there is an equivalent parity-NTA, in particular for every parity-AFA there is also an equivalent parity-NFA.

Theorem 2.7 ([MS95])**.** *For every parity-ATA $\mathcal{A}$ over Σ-labeled X-trees there is a parity-NTA $\mathcal{N}$ over Σ-labeled X-trees such that $L(\mathcal{A}) = L(\mathcal{N})$.*

Remark 2.8 ([KV99])**.** For every parity-NTA $\mathcal{A}$ over Σ-labeled $(X \times Y)$-trees there is a parity-NTA $\mathcal{B}$ over Σ-labeled X-trees such that $t \in L(\mathcal{B})$ if and only if $\mathsf{wide}_Y(t) \in L(\mathcal{A})$.

An alternating (one-way) pushdown tree automaton (APDTA) $\mathcal{P} = (Q, \Sigma, \Gamma, q_{\mathrm{in}}, \delta, \Omega)$ over Σ-labeled X-trees consists of a finite set Q of states with the initial state q_{in}, a labeling alphabet Σ, a stack alphabet Γ with the initial stack symbol $\bot \notin \Gamma$, a transition function

$$\delta\colon Q \times \Sigma_\varepsilon \times \Gamma_\bot \to \mathcal{B}^+(\mathsf{Dir}_{\circlearrowleft} \times Q \times \Gamma_\bot^{\leq 2}) \ ,$$

where $\Gamma_\bot = \Gamma \cup \{\bot\}$ and $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, and an acceptance component Ω which is either a parity acceptance condition $\mathrm{col}\colon Q \to [n]$, for some $n \in \mathbb{N}$, (in this case the automaton is denoted by parity-APDTA) or a Muller acceptance condition $\mathcal{F} \subseteq \mathcal{P}(Q)$ (Muller-APDTA). As for pushdown word automata we

assume that the initial stack symbol $\perp$ neither can be written to nor be deleted from the stack.

A run of a APDTA $\mathcal{P}$ on a Σ-labeled X-tree $t \in X_\Sigma$ is a not necessarily full $(X^* \times Q \times \Gamma^* \perp)$-labeled $\mathbb{N}$-tree (T, ρ) such that the following conditions hold.

1. $\varepsilon \in T$ and $\rho(\varepsilon) = (\varepsilon, q_{\text{in}}, \perp)$.

2. If $y \in T$ with $\rho(y) = (w, q, \gamma)$ and $\delta(q, t(w), \gamma(0)) = \varphi$ then there is a conjunct $\psi = \{(d_0, q_0, \alpha_0), \ldots, (d_{k-1}, q_{k-1}, \alpha_{k-1})\} \subseteq \mathsf{Dir}_{\circlearrowleft} \times Q \times \Gamma_\perp^{\leq 2}$ in φ such that the set of successors of y in T is precisely $\{y \cdot i \mid i \in [k]\}$ and $\rho(y \cdot i) = (w \cdot d_i, q_i, \alpha_i \gamma(1) \cdots \gamma(|\gamma| - 1))$.

A run (T, ρ) is accepting, if all its paths satisfy the acceptance condition Ω. That means, that for each infinite path π through T (starting in ε), in case of a parity acceptance condition $\min\{\text{Inf}(\text{col}(\rho(\pi)))\}$ has to be even, and $\text{Inf}(\text{Pr}_1(\rho(\pi))) \in \mathcal{F}$ has to be satisfied in case of a Muller acceptance condition. Here $\rho(\pi)$ is the infinite sequence of labelings of π, the coloring function col is extended to labelings $(w, q, \gamma) \in X^* \times Q \times \Gamma^* \perp$ such that $\text{col}(w, q, \gamma) = \text{col}(q)$ for every $w \in X^*$ and every configuration $(q, \gamma) \in Q \times \Gamma^* \perp$. Notice that Pr_1 projects to the set of states Q, i.e., $\text{Pr}_1(w, q, \gamma) = q$ for every $w \in X^*$ and every configuration $(q, \gamma) \in Q \times \Gamma^* \perp$..

A tree $t \in X_\Sigma$ is accepted by $\mathcal{P}$ if there is an accepting run (T, ρ) of $\mathcal{P}$ on t. The tree language recognized by $\mathcal{P}$ is

$$L(\mathcal{P}) = \{t \in X_\Sigma \mid t \text{ is accepted by } \mathcal{P}\} \ .$$

Let $X = \{x_0, \ldots, x_k\}$ for some $k \in \mathbb{N}$. An APDTA $\mathcal{P}$ is called nondeterministic (NPDTA) if, for every $q \in Q$ and every $A \in \Gamma_\perp$, either for all $a \in \Sigma$ there is some $(q', \alpha) \in Q \times \Gamma_\perp^{\leq 2}$ such that $\delta(q, a, A) = (\circlearrowleft, q', \alpha)$ or, for all $a \in \Sigma$, $\delta(q, a, A)$ has the form

$$\bigvee_{j=0}^{n} (\downarrow_{x_0}, q_0^j, \alpha_0^j) \wedge \ldots \wedge (\downarrow_{x_k}, q_k^j, \alpha_k^j)$$

for some $n \in \mathbb{N}$. We denote an NPDTA with a parity (Muller) acceptance condition by parity-NPDTA (Muller-NPDTA). Notice, that like for parity-NTA, the nonemptiness problem for parity-NPDTA, which is to decide, given a parity-NPDTA $\mathcal{P}$ over Σ-labeled X-trees whether $L(\mathcal{P}) \neq \emptyset$, is shown to be decidable.

Theorem 2.9 ([KPV02]). *The nonemptiness problem for parity-NPDTA is decidable.*

2.3 Infinite Games

2.3.1 Gale-Stewart Games

We consider so called Gale-Stewart games, turn-based infinite two-player games of perfect information, which were introduced in [GS53]. We present here a slightly modified version of the original definition.

Let Σ_I and Σ_O be two alphabets and let $\Sigma = \Sigma_I \times \Sigma_O$. We call Σ_I input alphabet and Σ_O output alphabet. An ω-language $L \subseteq \Sigma^\omega$ defines the Gale-Stewart game $\Gamma(L)$. The game $\Gamma(L)$ is played by two players, Player I (the input player which will also be denoted by Player 1) and Player O (the output player, also denoted by Player 0) in rounds $i \in \mathbb{N}$. The two players pick letters from their respective alphabets in alternation. In every round i, first Player I picks a letter a_i from Σ_I and then Player O (being aware of the choice a_i of Player I) picks a letter b_i from Σ_O.

A play of $\Gamma(L)$ is a sequence $a_0, b_0, a_1, b_1, a_2, b_2, \ldots$ of letters which yields two ω-words, the input word $\alpha = a_0 a_1 a_2 \cdots$ constructed by Player I and the output word $\beta = b_0 b_1 b_2 \cdots$ produced by Player O. The language L is used to determine the winner of the play, i.e., it provides the winning condition. Player O wins the play if and only if the ω-word $\alpha^\frown\beta = \binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)}\cdots$ induced by the play is contained in L.

A strategy for Player I is a function $\sigma_I \colon (\Sigma_O)^* \to \Sigma_I$, and a strategy for Player O is a function $\sigma_O \colon (\Sigma_I)^* \to \Sigma_O$. Consider a play $a_0, b_0, a_1, b_1, \ldots$ and its induced ω-word $\alpha^\frown\beta$. The play is consistent with a strategy σ_I if $\alpha(n) = \sigma_I(\mathsf{pref}_n(\beta))$, for all $n \in \mathbb{N}$. The play is consistent with a strategy σ_O if $\beta(n) = \sigma_O(\mathsf{pref}_{n+1}(\alpha))$, for all $n \in \mathbb{N}$. A strategy σ is winning for Player i, for $i \in \{0, 1\}$, if every play which is consistent with σ is won by Player i. We say that Player i wins $\Gamma(L)$ if there is a winning strategy for Player i. A game $\Gamma(L)$ is determined if it is won by either of the two players.

Consider Σ^ω as a topological space. Open sets are languages of the form $W \cdot \Sigma^\omega$ where $W \subseteq \Sigma^*$ and closed sets are complements of open sets. The class of open sets is denoted by $\mathbf{\Sigma}_1$ and the class of closed sets is denoted by $\mathbf{\Pi}_1$. A language $L \subseteq \Sigma^\omega$ is a Borel set if it is obtained from open and closed sets by repeatedly applying countable unions and countable intersections. Based on the classes $\mathbf{\Sigma}_1$ and $\mathbf{\Pi}_1$ the finite levels of the Borel Hierarchy are defined inductively, $\mathbf{\Sigma}_{n+1}$ is the class of countable unions of $\mathbf{\Pi}_n$-sets, and $\mathbf{\Pi}_{n+1}$ is the class of countable intersections of $\mathbf{\Sigma}_n$-sets, for $n > 0$. Let $\mathcal{B}(\mathbf{\Sigma}_n)$ denote the boolean closure of the n-th level of the Borel Hierarchy. The full Borel hierarchy is obtained by including the classes $\mathbf{\Sigma}_\alpha$, for countable ordinals α. Figure 2.1 illustrates how the classes are related.

Theorem 2.10 ([Mar75])**.** *$\Gamma(L)$ is determined, for every Borel set $L \subseteq \Sigma^\omega$.*

We will also represent strategies by functions which take into account the

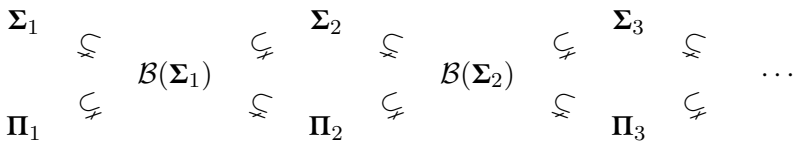


Figure 2.1: Finite Borel hierarchy

whole history of a play and not only sequences produced by the opponent, i.e., of the form $\sigma_I \colon (\Sigma_I \times \Sigma_O)^* \to \Sigma_I$ and $\sigma_O \colon (\Sigma_I \times \Sigma_O)^* \Sigma_I \to \Sigma_O$. Let $a_0, b_0, a_1, b_1, \ldots$ be a play and $\alpha^\frown\beta$ its induced ω-word. The play is consistent with σ_I if $\alpha(n) = \sigma_I(\mathsf{pref}_n(\alpha^\frown\beta))$, for all $n \in \mathbb{N}$. It is consistent with σ_O if $\beta(n) = \sigma_O(\mathsf{pref}_n(\alpha^\frown\beta)\alpha(n))$, for all $n \in \mathbb{N}$. Notice, that both representations can be converted into each other.

For a class $\mathcal{L}$ of ω-languages, we refer to a Gale-Stewart game $\Gamma(L)$ as an $\mathcal{L}$-game if $L \in \mathcal{L}$.

2.3.2 Games on Graphs

A game graph $G = (V, V_0, V_1, E, v_{\text{in}})$ consists of a (possibly countably infinite) directed graph (V, E) with set V of vertices and set $E \subseteq V \times V$ of edges, a partition $V_0 \cup V_1$ of the set of vertices V and the initial vertex $v_{\text{in}} \in V$. A vertex $v \in V$ is called reachable if there is a path from v_{in} to v. We say that a game graph G is deadlock-free if for every vertex $v \in V$ there is a vertex $v' \in V$ such that $(v, v') \in E$, i.e., every vertex has at least one outgoing edge. For the reason of convenience, in the following, we assume deadlock-free game graphs.

A game $\mathcal{G} = (G, \Omega)$ consists of a game graph G and a winning condition $\Omega \subseteq V^\omega$. A play in $\mathcal{G}$ is built up by moving a token on the game graph G. Initially, the token is placed on v_{in}. If the vertex v where the token is currently located is in V_i, then Player i has to choose an outgoing edge $(v, v') \in E$ and the token is moved to the vertex v'. Thus, a play in $\mathcal{G}$ is an infinite sequence of vertices $\rho \in V^\omega$ such that $\rho(0) = v_{\text{in}}$ and $(\rho(n), \rho(n+1)) \in E$, for all $n \in \mathbb{N}$. The winning condition Ω consists of all plays winning for Player O. A play ρ is winning for Player I if $\rho \in V^\omega \setminus \Omega$.

Let $\text{col} \colon V \to [n]$ be a coloring function assigning to every vertex in V a color from $[n]$, for some $n \in \mathbb{N}$. Analogously to acceptance conditions for automata (cf. 2.2), we call a winning condition Ω parity winning condition if $\Omega = \{\rho \in V^\omega \mid \min\{\text{Inf}(\text{col}(\rho))\} \text{ is even}\}$, it is called weak-parity winning condition if $\Omega = \{\rho \in V^\omega \mid \min\{\text{Occ}(\text{col}(\rho))\} \text{ is even}\}$. Moreover, reachability winning conditions corresponds to E-acceptance and safety winning conditions to A-acceptance conditions. We denote a parity (weak-parity,

reachability or safety) game by $\mathcal{G} = (G, \mathrm{col})$. Furthermore, for $\mathcal{F} \subseteq \mathcal{P}(V)$, A Muller winning condition is defined by $\Omega = \{\rho \in V^\omega \mid \mathrm{Inf}(\rho) \in \mathcal{F}\}$ and a Muller game is denoted by $\mathcal{G} = (G, \mathcal{F})$.

A strategy for Player i is a function $\sigma\colon V^*V_i \to V$ such that for every $w \in V^*V_i$, we have $(\mathsf{last}(w), \sigma(w)) \in E$. A strategy σ is called positional if $\sigma(w) = \sigma(w')$ holds for all $w, w' \in V^*V_i$ with $\mathsf{last}(w) = \mathsf{last}(w')$, i.e., the choice of the next move does not depend on the whole play prefix but only on the current vertex. A play ρ is consistent with a strategy σ for Player i if $\rho(n+1) = \sigma(\mathsf{pref}_{n+1}(\rho))$ for every $n \in \mathbb{N}$ with $\rho(n) \in V_i$. A strategy σ is winning for Player i if every play ρ which is consistent with σ is winning for Player i. We say that Player i wins a game $\mathcal{G}$ if there exists a winning strategy for Player i in $\mathcal{G}$. A game $\mathcal{G}$ is determined if it is won by either of the two players.

Theorem 2.11 ([EJ91], [Mos91])**.** *Parity games are determined with positional winning strategies.*

We will also use the following representations for strategies. Notice, that every play prefix can be described by a sequence of edges instead of a sequence of vertices. Hence, a strategy for Player i can be defined by $\sigma\colon E^* \to V$ such that for every $\eta \in E^+$ with $\mathsf{last}(\eta) = (v, v')$, for some $v \in V$ and $v' \in V_i$, we have $(v', \sigma(\eta)) \in E$, and if $v_{\mathrm{in}} \in V_i$, then $(v_{\mathrm{in}}, \sigma(\varepsilon)) \in E$. Moreover, a strategy for Player i can also be defined by $\sigma\colon E^* \to E$ such that for every $\eta \in E^+$ with $\mathsf{last}(\eta) = (v, v')$, for some $v \in V$ and $v' \in V_i$, we have $\sigma(\eta) = (v', v'')$, for some $v'' \in V$, and if $v_{\mathrm{in}} \in V_i$, then $\sigma(\varepsilon) = (v_{\mathrm{in}}, v'')$, for some $v'' \in V$. Finally, one can define a strategy for Player i by a function $\sigma\colon V^*V_i \to E$ such that for every $w \in V^*V_i$, we have $\sigma(w) = (\mathsf{last}(w), v)$, for some $v \in V$. Notice that all these representations can be converted into each other.

Pushdown Game Graphs

Let $\mathcal{S} = (Q, \Gamma, \Delta, q_{\mathrm{in}})$ be a PDS. The induced pushdown graph $G(\mathcal{S}) = (V, E)$, also called configuration graph, is an infinite directed graph where the set of vertices $V = \{(q, \gamma) \mid q \in Q, \gamma \in \Gamma^*\bot\}$ is the set of configurations, and for two configurations $v, v' \in V$, there exists the edge $(v, v') \in E$ if and only if $v \vdash_{\mathcal{S}} v'$. Notice that $G(\mathcal{S})$ is deadlock-free if $\mathcal{S}$ is deadlock-free.

Now, a partition $Q_0 \cup Q_1$ of the set of states Q induces the game graph $G = (V, V_0, V_1, E, v_{\mathrm{in}})$ which is called pushdown game graph, where $(V, E) = G(\mathcal{S})$, the partition $V_0 \cup V_1$ of the set of configurations V is defined by $V_i = \{(q, \gamma) \in V \mid q \in Q_i\}$, for $i \in \{0, 1\}$, and $v_{\mathrm{in}} = (q_{\mathrm{in}}, \bot)$.

A pushdown game $\mathcal{G} = (G, \Omega)$ consists of a pushdown game graph $G = (V, V_0, V_1, E, v_{\mathrm{in}})$ and a winning condition $\Omega \subseteq V^\omega$. In the most cases we

will consider winning conditions which do not depend on the stack contents but only on the states of configurations, i.e., winning conditions Ω of the following form. For every $\rho \in V^\omega$, if $\rho \in \Omega$ then every $\rho' \in V^\omega$ with $\mathrm{Pr}_0(\rho) = \mathrm{Pr}_0(\rho')$ is also contained in Ω. In such cases, we will also write $\mathrm{Pr}_0(\Omega)$ for the winning condition instead of Ω. Pushdown games with more general winning conditions (which also depend on the stack contents) are studied e.g. in [CDT02], [Ser04] and [Fin05]. In Chapter 3, we will introduce a winning condition which also depends on the stack contents, more precisely on the stack heights.

For a PDS $\mathcal{S} = (Q, \Gamma, \Delta, q_{\mathrm{in}})$ and its induced pushdown game graph $G = (V, V_0, V_1, E, v_{\mathrm{in}})$, let $\mathrm{col}\colon Q \to [n]$ be a coloring function extended to configurations via $\mathrm{col}(v) = \mathrm{col}(\mathrm{Pr}_0(v))$, for every configuration $v \in V$. We refer to a pushdown game $\mathcal{G} = (G, \mathrm{col})$ with a parity (weak-parity, reachability, or safety) winning condition col as a parity (weak-parity, reachability, or safety) pushdown game. Moreover, we refer to a pushdown game $\mathcal{G} = (G, \mathcal{F})$ with a Muller winning condition $\mathcal{F} \subseteq \mathcal{P}(Q)$ as a Muller pushdown game.

2.3.3 Game Reduction, Game Simulation

Let $\mathcal{G} = (G, \Omega)$ and $\mathcal{G}' = (G', \Omega')$ be games on graphs $G = (V, V_0, V_1, E, v_{\mathrm{in}})$ and $G' = (V', V_0', V_1', E', v_{\mathrm{in}}')$ with winning conditions Ω and Ω', respectively.

We say that $\mathcal{G} = (G, \Omega)$ is reducible to $\mathcal{G}' = (G', \Omega')$ if there exists a memory structure $\mathcal{M} = (Q, \delta, q_{\mathrm{in}})$ consisting of a finite set Q of states with initial state q_{in} and an update function $\delta\colon Q \times V \to Q$ such that

1. $V' = V \times Q$, $V_0' = V_0 \times Q$, $V_1' = V_1 \times Q$ and $v_{\mathrm{in}}' = (v_{\mathrm{in}}, q_{\mathrm{in}})$
2. $((v, q), (\overline{v}, \overline{q})) \in E'$ if and only if $(v, \overline{v}) \in E$ and $\delta(q, \overline{v}) = \overline{q}$
3. Player i wins a play ρ in $\mathcal{G}$ if and only if Player i wins the play $\rho' = (v_0, q_0)(v_1, q_1)\cdots$ in $\mathcal{G}'$ induced by ρ such that $(v_0, q_0) = v_{\mathrm{in}}'$ and $(v_n, q_n) = (\rho(n), \delta(\rho(n), q_{n-1}))$, for all $n > 0$.

Game reduction is used to transfer games with complex winning conditions which are hard to handle to games with possibly simpler winning conditions. The drawback which has to be accepted is the growth of the size of the game graphs which get larger by taking the cartesian product with the memory structure. A prominent example is the reduction of Muller games to parity games using the latest appearance record [Tho95].

Now, we formulate a more general notion of game simulation. We say that $\mathcal{G} = (G, \Omega)$ is simulated by $\mathcal{G}' = (G', \Omega')$ if there exist two functions $f\colon V^* V_i \to (V')^* V_i'$ and $g\colon E' \times V \to V$ such that

1. Player i wins $\mathcal{G}$ if and only if Player i wins $\mathcal{G}'$

2. if $\sigma' \colon (V')^* V_i' \to E'$ is a winning strategy for Player i in $\mathcal{G}'$ then $\sigma \colon V^* V_i \to V$ with $\sigma(w) = g(\sigma'(f(w)), \mathsf{last}(w))$, for every play prefix $w \in V^* V_i$, is a winning strategy for Player i in $\mathcal{G}$.

The function f transfers a play prefix $w \in V^* V_i$ in $\mathcal{G}$ into a play prefix $w' \in (V')^* V_i'$ in $\mathcal{G}'$. To deduce a winning strategy σ in $\mathcal{G}$ from a winning strategy σ' in $\mathcal{G}'$ the function g is used which maps an edge in E' (the next move given by σ') and the current vertex from V_i to the vertex which has to be chosen next.

In contrast to game reduction where the size of the game graph increases in the reduction process, the more general game simulation allows to transfer games on large (infinite) game graphs to games on smaller (finite) game graphs. A game simulation transferring parity pushdown games (on infinite pushdown graphs) to parity games on finite game graphs [Wal96, Wal01] will be used in Section 6.

Chapter 3

Pushdown Games and Pushdown Winning Strategies

From the Büchi-Landweber result and the known correspondence between finite automata and monadic second-order logic it follows that Gale-Stewart games with specifications definable in monadic second-order logic can be solved with winning strategies which are again definable in monadic second-order logic. This fact raises the question concerning the conceptual connection between formats of game specifications and corresponding solutions. This problem can be viewed as the following reformulation of Church's Problem focusing on the formats of the game specifications and their solutions. Given a winning condition L in a specific format, i.e., L is an ω-language from a particular class recognizable by a certain type of automata or definable in some specific logic, it is to decide whether there exist a winning strategy σ for the winner such that σ can be implemented in the same format as the specification L, i.e., σ is definable in the same logic or realizable by a transducer of the same type. This relation between formats of game specifications and corresponding solutions has been analyzed for several regular classes. Selivanov established a tight connection for the class of aperiodic regular languages [Sel07]. He showed that games with aperiodic regular winning conditions are determined with winning strategies realizable by aperiodic transducers. Rabinovich and Thomas established analogous result for a number of sublogics of the monadic second-order logic [RT07], among them first-order logic over $(\mathbb{N}, <)$, the extension of first-order logic over $(\mathbb{N}, <)$ by modular counting quantifiers, first order logic over $(\mathbb{N}, S)$ with successor relation S and the quantifier-free first-order logic over $(\mathbb{N}, 0, +1)$ called strictly bounded logic. Moreover, examples of logics where winning strategies of the same format don't suffice were exhibited. Chaturvedi et al. studied subclasses of star-free regular languages in [COT11] where piecewise testable languages and languages from the dot-depth hierarchy are considered.

In this chapter, we extend these results to contextfree specifications, hence, we address the relation between the formats of winning conditions and corresponding winning strategies for contextfree Gale-Stewart games as well as parity games played over pushdown game graphs. We consider several classes of contextfree specifications which we introduce in Section 3.1. We establish a tight connection between the formats of specifications and their solutions for a number of cases. Furthermore, we also present some cases where this correspondence fails. To prove this results, which we state in Section 3.2, we first recall the technique of Kupferman and Vardi [KV00a] in Subsection 3.2.1. The proofs are presented in Subsection 3.2.2.

3.1 Classes of Contextfree Languages

In this section we define several classes of contextfree $*$-languages and contextfree ω-languages. Various number of such classes is conceivable which can be defined by different kinds of pushdown automata recognizing those classes. We distinguish the types of pushdown automata by several properties of the underlying pushdown machine on the one hand and by the underlying acceptance conditions on the other hand.

For pushdown automata on finite words we consider acceptance by final states, and for those on infinite words we consider parity acceptance, as defined in Section 2.2.2. Beyond that, we introduce a new kind of acceptance conditions for ω-pushdown automata, so-called stair acceptance conditions [LMS04]. Notice that this notion is also naturally carried over to winning conditions for games on pushdown game graphs, called stair winning conditions.

Intuitively, for a finite or infinite path through a configuration graph, a configuration is said to be a stair configuration if no subsequent configuration of smaller stack height exists in this path.

Definition 3.1 (Stairs)**.** Let $V = Q \times \Gamma^* \bot$ be a set of configurations over some set Q of states and a pushdown alphabet Γ. Define the functions $\mathsf{StairPositions}\colon V^+ \cup V^\omega \to 2^{\mathbb{N}}$ and $\mathsf{Stairs}\colon V^+ \cup V^\omega \to V^+ \cup V^\omega$ by

$$\mathsf{StairPositions}(w) = \{n \in \mathbb{N} \mid \forall m \geq n : \mathsf{sh}(w(m)) \geq \mathsf{sh}(w(n))\},$$

and $\mathsf{Stairs}(w) = w(n_0)w(n_1)\cdots$, where $n_0 < n_1 < \cdots$ is the ascending enumeration of $\mathsf{StairPositions}(w)$, for $w \in V^+ \cup V^\omega$.

Using this notion we now define stair acceptance conditions for pushdown automata (and stair winning conditions for games on pushdown game graphs). Let $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}})$ be a PDM. A stair parity acceptance condition is given by a coloring function $\text{col}\colon Q \to [n]$, for some $n \in \mathbb{N}$. A

run ρ of the stair parity pushdown automaton (parity-StPDA) $\mathcal{P} = (\mathcal{M}, \mathrm{col})$ on a word $w \in \Sigma^\omega$ is accepting if $\mathsf{Stairs}(\rho)$ satisfies the parity condition given by col, i.e., if the minimal color seen infinitely often in $\mathsf{Stairs}(\rho)$ is even.

Analogously, a pushdown game $\mathcal{G} = (G, \mathrm{col})$ over a pushdown game graph $G = (V, V_0, V_1, E, v_{\mathrm{in}})$ with a stair parity winning condition given by the coloring function col is called stair parity pushdown game. A play $\rho \in V^\omega$ of $\mathcal{G}$ is winning for Player O if and only if $\mathsf{Stairs}(\rho)$ satisfies the parity condition given by col.

Now, we define several types of pushdown machines by considering different restrictions of the general model of (nondeterministic) PDM. In the previous chapter we already introduced one restriction, namely deterministic pushdown machines (DPDM) where the transition relation is restricted such that it is a (partial) transition function.

We continue by defining input driven PDM where the input symbol determines whether a push-, pop- or skip-transition is performed. Such PDM are called visibly pushdown machines (VPM) [AM04]. Intuitively, the transition relation is restricted such that for every input letter $a \in \Sigma$, all transitions in the transition relation processing a are either push-, pop- or skip-transitions.

Definition 3.2 (Visibly PDM)**.** A PDM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_{\mathrm{in}})$ is visibly (VPM) if there are a partition $\Sigma_{\mathrm{push}} \cup \Sigma_{\mathrm{pop}} \cup \Sigma_{\mathrm{skip}}$ of Σ and relations $\Delta_{\mathrm{push}} \subseteq Q \times \Sigma_{\mathrm{push}} \times Q \times \Gamma$, $\Delta_{\mathrm{pop}} \subseteq Q \times \Sigma_{\mathrm{pop}} \times \Gamma \times Q$ and $\Delta_{\mathrm{skip}} \subseteq Q \times \Sigma_{\mathrm{skip}} \times Q$ such that the transition relation Δ is of the following form. For all $X \in \Gamma_\perp$,

- $(q, X, a, q', AX) \in \Delta$ if and only if $(q, a, q', A) \in \Delta_{\mathrm{push}}$,
- $(q, A, a, q', \varepsilon), (q, \perp, a, q', \perp) \in \Delta$ if and only if $(q, a, A, q') \in \Delta_{\mathrm{pop}}$, and
- $(q, X, a, q', X) \in \Delta$ if and only if $(q, a, q') \in \Delta_{\mathrm{skip}}$.

We call such an alphabet $\Sigma = \Sigma_{\mathrm{push}} \cup \Sigma_{\mathrm{pop}} \cup \Sigma_{\mathrm{skip}}$, which is partitioned into three disjoint alphabets, a visibly pushdown alphabet. Symbols from Σ_{push} are called *call*-symbols, symbols from Σ_{pop} are called *return*-symbols, and Σ_{skip} contains *internal actions*. This notions originate from the context of program analysis which was the first motivation for the introduction of VPM to model recursive procedure calls.

When reading a *call*-symbol $a \in \Sigma_{\mathrm{push}}$, a VPM has to push a stack symbol $A \in \Gamma$ onto the stack which depends only on the input symbol a and the current state $q \in Q$, i.e., regardless of which symbol is on the top of the stack. *Internal actions* $a \in \Sigma_{\mathrm{skip}}$ do not affect the stack which remains untouched. When processing a *return*-symbol $a \in \Sigma_{\mathrm{pop}}$, a VPM has to pop the top stack symbol from the stack if the stack is not empty. In case of an empty stack, a *return*-symbol $a \in \Sigma_{\mathrm{pop}}$ is treated as an *internal action*.

Definition 3.3 (Realtime PDM)**.** A PDM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}})$ is called realtime (rt-PDM) if $\Delta \subseteq Q \times \Gamma_\perp \times \Sigma \times Q \times \Gamma_\perp^{\leq 2}$.

That means, the transition relation of a rt-PDM is restricted such that it contains no ε-transitions. Notice that every VPM is realtime. For the next definitions, the pushdown alphabet of a PDM is restricted.

Definition 3.4 (One-Counter PDM)**.** A PDM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}})$ is a one-counter machine (1CM) if $|\Gamma| = 1$.

Definition 3.5 (Blind 1CM)**.** A 1CM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}})$ is called blind one-counter (B1CM) if the following statement holds. For all $q, q' \in Q$ and all $a \in \Sigma_\varepsilon$, if $(q, \perp, a, q', A^n\perp) \in \Delta$, for $0 \leq n \leq 1$, then $(q, A, a, q', A^nA) \in \Delta$.

That means, that for B1CM every transition that is enabled with empty stack is also enabled with nonempty stack. Thus, a B1CM cannot check whether or not its stack is empty. In the literature this type of 1CM is sometimes also referred to as partially blind one-counter.

Now, using these definitions of PDM, several types of corresponding pushdown automata can be defined. A PDA $\mathcal{P} = (\mathcal{M}^\mathcal{P}, F)$ is a

- DPDA if $\mathcal{M}^\mathcal{P}$ is a DPDM,
- rt-DPDA if $\mathcal{M}^\mathcal{P}$ is a rt-DPDM,
- VPA if $\mathcal{M}^\mathcal{P}$ is a VPM,
- DVPA if $\mathcal{M}^\mathcal{P}$ is a DPDM and VPM,
- 1CA if $\mathcal{M}^\mathcal{P}$ is a 1CM,
- D1CA if $\mathcal{M}^\mathcal{P}$ is a DPDM and 1CM,
- V1CA if $\mathcal{M}^\mathcal{P}$ is a VPM and 1CM,
- DV1CA if $\mathcal{M}^\mathcal{P}$ is a DPDM, VPM and 1CM,
- B1CA if $\mathcal{M}^\mathcal{P}$ is a B1CM,
- DB1CA if $\mathcal{M}^\mathcal{P}$ is a DPDM and B1CM.

Analogously, for ω-pushdown automata, a parity-PDA (parity-StPDA) $\mathcal{P} = (\mathcal{M}^\mathcal{P}, \text{col})$ is a

- parity-DPDA (parity-StDPDA) if $\mathcal{M}^\mathcal{P}$ is a DPDM,
- parity-rt-DPDA (parity-rt-StDPDA) if $\mathcal{M}^\mathcal{P}$ is a rt-DPDM,
- parity-VPA (parity-StVPA) if $\mathcal{M}^\mathcal{P}$ is a VPM,

- parity-DVPA (parity-StDVPA) if $\mathcal{M}^{\mathcal{P}}$ is a DPDM and VPM,
- parity-1CA (parity-St1CA) if $\mathcal{M}^{\mathcal{P}}$ is a 1CM,
- parity-D1CA (parity-StD1CA) if $\mathcal{M}^{\mathcal{P}}$ is a DPDM and 1CM,
- parity-V1CA (parity-StV1CA) if $\mathcal{M}^{\mathcal{P}}$ is a VPM and 1CM,
- parity-DV1CA (parity-StDV1CA) if $\mathcal{M}^{\mathcal{P}}$ is a DPDM, VPM and 1CM,
- parity-B1CA (parity-StB1CA) if $\mathcal{M}^{\mathcal{P}}$ is a B1CM,
- parity-DB1CA (parity-StDB1CA) if $\mathcal{M}^{\mathcal{P}}$ is a DPDM and B1CM.

Notice that for all these types of pushdown automata except deterministic blind one-counter, an equivalent non-blocking pushdown automaton of the same type can be constructed. This can be shown analogously as in Lemma 2.3 and Lemma 2.6. For deterministic blind one-counter the ability to block computations with an empty stack is essential for rejecting input words. This cannot be resolved by adding a new rejecting sink state, since a deterministic blind one-counter cannot distinguish between empty and nonempty stack, hence configurations with stack heights greater zero might also lead to the rejecting sink state.

Remark 3.6.

1. For every PDA (parity-PDA, parity-StPDA) $\mathcal{P}$ of one of the above types except DB1CA (parity-DB1CA, parity-StDB1CA), there is a PDA (parity-PDA, parity-StPDA) $\mathcal{P}'$ of the same type as $\mathcal{P}$, having the continuity property, such that $L(\mathcal{P}) = L(\mathcal{P}')$.
2. There is a DB1CA (parity-DB1CA, parity-StDB1CA) $\mathcal{P}$ such that for every DB1CA (parity-DB1CA, parity-StDB1CA) $\mathcal{P}'$ with continuity property $L(\mathcal{P}) \neq L(\mathcal{P}')$.

Now, we fix our notation for classes of languages of finite and infinite words, respectively, recognized by pushdown automata of particular types presented above and state some results concerning their inclusion relations. In Chapter 2 we already introduced the classes of (nondeterministic) contextfree and deterministic contextfree languages of finite words CFL and DCFL, as well as for infinite words CFL_ω and DCFL_ω. We denote the class of languages recognized by parity-StPDA by StCFL_ω and those recognized by parity-StDPDA by StDCFL_ω.

The following two lemmata show that for nondeterministic pushdown automata, stair parity acceptance condition does not increase the power of the automata compared to parity acceptance condition. However, this is not the case for deterministic pushdown automata.

Lemma 3.7.

1. *For each parity-PDA $\mathcal{P}$ one can construct a parity-StPDA $\mathcal{P}'$ such that $L(\mathcal{P}) = L(\mathcal{P}')$.*
2. *For each parity-StPDA $\mathcal{P}$ one can construct a parity-PDA $\mathcal{P}'$ such that $L(\mathcal{P}) = L(\mathcal{P}')$.*

Proof. We prove only the second statement, the first statement can be shown similar to Lemma 3.9.

Let $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \text{col})$ be a parity-StPDA with the coloring function $\text{col}\colon Q \to [n]$ where we assume $\text{Par}(n) = 0$, without loss of generality. The idea for the construction of an equivalent parity-PDA $\mathcal{P}'$ is as follows. For a run ρ of $\mathcal{P}$, an equivalent parity-PDA $\mathcal{P}'$ guesses (and verifies) the positions from $\mathsf{StairPositions}(\rho)$. Moreover, the parity condition col' has to be defined such that exactly the colors of the stair positions contribute to the evaluation, i.e., a run ρ' of $\mathcal{P}'$ satisfies the parity condition col' if its stair positions are guessed correctly and the corresponding run ρ of $\mathcal{P}$ satisfies the stair parity condition col.

Formally, $\mathcal{P}' = (Q', \Sigma, \Gamma', \Delta', q_{\text{in}}, \text{col}')$ with $Q' = Q \cup \overline{Q} \cup \{q_{\mathsf{rej}}\}$ where $\overline{Q} = \{\overline{q} \mid q \in Q\}$ and $\Gamma' = \Gamma \cup \overline{\Gamma}$ where $\overline{\Gamma} = \{\overline{A} \mid A \in \Gamma\}$. A configuration $(q, \gamma) \in Q' \times (\Gamma')^* \bot$ is intended to indicate a stair configuration if $q \in Q$ and $\gamma(0) \in \Gamma_\bot$, whereas if $q \in \overline{Q}$ and $\gamma(0) \in \overline{\Gamma}$ it shall indicate a non-stair configuration. For this purpose, we define the following transitions to be contained in Δ'. For every push-transition $(q, X, a, p, YZ) \in \Delta$, where $X, Y, Z \in \Gamma_\bot$ and $a \in \Sigma_\varepsilon$, Δ' contains transitions

$$(q, X, a, p, YZ), \quad (q, X, a, \overline{p}, \overline{Y}Z) \text{ and } (\overline{q}, \overline{X}, a, \overline{p}, \overline{YZ}).$$

That means, being in a stair configuration the automaton can decide either to proceed to a stair configuration or to a non-stair configuration. And being in a non-stair configuration, the only possibility is to proceed to a further non-stair configuration. For every skip-transition $(q, X, a, p, Y) \in \Delta$, where $X, Y \in \Gamma_\bot$ and $a \in \Sigma_\varepsilon$, transitions

$$(q, X, a, p, Y) \text{ and } (\overline{q}, \overline{X}, a, \overline{p}, \overline{Y})$$

are contained in Δ', which means that the successor configuration is a stair configuration if and only if the current configuration is a stair configuration. Moreover, for every pop-transition $(q, X, a, p, \varepsilon) \in \Delta$, where $X \in \Gamma_\bot$ and $a \in \Sigma_\varepsilon$, we define

$$(\overline{q}, \overline{X}, a, \overline{p}, \varepsilon) \text{ and } (\overline{q}, \overline{X}, a, p, \varepsilon)$$

to be contained in Δ'. A pop-transition can only be performed from a non-stair configuration, where the automaton has to decide whether it will

proceed to a stair or a non-stair configuration. If this decision turns out to be incorrect the automaton proceeds to the rejecting state q_{rej}. Hence, for every $q \in Q$, $a \in \Sigma_\varepsilon$, $X \in \Gamma_\perp$ and $X' \in \Gamma'_\perp$, Δ' contains transitions

$$(q, \overline{X}, a, q_{\mathsf{rej}}, \overline{X}), \ \ (\overline{q}, X, a, q_{\mathsf{rej}}, X) \text{ and } (q_{\mathsf{rej}}, X', a, q_{\mathsf{rej}}, X').$$

Finally, we define the coloring function $\mathrm{col}' \colon Q' \to [n]$ by

$$\mathrm{col}(q') = \begin{cases} \mathrm{col}(q') & \text{if } q' \in Q, \\ n-1 & \text{otherwise,} \end{cases}$$

for $q' \in Q'$, which ensures that only stair configurations contribute to the acceptance. By this construction, for every word $w \in \Sigma^\omega$, we have the existence of an accepting run ρ of $\mathcal{P}$ on w implies the existence of an accepting run ρ' of $\mathcal{P}'$ on w. □

Corollary 3.8. $\mathsf{CFL}_\omega = \mathsf{StCFL}_\omega$.

Lemma 3.9. *For each parity-DPDA $\mathcal{P}$, a parity-StDPDA $\mathcal{P}'$ can be constructed such that $L(\mathcal{P}) = L(\mathcal{P}')$.*

Proof. The idea is to pass the minimal color occurring between two consecutive stair positions to the following stair configuration. For this, we extend the stack symbols by a further component to store the minimal color seen since the last position of a smaller stack height. Moreover, to be able to recolor states we also introduce a further component for the states.

Let $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_{\mathrm{in}}, \mathrm{col})$ be a parity-DPDA with $\mathrm{col} \colon Q \to [n]$, for some $n > 0$. Define the parity-StDPDA $\mathcal{P}' = (Q', \Sigma, \Gamma', \delta', q'_{\mathrm{in}}, \mathrm{col}')$ as follows. The stack alphabet is defined by $\Gamma' = \Gamma \times [n]$, and the set of states by $Q' = S \cup \overline{S}$ where $S = Q \times [n]$ and $\overline{S} = \{\overline{s} \mid s \in S\}$, the initial state is $q'_{\mathrm{in}} = (q_{\mathrm{in}}, \mathrm{col}(q_{\mathrm{in}}))$. States from $\overline{S}$ are auxiliary, and serve to update the top of the stack after a pop-transition. We define the coloring function $\mathrm{col}' \colon Q' \to [n]$ by $\mathrm{col}'(q') = \mathrm{Pr}_{[n]}(q')$, for every $q' \in Q'$.

The transition function δ' is defined as follows. For every $q, p \in Q$, $X, Y, Z \in \Gamma_\perp$, and $a \in \Sigma_\varepsilon$, if $\delta(q, X, a) = (p, YZ)$ then for every $c, d \in [n]$,

$$\delta'(\langle q, c\rangle, \langle X, d\rangle, a) = (\langle p, \mathrm{col}(p)\rangle, \langle Y, \mathrm{col}(p)\rangle \langle Z, d\rangle),$$

i.e., a push-transition is performed by updating the first component of the top of the stack (while the second component remains untouched) and by pushing a new symbol onto the stack where the second component is initialized by the color of the reached state. Obviously, this state is not recolored. If $\delta(q, X, a) = (p, Y)$ then for every $c, d \in [n]$,

$$\delta'(\langle q, c\rangle, \langle X, d\rangle, a) = (\langle p, \mathrm{col}(p)\rangle, \langle Y, \min\{d, \mathrm{col}(p)\}\rangle),$$

i.e., when performing a skip-transition the second component of the top stack symbol has to be updated to the minimum of the color stored so far and the color of the reached state. Finally, if $\delta(q, X, a) = (p, \varepsilon)$ then for every $c, d, e \in [n]$,

$$\delta'(\langle q, c\rangle, \langle X, d\rangle, a) = (\overline{\langle p, d\rangle}, \varepsilon) \text{ and}$$

$$\delta'(\overline{\langle p, d\rangle}, \langle Z, e\rangle, \varepsilon) = (\langle p, \min\{d, \mathrm{col}(p)\}\rangle, \langle Z, \min\{d, e, \mathrm{col}(p)\}\rangle).$$

That means, that in case of a pop-transition, the top stack symbol is deleted from the stack while its color (stored in the second component) is passed down to the reached state which is appropriately recolored. The new top stack symbol also has to be updated such that it comprises the minimal color seen since the last position of a smaller stack height. This is done using auxiliary states from $\overline{S}$. By this construction we have for every word $w \in \Sigma^\omega$, the unique runs ρ of $\mathcal{P}$ on w and ρ' of $\mathcal{P}'$ on w satisfy $\min\{\mathrm{Inf}(\mathrm{col}(\rho))\} = \min\{\mathrm{Inf}(\mathrm{col}(\mathsf{Stairs}(\rho')))\}$. □

Corollary 3.10. $\mathsf{DCFL}_\omega \subseteq \mathsf{StDCFL}_\omega$.

Now, we consider visibly pushdown languages. Visibly pushdown languages of finite words are denoted by VPL and DVPL recognized by VPA or DVPA respectively, and classes of visibly pushdown ω-languages are denoted by VPL_ω, DVPL_ω, StVPL_ω, StDVPL_ω which are recognized by parity-VPA, parity-DVPA, parity-StVPA or parity-StDVPA, respectively. In contrast to general pushdown automata on finite words, visibly pushdown automata on finite words can be determinized.

Lemma 3.11 ([AM04])**.** *For each VPA $\mathcal{P}$ over a partitioned alphabet $\Sigma = \Sigma_{\mathrm{push}} \cup \Sigma_{\mathrm{pop}} \cup \Sigma_{\mathrm{skip}}$ one can construct a DVPA $\mathcal{P}'$ over Σ with the same partition $\Sigma_{\mathrm{push}} \cup \Sigma_{\mathrm{pop}} \cup \Sigma_{\mathrm{skip}}$ such that $L(\mathcal{P}) = L(\mathcal{P}')$.*

Corollary 3.12. $\mathsf{DVPL} = \mathsf{VPL}$.

However, the above lemma does not hold for visibly pushdown automata on infinite words with parity acceptance conditions. The expressive power of parity-VPA exceeds the expressive power of parity-DVPA.

Lemma 3.13 ([AM04])**.** $\mathsf{DVPL}_\omega \subsetneq \mathsf{VPL}_\omega$.

Nevertheless, a determinization procedure is presented in [LMS04] which takes advantage of stair acceptance.

Lemma 3.14 ([LMS04])**.** *For each parity-VPA $\mathcal{P}$ over a partitioned alphabet $\Sigma = \Sigma_{\mathrm{push}} \cup \Sigma_{\mathrm{pop}} \cup \Sigma_{\mathrm{skip}}$ one can construct a parity-StDVPA $\mathcal{P}'$ over Σ with the same partition $\Sigma_{\mathrm{push}} \cup \Sigma_{\mathrm{pop}} \cup \Sigma_{\mathrm{skip}}$ such that $L(\mathcal{P}) = L(\mathcal{P}')$.*

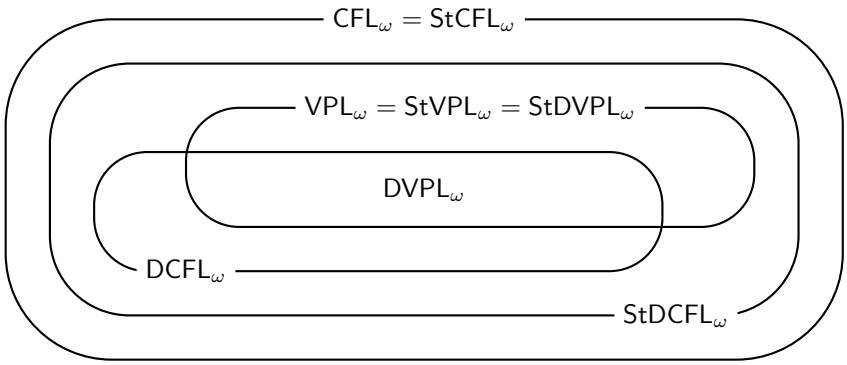


Figure 3.1: Classes of visibly and non-visibly pushdown languages

Proposition 3.15. $\mathsf{VPL}_\omega = \mathsf{StDVPL}_\omega = \mathsf{StVPL}_\omega$.

Proof. Inclusion $\mathsf{VPL}_\omega \subseteq \mathsf{StDVPL}_\omega$ follows from Lemma 3.14. Moreover, inclusion $\mathsf{StDVPL}_\omega \subseteq \mathsf{StVPL}_\omega$ obviously holds, and $\mathsf{StVPL}_\omega \subseteq \mathsf{VPL}_\omega$ is shown in exactly the same way as Lemma 3.7 by exploiting nondeterminism. □

Finally, the following remark clarifies the relationship between visibly and non-visibly pushdown languages of infinite words. The complete picture is depicted in Figure 3.1.

Remark 3.16.

(i) $\mathsf{DCFL}_\omega \setminus \mathsf{VPL}_\omega \neq \emptyset$,

(ii) $\mathsf{VPL}_\omega \setminus \mathsf{DCFL}_\omega \neq \emptyset$,

(iii) $\mathsf{VPL}_\omega \cup \mathsf{DCFL}_\omega \subsetneq \mathsf{StDCFL}_\omega$.

Proof. (i) Consider the language $L_1 = \{a^n b a^n b w \mid n > 0, w \in \{a,b\}^\omega\}$ over the alphabet $\Sigma = \{a,b\}$. Obviously, there is a parity-DPDA recognizing L_1, hence, $L_1 \in \mathsf{DCFL}_\omega$. However, for any partition of Σ into $\Sigma_{\text{push}} \cup \Sigma_{\text{pop}} \cup \Sigma_{\text{skip}}$, there is no parity-VPA recognizing L_1. If $a \in \Sigma_{\text{push}}$ then the information stored on the stack after processing a prefix $a^n b$ cannot be used while counting the number of the following a's, as the automaton is required to perform push-transitions. Otherwise, if $a \in \Sigma_{\text{pop}} \cup \Sigma_{\text{skip}}$, then after processing a prefix $a^n b$, there is not enough information on the stack for being able to compare the numbers of a's.

(ii) Consider the following language $L_2 \subseteq \{a,b\}^\omega$ with $w \in L_2$ if and only if there exists $k \in \mathbb{N}$ such that $|\mathsf{pref}_n(w)|_a - |\mathsf{pref}_n(w)|_b = k$, for infinitely many $n \in \mathbb{N}$, and $|\mathsf{pref}_n(w)|_a \geq |\mathsf{pref}_n(w)|_b$, for every $n \in \mathbb{N}$. A parity-StDVPA over the alphabet $\Sigma = \{a,b\}$ with $\Sigma_{\text{push}} = \{a\}$, $\Sigma_{\text{pop}} = \{b\}$

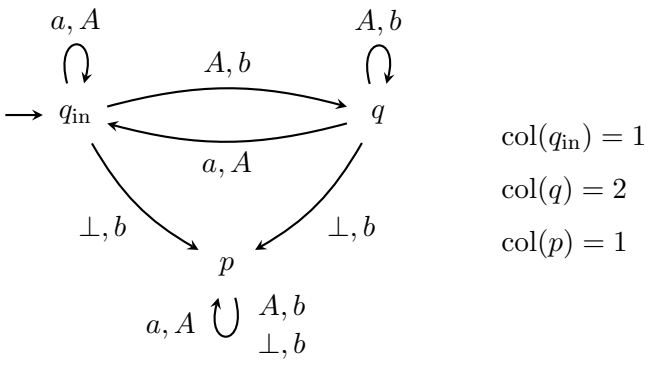


Figure 3.2: Parity-StDVPA recognizing $L_2 \in \mathsf{VPL}_\omega \setminus \mathsf{DCFL}_\omega$

and $\Sigma_{\text{skip}} = \emptyset$ recognizing L_2 (which can be converted into a parity-VPA according to Proposition 3.15) is depicted in Figure 3.2. The automaton pushes the pushdown symbol A onto the stack whenever it reads a and it pops A if it reads b. When reading b with an empty stack the automaton goes to the rejecting sink state p, since in this case $|\mathsf{pref}_n(w)|_a < |\mathsf{pref}_n(w)|_b$. Otherwise, every time the automaton reads a it proceeds to q_{in} and it proceeds to q if it reads b. A run ρ on a word w is accepting if there are only finitely many stair configurations in state q_{in} in ρ. This happens if there is a $k \in \mathbb{N}$ such that the stack height drops down to k again and again during the run, i.e. $\mathsf{sh}(\rho(n)) = k$ for infinitely many $n \in \mathbb{N}$. Which means that $|\mathsf{pref}_n(w)|_a - |\mathsf{pref}_n(w)|_b = k$, for infinitely many $n \in \mathbb{N}$.

On the other hand, one can show that the language L_2 is on a higher Borel level than every deterministic contextfree ω-language. More precisely, L_2 is a true $\mathbf{\Sigma}_3$-set, i.e., L_2 is in $\mathbf{\Sigma}_3$ but not in $\mathbf{\Pi}_3$ [CDT02]. However, any deterministic contextfree language $L \in \mathsf{DCFL}_\omega$ is contained in $\mathcal{B}(\mathbf{\Sigma}_2)$ [CG77a, CG77b], which implies $L_2 \notin \mathsf{DCFL}_\omega$, since $\mathcal{B}(\mathbf{\Sigma}_2) \subsetneq \mathbf{\Pi}_3$.

(iii) Let $L_3 = L_1 \cap L_2$. Obviously, $L_3 \notin \mathsf{DCFL}_\omega$ as well as $L_3 \notin \mathsf{VPL}_\omega$. However, it is easy to construct a parity-StDPDA recognizing L_3. First, the automaton checks whether a word starts with a prefix $a^n b a^n b$, for some $n > 0$. If this is not the case it proceeds to a rejecting sink state. Otherwise, it continues in exactly the same way as the parity-StDVPA recognizing L_2. □

We define further subclasses of DCFL_ω and StDCFL_ω by considering deterministic one-counter and realtime pushdown automata. We will fix our notation only for deterministic classes. By $\mathsf{D1CL}$ we denote deterministic one-counter languages recognized by D1CA and by $\mathsf{D1CL}_\omega$ and $\mathsf{StD1CL}_\omega$ one-counter ω-languages are denoted, which are recognized by parity-D1CA and parity-StD1CA. Visibly one-counter languages recognized by DV1CA

are denoted by DV1CL, and languages recognized by parity-DV1CA, parity-StDV1CA are denoted by $\mathsf{DV1CL}_\omega$ and $\mathsf{StDV1CL}_\omega$, respectively. Finally, we denote blind one-counter languages recognized by DB1CA by DB1CL, and languages recognized by parity-DB1CA, parity-StDB1CA are denoted by $\mathsf{DB1CL}_\omega$ and $\mathsf{StDB1CL}_\omega$, respectively. Classes of languages recognized by realtime deterministic pushdown automata are denoted by rt-DCFL, rt-DCFL_ω and rt-StDCFL_ω.

3.2 Formats of Contextfree Winning Conditions and of Corresponding Winning Strategies

After we presented various number of classes of contextfree languages, we consider contextfree games defined by those classes and study the connection between the formats of winning conditions and the corresponding winning strategies. We consider both Gale-Stewart games with contextfree winning conditions as well as games on pushdown game graphs. For the former only deterministic winning conditions are considered, as in general Gale-Stewart games with (nondeterministic) contextfree winning conditions cannot be solved.

Theorem 3.17 ([Fin01]). *For contextfree languages $L \in \mathsf{CFL}_\omega$, it is undecidable to determine which player has a winning strategy in $\Gamma(L)$.*

The proof uses the known undecidability of the universality problem for contextfree languages, which is to decide, given a parity-PDA $\mathcal{P}$ over an alphabet Σ, whether $L(\mathcal{P}) = \Sigma^\omega$.

On the other hand Walukiewicz showed that parity pushdown games played on pushdown game graphs can be simulated by parity games on finite game graphs from which deterministic pushdown winning strategies can be deduced. We will use this construction in Chapter 6.

Theorem 3.18 ([Wal96]). *Parity pushdown games are determined with deterministic pushdown winning strategies.*

This result directly implies the solvability of deterministic contextfree Gale-Stewart games since they can be modeled by pushdown games on (labeled) pushdown game graphs. Let $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_{\text{in}}, \text{col})$ be a parity-DPDA having the continuity property over an alphabet $\Sigma = \Sigma_I \times \Sigma_O$ with the coloring function $\text{col}\colon Q \to [n]$, for some $n \in \mathbb{N}$, and $\Gamma(L(\mathcal{P}))$ be the Gale-Stewart game defined by the language recognized by $\mathcal{P}$. The corresponding pushdown game on a pushdown game graph is obtained as follows. Define DPDM $\mathcal{M}^{\mathcal{P}} = (Q', \Sigma \cup \Sigma_I, \Gamma, \delta', q_{\text{in}})$ where $Q' = Q \cup (Q \times \Sigma_I)$ and for all $a \in \Sigma_I$, $b \in \Sigma_O$, $q, p \in Q$, $A \in \Gamma_\perp$ and $\alpha \in \Gamma_\perp^{\leq 2}$,

- if $\delta(q, A, \binom{a}{b}) = (p, \alpha)$ then
  $\delta'(q, A, a) = (\langle q, a \rangle, A)$ and $\delta'(\langle q, a \rangle, A, \binom{a}{b}) = (p, \alpha)$, and
- if $\delta(q, A, \varepsilon) = (p, \alpha)$ then $\delta'(q, A, \varepsilon) = (p, \alpha)$.

Moreover, let PDS $\mathcal{S}^{\mathcal{P}} = (Q', \Gamma, \Delta, q_{\text{in}})$ and the labeling function $\lambda\colon \Delta \to \mathcal{P}(\Sigma \cup \Sigma_I)$ be as follows. A transition (q, A, p, α) is contained in Δ if and only if there is $a \in \Sigma_\varepsilon \cup \Sigma_I$ such that $\delta'(q, A, a) = (p, \alpha)$. Furthermore, for every transition $t = (q, A, p, \alpha) \in \Delta$, $a \in \lambda(t)$ if $\delta'(q, A, a) = (p, \alpha)$. This means that $\mathcal{S}^{\mathcal{P}}$ is obtained from the DPDM $\mathcal{M}^{\mathcal{P}}$ by outsourcing the input alphabet of $\mathcal{M}^{\mathcal{P}}$ to the labeling function.

This construction subdivides the processing of a pair $\binom{a}{b} \in \Sigma$ into two steps where in the first step Player I can choose $a \in \Sigma_I$ and in the second step Player O completes the pair by choosing $b \in \Sigma_O$. Hence, we define the partition of the set of states Q' such that $Q_0 = Q \times \Sigma_I$ and $Q_1 = Q$. Notice, that since $\mathcal{P}$ has the continuity property, the pushdown game graph $G(\mathcal{S}^{\mathcal{P}})$ is of the following form. For each configuration $(q, \gamma) \in Q_1 \times \Gamma^*\bot$, either there is exactly one outgoing edge labeled by ε or there are $|\Sigma_I|$ many outgoing edges, namely one outgoing edge labeled by a for each $a \in \Sigma_I$. Furthermore, for each configuration $(\langle q, a \rangle, \gamma) \in Q_0 \times \Gamma^*\bot$, there exists exactly one outgoing edge containing $\binom{a}{b}$ in its labeling, for every $b \in \Sigma_O$.

Finally, we define the coloring function $\text{col}'\colon Q' \to [n]$ by $\text{col}'(q) = \text{col}(q)$ if $q \in Q_1$ and $\text{col}'(q) = n - 1$ if $q \in Q_0$. Thus, we obtain a parity pushdown game $(G(\mathcal{S}^{\mathcal{P}}), \text{col}')$ such that $\Gamma(L(\mathcal{P}))$ is simulated by $(G(\mathcal{S}^{\mathcal{P}}), \text{col}')$ and a winning strategy σ' in $(G(\mathcal{S}^{\mathcal{P}}), \text{col}')$ instantly implements a winning strategy σ in $\Gamma(L(\mathcal{P}))$.

Corollary 3.19. *Deterministic contextfree Gale-Stewart games are determined with deterministic pushdown winning strategies.*

We extend this result and establish a tight connection between the formats of contextfree winning conditions on the one hand and of the corresponding contextfree winning strategies on the other hand. To state the notion of a format of a winning strategy more precisely, recall that we use pushdown transducers as defined in Section 2.2.2 to define pushdown strategies. Analogously, as for pushdown automata, we define the format of a pushdown transducer by the format of its underlying PDM. Notice that a pushdown transducer $\mathcal{T} = (Q, \Sigma_I, \Sigma_O, \Gamma, \Delta, q_{\text{in}}, \lambda)$ can also be represented by a collection of PDA $(\mathcal{A}_a)_{a \in \Sigma_O}$ of the same format as $\mathcal{T}$ where $\mathcal{A}_a = (Q, \Sigma_I, \Gamma, \Delta, q_{\text{in}}, F_a)$ with $F_a = \{q \in Q \mid \lambda(q) = a\}$. Therefore, we associate classes of contextfree $*$-languages with pushdown transducers according to their formats.

Theorem 3.20. *Let $\mathcal{G}$ be a parity or a stair parity pushdown game played on the (labeled) pushdown game graph defined by a PDM $\mathcal{P}$.*

1. *$\mathcal{G}$ is determined with* DCFL *winning strategies if $\mathcal{P}$ is DPDM,*
2. *$\mathcal{G}$ is determined with* DVPL *winning strategies if $\mathcal{P}$ is DVPM,*
3. *$\mathcal{G}$ is determined with rt-*DCFL *winning strategies if $\mathcal{P}$ is rt-DPDM,*
4. *$\mathcal{G}$ is determined with* D1CL *winning strategies if $\mathcal{P}$ is D1CM.*

Notice that for deterministic contextfree Gale-Stewart games the following directly follows from this result.

Corollary 3.21.

1. *DCFL_ω- and StDCFL_ω-games are determined with winning strategies from* DCFL*,*
2. *DVPL_ω- and StDVPL_ω-games are determined with winning strategies from* DVPL*,*
3. *rt-DCFL_ω- and rt-StDCFL_ω-games are determined with winning strategies from rt-*DCFL*,*
4. *$\mathsf{D1CL}_\omega$- and $\mathsf{StD1CL}_\omega$-games are determined with winning strategies from* D1CL*.*

This is due to the fact that by the above construction a Gale-Stewart game given by a pushdown automaton $\mathcal{P}$ (of one of the formats mentioned in the corollary) can easily be reduced to a pushdown game on a pushdown game graph defined by a pushdown machine $\mathcal{M}^{\mathcal{P}}$ of the same format as $\mathcal{P}$. In the first step of this reduction, $\mathcal{P}$ is transformed to an equivalent pushdown automaton of the same format that has the continuity property, which is possible according to Remark 3.6. Notice that for this approach the continuity property is essential since it guarantees that the obtained pushdown game graph is such that in each round every letter $a \in \Sigma_I$ can be chosen by Player I and Player O can respond by any letter $b \in \Sigma_O$.

Theorem 3.20 is proved in Subsection 3.2.2 by a uniform proof method that uses the technique proposed by Kupferman and Vardi [KV00a], which we recall in the next subsection. It comprises a reduction to the emptiness problem for alternating parity two-way tree automata which is to decide, given a parity-A2TA $\mathcal{A}$, whether $L(\mathcal{A}) = \emptyset$. The crucial points for the proof of the theorem are the treatment of the stair condition and the construction of one-counter winning strategies for one-counter games.

Besides this four positive cases of pushdown games which turn out to be solvable by pushdown winning strategies of the same format as the winning conditions, for two further cases we prove that this correspondence fails, i.e., strategies of the same format are not sufficient.

Theorem 3.22. $\mathsf{DV1CL}_\omega$*-games and* $\mathsf{StDV1CL}_\omega$*-games are determined, however,* DV1CL *winning strategies do not suffice.*

Again, by the above construction it follows that this result also holds for pushdown games played on configuration graphs of deterministic visibly one-counter machines.

Corollary 3.23. *Parity and stair parity pushdown games played on (labeled) pushdown game graphs defined by DV1CM are determined, however,* DV1CL *winning strategies do not suffice.*

Moreover, pushdown games on configuration graphs of blind one-counter machines turn out to be not solvable by blind one-counter strategies. However, for blind one-counter Gale-Stewart games the question whether DB1CL winning strategies suffice is still open.

Theorem 3.24. *Parity and stair parity pushdown games played on (labeled) pushdown game graphs defined by DB1CM are determined, however,* DB1CL *winning strategies do not suffice.*

The proofs of Theorem 3.20, Theorem 3.22, and Theorem 3.24 are presented in Subsection 3.2.2.

3.2.1 Solving Pushdown Games using A2TA

A technique which uses alternating two-way tree automata to solve pushdown games was proposed by Kupferman and Vardi [KV00a]. The idea is to simulate a parity pushdown game by a parity-A2TA operating on an infinite tree which represents all possible stack contents. Formally, for a pushdown alphabet Γ, define the full $\Gamma_\perp$-labeled Γ-tree (Γ^*, t_Γ) by $t_\Gamma(\varepsilon) = \perp$ and for every $\gamma \in \Gamma^+$, $t_\Gamma(\gamma) = \mathsf{last}(\gamma)$. The root of t_Γ represents the empty stack, hence, it is labeled by $\perp$. A node $\gamma \in \Gamma^+$ corresponds to the stack content $\mathsf{rev}(\gamma)\perp$ and is labeled by the top stack symbol $\mathsf{last}(\gamma)$.

The technique comprises three essential steps. The first step is to construct a parity-A2TA which simulates the parity pushdown game. In the second step the obtained parity-A2TA is translated into an equivalent parity-N1TA. Finally, a pushdown winning strategy is deduced by solving the emptiness problem. We recall here the crucial points of the construction which is similarly presented in [Cac01].

From pushdown game $\mathcal{G}$ to parity-A2TA $\mathcal{A}$. Let $\mathcal{P} = (Q, \Gamma, \Delta, q_{\text{in}})$ be a PDS with a partition $Q_0 \cup Q_1$ of the set Q of states. For the sake of convenience, we assume Δ to be of the following form. For every push-transition $(q, A, q', BC) \in \Delta$ we assume $A = C$, and for every skip-transition

$(q, A, q', B) \in \Delta$ we assume $A = B$, i.e., the top stack symbol is not modified when performing a skip- or a push-transition, hence to modify a top stack symbol it has first to be deleted using a pop-transition and then a new symbol is pushed. Moreover, let $\lambda\colon \Delta \to \mathcal{P}(\Sigma)$ be a labeling function for some alphabet Σ which deterministically labels the pushdown graph $G(\mathcal{P})$, i.e., for all $a \in \Sigma$, all $q \in Q$ and all $A \in \Gamma_\bot$,

$$\big|\{(q', \alpha) \mid (q, A, q', \alpha) \in \Delta \text{ and } a \in \lambda(q, A, q', \alpha)\}\big| + \big|\{(q', \alpha) \mid (q, A, q', \alpha) \in \Delta \text{ and } \lambda(q, A, q', \alpha) = \emptyset\}\big| \leq 1.$$

Let $\mathrm{col}\colon Q \to [n]$ be a coloring function, for some $n \in \mathbb{N}$. Consider the parity pushdown game $\mathcal{G} = (G(\mathcal{P}), \mathrm{col})$ on the labeled pushdown game graph $G(\mathcal{P})$ with the parity condition col. Moreover, let δ be the following function. For every $q \in Q$ and every $A \in \Gamma_\bot$,

$$\delta(q, A) = \begin{cases} \bigvee\limits_{(q,A,q',A'A)\in\Delta} (\downarrow_{A'}, q') \vee \bigvee\limits_{(q,A,q',A)\in\Delta} (\circlearrowleft, q') \vee \bigvee\limits_{(q,A,q',\varepsilon)\in\Delta} (\uparrow, q'), & \\ & \text{if } q \in Q_0, \\ \bigwedge\limits_{(q,A,q',A'A)\in\Delta} (\downarrow_{A'}, q') \wedge \bigwedge\limits_{(q,A,q',A)\in\Delta} (\circlearrowleft, q') \wedge \bigwedge\limits_{(q,A,q',\varepsilon)\in\Delta} (\uparrow, q'), & \\ & \text{if } q \in Q_1. \end{cases}$$

The parity-A2TA $\mathcal{A} = (Q^{\mathcal{A}}, \Gamma_\bot, \delta^{\mathcal{A}}, q_{\mathrm{in}}^{\mathcal{A}}, \mathrm{col}^{\mathcal{A}})$ simulating $\mathcal{G}$ is defined by

- $Q^{\mathcal{A}} = Q \cup P \cup \{q_{\mathrm{in}}^{\mathcal{A}}\}$ where $P = \{p_A \mid A \in \Gamma\}$ such that $Q \cap P = \emptyset$ and $q_{\mathrm{in}}^{\mathcal{A}} \notin Q \cup P$,
- $\mathrm{col}^{\mathcal{A}}(q) = \begin{cases} \mathrm{col}(q) & \text{if } q \in Q, \\ 0 & \text{otherwise.} \end{cases}$
- for $q \in Q$, $p \in P$ and $A \in \Gamma$,

$$\begin{aligned} \delta^{\mathcal{A}}(q_{\mathrm{in}}^{\mathcal{A}}, \bot) &= (\circlearrowleft, q_{\mathrm{in}}) \wedge \bigwedge_{B\in\Gamma} (\downarrow_B, p_B), \\ \delta^{\mathcal{A}}(q, A) &= \delta(q, A), \\ \delta^{\mathcal{A}}(q, \bot) &= \delta(q, \bot), \\ \delta^{\mathcal{A}}(p, A) &= \begin{cases} \bigwedge_{B\in\Gamma} (\downarrow_B, p_B), & \text{if } p = p_A, \\ \mathsf{false}, & \text{if } p \neq p_A. \end{cases} \end{aligned}$$

The parity-A2TA $\mathcal{A}$ operates on the tree t_Γ which is used instead of the stack. The states from P are used to verify that the input tree is indeed t_Γ. For this, auxiliary computations are initiated starting in the initial state $q_{\text{in}}^{\mathcal{A}}$ on the root node which pass down the states from P to the respective child nodes (for every $A \in \Gamma$, state p_A is send in direction A). These computations succeed if every node $\gamma \in \Gamma^+$ is labeled by $\mathsf{last}(\gamma)$.

To simulate the pushdown game $\mathcal{G}$ alternation is used. Being in a state $q \in Q_0$ the parity-A2TA $\mathcal{A}$ guesses a transition from the disjunction over all possible pushdown transitions which should be chosen by Player O. On the other hand, being in some state $q \in Q_1$, we have to follow each pushdown transition, due to the conjunction over all possible pushdown transitions that can be chosen by Player I.

By this construction, there is an accepting run of $\mathcal{A}$ on t_Γ if and only if there exists a winning strategy for Player O in $\mathcal{G}$. Notice that, due to the auxiliary computations using the states from P, the only tree which can be accepted by $\mathcal{A}$ is t_Γ. Hence, in order to solve $\mathcal{G}$, $\mathcal{A}$ has to be tested for emptiness.

Theorem 3.25 ([KV00a]). *Player O wins $\mathcal{G}$ if and only if $L(\mathcal{A}) \neq \emptyset$.*

The emptiness problem for parity-A2TA is solved in [Var98] by a reduction to the emptiness problem for parity-N1TA. We recall the construction of an equivalent parity-N1TA from a parity-A2TA in the following paragraph.

From parity-A2TA $\mathcal{A}$ to parity-N1TA $\mathcal{B}$. Consider a parity-A2TA $\mathcal{A} = (Q^{\mathcal{A}}, \Gamma_\perp, \delta^{\mathcal{A}}, q_{\text{in}}^{\mathcal{A}}, \text{col}^{\mathcal{A}})$ with the coloring function $\text{col}^{\mathcal{A}} \colon Q^{\mathcal{A}} \to [n]$ obtained from a parity pushdown game $\mathcal{G}$ as described above. The idea for the construction of an equivalent one-way tree automaton is to guess a strategy for $\mathcal{A}$ that determines which copies to send at every node. In case of a conjunction, of course, all copies have to be sent, and for a disjunction, the strategy can choose one copy. The automaton has to verify the consistency of the guessed strategy in a one-way manner and furthermore a run of $\mathcal{A}$ has to be accepting if it accounts for this strategy.

We use the set $\mathsf{Str} = \mathcal{P}(Q^{\mathcal{A}} \times \mathsf{Dir}_{\uparrow,\circlearrowleft} \times Q^{\mathcal{A}})$ to represent strategies for $\mathcal{A}$ and the set $\mathsf{Ann} = \mathcal{P}(Q^{\mathcal{A}} \times [n] \times Q^{\mathcal{A}})$, called the set of annotations, will be used to check whether a strategy yields an accepting run of $\mathcal{A}$ on t_Γ.

Consider a $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann})$-labeled full Γ-tree (Γ^*, t) with $\text{Pr}_0(t(\gamma)) = t_\Gamma(\gamma)$ for every $\gamma \in \Gamma^*$, i.e., t can be regarded as t_Γ where every labeling is augmented by some element from Str and an element from Ann. By t_{Str} we denote the Str-labeled Γ-tree obtained from t by projecting to the second component of the labelings, i.e., $t_{\mathsf{Str}}(\gamma) = \text{Pr}_1(t(\gamma))$ for every $\gamma \in \Gamma^*$, and t_{Ann} denotes the Ann-labeled Γ-tree containing only the last component of the labeling of t, i.e., $t_{\mathsf{Ann}}(\gamma) = \text{Pr}_2(t(\gamma))$ for every $\gamma \in \Gamma^*$.

A strategy is encoded in t, if t_{Str} is consistent, i.e., if t_{Str} satisfies the following conditions. For every node $\gamma \in \Gamma^*$,

(i) if $(q, d, r) \in t_{\mathsf{Str}}(\gamma)$ then $\{(d', r') \mid (q, d', r') \in t_{\mathsf{Str}}(\gamma)\} \subseteq \mathsf{Dir}_{\uparrow,\circlearrowleft} \times Q^{\mathcal{A}}$ satisfies $\delta^{\mathcal{A}}(q, t_\Gamma(\gamma))$,

(ii) if $(q, d, r) \in t_{\mathsf{Str}}(\gamma)$ then there are $d' \in \mathsf{Dir}_{\uparrow,\circlearrowleft}$ and $q' \in Q^{\mathcal{A}}$ such that $(r, d', q') \in t_{\mathsf{Str}}(\gamma \cdot d)$ or $\emptyset$ satisfies $\delta^{\mathcal{A}}(q, t_\Gamma(\gamma \cdot d))$,

(iii) if $\gamma = \varepsilon$ then there are $d \in \mathsf{Dir}_{\uparrow,\circlearrowleft}$ and $q \in Q^{\mathcal{A}}$ such that $(q^{\mathcal{A}}_{\mathrm{in}}, d, q) \in t_{\mathsf{Str}}(\gamma)$ or $\emptyset$ satisfies $\delta^{\mathcal{A}}(q^{\mathcal{A}}_{\mathrm{in}}, t_\Gamma(\gamma))$.

The first condition requires that at every node the strategy satisfies the transition function $\delta^{\mathcal{A}}$. The second condition ensures that the strategy encoded in t can always be followed, i.e., being in node γ in some state q if the strategy $t_{\mathsf{Str}}(\gamma)$ contains (q, d, r) then at node $\gamma \cdot d$ a strategy for state r has also to be defined. The last condition states that at the root a strategy for the initial state has to be defined.

Notice that all three requirements can be checked locally, i.e., by inspecting the neighborhood of a node. Thus, a deterministic parity-D1TA $\mathcal{B}_{\mathsf{Str}}$ over $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann})$-labeled Γ-trees can be constructed such that it accepts an input tree t if $\mathrm{Pr}_0(t(\gamma)) = t_\Gamma(\gamma)$, for every $\gamma \in \Gamma^*$, and the corresponding tree t_{Str} is consistent, i.e., the above conditions are satisfied.

Next, it is to verify that the strategy encoded in t also induces an accepting run of $\mathcal{A}$. To be able to do this in a one-way manner we use annotations to store information about finite detours (paths which go down from a node and come back to the same node) induced by the strategy. A tuple $(q, m, r) \in Q^{\mathcal{A}} \times [n] \times Q^{\mathcal{A}}$ contained in an annotation of some node indicates the existence of a detour induced by the strategy which starts in state q and comes back to the same node in state r with c being the minimal color seen along this detour.

An annotation t_{Ann} encoded in t, is correct for t_{Str} if the following conditions are satisfied. For every node $\gamma \in \Gamma^*$,

(i) if $(q, \circlearrowleft, r) \in t_{\mathsf{Str}}(\gamma)$ then $(q, \mathrm{col}^{\mathcal{A}}(r), r) \in t_{\mathsf{Ann}}(\gamma)$,

(ii) if $(q, \downarrow_A, r) \in t_{\mathsf{Str}}(\gamma)$ and $(r, \uparrow, q') \in t_{\mathsf{Str}}(\gamma A)$ then $(q, \min\{c_r, c_{q'}\}, q') \in t_{\mathsf{Ann}}(\gamma)$ where $c_r = \mathrm{col}^{\mathcal{A}}(r)$ and $c_{q'} = \mathrm{col}^{\mathcal{A}}(q')$,

(iii) if $(q, c, r), (r, c', q') \in t_{\mathsf{Ann}}(\gamma)$ then $(q, \min\{c, c'\}, q') \in t_{\mathsf{Ann}}(\gamma)$,

(iv) if $(q, \downarrow_A, r) \in t_{\mathsf{Str}}(\gamma)$ and $(r, c, r') \in t_{\mathsf{Ann}}(\gamma A)$ and $(r', \uparrow, q') \in t_{\mathsf{Str}}(\gamma A)$ then $(q, \min\{c_r, c, c_{q'}\}, q') \in t_{\mathsf{Ann}}(\gamma)$ where $c_r = \mathrm{col}^{\mathcal{A}}(r)$, $c_{q'} = \mathrm{col}^{\mathcal{A}}(q')$.

Again, all conditions can be checked locally. Hence, a deterministic parity-D1TA $\mathcal{B}_{\mathsf{Ann}}$ over $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann})$-labeled Γ-trees can be constructed such that a tree t is accepted if the corresponding tree t_{Ann} is correct for t_{Str}.

Finally, a parity-D1TA $\mathcal{B}_{\mathrm{col}^{\mathcal{A}}}$ over $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann})$-labeled Γ-trees can be constructed which evaluates the parity condition $\mathrm{col}^{\mathcal{A}}$ of the run induced by t_{Str} and t_{Ann}. The colors stored in the annotations are used to avoid detours.

Consider a parity-D1TA $\mathcal{B}'$ over $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann})$-labeled Γ-trees which is the product of $\mathcal{B}_{\mathsf{Str}}$, $\mathcal{B}_{\mathsf{Ann}}$ and $\mathcal{B}_{\mathrm{col}^{\mathcal{A}}}$. Clearly, $\mathcal{B}'$ accepts those trees t which represent accepting runs of $\mathcal{A}$ on t_Γ, i.e., in the first component of the labelings of t the tree t_Γ is encoded, the corresponding trees t_{Str} and t_{Ann} are such that t_{Str} encodes a consistent strategy and t_{Ann} is correct for t_{Str}, and the run of $\mathcal{A}$ induced by t_{Str} satisfies the parity condition $\mathrm{col}^{\mathcal{A}}$.

Now, we can construct a parity-N1TA $\mathcal{B}$ over $\Gamma_\perp$-labeled Γ-trees which is equivalent to $\mathcal{A}$. The automaton $\mathcal{B}$ nondeterministically guesses the trees t_{Str} and t_{Ann} and verifies consistency of t_{Str} and correctness of t_{Ann} for t_{Str} as well as it evaluates the parity condition of the induced run of $\mathcal{A}$ using $\mathcal{B}'$.

Theorem 3.26 ([Var98])**.** *For every parity-A2TA $\mathcal{A}$ there exists a parity-N1TA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$.*

Notice, that due to decidability of the emptiness problem for parity-N1TA (cf. e.g. [EJ88, GTW02]) the winner of the pushdown game $\mathcal{G}$ can be determined applying Theorem 3.25 and Theorem 3.26.

Corollary 3.27. *Player O wins $\mathcal{G}$ if and only if $L(\mathcal{A}) = L(\mathcal{B}) \neq \emptyset$ if and only if $L(\mathcal{B}') \neq \emptyset$.*

In the following paragraph, we show how a pushdown winning strategy for Player O can be deduced from $\mathcal{B}'$ in case $L(\mathcal{B}')$ is nonempty. Note that if $L(\mathcal{B}')$ is empty then a pushdown winning strategy for Player I can be deduced by swapping the roles of the players for the construction of the parity-A2TA simulating $\mathcal{G}$.

From parity-D1TA $\mathcal{B}'$ to Pushdown Strategy $\mathcal{S}$. Let $\mathcal{B}'$ be the parity-D1TA as defined in the previous paragraph. Moreover, we assume $L(\mathcal{B}') \neq \emptyset$. It is well known that every nonempty tree language recognized by a parity tree automaton contains a regular tree and furthermore, a DFT generating such a tree can be constructed effectively from the parity tree automaton (see e.g. [Tho97, GTW02]). Let $\mathcal{R} = (Q^{\mathcal{R}}, \Gamma, \Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann}, \delta^{\mathcal{R}}, q_{\mathrm{in}}^{\mathcal{R}}, \lambda^{\mathcal{R}})$ be a DFT obtained from $\mathcal{B}'$ generating a regular $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann})$-labeled Γ-tree $t_{\mathrm{reg}} \in L(\mathcal{B}')$.

We define a DPDT $\mathcal{S} = (Q, \Delta, \Delta, Q^{\mathcal{R}}, \delta^{\mathcal{S}}, q_{\mathrm{in}}, \lambda^{\mathcal{S}})$ implementing a pushdown strategy for Player O in $\mathcal{G}$ using $\mathcal{R}$ as follows. The pushdown transducer $\mathcal{S}$ reads transitions from Δ corresponding to the edges chosen by the

two players in $\mathcal{G}$ and outputs transitions from Δ corresponding to the next choice of Player O. For the set of states, states from $\mathcal{G}$ are utilized, and states from $\mathcal{R}$ are used as the stack alphabet.

The idea is to keep track of the labeling of the current node of t_{reg} corresponding to the stack content of the current configuration to be able to deduce the next choice for Player O from the strategy encoded in the labeling of the node. For this, we store the appropriate states of $\mathcal{R}$ on the stack of $\mathcal{S}$. By inspecting the top stack symbol, the labeling $(A, S, H) \in \Gamma_\bot \times \mathsf{Str} \times \mathsf{Ann}$ is then easily deduced by applying $\lambda^{\mathcal{R}}$. Formally, for $q, q' \in Q$, $r \in Q^{\mathcal{R}}$ and $A, B \in \Gamma$, define the transition function $\delta^{\mathcal{S}}$ by

$$\begin{aligned}
&\delta^{\mathcal{S}}(q, \bot, (q, \bot, q', \bot)) = (q', \bot),\\
&\delta^{\mathcal{S}}(q, r, (q, A, q', A)) = (q', r),\\
&\delta^{\mathcal{S}}(q, \bot, (q, \bot, q', A\bot)) = (q', r'\bot) \text{ where } r' = \delta^{\mathcal{R}}(q^{\mathcal{R}}_{\text{in}}, A),\\
&\delta^{\mathcal{S}}(q, r, (q, A, q', BA)) = (q', r'r) \text{ where } r' = \delta^{\mathcal{R}}(r, B),\\
&\delta^{\mathcal{S}}(q, r, (q, A, q', \varepsilon)) = (q', \varepsilon).
\end{aligned}$$

Now, it remains to define the output function $\lambda^{\mathcal{S}}$. Clearly, there might be several possible transitions encoded in a strategy $S \in \mathsf{Str}$ which can be chosen by Player O in order to win a play. Hence, to define a deterministic output function, let $\mathsf{f} \colon (\mathsf{Str} \times Q) \to (Q \times \mathsf{Dir}_{\uparrow,\circlearrowleft} \times Q)$ be some choice function such that for every $S \in \mathsf{Str}$ and every $q \in Q$, $\mathsf{f}(S, q) \in \{s \in S \mid \mathrm{Pr}_0(s) = q\}$, i.e., for a strategy S and a state q the choice function f selects an element s from S containing q in its first component. So, we define $\lambda^{\mathcal{S}} \colon Q \times (Q^{\mathcal{R}} \cup \{\bot\}) \to \Delta$ for $q \in Q_0$ and $r \in Q^{\mathcal{R}}$ by

$$\lambda^{\mathcal{S}}(q, \bot) = \begin{cases} (q, \bot, q', \bot), & \text{if } \lambda^{\mathcal{R}}(q^{\mathcal{R}}_{\text{in}}) = (\bot, S, H) \text{ and } \mathsf{f}(S, q) = (q, \circlearrowleft, q'),\\ (q, \bot, q', A\bot), & \text{if } \lambda^{\mathcal{R}}(q^{\mathcal{R}}_{\text{in}}) = (\bot, S, H) \text{ and } \mathsf{f}(S, q) = (q, \downarrow_A, q'), \end{cases}$$

$$\lambda^{\mathcal{S}}(q, r) = \begin{cases} (q, A, q', A), & \text{if } \lambda^{\mathcal{R}}(r) = (A, S, H) \text{ and } \mathsf{f}(S, q) = (q, \circlearrowleft, q'),\\ (q, A, q', BA), & \text{if } \lambda^{\mathcal{R}}(r) = (A, S, H) \text{ and } \mathsf{f}(S, q) = (q, \downarrow_B, q'),\\ (q, A, q', \varepsilon), & \text{if } \lambda^{\mathcal{R}}(r) = (A, S, H) \text{ and } \mathsf{f}(S, q) = (q, \uparrow, q'). \end{cases}$$

Notice, that a pushdown strategy which reads the labelings of the transitions in $\mathcal{G}$ instead of the transitions themselves can be defined in a similar way. In particular, this is reasonable in case where the labeled pushdown graph models a Gale-Stewart game, since for Gale-Stewart games the winning strategy has to produce output letters. We define a DPDT $\mathcal{S}' = (Q, \Sigma, \Sigma, Q^{\mathcal{R}}, \delta^{\mathcal{S}'}, q_{\text{in}}, \lambda^{\mathcal{S}'})$ implementing such a winning strategy as follows. The transition function $\delta^{\mathcal{S}'}$ is defined by

$$\delta^{\mathcal{S}'}(q, r, a) = \delta^{\mathcal{S}}(q, r, t) \text{ where } t \in \Delta \text{ with } a \in \lambda(t),$$

$$\delta^{\mathcal{S}'}(q, r, \varepsilon) = \delta^{\mathcal{S}}(q, r, t) \text{ where } t \in \Delta \text{ with } \emptyset = \lambda(t),$$

for $q, \in Q$, $r \in Q^{\mathcal{R}}$, and $a \in \Sigma$. This means, the input letter is mapped to the corresponding transition, which is possible as the labeling λ deterministically labels the pushdown graph, and the transition function $\delta^{\mathcal{S}}$ is applied. To define the output function $\lambda^{\mathcal{S}'}$, again we use the output function $\lambda^{\mathcal{S}}$. Let $\mathsf{f}' \colon \mathcal{P}(\Sigma) \to \Sigma_\varepsilon$ be a choice function such that $\mathsf{f}'(\emptyset) = \varepsilon$ and $\mathsf{f}'(\Xi) \in \Xi$ for every $\Xi \in \mathcal{P}(\Sigma)$. For $q \in Q$ and $r \in Q_\perp^{\mathcal{R}}$, we define

$$\lambda^{\mathcal{S}'}(q, r) = \mathsf{f}'(\lambda(t)) \text{ where } t = \lambda^{\mathcal{S}}(q, r).$$

3.2.2 Proof of Theorems

In this subsection we present the proofs of Theorems 3.20, 3.22 and 3.24. We show how the technique presented in the previous subsection can be adapted to solve parity pushdown games and stair parity pushdown games played on labeled pushdown graphs defined by deterministic, visibly, real-time, and one-counter machines such that the obtained pushdown transducers implementing winning strategies are of corresponding formats. Moreover, we present winning conditions which show that visibly one-counter strategies do not suffice to solve visibly one-counter games as well as blind one-counter strategies do not suffice to solve blind one-counter games.

Stair Parity Games

We begin by showing how to handle stair parity winning conditions. The idea is to use alternating two-way tree automata to simulate stair parity pushdown games. For this, we define alternating two-way tree automata which evaluate stair parity conditions.

Definition 3.28. Let Γ be a finite alphabet. An alternating stair parity two-way tree automaton (parity-StA2TA) $\mathcal{A} = (Q, \Sigma, \delta, q_{\text{in}}, \text{col})$ over Σ-labeled Γ-trees consists of a finite set of states Q with the initial state q_{in}, a labeling alphabet Σ, a coloring function $\text{col} \colon Q \to [n]$, for some $n \in \mathbb{N}$, and a transition function $\delta \colon Q \times \Sigma \to \mathcal{B}^+(\mathsf{Dir}_{\uparrow,\circlearrowleft} \times Q)$. Just as for parity-A2TA, a

run of $\mathcal{A}$ on a tree $t \in \Gamma_\Sigma$ is a $(Q \times \Gamma^*)$-labeled $\mathbb{N}$-tree (T, ρ) satisfying the following conditions.

1. $\varepsilon \in T$ and $\rho(\varepsilon) = (q_{\text{in}}, \varepsilon)$.
2. If $y \in T$ with $\rho(y) = (q, \gamma)$ and $\delta(q, t(\gamma)) = \varphi$ then there is a conjunct $\psi = \{(d_0, q_0), \ldots, (d_{k-1}, q_{k-1})\} \subseteq \mathsf{Dir}_{\uparrow,\circlearrowleft} \times Q$ in φ such that the set of successors of y in T is precisely $\{y \cdot i \mid i \in [k]\}$ and $\rho(y \cdot i) = (q_i, \gamma \cdot d_i)$.

For a path π through T starting in ε, let $\rho(\pi) \in (Q \times \Gamma^*)^\omega$ denote the sequence of labelings of π. Extending each labeling of $\rho(\pi)$ by the $\bot$-symbol yields a sequence of configurations $\rho'(\pi) \in (Q \times \Gamma^*\bot)^\omega$ with $(\rho'(\pi))(i) = (\rho(\pi))(i)\bot$, for $i \in \mathbb{N}$. Moreover, let col be extended to configurations such that $\text{col}(q, \gamma\bot) = \text{col}(q)$ for every $q \in Q$ and every $\gamma \in \Gamma^*$. A run (T, ρ) is accepting if all its paths satisfy the stair parity condition col, i.e., for each path π through T, $\min\{\text{Inf}(\text{col}(\mathsf{Stairs}(\rho'(\pi))))\}$ is even. A tree $t \in \Gamma_\Sigma$ is accepted by $\mathcal{A}$ if there is an accepting run (T, ρ) of $\mathcal{A}$ on t.

Now, given a stair parity pushdown game $\mathcal{G}$ on a deterministically labeled pushdown game graph, a parity-StA2TA $\mathcal{A}$ simulating $\mathcal{G}$ is defined in exactly the same manner as parity-A2TA for parity pushdown games as described in the previous subsection. Notice, that the auxiliary computations which check whether the input tree is t_Γ are not bidirectional, since only directions from $\mathsf{Dir} = \{\downarrow_A \mid A \in \Gamma\}$ are used. This means, that there is no difference between the evaluation of the stair parity condition and the evaluation of the parity condition for those paths in a run (T, ρ) of $\mathcal{A}$, i.e., this auxiliary computations check whether the input tree is t_Γ, now evaluating the stair parity condition. All other paths π in T correspond to plays $\rho'(\pi)$ in $\mathcal{G}$ (by $\rho'(\pi)$ we denote the sequence of configurations for a path π as above). Hence, Theorem 3.25 can be lifted to stair pushdown games.

Theorem 3.29. *Player O wins $\mathcal{G}$ if and only if $L(\mathcal{A}) \neq \emptyset$.*

In order to solve the emptiness problem for parity-StA2TA, we again use a reduction to the emptiness problem for parity-N1TA. For this, a parity-StA2TA is transformed into an equivalent parity-N1TA. This is done by adapting the construction presented in the previous subsection.

From parity-StA2TA $\mathcal{A}$ to parity-N1TA $\mathcal{B}$. Let parity-StA2TA $\mathcal{A} = (Q^\mathcal{A}, \Gamma_\bot, \delta^\mathcal{A}, q_{\text{in}}^\mathcal{A}, \text{col}^\mathcal{A})$ with the coloring function $\text{col}^\mathcal{A} \colon Q^\mathcal{A} \to [n]$ be obtained from a stair parity pushdown game $\mathcal{G}$ according to the construction described on page 38. Again, an equivalent parity-N1TA will guess a strategy for $\mathcal{A}$, verify its consistency, and, using annotations, evaluate the stair parity condition for the run of $\mathcal{A}$ induced by the guessed strategy.

We use the set Str to represent strategies. Notice, that the only colors relevant for the evaluation of a stair condition seen during a finite detour starting in a node $\gamma \in \Gamma^*$ are those seen at the node γ itself. All other colors can be ignored, since they are seen at nodes corresponding to non-stair configurations, as there is a subsequent configuration of smaller stack height, namely when the detour returns to node γ. Hence, to keep track of the colors at the stair configurations of a run, we add an additional annotation component. Let $\mathsf{Ann}' = \mathcal{P}(Q^{\mathcal{A}} \times Q^{\mathcal{A}})$.

For a $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann} \times \mathsf{Ann}')$-labeled full Γ-tree (Γ^*, t), we denote by t_{Str} the Str-labeled full Γ-tree with $t_{\mathsf{Str}}(\gamma) = \mathrm{Pr}_1(t(\gamma))$, for every node $\gamma \in \Gamma^*$. By t_{Ann} we denote the Ann-labeled full Γ-tree with $t_{\mathsf{Ann}}(\gamma) = \mathrm{Pr}_2(t(\gamma))$, and by $t_{\mathsf{Ann}'}$ the Ann'-labeled full Γ-tree with $t_{\mathsf{Ann}'}(\gamma) = \mathrm{Pr}_3(t_{\mathsf{Ann}'}(\gamma))$ is denoted, for $\gamma \in \Gamma^*$. The idea is to store the information about finite detours which return to the current node only once in $t_{\mathsf{Ann}'}$, the information about all possible finite detours and the minimal color of the states seen at the current node during the detour is enclosed in t_{Ann}. Hence, for a node $\gamma \in \Gamma^*$, the meaning of $(q, q') \in t_{\mathsf{Ann}'}(\gamma)$ is that there is a finite detour induced by t_{Str} which starts in state q and ends in state q' and which returns to γ exactly once, whereas $(q, c, q') \in t_{\mathsf{Ann}}(\gamma)$ means that there is a finite detour that starts in q and ends in q' with c being the minimal color of the states seen at node γ during this detour.

Consider a $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann} \times \mathsf{Ann}')$-labeled Γ-tree (Γ^*, t) with $\mathrm{Pr}_0(t(\gamma)) = t_\Gamma(\gamma)$ for every $\gamma \in \Gamma^*$. The consistency of t_{Str} is checked by a deterministic parity-D1TA obtained from $\mathcal{B}_{\mathsf{Str}}$ from the previous subsection now operating on $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann} \times \mathsf{Ann}')$-labeled Γ-trees. To verify the correctness of the annotation $t_{\mathsf{Ann}'}$ for the strategy t_{Str} the following conditions have to be checked. For every node $\gamma \in \Gamma^*$,

(i) if $(q, \circlearrowleft, r) \in t_{\mathsf{Str}}(\gamma)$ then $(q, r) \in t_{\mathsf{Ann}'}(\gamma)$,

(ii) if $(q, \downarrow_A, r) \in t_{\mathsf{Str}}(\gamma)$ and $(r, \uparrow, q') \in t_{\mathsf{Str}}(\gamma A)$ then $(q, q') \in t_{\mathsf{Ann}'}(\gamma)$,

(iii) if $(q, r) \in t_{\mathsf{Ann}'}(\gamma)$ then $(q, \mathrm{col}^{\mathcal{A}}(r), r) \in t_{\mathsf{Ann}}$,

(iv) if $(q, c, r), (r, c', q') \in t_{\mathsf{Ann}}(\gamma)$ then $(q, \min\{c, c'\}, q') \in t_{\mathsf{Ann}}(\gamma)$,

(v) if $(q, \downarrow_A, r) \in t_{\mathsf{Str}}(\gamma)$ and $(r, c, r') \in t_{\mathsf{Ann}}(\gamma A)$ and $(r', \uparrow, q') \in t_{\mathsf{Str}}(\gamma A)$ then $(q, q') \in t_{\mathsf{Ann}'}(\gamma)$.

All conditions can be checked locally. Hence, a parity-D1TA over $(\Gamma_\perp \times \mathsf{Str} \times \mathsf{Ann} \times \mathsf{Ann}')$-labeled Γ-trees can be constructed that checks the correctness of an annotation $t_{\mathsf{Ann}'}$ for a strategy t_{Str}. Finally, to evaluate the stair parity condition $\mathrm{col}^{\mathcal{A}}$ of a run induced by t_{Str} and t_{Ann} the parity-D1TA $\mathcal{B}_{\mathrm{col}^{\mathcal{A}}}$ from the previous subsection is used which now ignores the Ann'-components.

Corollary 3.30. *For every parity-StA2TA $\mathcal{A}$ there exists a parity-N1TA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$.*

Hence, a pushdown transducer implementing a winning strategy can be deduced as described in the previous subsection. Clearly, by this construction the underlying PDM of the transducer is deterministic if the underlying PDM of the game is deterministic.

One-Counter Games

Now, we consider parity and stair parity pushdown games played on labeled configuration graphs of deterministic one-counter machines. Let $\mathcal{C} = (Q, \Sigma, \Gamma, \delta^{\mathcal{C}}, q_{\text{in}})$ be a D1CM with $\Gamma = \{A\}$, a partition $Q = Q_0 \cup Q_1$, and $\mathcal{P} = (Q, \Gamma, \Delta, q_{\text{in}})$ be the corresponding PDS with the labeling function $\lambda\colon \Delta \to \mathcal{P}(\Sigma)$. Moreover, let $\text{col}\colon Q \to [n]$ be a coloring function and $\mathcal{G} = (G(\mathcal{P}), \text{col})$ the parity or stair parity one-counter game.

Notice, that since the stack alphabet is a singleton, the tree t_Γ representing all possible stack contents consists of a single branch with the root labeled by $\bot$ and all other nodes are labeled by A. Thus, in this case t_Γ can be considered as the word $\bot A^\omega$. Moreover, since the set Dir is a singleton as well, the parity-A2TA (parity-StA2TA) $\mathcal{A}$ simulating $\mathcal{G}$ can be viewed as an alternating two-way parity word automaton.

Clearly, $\mathcal{G}$ can be solved using the presented technique. However, the PDM $\mathcal{M}^{\mathcal{S}}$ of the obtained pushdown transducer $\mathcal{S}$ implementing a winning strategy is not a one-counter. We show how a one-counter strategy can be constructed nevertheless.

Assume that $L(\mathcal{A}) \neq \emptyset$ and let $\mathcal{R} = (Q^{\mathcal{R}}, \Gamma, \Gamma', \delta^{\mathcal{R}}, q_{\text{in}}^{\mathcal{R}}, \lambda^{\mathcal{R}})$ be the DFT obtained by the above construction that generates a regular (Γ')-labeled Γ-tree t_{reg} with $\Gamma' = \Gamma_\bot \times \mathsf{Str} \times \mathsf{Ann}$ in case of $\mathcal{G}$ being a parity game and $\Gamma' = \Gamma_\bot \times \mathsf{Str} \times \mathsf{Ann} \times \mathsf{Ann}'$ in case of a stair parity game. As $|\Gamma| = 1$, we can consider the regular tree t_{reg} as an ultimately periodic word $w_{\text{reg}} \in (\Gamma')^\omega$ with $t_{\text{reg}}(A^i) = w_{\text{reg}}(i)$, for $i \in \mathbb{N}$. Hence, for some $k, l \in \mathbb{N}$ with $l < k$,

$$w_{\text{reg}} = \big(w_{\text{reg}}(0) \cdots w_{\text{reg}}(l-1)\big)\big(w_{\text{reg}}(l) \cdots w_{\text{reg}}(k)\big)^\omega.$$

and $\mathcal{R}$ is of the following form. The set of states is $Q^{\mathcal{R}} = \{r_0, \ldots, r_k\}$ with $r_0 = r_{\text{in}}^{\mathcal{R}}$. The transition function is such that $\delta^{\mathcal{R}}(r_i, A) = r_{i+1}$, for all $i \in [k]$, and $\delta^{\mathcal{R}}(r_k, A) = r_l$. The output function is $\lambda^{\mathcal{R}}(r_i) = w_{\text{reg}}(i)$, for $i \in [k+1]$.

To construct a one-counter winning strategy from $\mathcal{R}$, we store the required information in the states of the one-counter strategy instead of its stack. However, the stack is used to count the number of times $\mathcal{R}$ goes into its loop. Formally, we define the DPDT $\mathcal{S}$ implementing a winning strategy for Player O by $\mathcal{S} = (Q^{\mathcal{S}}, \Delta, \Delta, \{A\}, \delta^{\mathcal{S}}, (q_{\text{in}}, r_0), \lambda^{\mathcal{S}})$ with $Q^{\mathcal{S}} = Q \times Q^{\mathcal{R}}$. For

$q, q' \in Q$, $i \in [k]$ and $X, Y \in \{A, \bot\}$, the transition function $\delta^{\mathcal{S}}$ is defined by

$$\delta^{\mathcal{S}}((q, r_i), Y, (q, X, q', X)) = ((q', r_i), Y),$$

$$\delta^{\mathcal{S}}((q, r_i), Y, (q, X, q', AX)) = \begin{cases} ((q', r_{i+1}), Y) & \text{if } i < k, \\ ((q', r_l), AY) & \text{if } i = k, \end{cases}$$

$$\delta^{\mathcal{S}}((q, r_i), Y, (q, X, q', \varepsilon)) = \begin{cases} ((q', r_{i-1}), Y) & \text{if } i \neq l \text{ or } Y = \bot, \\ ((q', r_k), \varepsilon) & \text{if } i = l \text{ and } Y = A. \end{cases}$$

Essentially, the stack is used to appropriately update the second component of $Q^{\mathcal{S}}$. In particular, the stack is increased every time the loop in $\mathcal{R}$ is completed, i.e., when $\mathcal{S}$ proceeds from r_k to r_l. Hence, when reading a pop-transition in a state with r_l in its second component, the top stack symbol determines whether the second component is updated to r_{l-1} or to r_k. If the stack is empty then r_l was reached from r_{l-1} while a matching push-transition was read, otherwise if the top stack symbol is A then the loop was already completed and the predecessor of r_l is r_k.

The output function $\lambda^{\mathcal{S}}$ can be adopted as defined in the previous subsection. Notice, that here the output depends only on the state and not on the top of the stack. However, since the states from $Q^{\mathcal{R}}$ are moved to the second component of $Q^{\mathcal{S}}$, we have the appropriate domain $Q \times Q^{\mathcal{R}}$ for $\lambda^{\mathcal{S}}$.

Visibly and Realtime Games

For parity or stair parity pushdown games played on configuration graphs of DVPM, consider the pushdown strategies $\mathcal{S}$ as well as $\mathcal{S}'$ (which consumes and outputs letters from Σ rather than transitions from Δ) obtained by the above technique. Notice that if a push-transition or respectively a letter from Σ_{push} is processed then $\mathcal{S}$ and $\mathcal{S}'$ perform a push-transition as well. For a pop-transition or respectively a letter from Σ_{pop}, a pop-transition is performed by $\mathcal{S}$ and $\mathcal{S}'$. Furthermore, skip-transitions or, respectively, letters from Σ_{skip} induce also skip-transitions of the transducers. However, a VPM has no access to the top of the stack when performing skip- and push-transitions. Therefore, we modify $\mathcal{S}$ and $\mathcal{S}'$, respectively, by extending the stack alphabet and the set of states by shifting the top stack symbol to the current state to obtain a DVPT $\mathcal{S}_{\text{vis}}$ implementing a visibly winning strategy. We give the construction for $\mathcal{S}_{\text{vis}}$ from $\mathcal{S}'$. Formally,

$$\mathcal{S}_{\text{vis}} = (Q \times Q^{\mathcal{R}}, \Sigma, \Sigma, Q^{\mathcal{R}} \times Q^{\mathcal{R}}, \Delta^{\text{vis}}, (q_{\text{in}}, q^{\mathcal{R}}_{\text{in}}), \lambda^{\text{vis}}),$$

such that for every $q, q' \in Q$, $a \in \Sigma$, and $X, Y \in Q^{\mathcal{R}}_{\bot}$,

$$((q, X), a, (q', X)) \in \Delta^{\text{vis}}_{\text{skip}} \text{ if } \delta^{\mathcal{S}'}(q, X, a) = (q', X),$$

$$((q,X),a,(X,Y),(q',Y)) \in \Delta^{\text{vis}}_{\text{pop}} \text{ if } \delta^{\mathcal{S}'}(q,X,a) = (q',\varepsilon),$$
$$((q,X),a,(q',Y),(Y,X)) \in \Delta^{\text{vis}}_{\text{push}} \text{ if } \delta^{\mathcal{S}'}(q,X,a) = (q',YX).$$

Hence, by this simple modification, we obtain a transducer implementing a winning strategy of the same format as the pushdown machine defining the pushdown game.

Now, consider a parity or stair parity pushdown game played on a configuration graph of rt-DPDM, and the corresponding pushdown strategies $\mathcal{S}$ and $\mathcal{S}'$. By definition of $\mathcal{S}$, the corresponding DPDM $\mathcal{M}^{\mathcal{S}}$ is realtime, as no ε-transition are contained in the transition function. Notice, that since the game graph is a configuration graph of a rt-DPDM there is no transition $t \in \Delta$ with $\lambda(t) = \emptyset$. Hence, the transition function of $\mathcal{S}'$ contains only non-ε-transitions, as well.

Visibly and Blind One-Counter Games

First, we show that visibly one-counter strategies do not suffice to solve visibly one-counter Gale-Stewart games. Let $\Sigma = \Sigma_I \times \Sigma_O$ where $\Sigma_I = \{a,c\}$ and $\Sigma_O = \{a,r\}$. Consider the following $\mathsf{DV1CL}_\omega$-game $\Gamma(L_{\text{vc}})$ with $L_{\text{vc}} \subseteq \Sigma^\omega$ such that $w \in L_{\text{vc}}$ if and only if

- $\text{Pr}_0(w) \neq c^n a^\omega$, for some $n > 1$, or
- $w = \binom{c}{r}^n \binom{a}{r}^{n-1} \binom{a}{a} \binom{a}{r}^\omega$, for some $n > 1$.

A parity-DV1CA recognizing L_{vc} for the visibly pushdown alphabet Σ with $\Sigma_{\text{push}} = \{\binom{c}{a}, \binom{c}{r}\}$, $\Sigma_{\text{pop}} = \{\binom{a}{r}\}$ and $\Sigma_{\text{skip}} = \{\binom{a}{a}\}$ is depicted in Figure 3.3 where the symbol asterisk stands for any symbol from Σ_O. To facilitate readability the stack symbol A is omitted in the labelings of the push-transitions, since it is the only symbol which can be pushed onto the stack. Moreover, if the stack symbol is omitted in the labeling of an $\binom{a}{r}$-transition then this transition is defined for both stack symbols $\bot$ and A.

The automaton checks whether an input word w satisfies $\text{Pr}_0(w) = c^n a^\omega$, for some $n > 1$. If this is violated then the automaton proceeds to the accepting sink state q_7 and remains in the loop forever. In case where $\text{Pr}_0(w) = c^n a^\omega$, for some $n > 1$, the stack is used to check the second condition of the definition. Being in state q_2 the length n of the prefix $\binom{c}{r}^n$ of the input word is stored on the stack. State q_6 can only be reached from a configuration $(q_2, A^n\bot)$ if the infix $\binom{a}{r}^{n-1}\binom{a}{a}\binom{a}{r}^2$ is read. There the automaton remains as long as letters $\binom{a}{r}$ are processed. The coloring ensures that an input word w is accepted if the unique run on w eventually remains in q_7 or in q_6 forever corresponding to the both conditions of the definitions.

Clearly, a winning strategy $\sigma_O \colon (\Sigma_I \times \Sigma_O)^*\Sigma_I \to \Sigma_O$ for Player O in $\Gamma(L_{\text{vc}})$ has to respond by letter a at the appropriate position, if the input

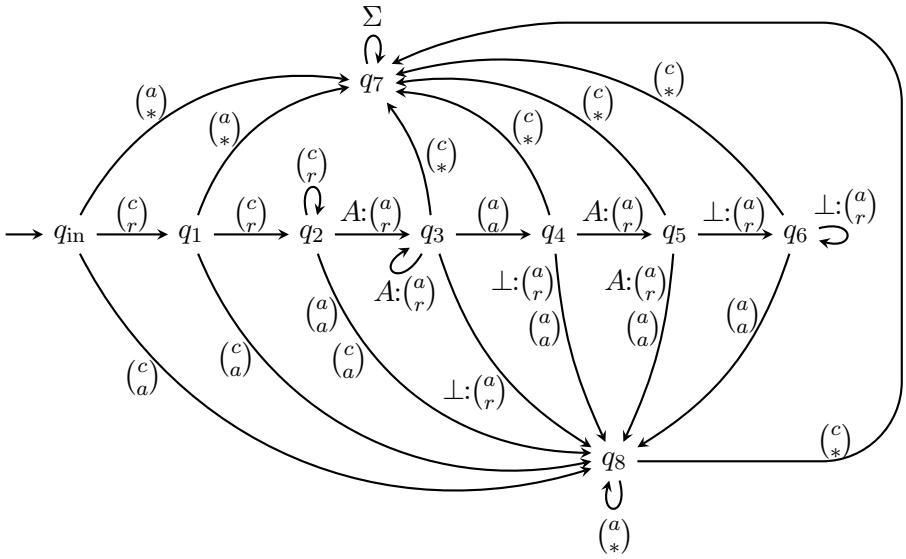


Figure 3.3: Parity-DV1CA recognizing L

produced by Player I is $c^n a^\omega$ for $n > 1$, i.e., $\sigma_O(\binom{c}{r}^n \binom{a}{r}^{n-1} a) = a$, for every $n > 1$. We use a language theoretic argument to show that this cannot be implemented by any DV1CT. We use the representation of a pushdown transducer given by a collection of pushdown automata (cf. Section 3.2).

Lemma 3.31. *The language* $L = \{\binom{c}{r}^n \binom{a}{r}^{n-1} a \mid n > 1\}$ *is not accepted by any DV1CA.*

Proof. Assume L is recognizable by DV1CA $\mathcal{A}_a = (Q, \Sigma, \{A\}, \delta, q_{\text{in}}, F_a)$. Moreover, let $|Q| = k$. Consider a word $w_m = \binom{c}{r}^{m+1} \binom{a}{r}^m a$ with $m > k$. In case $\binom{c}{r} \in \Sigma_{\text{pop}} \cup \Sigma_{\text{skip}}$, there exist $0 < i < j \leq m+1$ such that for some state $q \in Q$ the run of $\mathcal{A}_a$ on $\mathsf{pref}_i(w_m) = \binom{c}{r}^i$ as well as the run on $\mathsf{pref}_j(w_m) = \binom{c}{r}^j$ ends in the configuration $(q, \bot)$. Since $w_m \in L$, i.e., the run of $\mathcal{A}_a$ on w_m ends in a final state $q' \in F_a$, the run of $\mathcal{A}_a$ on the word $\binom{c}{r}^{m+1-(j-i)} \binom{a}{r}^m a \notin L$ also ends in the final state q' which yields the contradiction. If $\binom{c}{r} \in \Sigma_{\text{push}}$, there exist $0 < i < j \leq m+1$ such that for some state $q \in Q$ the run of $\mathcal{A}_a$ on $\mathsf{pref}_i(w_m)$ ends in the configuration $(q, A^i\bot)$ and the run on $\mathsf{pref}_j(w_m)$ ends in $(q, A^j\bot)$. Now, a similar pumping argument can be applied. Since the run of $\mathcal{A}_a$ on w_m ends in some final state $q' \in F_a$ and the stack height of every configuration of the run (except the initial one) is greater than zero, we can pump the infix $w_m(i+1) \cdots w_m(j)$ such that

the new word is accepted by $\mathcal{A}_a$ as the empty stack is never reached. For instance, the word $\binom{c}{r}^{m+1+(j-i)}\binom{a}{r}^m a \notin L$ is accepted by $\mathcal{A}_a$ which yields the contradiction. □

We conclude this section by presenting a parity pushdown game which is played on a labeled configuration graph of a deterministic blind one-counter machine which is not solvable by deterministic blind one-counter strategies. Consider the following DB1CM $\mathcal{M} = (Q, \Sigma, \{A\}, \delta, q_{\text{in}})$ with $Q = \{q_{\text{in}}, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b, c, d\}$ and the transition function defined by

- $\delta(q_{\text{in}}, a, X) = (q_{\text{in}}, AX)$, $\delta(q_{\text{in}}, b, A) = (q_1, \varepsilon)$,
- $\delta(q_1, b, A) = (q_1, \varepsilon)$, $\delta(q_1, c, X) = (q_2, X)$,
- $\delta(q_2, a, X) = (q_3, X)$, $\delta(q_2, b, X) = (q_4, X)$,
- $\delta(q_3, c, X) = (q_4, AX)$, $\delta(q_3, d, A) = (q_3, A)$,
- $\delta(q_4, c, X) = (q_3, AX)$, $\delta(q_4, d, A) = (q_4, A)$

where $X \in \{A, \bot\}$. Furthermore, let $Q_0 = \{q_2, q_3\}$ and $Q_1 = Q \setminus Q_0$ and the coloring function

$$\text{col}(q) = \begin{cases} 1, & \text{if } q = q_4, \\ 0, & \text{otherwise.} \end{cases}$$

The parity pushdown game $\mathcal{G} = (G(\mathcal{M}), \text{col})$ is depicted in Figure 3.4 where Player I configurations are represented by rectangle nodes and Player O configurations are rounded.

Player I begins a play by building up a finite prefix $a^n b^m c$ with $m \leq n$. If he picks an infinite number of a's by remaining in the initial state q_{in} forever, he loses, since $\text{col}(q_{\text{in}}) = 0$. After a prefix $a^n b^m c$ is provided, Player O has to decide whether to pick letter a or b. Player O wins if he can force to reach a loop in state q_3. On the other hand, he loses if a loop in q_4 is reached where Player I can stay forever by choosing d. Hence, a winning strategy for Player O has to pick letter a from a configuration $(q_2, A^i \bot)$ for $i > 0$, this is the case, if the prefix constructed by Player I contains more letters a than b. On the other hand, in order to win Player O has to pick letter b being in configuration $(q_2, \bot)$ which is the case if the prefix constructed by Player I contains equal number of a's and b's.

Notice, that such a winning strategy can easily be realized by a deterministic pushdown transducer. However, a winning strategy cannot be implemented by any DB1CT. To show this, we again use a language theoretic argument.

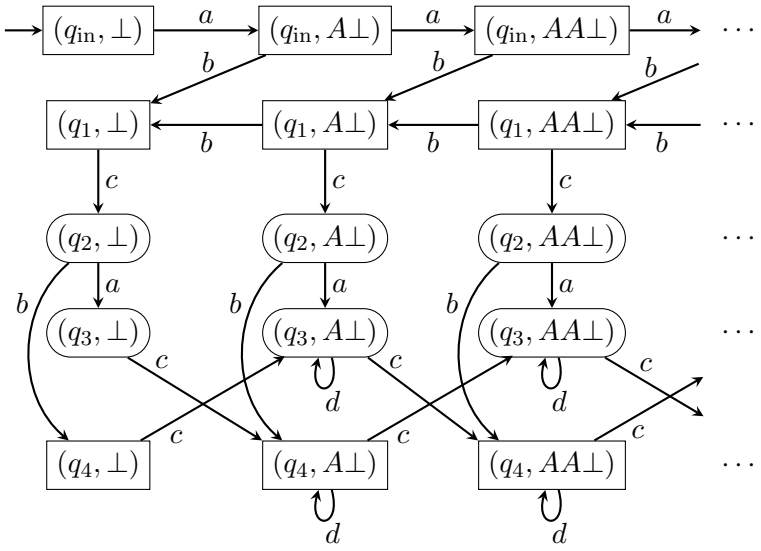


Figure 3.4: Parity blind one-counter game

Lemma 3.32. *The language $L = \{a^n b^n c \mid n > 0\}$ is not accepted by any DB1CA.*

Proof. Assume L is recognizable by DB1CA $\mathcal{A}_b = (Q, \Sigma, \{A\}, \delta, q_{\text{in}}, F_b)$. Consider a word $w = a^m b^m c \in L$ with $m > |Q|$. Moreover, let ρ be the accepting run of $\mathcal{A}_b$ on w. Clearly, there exist two positions $0 \leq i < j \leq m+1$ such that $\rho(i) = (q, A^{s_i}\bot)$ and $\rho(j) = (q, A^{s_j}\bot)$, for some $q \in Q$ and $s_i, s_j \in \mathbb{N}$. Let $\rho_1 = \rho(0) \cdots \rho(i)$, $\rho_2 = \rho(i+1) \cdots \rho(j)$ and $\rho_3 = \rho(j+1) \cdots \rho(|w|)$. We distinguish two cases. First, let $s_i < s_j$. Consider $\overline{\rho} = \rho_1 \rho_2 \rho_2' \rho_3'$ where $\mathrm{Pr}_0(\rho_2') = \mathrm{Pr}_0(\rho_2)$, $\mathrm{Pr}_0(\rho_3') = \mathrm{Pr}_0(\rho_3)$, $\mathsf{sh}(\rho_2'(k)) = \mathsf{sh}(\rho_2(k)) + |s_j - s_i|$ and $\mathsf{sh}(\rho_3'(k)) = \mathsf{sh}(\rho_3(k)) + |s_j - s_i|$, for $k \in \mathbb{N}$. Since $\mathcal{A}_b$ is blind, i.e., every transition enabled with empty stack is also enabled with nonempty stack, we have $\overline{\rho}$ is an accepting run of $\mathcal{A}_b$ on the word $a^{m+(j-i)} b^m c \notin L$. Otherwise, if $s_i \geq s_j$, then with the same argument $\rho_1 \rho_3'$ is an accepting run of $\mathcal{A}_b$ on the word $a^{m-(j-i)} b^m c \notin L$. Contradiction. □

3.3 Summary of Results

We investigated Gale-Stewart games defined by several types of contextfree winning conditions as well as games played on pushdown game graphs defined by several types of pushdown machines. In particular, the relation between the formats of the underlying pushdown machines of those games and the formats of their winning strategies was addressed.

By a uniform proof method we have shown that parity and stair parity pushdown games played on configuration graphs of deterministic, deterministic visibly, deterministic realtime and deterministic one-counter machines are determined with pushdown strategies of corresponding format. This directly implies the corresponding result for Gale-Stewart games defined by deterministic, deterministic visibly, deterministic realtime and deterministic one-counter parity and stair parity pushdown automata, since they can be modeled by pushdown games on corresponding configuration graphs.

On the contrary, besides these positive cases where the correspondence between the formats could be established, we showed that there are cases where strategies of the same format do not suffice. There are Gale-Stewart games with deterministic visibly one-counter winning conditions which are not determined with deterministic visibly one-counter strategies. Thus, this also holds for parity and stair parity pushdown games on visibly one-counter configuration graphs. Furthermore, we showed that deterministic blind one-counter strategies are not sufficient to solve pushdown games on configuration graphs of deterministic blind one-counter machines.

Chapter 4

Pushdown Delay Games

Delay games are used to model systems where an agent is allowed or even forced to defer the delivery of his actions for an arbitrary finite number of rounds. Such situations occur when, for instance, transmission of data in networks or processes equipped with buffers are modeled. Delay games can be viewed as a generalization of Gale-Stewart games where the strict alternation between the moves of the players is modified by allowing one of the players to postpone his moves for some time, thus obtaining a lookahead on the moves of the other player. Alternatively, one can require the other player to produce words of appropriate lengths instead of producing single symbols. For this purpose, so-called delay functions are provided that determine the lengths of the words to be produced in the individual rounds.

Hosch and Landweber proved for regular winning conditions that it is decidable whether the corresponding delay game can be won with bounded lookahead [HL72], i.e., from some round on both players produce single symbols in a strict alternation. This result was improved by Holtmann et al. in [HKT10], the authors showed that if a player wins a regular delay game with arbitrary lookahead, then he also can win with bounded lookahead which is doubly-exponential in the size of the parity automaton recognizing the winning condition. Moreover, the authors observe that this does not hold for deterministic contextfree delay games, i.e., there is a deterministic contextfree winning condition which can be won with arbitrary, but not with bounded lookahead.

Besides the application of delay games for modeling different scenarios where delay occurs, the study of delay games is also of basic theoretical interest. Given a binary relation $R \subseteq A \times B$, for sets A and B, a function $f\colon A \to B$ is called uniformization of R if $(a, f(a)) \in R$, for every $a \in A$. The uniformization problem asks, for a given class of relations $\mathcal{R}$ and a given class of functions $\mathcal{F}$, whether for every relation $R \in \mathcal{R}$ there exists a function $f_R \in \mathcal{F}$ which uniformizes R. Notice that a strategy of a Gale-Stewart game $\Gamma(L)$

with the winning condition $L \subseteq \Sigma^\omega \times \Sigma^\omega$ induces a mapping $\sigma\colon \Sigma^\omega \to \Sigma^\omega$ such that it is winning for Player O if $(\alpha, \sigma(\alpha)) \in L$, i.e., if σ uniformizes L. Since in the classical setting of strict alternation of the symbols picked by the players, the n-th symbol of $\sigma(\alpha)$ depends only on the first n letters of α, strategies in the classical setting give special kinds of functions. By means of delay games uniformization is studied for more general functions. Arbitrary finite lookahead, corresponding to the case where the n-th symbol of $\sigma(\alpha)$ depends on a finite prefix α, induces a continuous function in the Cantor topology over Σ^ω. Bounded lookahead, where there is some value which is not exceeded by the lookahead, i.e., eventually both players produce single symbols in a strict alternation, induces a Lipschitz-continuous function in the Cantor topology over Σ^ω.

Stated in terms of uniformization of binary relations, Hosch and Landweber proved that the uniformization problem is decidable for regular relations with Lipschitz-continuous functions. Holtmann et al. showed the equivalence of the existence of a continuous uniformization function and the existence of a Lipschitz-continuous uniformization function for regular relations. However, this equivalence does not hold for deterministic contextfree relations.

In this chapter we investigate contextfree delay games. We answer the following questions stated in [HKT10]:

1. Can the winner of a deterministic contextfree delay game be determined effectively, and

2. what amount of lookahead is necessary to win deterministic contextfree delay games?

We introduce delay games formally in Section 4.1. Then, in Section 4.2, the first question is answered. First, we show that for a fixed bounded lookahead determining the winner of a deterministic contextfree delay game is decidable. However, it is undecidable to determine whether a given player wins a deterministic contextfree game with arbitrary lookahead. Hence, we show undecidability of the uniformization problem for deterministic contextfree relations with continuous functions. Finally, we characterize sets of functions for which it is decidable whether a given player wins a deterministic contextfree delay game if the lookahead is restricted to one of this functions. In Section 6.4 the second question is answered. We present a deterministic contextfree delay game which is won if arbitrary lookahead is available, however, lookahead that is bounded by an elementary function does not suffice. Thus, a non-elementary lower bound is established.

4.1 Games with Delay

In this section, we introduce our notation for delay games. Let Σ_I be an input alphabet, Σ_O an output alphabet and $\Sigma = \Sigma_I \times \Sigma_O$. An ω-language $L \subseteq \Sigma^\omega$ and a so called delay function

$$f\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$$

define the delay game $\Gamma_f(L)$. The game $\Gamma_f(L)$ is played by the two players, Player I and Player O, in rounds $i \in \mathbb{N}$ as follows. In every round i, first Player I picks a word

$$u_i \in (\Sigma_I)^* \text{ with } |u_i| = f(i),$$

then Player O (being aware of the choice u_i of Player I) picks a letter $b_i \in \Sigma_O$.

A play of $\Gamma_f(L)$ is a sequence $u_0, b_0, u_1, b_1, u_2, b_2, \ldots$ which yields two ω-words, the input word $\alpha = u_0u_1u_2\cdots$ constructed by Player I and the output word $\beta = b_0b_1b_2\cdots$ produced by Player O. As for Gale-Stewart games, the language L provides the winning condition. Player O wins the play if and only if the ω-word $\alpha^\frown\beta$ induced by the play is contained in L.

A strategy for Player I is a function

$$\sigma_I\colon (\Sigma_O)^* \to (\Sigma_I)^* \text{ such that } |\sigma_I(w)| = f(|w|),$$

for every $w \in (\Sigma_O)^*$. A strategy for Player O is a function $\sigma_O\colon (\Sigma_I)^* \to \Sigma_O$. Consider a play $u_0, b_0, u_1, b_1, \ldots$ of $\Gamma_f(L)$. The play is consistent with a strategy σ_I for Player I if $u_n = \sigma_I(b_0 \cdots b_{n-1})$, for all $n \in \mathbb{N}$. The play is consistent with a strategy σ_O for Player O if $b_n = \sigma_O(u_0 \cdots u_n)$, for all $n \in \mathbb{N}$. Accordingly, a strategy σ is winning for Player i, for $i \in \{0, 1\}$, if every play which is consistent with σ is won by Player i, and in this case we say that Player i wins $\Gamma_f(L)$.

For a delay function $f\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$, we define the distance function d_f which we also refer to as lookahead function by

$$d_f(i) = \left(\sum_{j=0}^{i} f(j)\right) - (i+1),$$

i.e., $d_f(i)$ is the lookahead attained by Player O after choosing letter b_i in the i-th round. We classify delay functions according to their lookahead functions (and not to the kind of a delay function itself). Hence, for a delay function f, we say that f is a constant-delay function with delay d if

$$d_f(i) = d, \text{ for all } i \in \mathbb{N},$$

which is the case if $f(0) = d+1$ and $f(n) = 1$ for all $n > 0$. We say that f is a bounded-delay function if the corresponding distance function is bounded, i.e., there exists $N \in \mathbb{N}$ such that

$$|d_f(i)| \leq N \text{ for all } i \in \mathbb{N}.$$

This is the case, if $\{i \in \mathbb{N} \mid f(i) \neq 1\}$ is finite. Notice, that every constant-delay function is also a bounded-delay function. We say that f is a linear-delay function with delay $k > 0$ if

$$d_f(i) = (k-1)(i+1), \text{ for all } i \in \mathbb{N},$$

i.e., if $f(n) = k$ for all $n \in \mathbb{N}$. Finally, f is an elementary-delay function if

$$d_f \in \mathcal{O}(\exp_k), \text{ for some fixed } k \in \mathbb{N},$$

where $\exp_k$ denotes the k-fold exponential function $\exp_k \colon \mathbb{N} \to \mathbb{N}$ which is inductively defined by $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$.

Example 4.1. Let $\Sigma = \Sigma_I \times \Sigma_O$ with $\Sigma_I = \{a, \sharp\}$ and $\Sigma_O = \{a, b, \sharp\}$. Consider the following language $L \subseteq \Sigma^\omega$ with $w \in L$ if and only if

- $\mathrm{Pr}_0(w) = a^\omega$, or
- $\mathrm{Pr}_0(w)$ contains an infix $\sharp\sharp$ or an infix $\sharp a^{2n+1}\sharp$ for some $n \in \mathbb{N}$, or
- $w = \binom{a}{a}^{n_0}\binom{a}{b}^{n_0}\binom{\sharp}{\sharp}\binom{a}{a}^{n_1}\binom{a}{b}^{n_1}\binom{\sharp}{\sharp}\binom{a}{a}^{n_2}\binom{a}{b}^{n_2}\binom{\sharp}{\sharp}\cdots$ with $n_i \in \mathbb{N}$, for all $i \in \mathbb{N}$.

Notice, that a parity-DPDA recognizing L can easily be constructed. Consider the game $\Gamma(L)$. Clearly, Player I has a winning strategy in $\Gamma(L)$ by first producing two a's. If Player O responds by outputting ab then Player I continues by $aa(\sharp aa)^\omega$ and wins. Otherwise, if the first two output letters produced by Player O are not ab then Player I continues by $(\sharp aa)^\omega$ and wins.

However, Player O wins the delay game $\Gamma_f(L)$ with linear-delay function $f(n) = 2$ for all $n \in \mathbb{N}$. This is because in every round i, according to the delay function f, Player O is already informed about the prefix of the input word produced by Player I of length $2(i+1)$. Clearly, this lookahead is sufficient to determine whether to respond by the output letter a or b. $\diamond$

We conclude this section by briefly discussing determinacy of delay games with deterministic contextfree winning conditions. We show that deterministic contextfree delay games can be modeled as parity games on countable game graphs with finitely many colors. Then, determinacy of such games follows by Theorem 2.11. For this, we use the queue structure. For an alphabet Σ let the function $\mathsf{deq} \colon \Sigma^* \to \Sigma^*$ be defined by $\mathsf{deq}(aw) = w$ and $\mathsf{deq}(\varepsilon) = \varepsilon$, for $w \in \Sigma^*$ and $a \in \Sigma$.

Theorem 4.2. *Let $L \in \mathsf{DCFL}_\omega$ and $f \colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$ be a delay function. Then, $\Gamma_f(L)$ is determined.*

Proof. Let $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_{\text{in}}, \text{col})$ with $\Sigma = \Sigma_I \times \Sigma_O$ be a parity-DPDA with continuity property such that $L(\mathcal{P}) = L$. Consider the following parity game (G, col') where $G = (V, V_0, V_1, E, v_{\text{in}})$ is defined as follows. The set of vertices is given by

$$V = Q \times \Gamma^{*}\bot \times (\Sigma_I)^{*} \times \mathbb{N} \times \mathbb{N}.$$

The first two components of a vertex are used to store the current configuration, the third component stores the content of the current lookahead, and the last two components are used to keep track of the current round and the length of the word to be chosen by Player I given by the delay function. The initial vertex is defined by

$$v_{\text{in}} = (q_{\text{in}}, \bot, \varepsilon, 0, f(0)).$$

We define the edge relation E as follows. For every $(q, \gamma, w, i, j) \in V$, if $j > 0$ then for all $a \in \Sigma_I$

$$\big((q, \gamma, w, i, j),\ (q, \gamma, wa, i, j-1)\big) \in E,$$

otherwise, in case $j = 0$, if an ε-transition is defined from (q, γ), then

$$\big((q, \gamma, w, i, j),\ (q', \gamma', w, i, j)\big) \in E,$$

where $(q, \gamma) \overset{\varepsilon}{\vdash}_{\mathcal{P}} (q', \gamma')$, or else, for all $b \in \Sigma_O$,

$$\big((q, \gamma, w, i, j),\ (q', \gamma', \mathsf{deq}(w), i+1, f(i+1))\big) \in E,$$

where $(q, \gamma) \overset{\sigma}{\vdash}_{\mathcal{P}} (q', \gamma')$ with $\sigma = \binom{w(0)}{b}$. Define the partition of the set V of vertices by $V_0 = \{(q, \gamma, w, i, j) \in V \mid j = 0\}$, and $V_1 = V \setminus V_0$, accordingly. And let the coloring function be defined by $\text{col}'(v) = \text{col}(\text{Pr}_0(v))$. Clearly, by this construction Player p wins $\Gamma_f(L)$ if and only if Player p wins the parity game (G, col'), for $p \in \{0, 1\}$. □

4.2 Decision Problems

In this section, we consider various decision problems regarding delay games with contextfree winning conditions. One type of questions that we address is concerned with the existence of delay functions such that for a given deterministic contextfree winning condition Player O has a winning strategy in the corresponding delay games. The second kind of decision problems that we tackle ask whether Player O can win a delay game for a given winning condition and a given delay function. Here, we analyze for which classes of delay functions this problem is decidable.

We begin by showing that for a fixed bounded-delay function the winner can be determined effectively. It turns out that this result is the most general decidability result. Hence, relaxing the boundedness condition on the distance function makes the problem undecidable (see Theorem 4.6).

Theorem 4.3. *The following problem is decidable.*

Given: *Parity-DPDA $\mathcal{P}$ and a bounded-delay function f.*

Question: *Does Player O win the delay game $\Gamma_f(L(\mathcal{P}))$?*

Proof. Using the construction of the proof of Theorem 4.2 we show that $\Gamma_f(L(\mathcal{P}))$ can be reduced to a parity game played on a configuration graph of a deterministic pushdown machine. We exploit the fact that in case of bounded-delay functions, the queue which stores the content of the lookahead is bounded, as well as the number of rounds in which Player I picks a word containing more than one letter is finite.

Let $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_{\text{in}}, \text{col})$ with $\Sigma = \Sigma_I \times \Sigma_O$ and let $N \in \mathbb{N}$ be the bound of the distance function, i.e., $d_f(i) \leq N$ for all $i \in \mathbb{N}$. Moreover, let $M \in \mathbb{N}$ such that $f(i) = 1$ for all $i \geq M$, which exists as $\{i \in \mathbb{N} \mid f(i) \neq 1\}$ is finite. Furthermore, let $F = \max\{f(i) \mid i \in \mathbb{N}\}$.

Consider the following DPDM $\mathcal{M} = (Q^{\mathcal{M}}, \Sigma \cup \Sigma_I, \Gamma, \delta^{\mathcal{M}}, q_{\text{in}}^{\mathcal{M}})$ where

$$Q^{\mathcal{M}} = Q \times (\Sigma_I)^{\leq N} \times [M+1] \times [F+1],$$

$$q_{\text{in}}^{\mathcal{M}} = (q_{\text{in}}, \varepsilon, 0, f(0)),$$

and for every $s = (q, w, i, j) \in Q^{\mathcal{M}}$, if $j > 0$, then for every $A \in \Gamma_\perp$, $a \in \Sigma_I$

$$\delta^{\mathcal{M}}(s, A, a) = \big((q, wa, i, j-1), A\big),$$

otherwise, if $j = 0$, then for every $A \in \Gamma_\perp$ and $b \in \Sigma_O$

$$\delta^{\mathcal{M}}(s, A, \varepsilon) = \big((q', w, i, j), \gamma\big) \text{ if } \delta(q, A, \varepsilon) = (q', \gamma),$$

$$\delta^{\mathcal{M}}(s, A, b) = \big((q', \mathsf{deq}(w), \mathsf{succ}(i), f(\mathsf{succ}(i))), \gamma\big) \text{ if } \delta(q, A, \sigma) = (q', \gamma),$$

where $\sigma = \binom{w(0)}{b}$ and $\mathsf{succ}(i) = i + 1$ for $i \in [M]$ and $\mathsf{succ}(M) = M$. Clearly, with partition of $Q^{\mathcal{M}}$ given by $Q_0 = \{(q, w, i, j) \in Q^{\mathcal{M}} \mid j = 0\}$ and $Q_1 = Q^{\mathcal{M}} \setminus V_0$ and coloring $\text{col}'(s) = \text{col}(\text{Pr}_0(s))$, for $s \in Q^{\mathcal{M}}$, a parity game $(G(\mathcal{M}), \text{col}')$ is obtained such that Player p wins $\Gamma_f(L(\mathcal{P}))$ if and only if Player p wins $(G(\mathcal{M}), \text{col}')$, for $p \in \{0, 1\}$. □

We continue with the undecidability results which are obtained by a reduction from the halting problem for 2-register machines. Such a machine comprises two registers, each holding a non-negative integer, and a finite control which can increment or decrement the value of a register, furthermore

it can be tested whether the content of a register equals zero. Formally, a 2-register machine $\mathcal{R}$ is a list of pairs

$$\mathcal{R} = \big((0\colon I_0), \ldots, (k\colon I_k)\big)$$

where $I_\ell \in \{\texttt{INC(X}_i\texttt{)}, \texttt{DEC(X}_i\texttt{)}, \texttt{IF X}_i\texttt{=0 GOTO m} \mid i \in \{0,1\} \text{ and } m \in [k{+}1]\}$, for every $\ell \in [k]$, and $I_k = \texttt{HALT}$. The first component of a pair is called line number and the second one is the instruction.

The computation of $\mathcal{R}$ starts in the initial pair $(0\colon I_0)$. For a pair $(\ell\colon I_\ell)$ with $\ell \in [k+1]$, if $I_\ell = \texttt{INC(X}_i\texttt{)}$ then the value of register i, for $i \in \{0,1\}$, is incremented by 1, and $\mathcal{R}$ proceeds to the next pair with the line number $\ell + 1$. if $I_\ell = \texttt{DEC(X}_i\texttt{)}$ then the value of register i is decremented by 1 in case it holds a nonzero value, else the decrease operation has no effect, i.e., the value remains zero, accordingly the register machine proceeds to the next pair with the line number $\ell{+}1$. If $I_\ell = \texttt{IF X}_i\texttt{=0 GOTO m}$, for $m \in [k{+}1]$, then the next line number is determined by examining the content of register i, if it holds value 0 then $\mathcal{R}$ proceeds to line number m, else if the value is nonzero the computation continues with line number $\ell{+}1$. Finally, if $I_\ell = \texttt{HALT}$ then the computation halts.

A configuration of $\mathcal{R}$ is a tuple (ℓ, n_0, n_1) where $\ell \in [k+1]$ is a line number and $n_i \in \mathbb{N}$ is the value of register i, for $i \in \{0,1\}$. Configuration $c_{\text{in}} = (0,0,0)$ is the initial configuration. Moreover, we call a configuration c halting configuration if $\text{Pr}_0(c) = k$, i.e., c contains line number k for which the corresponding instruction is $I_k = \texttt{HALT}$. For two configurations $c, c' \in [k+1] \times \mathbb{N} \times \mathbb{N}$, we write $c \vdash_{\mathcal{R}} c'$ if c' is the successor configuration of c, i.e., for all $\ell \in [k+1]$, $n_0, n_1 \in \mathbb{N}$ and $i \in \{0,1\}$

$$\begin{aligned}
&(\ell, n_0, n_1) \vdash_{\mathcal{R}} (\ell+1, n_0+1, n_1) && \text{if } I_l = \texttt{INC(X}_0\texttt{)},\\
&(\ell, n_0, n_1) \vdash_{\mathcal{R}} (\ell+1, n_0, n_1+1) && \text{if } I_l = \texttt{INC(X}_1\texttt{)},\\
&(\ell, n_0, n_1) \vdash_{\mathcal{R}} (\ell+1, n_0-1, n_1) && \text{if } I_l = \texttt{DEC(X}_0\texttt{)} \text{ and } n_0 \neq 0,\\
&(\ell, n_0, n_1) \vdash_{\mathcal{R}} (\ell+1, n_0, n_1-1) && \text{if } I_l = \texttt{DEC(X}_1\texttt{)} \text{ and } n_1 \neq 0,\\
&(\ell, n_0, n_1) \vdash_{\mathcal{R}} (\ell+1, n_0, n_1) && \text{if } I_l = \texttt{DEC(X}_i\texttt{)} \text{ and } n_i = 0,\\
&(\ell, n_0, n_1) \vdash_{\mathcal{R}} (\ell+1, n_0, n_1) && \text{if } I_l = \texttt{IF X}_i\texttt{=0 GOTO m} \text{ and } n_i \neq 0,\\
&(\ell, n_0, n_1) \vdash_{\mathcal{R}} (m, n_0, n_1) && \text{if } I_l = \texttt{IF X}_i\texttt{=0 GOTO m} \text{ and } n_i = 0.
\end{aligned}$$

The run of $\mathcal{R}$ is either a finite sequence $\rho = \rho(0) \cdots \rho(r)$ of configurations such that $\rho(0) = c_{\text{in}}$, $\rho(r)$ is halting and $\rho(n) \vdash_{\mathcal{R}} \rho(n+1)$ for all $n \in [r]$ or it is an infinite sequence $\rho = \rho(0)\rho(1)\cdots$ of configurations such that $\rho(0) = c_{\text{in}}$ and $\rho(n) \vdash_{\mathcal{R}} \rho(n+1)$ for all $n \in \mathbb{N}$. We say that $\mathcal{R}$ halts if the run of $\mathcal{R}$ is finite. It is well-known that the halting problem for 2-register machines (which is to decide, given a 2-register machine $\mathcal{R}$, whether $\mathcal{R}$ halts) is undecidable [SS63].

Now, we show that for deterministic contextfree winning conditions determining whether there is an arbitrary delay function such that Player O has a winning strategy in the corresponding delay game is undecidable.

Theorem 4.4. *The following problem is undecidable.*

Given: *Parity-DPDA $\mathcal{P}$.*

Question: *Does there exist a delay function f such that Player O wins the delay game $\Gamma_f(L(\mathcal{P}))$?*

Proof. We proceed by a reduction from the halting problem for 2-register machines. For such a machine $\mathcal{R} = \big((0\colon I_0), \ldots, (k{-}1\colon I_{k-1}), (k\colon \texttt{HALT})\big)$, we encode configurations by words over $\{r_0, r_1\} \cup [k{+}1]$ such that a configuration (ℓ, n_0, n_1) is encoded by $\ell\, r_0^{n_0}\, r_1^{n_1}$. Define

$$\mathsf{Conf} = \{\ell\, r_0^{n_0}\, r_1^{n_1} \mid \ell \in [k+1],\, n_0, n_1 \in \mathbb{N}\}$$

and $\mathsf{Conf}_{\text{in}} = 0$, i.e., Conf is the set of all encodings of configurations and $\mathsf{Conf}_{\text{in}}$ is the encoding of the initial configuration. For two encodings of configurations $c = \ell\, r_0^{n_0}\, r_1^{n_1}$ and $c' = \ell'\, r_0^{n_0'}\, r_1^{n_1'}$ we write $c \vdash_{\mathcal{R}} c'$ if for the corresponding configurations $(\ell, n_0, n_1) \vdash_{\mathcal{R}} (\ell', n_0', n_1')$ holds. Notice, that for two encodings $c, c' \in \mathsf{Conf}$, if $c \vdash_{\mathcal{R}} c'$ then we have $|c'| \leq |c| + 1$.

Now, consider the following winning condition $L_{\mathcal{R}}$ over $\Sigma = \Sigma_I \times \Sigma_O$, where $\Sigma_I = \{\sharp, r_0, r_1\} \cup [k+1]$ and $\Sigma_O = \{N, E_0, E_1, L\}$. A word $w \in \Sigma^\omega$ is contained in $L_{\mathcal{R}}$ if

- $\mathrm{Pr}_0(w) \neq \sharp\, c_0 \sharp\, c_1 \sharp\, c_2 \cdots$ where $c_i \in \mathsf{Conf}$, for $i \in \mathbb{N}$, and $c_0 = \mathsf{Conf}_{\text{in}}$, or
- there exists exactly one $n \in \mathbb{N}$ such that $w(n) \in \{\sharp\} \times \{E_0, E_1, L\}$ and there is an infix v of w such that
  - $\mathrm{Pr}_0(v) = \sharp\, \ell\, r_0^{n_0}\, r_1^{n_1}\, \sharp\, \ell'\, r_0^{n_0'}\, r_1^{n_1'}\, \sharp$, for some $n_0, n_1, n_0', n_1' \in \mathbb{N}$, and
  - $\mathrm{Pr}_1(v) = X\, N^{|v|-1}$ where $X \in \{E_0, E_1, L\}$, and
  - if $X = E_i$ then $n_i' \neq m_i$, for $i \in \{0, 1\}$, else if $X = L$ then $\ell' \neq \ell_s$ where $\ell\, r_0^{n_0}\, r_1^{n_1} \vdash_{\mathcal{R}} \ell_s\, r_0^{m_0}\, r_1^{m_1}$.

According to the first condition, Player I loses if he does not build up a word of the form $\sharp\mathsf{Conf}_{\text{in}}(\sharp\mathsf{Conf})^\omega$. So, consider such a word $\sharp c_0 \sharp c_1 \sharp c_2 \sharp \cdots$ with $c_i \in \mathsf{Conf}$ for all $i \in \mathbb{N}$ and $c_0 = \mathsf{Conf}_{\text{in}}$. In order to win, Player O has to find an infix $\sharp c_j \sharp c_{j+1}$ such that $c_j \vdash_{\mathcal{R}} c_{j+1}$ does not hold. Furthermore, he has to state why it does not hold. So, at each position where Player I has picked a $\sharp$ symbol Player O indicates whether he believes that the following two encodings indeed encode successive configurations or not. By choosing the letter N Player O states that he believes that the following two encodings

$$\big(\ldots,(7\colon \texttt{INC(X}_1\texttt{)}),(8\colon \texttt{DEC(X}_1\texttt{)}),\ldots\big)$$

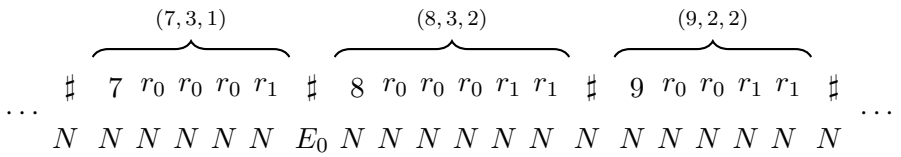


Figure 4.1: Part of a play encoding three configurations

adhere to the successor relation $\vdash_{\mathcal{R}}$. On the other hand, if Player O believes that the successor relation $\vdash_{\mathcal{R}}$ is violated the error which occurs is indicated by a letter from $\{E_0, E_1, L\}$. Letter E_0 is used to declare that the first register is not updated correctly, accordingly E_1 indicates that the second register is updated incorrectly. An incorrect update of the line number is indicated by letter L. At any other position, letter N has to be chosen.

Figure 4.1 shows an example of a play infix with the encoding of three configurations. The first update is correct, since $(7,3,1) \vdash_{\mathcal{R}} (8,3,2)$, while the second one is not, since $(8,3,2) \vdash_{\mathcal{R}} (9,3,1)$. Both registers are updated incorrectly, letter E_0 claims the violation of the successor relation for the first register in front of the the encodings of configurations $(8,3,2)$ and $(9,2,2)$.

We show, that $L_{\mathcal{R}}$ is recognized by a parity-DPDA $\mathcal{P}_{\mathcal{R}}$. Notice, that the first condition of the definition of $L_{\mathcal{R}}$, whether the first component is a word in $\sharp\mathsf{Conf}_{\text{in}}(\sharp\mathsf{Conf})^{\omega}$, is regular. Moreover, the existence of exactly one letter from $\{\sharp\} \times \{E_0, E_1, L\}$ can also be checked in terms of a finite automaton. Furthermore, if a letter from $\{\sharp\} \times \{E_0, E_1, L\}$ is encountered while reading an input word, $\mathcal{P}_{\mathcal{R}}$ has to check whether the following infix v which consists of the next two encodings of configurations satisfies the above requirements. Let $\mathrm{Pr}_0(v) = \sharp\, \ell\, r_0^{n_0}\, r_1^{n_1}\, \sharp\, \ell'\, r_0^{n_0'}\, r_1^{n_1'}\, \sharp$, and $\ell\, r_0^{n_0}\, r_1^{n_1} \vdash_{\mathcal{R}} \ell_s\, r_0^{m_0}\, r_1^{m_1}$ for $n_0, n_1, n_0', n_1', m_0, m_1 \in \mathbb{N}$ and $\ell, \ell', \ell_s \in [k+1]$. if $\mathrm{Pr}_1(v(0)) = E_i$ for $i \in \{0,1\}$, then $n_i' \neq m_i$ if and only if $n_i' \neq n_i + s$, where

$$s = \begin{cases} 1 & \text{if } I_\ell = \texttt{INC(X}_i\texttt{)}, \\ -1 & \text{if } I_\ell = \texttt{DEC(X}_i\texttt{)} \text{ and } n_i > 0, \\ 0 & \text{otherwise.} \end{cases}$$

which can be checked using the pushdown stack by pushing a symbol onto the stack while $r_i^{n_i}$ is read. The stack content is then used to check whether $n_i' = n_i + s$. Otherwise, if $\mathrm{Pr}_1(v(0)) = L$ then $\ell' \neq \ell_s$ if and only if

- $I_\ell \in \{\texttt{INC(X}_i\texttt{)}, \texttt{DEC(X}_i\texttt{)}\}$ and $\ell' \neq \ell + 1$, or

- $I_\ell = \texttt{IF X}_i\texttt{=0 GOTO m}$ and $n_i > 0$ and $\ell' \neq \ell + 1$, or
- $I_\ell = \texttt{IF X}_i\texttt{=0 GOTO m}$ and $n_i = 0$ and $\ell' \neq m$.

for $i \in \{0, 1\}$, $m \in [k{+}1]$. This can be checked in terms of a finite automaton.

It remains to show that $\mathcal{R}$ halts if and only if there exists a delay function f such that Player O wins the game $\Gamma_f(L_\mathcal{R})$.

Suppose $\mathcal{R}$ halts and consider the linear-delay function with delay 6, i.e., $f(i) = 6$ for all $i \in \mathbb{N}$. We claim that Player O has a winning strategy for $\Gamma_f(L_\mathcal{R})$ which finds the first violation of the successor relation introduced by Player I. In round 0 Player I picks 6 letters which are sufficient for Player O to check whether Player I has encoded the initial configuration and its successor configuration, as the length of such an encoding is bounded by 6. Now consider a round $i > 0$. If the i-th input letter is not a $\sharp$, then Player O chooses an N as the output letter in this round. So suppose that the i-th input letter is a $\sharp$ and that Player O has not yet signaled a violation of the successor relation by choosing a letter from $\{E_0, E_1, L\}$ up to this position. Player I has produced a word $\sharp x \sharp y$ of length $6(i+1)$ where $|x| = i - 1$ and hence, $|y| = 5(i + 1)$. Note that both x and y might contain the letter $\sharp$. Let c denote the last encoding of a configuration in x and c' the first encoding of a configuration in y. As Player O has not signaled an violation of the successor relation at the previous $\sharp$, we know that c' is well-defined and that $c \vdash_\mathcal{R} c'$ holds. We have $|c| \leq |x| = i - 1$ and hence $|c'| \leq i$. Thus, the successor configuration of c' is encoded by at most $i + 1$ letters. As $i + (i + 1) + 2 < 5(i + 1)$ for all $i > 0$, in every round Player O has enough information to detect a violation of the successor relation if one is introduced. This strategy is indeed winning for Player O as Player I has eventually to violate the successor relation, since a halting configuration has no successor.

Now suppose $\mathcal{R}$ does not halt. For any delay function f, Player I has a winning strategy in the delay game $\Gamma_f(L_\mathcal{R})$ by producing the word $\sharp c_0 \sharp c_1 \sharp c_2 \sharp \cdots$ where $c_0 = \mathsf{Conf}_{\text{in}}$, $c_i \in \mathsf{Conf}$ and $c_i \vdash_\mathcal{R} c_{i+1}$ for all $i \in \mathbb{N}$. This means, Player I builds up the encoding of the infinite run of $\mathcal{R}$. Hence, due to determinacy, Player O does not win $\Gamma_f(L_\mathcal{R})$. □

Notice, that if Player O wins $\Gamma_f(L_\mathcal{R})$ for some delay function f then there is also a constant-delay function g such that Player O wins $\Gamma_g(L_\mathcal{R})$. The reason is the following. Player O wins $\Gamma_f(L_\mathcal{R})$ if and only if $\mathcal{R}$ halts. So let $\rho = \rho(0) \cdots \rho(r)$ be the run of $\mathcal{R}$ and let c_i denote the corresponding encoding of the configuration $\rho(i)$, for $i \in [r + 1]$. Clearly, since the length of every encoding c_i is bounded by $|c_i| \leq i + 1$ for $i \in [r + 1]$, we have $|\sharp c_0 \sharp \cdots \sharp c_r \sharp| \leq r + 2 + (r + 1)(r + 2)/2 = d_\mathcal{R}$. Hence, for Player O it suffices to be informed about the first $d_\mathcal{R}$ letters of the input produced by Player I

to be able to react in a proper manner. Thus, Player O wins $\Gamma_g(L_{\mathcal{R}})$ for the constant-delay function g with $g(0) = d_{\mathcal{R}}$ and $g(n) = 1$ for all $n > 0$.

Corollary 4.5. *The following problems are undecidable.*

Given: *Parity-DPDA $\mathcal{P}$.*

Question: *Does there exist a constant-delay function f such that Player O wins the delay game $\Gamma_f(L(\mathcal{P}))$?*

Given: *Parity-DPDA $\mathcal{P}$.*

Question: *Does there exist a linear-delay function f such that Player O wins the delay game $\Gamma_f(L(\mathcal{P}))$?*

Given: *Parity-DPDA $\mathcal{P}$ and $k > 0$.*

Question: *Does Player O win the delay game $\Gamma_f(L(\mathcal{P}))$ where f is the linear-delay function $f(i) = k$ for all $i \in \mathbb{N}$?*

By slightly modifying the winning condition $L_{\mathcal{R}}$ described above, we show that all undecidability results hold even for a very restricted class of deterministic contextfree winning conditions, namely winning conditions recognizable by E-DV1CA. The idea for defining a visibly winning condition $L'_{\mathcal{R}}$ is to let Player O control the behavior of the stack. To do this, Player O is supplied with additional letters to indicate which kind of a transition has to be performed, push-, pop- or a skip-transition, respectively. We define $\Sigma_O = \{\mathsf{Push}, \mathsf{Pop}, N, L, E_0, E_1, C\}$, the input alphabet remains unchanged $\Sigma_I = \{\sharp, r_0, r_1\} \times [k+1]$ and let $\Sigma_{\text{push}} = \Sigma_I \times \{\mathsf{Push}\}$, $\Sigma_{\text{pop}} = \Sigma_I \times \{\mathsf{Pop}\}$, and $\Sigma_{\text{skip}} = \Sigma_I \times \{N, L, E_0, E_1\}$, i.e., the membership of a letter to one of the alphabets depends only on the second component of the letter.

Furthermore, in order $L'_{\mathcal{R}}$ to be E-recognizable, we have to relax the requirement on the input word constructed by Player I such that it is no longer required to be of the form $\sharp\mathsf{Conf}_{\text{in}}(\sharp\mathsf{Conf})^\omega$. To compensate this, we require that Player I starts with the encoding of the initial configuration $\sharp 0\sharp$. Furthermore, Player O is provided with an additional symbol C. At any $\sharp$ position Player O can signal that the subsequent input word produced by Player I is not of the form $c\sharp c'\sharp\cdots$ where $c, c' \in \mathsf{Conf}$ such that $|c'| \leq |c|+1$ by answering this $\sharp$ by the letter C. If Player I does not produce a valid sequence of encodings of configurations, then Player 0 can claim this by indicating the appropriate position.

Finally, as soon as a $\sharp$ symbol is answered by one of the letters E_0, E_1, L or C for the first time, letters $\mathsf{Push}, \mathsf{Pop}$ and N are used to enable the automaton to perform the corresponding tests. All of them involve only comparison

of the lengths of two appropriate infixes. Hence, all necessary tests can be implemented using a single stack symbol.

We conclude this section by characterizing sets $\mathcal{F}$ of delay functions for which it is decidable whether Player O wins a given delay game with some delay function from $\mathcal{F}$. To give a general criterion, we define bounded sets of delay functions. We say that a set $\mathcal{F}$ of delay functions $f\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$ is bounded, if there exists a bound $N \in \mathbb{N}$ such that for every $f \in \mathcal{F}$ and every $i \in \mathbb{N}$ we have $d_f(i) \leq N$, i.e., there is a global bound on the lookahead for Player O given by the functions in $\mathcal{F}$. Notice, that every bounded set of delay function consists of bounded-delay functions. However, the converse does not hold, e.g. the set of all bounded-delay functions is unbounded.

Theorem 4.6. *Let $\mathcal{F}$ be a set of delay functions. The following problem is decidable if and only if $\mathcal{F}$ is bounded.*

Given: *Parity-DPDA $\mathcal{P}$.*

Question: *Does there exist an $f \in \mathcal{F}$ such that Player O wins the delay game $\Gamma_f(L(\mathcal{P}))$?*

Proof. Consider a bounded set $\mathcal{F}$ of delay functions. We define a partial order on delay functions as follows. Let f and g be delay functions, define $f \leq g$ if and only if $d_f(i) \leq d_g(i)$ for all $i \in \mathbb{N}$, i.e., the delay function g allows in any round at least as much lookahead as the delay function f does. Moreover, we say that a delay function $f \in \mathcal{F}$ is maximal if for all $g \in \mathcal{F}$, $f \leq g$ implies $f = g$. Let

$$\mathcal{F}_{\max} = \{f \in \mathcal{F} \mid f \text{ is maximal}\}$$

denote the set of all maximal delay functions from $\mathcal{F}$. Applying Dickson's Lemma [Dic13] and the boundedness of $\mathcal{F}$ it follows that $\mathcal{F}_{\max}$ is finite.

We claim that there exists a delay function $f \in \mathcal{F}$ such that Player O wins the delay game $\Gamma_f(L(\mathcal{P}))$ if and only if there exists a delay function $g \in \mathcal{F}_{\max}$ such that Player O wins the delay game $\Gamma_g(L(\mathcal{P}))$. As $\mathcal{F}_{\max}$ is finite and every $g \in \mathcal{F}_{\max}$ is also a bounded-delay function, the latter property can be decided by Theorem 4.3.

The implication from right to left is trivially true, so assume there exists an $f \in \mathcal{F} \setminus \mathcal{F}_{\max}$ such that Player O wins $\Gamma_f(L(\mathcal{A}))$. Then, there is a function $g \in \mathcal{F}_{\max}$ such that $f \leq g$, i.e., the function g admits Player O at least as much lookahead as f. Hence, a winning strategy for Player O in $\Gamma_f(L(\mathcal{P}))$ can easily be turned into a winning strategy for her in $\Gamma_g(L(\mathcal{P}))$.

Now consider an unbounded set $\mathcal{F}$ of delay functions, i.e., for every $N \in \mathbb{N}$ there exists a delay function $f \in \mathcal{F}$ and $i \in \mathbb{N}$ such that $d_f(i) > N$. We adapt the winning condition $L_{\mathcal{R}}$ described in the proof of Theorem 4.4 by allowing

Player O to postpone the beginning of the simulation of the computation of the 2-register machine $\mathcal{R}$ until enough lookahead is attained for Player O to inspect the complete halting computation of $\mathcal{R}$, in case there exists one, before potential violations of the successor relation have to be indicated.

Given a 2-register machine $\mathcal{R} = \big((0\colon I_0), \ldots, (k\colon I_k)\big)$, define $\mathsf{Conf}_{\text{in}}$ and Conf as in the proof of Theorem 4.4, and consider the following winning condition $L'_{\mathcal{R}}$ over $\Sigma = \Sigma_I \times \Sigma_O$, where $\Sigma_I = \{\sharp, r_0, r_1, N\} \cup [k+1]$ and $\Sigma_O = \{N, E_0, E_1, L, S\}$. A word $w \in \Sigma^\omega$ is contained in $L'_{\mathcal{R}}$ if

- $\mathrm{Pr}_0(w) \neq uv$ with $u = N^*$ and v is either N^ω or $\sharp c_0 \sharp c_1 \sharp c_2 \cdots$ where $c_i \in \mathsf{Conf}$, for $i \in \mathbb{N}$, and $c_0 = \mathsf{Conf}_{\text{in}}$, or
- $\mathrm{Pr}_1(w) = N^\omega$ and $\mathrm{Pr}_0(w) \neq N^\omega$, or
- $w = uv$ with $u = \binom{N}{N}^* \binom{N}{S} \binom{N}{N}^*$ and either $v \in L_{\mathcal{R}}$ or $\mathrm{Pr}_0(v) = N^\omega$

Player I has to build a word of the form $N^*\sharp\mathsf{Conf}_{\text{in}}(\sharp\mathsf{Conf})^\omega$ or N^ω. If he does not adhere to the format, he loses. Furthermore, in order to win Player I may produce the word N^ω if and only if Player O never plays letters E_0, E_1, L, S. Letter S is used to indicate that the simulation eventually has to start. Thus, if Player O plays the letter S, then Player I has to play a word of the form $N^*\sharp c_0 \sharp c_1 \sharp c_2 \cdots$ with $c_i \in \mathsf{Conf}$ for every $i > 0$, and $c_0 = \mathsf{Conf}_{\text{in}}$.

Again, in order to win, Player O has to find a pair c_j, c_{j+1} such that $c_j \vdash_{\mathcal{R}} c_{j+1}$ does not hold. The mechanism to do so is the same as the one described in the proof of Theorem 4.4.

Suppose $\mathcal{R}$ halts after r computation steps. Then, as mentioned above, the finite run of $\mathcal{R}$ is encoded by at most $d_{\mathcal{R}}$ letters. Let $f \in \mathcal{F}$ and $i \in \mathbb{N}$ such that $d_f(i) \geq d_{\mathcal{R}}$. Player O has a winning strategy in $\Gamma_f(L'_{\mathcal{R}})$. In the first i rounds, she chooses N. If Player I has picked in a round $j \leq i+1$ a word $u_j \neq N^{f(j)}$, then Player O wins by playing N ad infinitum. Otherwise, Player O plays S in round $i+1$. Hence, in order to win Player I eventually has to start simulating $\mathcal{R}$, say at position $j > i$. As d_f is non-decreasing, Player O has at least $d_{\mathcal{R}}$ letters lookahead when picking her letter in any round $j' \geq j$. As the machine halts, this lookahead enables Player O to detect a position where the successor relation $\vdash_{\mathcal{R}}$ is violated which Player I has to introduce, since a halting configuration does not have a successor configuration.

If $\mathcal{R}$ does not halt, then Player I has a winning strategy in $\Gamma_f(L'_{\mathcal{R}})$ for every delay function $f \in \mathcal{F}$. As long as Player O has not played S, Player I picks $N^{f(i)}$ in round i. As soon as letter S is picked by Player O, he starts producing the word $\sharp c_0 \sharp c_1 \sharp c_2 \cdots$, where $c_0, c_1, c_2, \ldots$ are encodings of the infinite run of $\mathcal{R}$ starting in the initial configuration. □

We conclude by giving two remarks concerning the above theorem. First, notice that $\mathcal{F}$ is not part of the input of the decision problem considered above. Hence, Theorem 4.6 is stated for any set $\mathcal{F}$ of delay functions without having to represent $\mathcal{F}$ effectively. Moreover, Using the ideas presented above, one can show that Theorem 4.6 holds even for weak-parity-DV1CA. However, E-acceptance and A-acceptance are not sufficient in this case, because of the modified interplay between the two players. On the one hand, Player O has to be forced to play letter S and on the other hand Player I has to be forced to start the simulation after Player O picked letter S.

4.3 Lower Bounds on Delays

In this section we analyze the extent of the lookahead necessary to win delay games with deterministic contextfree winning conditions. We show that there exists a deterministic contextfree winning condition L and a delay function f such that Player O wins the game $\Gamma_f(L)$, but for any elementary-delay function g $\Gamma_g(L)$ is won by Player I. To this end, the idea of the previous section is adapted. We define an infinite set of finite words and a successor relation on this set such that a successor word is exponentially longer than its predecessor. Again, the task of Player I is to produces an infinite sequence of such words and Player O has to find and indicate a violation of the successor relation. In contrast to the specifications of the previous section, first, Player I is required to eventually introduce a violation of the successor relation and second, Player O does not indicate a potential violation in front of the i-th word, but with her i-th bit. Due to the exponential growth of the successive words our result is obtained.

Theorem 4.7. *There exists a parity-DPDA $\mathcal{P}$ and a delay function f such that Player O wins the delay game $\Gamma_f(L(\mathcal{P}))$, but Player I wins $\Gamma_g(L(\mathcal{P}))$ for every elementary-delay function g.*

Proof. Let $\Sigma^\sharp = \{\binom{\sharp}{N}, \binom{\sharp}{D}, \binom{\sharp}{C}\}$ and $\Sigma^\flat = \{\binom{\flat}{N}, \binom{\flat}{H}\}$ be two alphabets. We define the language $L_B \subseteq (\Sigma^\flat \cup \{0\})^*$ as follows. A word w is contained in L_B if and only if w is of the form

$$w = \flat_0\, 0^{n_0}\, \flat_1\, 0^{n_1}\, \flat_2 \cdots \flat_{k-1}\, 0^{n_{k-1}},$$

where $k > 0$, $n_0 = 1$, $n_i > 0$ and $\flat_i \in \Sigma^\flat$, for all $i \in [k]$. We call a word $w \in L_B$ block. Moreover, we say that a block $w = \flat_0 0^{n_0} \flat_1 0^{n_1} \flat_2 \cdots \flat_{k-1} 0^{n_{k-1}}$ consists of k $\flat$-blocks $\flat_i 0^{n_i}$. Moreover, a block $w \in L_B$ consisting of only one $\flat$-block is called initial block, we denote the set of initial blocks by $L_{B_{\text{in}}} = \{\flat 0 \mid \flat \in \Sigma^\flat\}$.

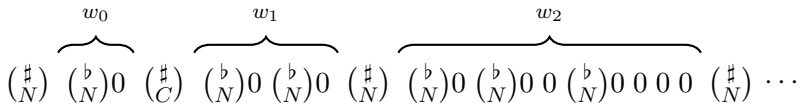


Figure 4.2: A prefix containing three blocks w_0, w_1 and w_2

Now, we define a successor relation $\vdash$ on L_B. For two blocks

$$w = \flat_0\, 0^{n_0}\, \flat_1\, 0^{n_1}\, \flat_2\, \cdots\, \flat_{k-1}\, 0^{n_{k-1}} \text{ and}$$
$$v = \flat'_0\, 0^{m_0}\, \flat'_1\, 0^{m_1}\, \flat'_2\, \cdots\, \flat'_{l-1}\, 0^{m_{l-1}},$$

where $k, l, n_i, m_j > 0$, $n_0 = m_0 = 1$, and $\flat_i, \flat'_j \in \Sigma^\flat$, for all $i \in [k]$ and $j \in [l]$, define $w \vdash v$ if

- $l = |w|$, i.e., v consists of $|w|$ many $\flat$-blocks, and
- $m_{j+1} = 2m_j$ for all $j \in [l-1]$, i.e., for every $\flat$-block $\flat'_j 0^{m_j}$ in v the consecutive $\flat$-block $\flat'_{j+1} 0^{m_{j+1}}$ contains twice as much letters 0.

Consider a word $\sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots \in (\Sigma^\sharp \cup \Sigma^\flat \cup \{0\})^\omega$, where $w_0 \in L_{B_{\text{in}}}$ and $w_i \in L_B$, $\sharp_i \in \Sigma^\sharp$ for all $i \in \mathbb{N}$. For two consecutive blocks

$$w_i = \flat_{i,0}\, 0^{n_{i,0}}\, \flat_{i,1}\, 0^{n_{i,1}}\, \flat_{i,2}\, \cdots\, \flat_{i,k_i-1}\, 0^{n_{i,k_i-1}} \text{ and}$$
$$w_j = \flat_{j,0}\, 0^{n_{j,0}}\, \flat_{j,1}\, 0^{n_{j,1}}\, \flat_{j,2}\, \cdots\, \flat_{j,k_j-1}\, 0^{n_{j,k_j-1}},$$

with $j = i+1$, for $i \in \mathbb{N}$, we say that the block w_j has a *doubling error* at position p in the range $0 \leq p < k_j - 1$ if $n_{j,p+1} \neq 2n_{j,p}$. The doubling error at position p in block w_j is *signaled*, if $\sharp_j = \binom{\sharp}{D}$, $\flat_{j,p} = \binom{\flat}{H}$, and $\flat_{j,p'} = \binom{\flat}{N}$ for all $p' < p$. Furthermore, we say that w_i and w_j constitute a *copy error*, if $|w_i| \neq k_j$. The copy error for w_i and w_j is *signaled* if $\sharp_i = \binom{\sharp}{C}$. Clearly, for two blocks $w, v \in L_B$, v constitutes a doubling error or w and v constitute a copy error if and only if $w \vdash v$ does not hold.

Consider the prefix of such a word $\sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots$ depicted in Figure 4.2. The blocks w_1 and w_2 constitute a copy error, as w_2 contains only three $\flat$-blocks, and not the required $4 = |w_1|$. This error is signaled by the letter $\binom{\sharp}{C}$ in front of w_1. Furthermore, the block w_1 contains a doubling error at position 0 which is not signaled.

Now, consider the following winning condition L over $\Sigma = \Sigma_I \times \Sigma_O$ where $\Sigma_I = \Sigma^\sharp \cup \Sigma^\flat \cup \{0\}$ and $\Sigma_O = \{N, D, C, H\}$. A word $w \in \Sigma^\omega$ is contained in L if at least one of the following conditions is satisfied,

- $\text{Pr}_0(w) \neq \sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots$ where $w_0 \in L_{B_{\text{in}}}$, $w_i \in L_B$ and $\sharp_i \in \Sigma^\sharp$ for all $i \in \mathbb{N}$, or

- $\mathrm{Pr}_0(w) = \sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots$ where $w_0 \in L_{B_{\mathrm{in}}}$, $\sharp_i \in \Sigma^\sharp$ and

  $$w_i = \flat_{i,0}\, 0^{n_{i,0}}\, \flat_{i,1}\, 0^{n_{i,1}}\, \flat_{i,2}\, \cdots\, \flat_{i,k_i-1}\, 0^{n_{i,k_i-1}} \in L_B$$

  for all $i \in \mathbb{N}$, and there exists $j \in \mathbb{N}$ such that

  - $(\mathrm{Pr}_1(w))(j) = D$ and $(\mathrm{Pr}_1(w))(j') = N$ for all $j' < j$, and
  - $\sharp_{j'} = \binom{\sharp}{N}$ for all $j' < j$, and
  - w_j constitutes a doubling error at position $0 \leq p \leq k_j - 1$, and
  - the second component in w_j corresponding to $\flat_{j,p}$ is H, and for all $p' < p$, the second component in w corresponding to $\flat_{j,p'}$ is N.

- $\mathrm{Pr}_0(w) = \sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots$ where $w_0 \in L_{B_{\mathrm{in}}}$, $\sharp_i \in \Sigma^\sharp$ and

  $$w_i = \flat_{i,0}\, 0^{n_{i,0}}\, \flat_{i,1}\, 0^{n_{i,1}}\, \flat_{i,2}\, \cdots\, \flat_{i,k_i-1}\, 0^{n_{i,k_i-1}} \in L_B$$

  for all $i \in \mathbb{N}$, and there exists $j \in \mathbb{N}$ such that

  - $(\mathrm{Pr}_1(w))(j) = C$ and $(\mathrm{Pr}_1(w))(j') = N$ for all $j' < j$, and
  - $\sharp_{j'} = \binom{\sharp}{N}$ for all $j' < j$, and
  - the pair w_j and w_{j+1} constitutes a copy error.

- $\mathrm{Pr}_0(w) = \sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots$ where $w_0 \in L_{B_{\mathrm{in}}}$, $\sharp_i \in \Sigma^\sharp$ and

  $$w_i = \flat_{i,0}\, 0^{n_{i,0}}\, \flat_{i,1}\, 0^{n_{i,1}}\, \flat_{i,2}\, \cdots\, \flat_{i,k_i-1}\, 0^{n_{i,k_i-1}} \in L_B$$

  for all $i \in \mathbb{N}$, and there exists $j \in \mathbb{N}$ such that

  - $\sharp_j = \binom{\sharp}{D}$ and $\sharp_{j'} = \binom{\sharp}{N}$ for all $j' < j$, and
  - $(\mathrm{Pr}_1(w))(j') = N$ for all $j' \leq j$, and
  - $\flat_{j,p} \neq \binom{\flat}{H}$ for all $p \in [k_j - 1]$ or w_j does not constitute a doubling error at position p satisfying $\flat_{j,p} = \binom{\flat}{H}$ and $\flat_{j,p'} \neq \binom{\flat}{H}$ for $p' < p$.

- $\mathrm{Pr}_0(w) = \sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots$ where $w_0 \in L_{B_{\mathrm{in}}}$, $\sharp_i \in \Sigma^\sharp$ and

  $$w_i = \flat_{i,0}\, 0^{n_{i,0}}\, \flat_{i,1}\, 0^{n_{i,1}}\, \flat_{i,2}\, \cdots\, \flat_{i,k_i-1}\, 0^{n_{i,k_i-1}} \in L_B$$

  for all $i \in \mathbb{N}$, and there exists $j \in \mathbb{N}$ such that

  - $\sharp_j = \binom{\sharp}{C}$ and $\sharp_{j'} = \binom{\sharp}{N}$ for all $j' < j$, and
  - $(\mathrm{Pr}_1(w))(j') = N$ for all $j' \leq j$, and
  - the pair w_j and w_{j+1} does not constitute a copy error.

- $(\mathrm{Pr}_0(w))(i) \neq \binom{\sharp}{C}$ and $(\mathrm{Pr}_0(w))(i) \neq \binom{\sharp}{D}$, for all $i \in \mathbb{N}$.

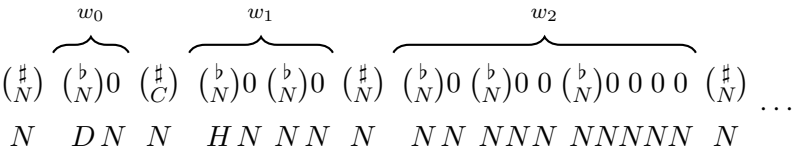


Figure 4.3: A play prefix containing violations of the successor relation $\vdash$ indicated by the players

According to the first condition, in order to win a play, Player I has to produce an input word of the form $\alpha = \sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots$ where $w_0 \in L_{B_{\text{in}}}$, $w_i \in L_B$ and $\sharp_i \in \Sigma^\sharp$ for all $i \in \mathbb{N}$, if he does not he loses. Player O produces an output word $\beta \in \Sigma_O^\omega$ where the letters from Σ_O are used to indicate violations of the successor relation $\vdash$ in the input word α by announcing the corresponding errors. If $\beta(j) = D$, then Player O claims that block w_j constitutes a doubling error at some position p which Player O also has to indicate by answering $\flat_{j,p}$ by letter H. According to the second condition of the definition of L, Player O wins a play if his first claim indicates an existing doubling error in a block w_j and furthermore identifies the appropriate position p in w_j, and for all preceding blocks $w_{j'}$ with $j' < j$ there are no errors signaled by Player I. If $\beta(j) = C$, then Player O claims that blocks w_j and w_{j+1} constitute a copy error. According to the third condition of the definition of L, Player O wins a play if his first claim indicates an existing copy error in blocks w_j and w_{j+1}, and for all preceding blocks $w_{j'}$ with $j' < j$ there are no errors signaled by Player I. The next two conditions of the definition of L require that the first error signaled by Player I indeed occurs, i.e., if Player I signals a doubling error in block w_j and Player O did not claim any error for any preceding block $w_{j'}$ with $j' \leq j$, but Player I does not indicate the position of this doubling error correctly, Player I loses. Analogously, if Player I signals a copy error in blocks w_j and w_{j+1} and Player O did not claim any error for any preceding block $w_{j'}$ with $j' \leq j$, but Player I does not produce this error, he loses. Finally, the last condition forces Player I eventually to signal some error.

Consider the play prefix depicted in Figure 4.3 obtained by extending the example of Figure 4.2. Player O claims the doubling error in w_1 by choosing $\beta(1) = D$ and identifying its position by answering $\flat_{1,0}$ with letter H. Notice, that although Player I correctly signals a copy error in front of w_1, this play is nevertheless winning for Player O, since the claim of Player O is correct as well. Moreover, according to the definition of L, if the first claim of Player O and the first signal of Player I refer to the same block w_j the claim of Player O prevails.

Now, we show that L can be recognized by a parity-DPDA $\mathcal{P}$. No-

tice, that the first and the last conditions of the definition of L are regular. Hence, words satisfying these conditions can be recognized by means of a parity-DFA. So, we explain how $\mathcal{P}$ processes words $w \in \Sigma^\omega$ with $\mathrm{Pr}_0(w) = \sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots$ where $w_0 \in L_{B_{\mathrm{in}}}$, $w_i \in L_B$ and $\sharp_i \in \Sigma^\sharp$ for all $i \in \mathbb{N}$, and $\sharp_i \in \{\binom{\sharp}{C}, \binom{\sharp}{D}\}$ for some $i \in \mathbb{N}$. The parity-DPDA $\mathcal{P}$ proceeds in four phases

1. The stack is prepared to be able to find the beginning of block w_i in w when starting at letter $w(i)$ as required in the second phase. To do so, $\mathcal{P}$ counts the number of letters processed so far minus the number of letters $\binom{\sharp}{N}$ in the first component and stores it on the stack, i.e., after a prefix $u \sqsubset w$ is precessed the stack height of the reached configuration is

   $$|u| - |\mathrm{Pr}_0(u)|_{\binom{\sharp}{N}}.$$

   This phase stops as soon as a letter $w(i)$ with $\mathrm{Pr}_0(w(i)) \in \{\binom{\sharp}{C}, \binom{\sharp}{D}\}$ or $\mathrm{Pr}_1(w(i)) \in \{C, D\}$ is read. If the first component of $w(i)$ is $\binom{\sharp}{C}$ or $\binom{\sharp}{D}$, then $\mathcal{P}$ goes to phase four. On the other hand, if the second component of $w(i)$ is C or D and $\mathrm{Pr}_0(w(i)) \notin \{\binom{\sharp}{C}, \binom{\sharp}{D}\}$, $\mathcal{P}$ proceeds to phase two.

2. This phase starts if $\mathrm{Pr}_1(w(i)) = C$ or $\mathrm{Pr}_1(w(i)) = D$, for some $i \in \mathbb{N}$, $\mathrm{Pr}_0(w(j)) \notin \{\binom{\sharp}{C}, \binom{\sharp}{D}\}$ for $j \leq i$, and $\mathrm{Pr}_1(w(j)) \notin \{C, D\}$ for $j < i$. The letter $\mathrm{Pr}_1(w(i))$ is held in the states during this phase for the later use in phase three. Now, $\mathcal{P}$ uses the information stored on the stack to find the beginning of w_i by decreasing the stack every time a letter from $\{\binom{\sharp}{N}\} \times \Sigma_O$ is processed. If a letter from $\{\binom{\sharp}{C}, \binom{\sharp}{D}\} \times \Sigma_O$ is processed before the stack is empty, i.e., $\sharp_j = \binom{\sharp}{C}$ or $\sharp_j = \binom{\sharp}{D}$ for $j < i$, then $\mathcal{P}$ jumps to phase four. Otherwise, if the empty stack is reached, then certainly the beginning of block w_i is found, and $\mathcal{P}$ continues with phase three.

3. In this phase, $\mathcal{P}$ checks whether the error indicated by $\mathrm{Pr}_1(w(i))$ occurs. Notice, that $\mathrm{Pr}_1(w(i))$ is available at the beginning of w_i, as this letter is carried through the whole computation of phase two. If $\mathrm{Pr}_1(w(i)) = C$, then $\mathcal{P}$ checks whether w_i and w_{i+1} constitute a copy error. This can be easily implemented by means of a parity-DPDA, as the length of w_i and the number of $\flat$-blocks in w_{i+1} have to be tested for equality. If the length of w_i and the number of $\flat$-blocks in w_{i+1} are equal, then the copy error is detected. If $\mathrm{Pr}_1(w(i)) = D$, then $\mathcal{P}$ checks whether w_i contains a doubling error, which has to be indicated in the second component by an H at the appropriate position. Again, this can be implemented by means of a parity-DPDA which compares the

lengths of two consecutive $\flat$-blocks, namely whether the first $\flat$-block contains half as much letters 0 as the second $\flat$-block. If there is an H in the second component of w_i and the first occurrence of H in the second component of w_i indicates an error correctly, then the doubling error is detected. The automaton $\mathcal{P}$ accepts w if and only if the error indicated by $\mathrm{Pr}_1(w(i))$ is detected.

4. In this phase, $\mathcal{P}$ checks whether the error signaled in $\sharp_j$ occurs. If $\sharp_j = \binom{\sharp}{C}$, then it checks whether w_j and w_{j+1} constitute a copy error. This is done by means of a parity-DPDA in the same manner as in phase three. Again, if the length of w_i and the number of $\flat$-blocks in w_{i+1} are equal, then the copy error is detected. If $\sharp_j = \binom{\sharp}{D}$, then $\mathcal{P}$ checks whether w_j contains a doubling error, which is indicated properly by a $\binom{\flat}{H}$ in the first component at the appropriate position. If the first component of w_j contains a $\binom{\flat}{H}$ and the first occurrence of $\binom{\flat}{H}$ in the first component of w_j indicates an error correctly, then the doubling error is detected. The automaton $\mathcal{P}$ accepts w if and only if the error indicated by $\sharp_j$ is not detected.

We continue by showing that there exists a delay function f such that Player O wins the delay game $\Gamma_f(L)$. To this end, consider two blocks $u, v \in L_B$ such that $u \vdash v$. Clearly, $v = \flat_0\, 0^{n_0}\, \flat_1\, 0^{n_1}\, \flat_2 \cdots \flat_{m-1}\, 0^{n_{m-1}}$ where $m = |u|$, $n_i = 2^i$ and $\flat_i \in \Sigma^\flat$ for $i \in [m]$. Hence, the length of v is

$$|v| = |u| + \sum_{i=0}^{|u|-1} 2^i = |u| + \frac{1-2^{|u|}}{1-2} = 2^{|u|} + |u| - 1.$$

Let the auxiliary function $g\colon \mathbb{N} \to \mathbb{N}$ be defined by $g(0) = 2$ and

$$g(i+1) = 2^{g(i)} + g(i) - 1,$$

for every $i \in \mathbb{N}$. For a sequence of blocks $w_0, w_1, w_2, \ldots$ with $w_0 \in L_{B_{\mathrm{in}}}$, $w_i \in L_B$ and $w_i \vdash w_{i+1}$ for all $i \in \mathbb{N}$, we have $g(i) = |w_i|$ for $i \in \mathbb{N}$. Now, using the function g we define the delay function $f\colon \mathbb{N} \to \mathbb{N}$ by

$$f(0) = g(0) + g(1) + 3 \text{ and}$$
$$f(i) = g(i+1) + 1 \text{ for every } i > 0.$$

Notice, that f and also d_f are non-elementary.

We claim that Player O has a winning strategy for $\Gamma_f(L)$. If Player I does not pick in the first round a word of the form

$$\sharp_0\, \flat_{0,0}\, 0\, \sharp_1\, \flat_{1,0}\, 0\, \flat_{1,1}\, 0\, 0\, \sharp_2$$

with $\sharp_0, \sharp_1, \sharp_2 \in \Sigma^\sharp$ and $\flat_{0,0}, \flat_{1,0}, \flat_{1,1} \in \Sigma^\flat$, then he has committed some error within his first two blocks, which can be claimed by Player O with the first letter. Now assume Player I has produced an input prefix

$$\sharp_0\, w_0\, \sharp_1\, w_1\, \sharp_2\, \cdots\, \sharp_i\, w_i\, \sharp_{i+1}$$

after round $i-1$ without introducing a doubling error in the blocks w_j for all $j < i$ and no copy error in the pairs w_j and w_{j+1} for all $j < i$. If he produces an x in the next round i that is of the form $v\sharp$ such that w_i and v do not constitute a copy error and if w_i does not contain a doubling error, then Player O picks N in round i. Otherwise, the error that occurs is claimed by picking C or D, respectively. This strategy is winning for Player O, as Player I is not able to signal and produce an error that cannot be claimed by Player O.

Finally, we show that for every elementary-delay function $f_\mathsf{e} \in \mathcal{O}(\exp_k)$, Player I has a winning strategy in $\Gamma_{f_\mathsf{e}}(L)$. Started with an initial block Player I can always play successive blocks without introducing errors until the length of the block w_i exceeds the lookahead

$$d_{f_\mathsf{e}}(i) = \left(\sum_{j=0}^{i} f_\mathsf{e}(j)\right) - (i+1)$$

of Player O. At such a round i, Player O has to make a claim concerning a block which is not completed yet. So, Player I signals a doubling error for this incomplete block. If Player O does not claim a doubling error, then Player I introduces a doubling error while completing the block in the next round. Subsequently, he produces arbitrary blocks and wins, since he is the first who indicates an existing error. Vice versa, if Player O claims the doubling error, then Player I does not introduce a doubling error while completing the block during the next round. Then, he again continues to produce arbitrary blocks and wins, as his claim is preceded by the claim of Player O which indicates a nonexistent error. □

Using ideas as presented in Section 4.2 one can show that Theorem 4.7 holds even for the restricted class of winning conditions recognizably by A-DV1CA. However, the game specification L as described above has to be modified even more than just providing additional letter for Player O to indicate which kind of a transition is to be performed. For a visibly one-counter the problem arises in the situation where the automaton has to change from phase two to phase four, since a claimed error has to be checked while the stack is not yet empty. To do this using one stack symbol, the stack has to be emptied before the next letter is processed, which cannot be done by a visibly automaton, as it has no ε-transitions. We avoid this situation by modifying the winning condition L such that if Player O indicates his first

error in round i and Player I signals an error in front of block w_j with $j < i$, then Player O loses immediately. Player O has still the possibility to win by additionally never claiming an error in a block w_i if Player I already claimed an error in a block w_j for some $j < i$. This modified winning condition is visibly one-counter. Moreover, as a play is only winning for Player I, if he claims an existing error before Player O does, the set of winning plays for Player O of the modified winning condition can be accepted by an A-DV1CA.

4.4 Summary of Results

We investigated deterministic contextfree delay games, a modified version of Gale-Stewart games where Player I has to pick words of some length greater than zero given by a delay function instead of single letters, thereby Player O obtains a lookahead on the moves of Player I. The winning condition is provided by a deterministic contextfree language. For the case where the lookahead is bounded, i.e., after some round i the lookahead remains constant, we showed that a delay game can be reduced to a game without delay, hence, the winner of such a delay game is decidable. Then, we showed that deciding the winner in a deterministic contextfree delay game with arbitrary lookahead is undecidable. We complemented by characterizing classes of delay functions for which it is decidable whether there is some delay function in the fixed class such that Player O wins the corresponding delay game. It turned out that the problem is decidable if and only if there is a global bound on the lookahead provided by the class of delay functions.

Moreover, we considered the lookahead necessary to win deterministic contextfree delay games. A non-elementary lower bound was established. For this, we presented a deterministic contextfree winning condition such that there exists a delay function such that Player O wins the corresponding delay game. However, Player O loses, if the lookahead is bounded by any elementary function.

Furthermore, we showed that our results hold even for restricted classes of deterministic contextfree winning conditions recognizable by visibly one-counter with weak acceptance conditions.

Our results show that, unlike the regular case, adding lookahead to deterministic contextfree games significantly changes their algorithmic properties. For regular delay games, bounded lookahead is sufficient, i.e., if Player O wins with arbitrary lookahead then he also wins with bounded lookahead. Bounded lookahead can be encoded into the winning condition, hence the classical algorithms to solve regular games without lookahead are still applicable for regular delay games. However, in case of deterministic contextfree delay games unbounded lookahead can not be reduced to the case of bounded

one. This is a reason why deterministic contextfree delay games are hard to handle algorithmically.

Chapter 5

Distributed Synthesis with Pushdown Specifications

A more realistic and relevant setting for practical application of controller synthesis than standard two-player games is that of distributed systems. In practice, systems rarely consist of just one process, but they are rather composed of several processes which communicate with each other and with the environment via certain communication channels. Thus, the task of distributed synthesis is to construct, for a given system specification and an architecture, which describes the communication structure of a distributed system, several implementations, one for each process, such that every overall system behavior satisfies the given specification. Clearly, in general the system processes have to produce their actions based on incomplete information about the global system state.

Distributed synthesis has been first considered by Pnueli and Rosner who proved, based on Peterson and Reif's results concerning multi-player games [PR79], that in general distributed synthesis problem is undecidable for specifications from the linear time temporal logic [PR90]. However, they also established decidability for linear time temporal logic specifications for the special case of acyclic architectures, called pipelines, where the processes are linearly ordered and the information flows from the environment in direction of the worst informed process.

This decidability result was extended by Kupferman and Vardi in two directions, they proved decidability for further special cases of architectures which also may contain cycles as well as for branching time temporal specifications [KV01]. Finally, a comprehensive criterion characterizing all decidable architectures for specifications given in linear time, branching time temporal logic or in μ-calculus was figured out by Finkbeiner and Schewe [FS05] who showed that an architecture is decidable if and only if it does not contain at least two incomparably informed processes, i.e., the processes can be or-

dered according to their informedness. The authors also gave a uniform tree automata-based synthesis procedure for decidable cases (see also [Sch08]).

Madhusudan and Thiagarajan introduced the concept of local specifications [MT01] where a system specification is given by a conjunction of local specifications for each of the system processes. Hence, every individual local specification defines the correct behaviors of the corresponding process. For such kind of specifications with regular local specifications, a characterization of all decidable acyclic architectures was established. It turned out, that there are architectures, called two-flanked pipelines, which contain incomparably informed processes but are still decidable for specification given by a conjunction of local regular specifications.

In this chapter we consider distributed synthesis for deterministic contextfree global specifications as well as specification given by a conjunction of regular or deterministic contextfree local specifications. The main result of this chapter is a full characterization of decidable architectures for the case of local specifications which is an extension of the result of [MT01], where acyclic architectures and regular local specifications are considered. Here we consider general architectures where also cycles are allowed, and furthermore, the local specifications may also be deterministic contextfree. In Section 5.1, we give some definitions and fix our notations which will be used throughout this chapter. The notion of architectures that specify the communication structure of distributed systems is introduced formally in Section 5.2. We prove undecidability for almost all architectures with deterministic contextfree global specifications in Section 5.3. Basically, only the corner cases are decidable for deterministic contextfree global specifications, namely those corresponding to nondistributed setting and where the environment does not send information at all. In Section 5.4, we exhibit several decidable and undecidable special cases of architectures with local specifications. These decidability and undecidability results, presented in Subsection 5.4.1 and Subsection 5.4.2, respectively, serve as prerequisite for the characterization of the decidable architectures with regular or deterministic contextfree local specifications. The effective criterion concerning the graph structure of the given architecture and the assignment of regular and deterministic contextfree local specifications to the individual processes is presented in Section 5.5.

5.1 Preliminaries

In distributed settings we deal with architectures which we formally define in the next section. Basically, an architecture can be regarded as a directed graph where the nodes correspond to the processes of the system and edges

correspond to channels via which the processes communicate. The following definitions will be needed in the subsequent sections to tackle the distributed realizability problem which is to decide, given an architecture and a system specification, whether there are strategies for the system processes which guarantee that for any input produced by the environment process the overall behavior of the system satisfies the system specification, i.e., whether there is a joint winning strategy for the system processes.

For any functions $f\colon A \to B$ and $g\colon B \to C$ the composition $f \circ g\colon A \to C$ is defined by $(f \circ g)(a) = g(f(a))$, for $a \in A$. Moreover, for a subset $A' \subseteq A$, we denote $f(A') = \{f(a) \mid a \in A'\}$. For two alphabets Σ_I, Σ_O and a strategy $\sigma\colon \Sigma_I^* \to \Sigma_O$ we define the functions $\sigma^{(*)}\colon \Sigma_I^* \to \Sigma_O^*$ and $\sigma^{(\omega)}\colon \Sigma_I^\omega \to \Sigma_O^\omega$ by

$$\sigma^{(*)}(w) = \sigma(\mathsf{pref}_0(w))\sigma(\mathsf{pref}_1(w)) \cdots \sigma(\mathsf{pref}_{|w|-1}(w)) \text{ and}$$
$$\sigma^{(\omega)}(\alpha) = \sigma(\mathsf{pref}_0(\alpha))\sigma(\mathsf{pref}_1(\alpha)) \cdots$$

for $w \in \Sigma_I^*$ and $\alpha \in \Sigma_I^\omega$, i.e., $\sigma^{(*)}(w)$ is the output word produced by the strategy σ on the finite prefix w, and $\sigma^{(\omega)}(\alpha)$ is the infinite output word produced by σ on the input ω-word α. For a language $L_{\text{in}} \subseteq \Sigma_I^\omega$, we say that a strategy σ generates the ω-language

$$L^{(\omega)}(\sigma) = \{\sigma^{(\omega)}(\alpha) \mid \alpha \in L_{\text{in}}\}$$

over L_{in}. Moreover, the $*$-language generated by σ over L_{in} is

$$L^{(*)}(\sigma) = \{\sigma^{(*)}(w) \mid w \sqsubset \alpha \text{ for some } \alpha \in L_{\text{in}}\}.$$

Clearly, $L^{(*)}(\sigma) = \{w \mid w \sqsubset \alpha \text{ for some } \alpha \in L^{(\omega)}(\sigma)\}$ for every strategy σ and any language $L_{\text{in}} \subseteq \Sigma_I^\omega$.

Let the boolean alphabet be denoted by $\mathbb{B} = \{\bot, \top\}$. We represent sets of sequences which can be sent along certain channels by the processes in a given architecture, called communication languages, by $\mathbb{B}$-labeled trees, as they are used in [MT01], which we call communication trees. For an alphabet Σ, we say that a $\mathbb{B}$-labeled full Σ-tree t is a communication tree over Σ if for every node $w \in \Sigma^*$ the following is satisfied:

- $t(\varepsilon) = \top$ and
- if $t(w) = \bot$ then $t(wa) = \bot$ for all $a \in \Sigma$ and
- if $t(w) = \top$ then $t(wa) = \top$ for some $a \in \Sigma$.

The set of all communication trees over Σ is denoted by $\mathbb{T}_{\mathsf{com}}(\Sigma)$. A communication tree $t \in \mathbb{T}_{\mathsf{com}}(\Sigma)$ represents the communication languages

$$L^{(*)}(t) = \{w \in \Sigma^* \mid t(w) = \top\} \text{ and}$$
$$L^{(\omega)}(t) = \{\alpha \in \Sigma^\omega \mid t(\mathsf{pref}_k(\alpha)) = \top \text{ for all } k \in \mathbb{N}\}.$$

Clearly, $L^{(*)}(t) = \{w \mid w \sqsubset \alpha \text{ for some } \alpha \in L^{(\omega)}(t)\}$ for every communication tree $t \in \mathbb{T}_{\mathsf{com}}(\Sigma)$.

Finally, we define a product of nondeterministic parity tree automata and nondeterministic parity pushdown tree automata. Let X be a set and Σ an alphabet. For a parity-NTA $\mathcal{A} = (Q^{\mathcal{A}}, \Sigma, q_{\text{in}}^{\mathcal{A}}, \delta^{\mathcal{A}}, \text{col}^{\mathcal{A}})$ and a parity-NPDTA $\mathcal{P} = (Q^{\mathcal{P}}, \Sigma, \Gamma, q_{\text{in}}^{\mathcal{P}}, \delta^{\mathcal{P}}, \text{col}^{\mathcal{P}})$ both over Σ-labeled X-trees, we define the Muller-NPDTA $\mathcal{A} \times \mathcal{P} = (Q, \Sigma, \Gamma, q_{\text{in}}, \delta, \mathcal{F})$ over Σ-labeled X-trees such that

- $Q = Q^{\mathcal{A}} \times Q^{\mathcal{P}}$,
- $q_{\text{in}} = (q_{\text{in}}^{\mathcal{A}}, q_{\text{in}}^{\mathcal{P}})$,
- for every $F \subseteq Q$, $F \in \mathcal{F}$ if and only if $\min\{\text{col}^{\mathcal{A}}(p) \mid (p,q) \in F\}$ is even and $\min\{\text{col}^{\mathcal{P}}(q) \mid (p,q) \in F\}$ is even,
- for all $(p,q) \in Q$, all $a \in \Sigma$ and all $A \in \Gamma$, if $\delta^{\mathcal{P}}(q,a,A) = (\circlearrowleft, q', \gamma)$, for some $q' \in Q^{\mathcal{P}}$ and $\gamma \in \Gamma_{\perp}^{\leq 2}$ then

  $$\delta((p,q),a,A) = (\circlearrowleft, (p,q'), \gamma).$$

  Otherwise, if $\delta^{\mathcal{P}}(q,a,A)$ is not an ε-transition, then

  $$\delta((p,q),a,A) = \bigvee_{[\varphi^{\mathcal{A}} \in \delta^{\mathcal{A}}(p,a)]} \bigvee_{[\varphi^{\mathcal{P}} \in \delta^{\mathcal{P}}(q,a,A)]} \bigwedge_{[x \in X]} (\downarrow_x, (p_x, q_x), \gamma_x)$$

  where $(\downarrow_x, p_x) \in \varphi^{\mathcal{A}}$ and $(\downarrow_x, q_x, \gamma_x) \in \varphi^{\mathcal{P}}$.

Clearly, a tree $t \in X_\Sigma$ is accepted by $\mathcal{A} \times \mathcal{P}$ if and only if it is accepted by $\mathcal{A}$ as well as by $\mathcal{P}$. Hence, $L(\mathcal{A} \times \mathcal{P}) = L(\mathcal{A}) \cap L(\mathcal{P})$.

5.2 Architectures

In this section we fix our notations for architectures which are used to specify the communication structure of distributed systems.

An architecture $\mathcal{A} = (P, C, r)$ consists of

- a finite set of processes $P = \{p_0, p_1, \ldots, p_n\}$ with $n > 0$, process p_0 is called environment process, also denoted by $p_{\mathsf{env}} = p_0$, processes from $P_{\mathsf{sys}} = \{p_1, \ldots, p_n\}$ are called system processes,
- a finite set of channels $C = C_{p_0} \cup C_{p_1} \cup \cdots \cup C_{p_n}$ with pairwise disjoint sets C_p, for $p \in P$, containing the channels process p writes to, we call a channel $c \in C_p$, for $p \in P$, output channel of process p,

- a function $r\colon C \to P$ assigning for each channel a process which reads it, if $r(c) = p$, for $c \in C$ and $p \in P$, then c is called input channel of process p.

We require that the function r satisfies $r(c) \in P_{\mathsf{sys}}$, for all $c \in C \setminus C_{p_{\mathsf{env}}}$, i.e., each output channel of a system process is also an input channel of some system process. Moreover, we assume that, for all system processes $p \in P_{\mathsf{sys}}$, $r^{-1}(p) \neq \emptyset$ and $C_p \neq \emptyset$, i.e., each system process has at least one input and one output channel.

An architecture $\mathcal{A} = (P, C, r)$ induces a directed graph $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ where $V_{\mathcal{A}} = P$ and $(p, p') \in E_{\mathcal{A}}$ if and only if there is a channel $c \in C_p$ such that $r(c) = p'$. For any directed graph $G = (V, E)$ and a subset $S \subseteq V$, let $G[S]$ denote the subgraph of G induced by S, i.e., $G[S] = (S, E')$ where $E' = E \cap (S \times S)$. We say that an architecture $\mathcal{A}$ is connected if the subgraph $G_{\mathcal{A}}[P_{\mathsf{sys}}]$ induced by the set of the system processes is weakly connected. We say that an architecture $\mathcal{A}$ is cyclic if $G_{\mathcal{A}}$ contains at least one nontrivial directed cycle, otherwise $\mathcal{A}$ is called acyclic.

Let $\mathcal{A} = (P, C, r)$ be an architecture. For a subset $Q \subseteq P_{\mathsf{sys}}$, let $C[Q]$ be the set of channels restricted to processes from $Q \cup \{p_{\mathsf{env}}\}$, i.e.,

$$C[Q] = C^Q_{p_{\mathsf{env}}} \cup \bigcup_{p \in Q} C^Q_p \text{ where } C^Q_p = \big\{c \in C_p \mid r(c) \in Q \cup \{p_{\mathsf{env}}\}\big\},$$

and let the function $r_Q\colon C[Q] \to Q$ be defined by $r_Q(c) = r(c)$, for every $c \in C[Q]$. If $(Q \cup \{p_{\mathsf{env}}\}, C[Q], r_Q)$ is an architecture, then it is denoted by $\mathcal{A}[Q]$ and we say that $\mathcal{A}[Q]$ is the subarchitecture of $\mathcal{A}$ induced by Q. Notice, that not each subset Q of system processes induces a subarchitecture of $\mathcal{A}$, since the requirement that each system process has at least one input and one output channel may be not fulfilled. An architecture $\mathcal{A}'$ is a subarchitecture of $\mathcal{A}$ if $\mathcal{A}' = \mathcal{A}[Q]$ for some $Q \subseteq P_{\mathsf{sys}}$.

For a process $p \in P$, we call a channel $c \in C_p$ hidden if $r(c) = p$, i.e., channel c is read by process p and not by any other process. The set of all hidden channels of process $p \in P$ is denoted by $H_p = \{c \in C_p \mid r(c) = p\}$. We call the output channels $C_{p_{\mathsf{env}}}$ of the environment process, external input channels of the system or just external input channels for short. Channels from $H_{p_{\mathsf{env}}} \subseteq C_{p_{\mathsf{env}}}$ are hidden external input channels. We call the channels from $\bigcup_{p \in P_{\mathsf{sys}}} C_p \setminus H_p$ internal communication channels. For every system process $p \in P_{\mathsf{sys}}$, channels from H_p cannot be read by any other system process, so they are used to model channels transmitting information which is destined for the outside world. Hence, we call channels from $\bigcup_{p \in P_{\mathsf{sys}}} H_p$ external output channels of the system or external output channels for short.

We say that a process $p \in P_{\mathsf{sys}}$ is reachable if there is a directed path from p_{env} to p in the induced graph $G_{\mathcal{A}}$. The set of all reachable system

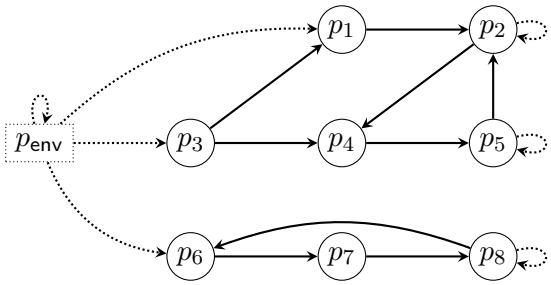


Figure 5.1: Induced graph $G_{\mathcal{A}}$ of an architecture $\mathcal{A}$

processes is denoted by $P_{\mathsf{reach}} \subseteq P_{\mathsf{sys}}$. Let $p, p' \in P$ be two processes with $p \neq p'$. We say that p sends information to p' if there is a channel $c \in C_p$ such that $r(c) = p'$. We say that p is better informed than p' if $p \in P_{\mathsf{reach}}$ and each directed path from p_{env} to p' in $G_{\mathcal{A}}$ goes through p.

To illustrate the above definitions, consider the graph $G_{\mathcal{A}}$ induced by an architecture $\mathcal{A}$ depicted in Figure 5.1. $\mathcal{A}$ is not connected, as the subgraph $G_{\mathcal{A}}[P_{\mathsf{sys}}]$ is not connected. Subarchitectures are, for instance, induced by subsets $\{p_1, p_2, p_4, p_5\}$ and $\{p_6, p_7, p_8\}$. However, $\{p_1, p_2, p_3, p_4\}$ does not induce a subarchitecture, since p_4 has no output channel in $G_{\mathcal{A}}[\{p_1, p_2, p_3, p_4\}]$. Dashed edges correspond to external input and external output channels where the loop in p_{env} corresponds to hidden external input channels and all other loops to external output channels. Solid edges arise from internal communication channels. All system processes are reachable. Moreover, process p_3 sends information to p_1 and to p_4, but not to p_2. Furthermore, process p_6 is better informed than p_7 and p_8, also p_4 is better informed than p_5. However, p_3 is incomparably informed with any other system process, i.e., p_3 is not better informed than any other system process, and there is also no other system process which is better informed than p_3.

Now, we define hierarchical architectures. We say that an architecture $\mathcal{A} = (P, C, r)$ with $P = \{p_0, \ldots, p_n\}$ is a pipeline if

$$r(C_{p_{\mathsf{env}}}) = \{p_1\},\ r(C_{p_i}) \subseteq \{p_i, p_{i+1}\} \text{ for } i \in [n] \text{ and } r(C_{p_n}) = \{p_n\}.$$

This means, the environment process p_{env} sends information only to one process $p_1 \in P_{\mathsf{sys}}$. Furthermore, there is an linear informedness order on P_{sys}, since each process p_i is allowed to send information only to the consecutive process p_{i+1}, for $i \in [n]$, as well as to have external output channels. Moreover, process p_n has only external output channels. Hence, for any two system processes p_i and p_j, if $i < j$, then p_i is better informed than p_j. We say that the architecture $\mathcal{A}$ is a two-flanked pipeline if

$$r(C_{p_{\mathsf{env}}}) = \{p_1, p_n\},\ r(C_{p_i}) \subseteq \{p_i, p_{i+1}\} \text{ for } i \in [n] \text{ and } r(C_{p_n}) = \{p_n\}.$$

In contrast to pipelines, in a two-flanked pipeline the environment process p_{env} sends information to the first process p_1 and to the last process p_n. In this case, we have for $0 < i < j < n$, process p_i is better informed than process p_j. However, for every $p \in P_{\mathsf{sys}} \setminus \{p_n\}$, the informedness of processes p and p_n is incomparable, i.e., p is not better informed than p_n, as well as p_n is not better informed than p.

We extend the two classes of architectures described above by slightly relaxing the requirement concerning the internal communication channels. We allow any system process p_i to send information not only to the consecutive process p_{i+1}, but also to any better informed system process. We say that the architecture $\mathcal{A}$ is a pipeline with backward-channels if

$$r(C_{p_{\mathsf{env}}}) = \{p_1\},\ r(C_{p_i}) \subseteq \{p_j \mid 0 < j \leq i+1\} \text{ for } i \in [n], \text{ and}$$
$$r(C_{p_n}) \subseteq P_{\mathsf{sys}}.$$

Notice, that in contrast to pipelines the last process p_n of a pipeline with backward-channels is not constrained to have an external output channel, as it may send information to some better informed process. Finally, we say that $\mathcal{A}$ is a two-flanked pipeline with backward-channels if

$$r(C_{p_{\mathsf{env}}}) = \{p_1, p_n\},\ r(C_{p_i}) \subseteq \{p_j \mid 0 < j \leq i+1\} \text{ for } i \in [n], \text{ and}$$
$$r(C_{p_n}) \subseteq P_{\mathsf{sys}}.$$

A channel $c \in C_{p_i}$, for $0 < i \leq n$, is called backward-channel if $r(c) = p_j$ with $j < i$ and it is called forward-channel if $r(c) = p_j$ with $j = i+1$. We say that an architecture $\mathcal{A}$ is hierarchical if $\mathcal{A}$ is a pipeline, a two-flanked pipeline, a pipeline with backward-channels, or a two-flanked pipeline with backward-channels. Notice, that our notion of hierarchical architectures slightly differs from the classical one, since in the literature architectures are usually referred to as hierarchical if there is no pair of system processes which is incomparable with respect to the informedness relation.

Figure 5.2 shows four directed graphs $G_{\mathcal{A}_i}$ induced by example architectures $\mathcal{A}_i$ for $i \in [4]$, where $\mathcal{A}_0$ is a pipeline, $\mathcal{A}_1$ is a two-flanked pipeline, $\mathcal{A}_2$ is a pipeline with backward-channels and $\mathcal{A}_3$ is a two-flanked pipeline with backward-channels.

A labeling $(\Sigma_c)_{c \in C}$ for an architecture $\mathcal{A} = (P, C, r)$ assigns to any channel $c \in C$ a finite alphabet Σ_c containing the symbols which can be sent along c. For every process $p \in P_{\mathsf{sys}}$, we define the input alphabet of p as

$$\Sigma_{\text{in}}^{p} = \prod_{c \in r^{-1}(p)} \Sigma_c \ ,$$

i.e., we take the Cartesian product over all alphabets of the channels which are read by process p. Accordingly, the output alphabet of a process is the

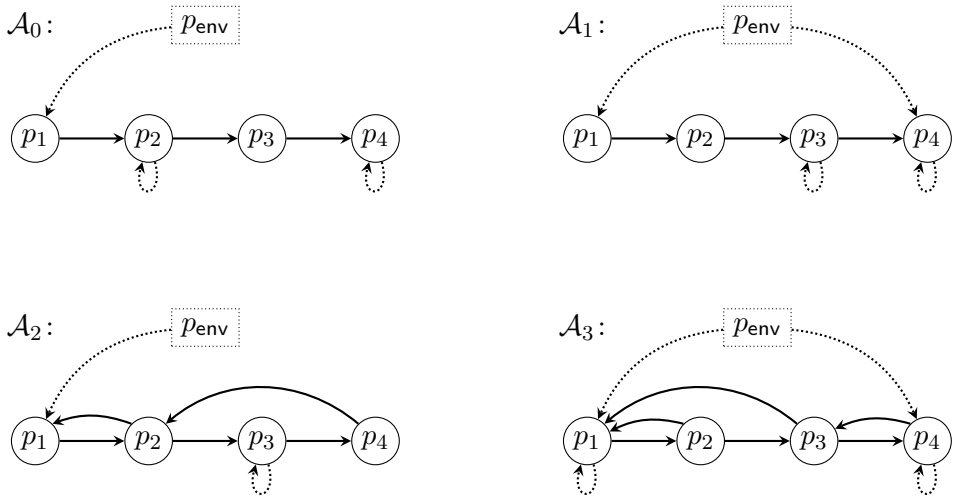


Figure 5.2: Hierarchical architectures

Cartesian product over all alphabets to which the process writes. Hence, for every $p \in P$, the output alphabet of p is defined as

$$\Sigma^p_{\text{out}} = \prod_{c \in C_p} \Sigma_c \ .$$

For every system process $p \in P_{\mathsf{sys}}$, we define the local alphabet of p as

$$\Sigma^p = \Sigma^p_{\text{in}} \times \Sigma^p_{\text{out}} \ .$$

Furthermore, the global system alphabet of $\mathcal{A}$ is defined as

$$\Sigma^{\mathcal{A}} = \prod_{c \in C} \Sigma_c = \prod_{i=0}^{n} \Sigma^{p_i}_{\text{out}} \ .$$

The overall system operates in rounds $i \in \mathbb{N}$. In each round i, every process $p \in P$ produces an output letter $\alpha_p(i) \in \Sigma^p_{\text{out}}$ and writes the appropriate components of $\alpha_p(i)$ to the corresponding channels $c \in C_p$. The ω-word

$$\alpha_{\mathcal{A}} = \alpha_{p_0} \frown \alpha_{p_1} \frown \ldots \frown \alpha_{p_n} \in (\Sigma^{\mathcal{A}})^\omega$$

obtained from the output sequences α_p of all the processes $p \in P$ is the global system behavior. Moreover, for every system process $p \in P_{\mathsf{sys}}$, the ω-word $\beta_p \frown \alpha_p$ with $\beta_p = \Pr_{\Sigma^p_{\text{in}}}(\alpha_{\mathcal{A}})$ is the local process behavior of p.

A global system specification is a language $L \subseteq (\Sigma^{\mathcal{A}})^\omega$ consisting of all correct system behaviors. A local specification for process $p \in P_{\mathsf{sys}}$ is a language $L_p \subseteq (\Sigma^p)^\omega$. Let $\ell = (L_{p_1}, \ldots, L_{p_n})$ be a list of local specifications

for the system processes. Then, the corresponding global system specification is the language $L(\ell) \subseteq (\Sigma^{\mathcal{A}})^\omega$ such that for every system process $p \in P_{\mathsf{sys}}$, we have $\mathrm{Pr}_{\Sigma^p}(L(\ell)) = L_p$.

A local strategy for process $p \in P_{\mathsf{sys}}$ is a function $\sigma_p \colon (\Sigma^p_{\mathrm{in}})^* \to \Sigma^p_{\mathrm{out}}$. A local behavior $\beta_p{}^\frown\alpha_p$ of p is consistent with σ_p, if $\alpha_p(i) = \sigma_p(\mathsf{pref}_i(\beta_p))$, for all $i \in \mathbb{N}$. For a local specification L_p for p, we say that σ_p is winning if every local behavior $\beta_p{}^\frown\alpha_p$ of p that is consistent with σ_p is contained in L_p. Since system processes form a coalition against the environment process, particular input sequences can be precluded for some system processes due to the local strategies of the other processes. Hence, we introduce the notion of strategies winning on some input language where the set of possible input words for a system process is restricted. For an input language $L_{\mathrm{in}} \subseteq (\Sigma^p_{\mathrm{in}})^\omega$, we say that σ_p is winning on L_{in} if every local behavior $\beta_p{}^\frown\alpha_p$ of p with $\beta_p \in L_{\mathrm{in}}$ which is consistent with σ_p is contained in L_p. Clearly, a strategy σ_p for process p is winning, if it is winning on $(\Sigma^p_{\mathrm{in}})^\omega$.

A joint strategy for all system processes is a tuple $\sigma_{\mathsf{sys}} = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ where each σ_{p_i} is a local strategy for process p_i, for $1 \leq i \leq n$. A global system behavior $\alpha_{\mathcal{A}}$ is consistent with σ_{sys} if the corresponding local process behavior of each system process $p_i \in P_{\mathsf{sys}}$ is consistent with σ_{p_i}. For a global system specification L, we say that σ_{sys} is winning if every global system behavior which is consistent with σ_{sys} is contained in L.

We say that a global system specification $L \subseteq (\Sigma^{\mathcal{A}})^\omega$ is realizable in an architecture $\mathcal{A} = (P, C, r)$ with labeling $(\Sigma_c)_{c \in C}$ if there is a joint winning strategy for the system processes. The realizability problem is the following.

Given: Architecture $\mathcal{A} = (P, C, r)$ with labeling $(\Sigma_c)_{c \in C}$ and global system specification $L \subseteq (\Sigma^{\mathcal{A}})^\omega$.

Question: Is L realizable in $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$?

For a class $\mathcal{L}$ of global system specifications and an architecture $\mathcal{A} = (P, C, r)$ we say that the realizability problem is decidable for global specifications from $\mathcal{L}$ for the architecture $\mathcal{A}$ if for every labeling $(\Sigma_c)_{c \in C}$ of $\mathcal{A}$ the realizability problem is decidable for $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$ when the specifications are restricted to $\mathcal{L}$. We will also refer to an architecture $\mathcal{A}$ as decidable for $\mathcal{L}$ if the realizability problem is decidable for specifications from $\mathcal{L}$ for $\mathcal{A}$.

Theorem 5.1 ([FS05])**.** *The realizability problem for global specifications from* REG_ω *is decidable for an architecture* $\mathcal{A}$ *if and only if there are no two reachable and incomparably informed*[2] *system processes in* $\mathcal{A}$*.*

[2] In [FS05] this situation where two reachable system processes are incomparably informed is called information fork.

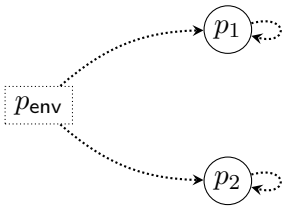


Figure 5.3: Undecidable architecture for global regular specifications

Basically, this theorem is established by showing that every architecture without a pair of reachable and incomparably informed system processes can be transformed into a pipeline such that a regular global specification is realizable in the pipeline if and only if it is realizable in the original architecture (for decidability of pipelines see also [PR90, KV01]). On the other hand, if an architecture contains a pair of reachable and incomparably informed system processes, then ideas from [PR90] are applied where undecidability is shown for the architecture depicted in Figure 5.3 using a reduction from the halting problem for Turing machines.

However, in case where the global specification is given by a list of local regular specifications, even more architectures turn out to be decidable. A characterization for the class of acyclic architectures is established in [MT01].

Theorem 5.2 ([MT01])**.** *The realizability problem for specifications given by a list of local specifications, each from* REG_ω*, is decidable for an acyclic architecture* $\mathcal{A}$ *if and only if each connected subarchitecture of* $\mathcal{A}$ *is a pipeline or a two-flanked pipeline.*

5.3 Global Specifications

In this section we consider the realizability problem for global deterministic contextfree specifications. We show that unlike the regular case almost all architectures, even pipelines, are undecidable. We proceed by a reduction from the Post's Correspondence Problem (PCP). For an alphabet Θ, a sequence of pairs $\mathcal{I} = ((u_0, v_0), \ldots, (u_{m-1}, v_{m-1}))$ where $u_i, v_i \in \Theta^*$, for $i \in [m]$, is called instance of the PCP. A word $s = i_1 \cdots i_k \in [m]^+$ is called solution of $\mathcal{I}$ if $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$. The Post's Correspondence Problem is to decide, given an instance $\mathcal{I}$, whether there exists a solution s of $\mathcal{I}$. It is well-known that PCP is undecidable [Pos46].

Theorem 5.3. *The realizability problem for global specifications from* DCFL_ω *is undecidable for an architecture* $\mathcal{A} = (P, C, r)$ *if and only if*

- $C_{p_{\mathsf{env}}} \neq \emptyset$*, and*

- $|P_{\mathsf{sys}}| \geq 2$ *or* $H_{p_{\mathsf{env}}} \neq \emptyset$.

Proof. Let $(\Sigma_c)_{c \in C}$ be a labeling of $\mathcal{A}$ and let parity-DPDA $\mathcal{P}$ define a global deterministic contextfree specification $L(\mathcal{P}) \subseteq (\Sigma^{\mathcal{A}})^\omega$. If $C_{p_0} = \emptyset$ then $L(\mathcal{P})$ is realizable in $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$ if $L(\mathcal{P}) \neq \emptyset$, since the system processes are not obliged to react to the inputs of the environment process, but just have jointly to produce a word from $L(\mathcal{P})$. Hence, in this case the realizability problem is reduced to the nonemptiness problem for parity-DPDA which is decidable. Moreover, if $|P_{\mathsf{sys}}| = |\{p_1\}| = 1$ and $H_{p_{\mathsf{env}}} = \emptyset$, then $\Sigma^{\mathcal{A}} = \Sigma_{\text{out}}^{p_{\mathsf{env}}} \times \Sigma_{\text{out}}^{p_1}$ and we have $L(\mathcal{P})$ is realizable in $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$ if Player I wins $\Gamma(L')$ where L' is defined as

$$L' = (\Sigma^{\mathcal{A}})^\omega \setminus \big\{\alpha^\frown\beta \in (\Sigma_{\text{out}}^{p_1} \times \Sigma_{\text{out}}^{p_{\mathsf{env}}})^\omega \mid \beta^\frown\alpha \in L(\mathcal{P})\big\}.$$

Notice, that the transfer from $L(\mathcal{P})$ to L' is necessary to resolve the discrepancy between both settings where in an architecture in every round process environment and the system process produce letters concurrently, whereas in Gale-Stewart games Player I and Player O proceed in alternation.

Now, assume that $C_{p_{\mathsf{env}}} \neq \emptyset$ and $|P_{\mathsf{sys}}| \geq 2$. Let $p, q \in P_{\mathsf{sys}}$ be two system processes and let $c_I \in C_{p_{\mathsf{env}}}$ and $c_O \in C_q$. W.l.o.g., assume that $r(c_I) = p$. Given an instance $\mathcal{I} = ((u_0, v_0), \ldots, (u_{m-1}, v_{m-1}))$ of the PCP over some alphabet Θ, define the labeling $(\Sigma_c)_{c \in C}$ of $\mathcal{A}$ by

$$\begin{aligned}
&\Sigma_{c_I} = \{U, V\},\\
&\Sigma_{c_O} = [m] \cup \Theta \cup \{\sharp\} \text{ and}\\
&\Sigma_c = \{\sharp\} \text{ for all } c \in C \setminus \{c_I, c_O\},
\end{aligned}$$

and consider the following global specification L over $\Sigma^{\mathcal{A}}$. A word $\alpha \in (\Sigma^{\mathcal{A}})^\omega$ is contained in L if and only if $\Pr_{\Sigma_{c_O}}(\alpha)$ is of the form

$$\Pr_{\Sigma_{c_O}}(\alpha) = \sharp\, i_k \cdots i_1 \,\sharp\, w \,\sharp\, \Sigma_{c_O}^\omega$$

with $k \in \mathbb{N}, i_j \in [m]$ for $j \in [k+1]$ and $w \in \Theta^*$ such that

$$w = \begin{cases} u_{i_1} \cdots u_{i_k} & \text{if } (\Pr_{\Sigma_{c_I}}(\alpha))(0) = U,\\ v_{i_1} \cdots v_{i_k} & \text{if } (\Pr_{\Sigma_{c_I}}(\alpha))(0) = V. \end{cases}$$

Clearly, L is deterministic contextfree. Consider the following parity-DPDA $\mathcal{P}$ over $\Sigma^{\mathcal{A}}$. For an input word $\alpha \in (\Sigma^{\mathcal{A}})^\omega$, the automaton $\mathcal{P}$ checks whether $\Pr_{\Sigma_{c_O}}(\alpha)$ has a prefix of the form $\sharp\, i_k \cdots i_1 \,\sharp\, w \,\sharp$, for $i_j \in [m]$ and $w \in \Theta^*$. Furthermore, if the Σ_{c_I}-component of the first letter of α is $(\Pr_{\Sigma_{c_I}}(\alpha))(0) = U$, then $\mathcal{P}$ pushes $\mathsf{rev}(u_{i_j})$ (letter by letter using ε-transitions) onto the stack when reading i_j in the Σ_{c_O}-component. Otherwise, if $(\Pr_{\Sigma_{c_I}}(\alpha))(0) = V$, then $\mathcal{P}$ pushes $\mathsf{rev}(v_{i_j})$ onto the stack when

Figure 5.4: Decidable architectures for global specifications from DCFL_ω

reading i_j in the Σ_{c_O}-component. Clearly, after processing $\sharp\, i_k \cdots i_1\, \sharp$ in the Σ_{c_O}-component, depending on $(\mathrm{Pr}_{\Sigma_{c_I}}(\alpha))(0)$ the stack content is either $u_{i_1} \cdots u_{i_k} \bot$ or $v_{i_1} \cdots v_{i_k} \bot$. Now, $\mathcal{P}$ has to check whether w coincides with the stack content.

We show that $\mathcal{I}$ has a solution if and only if there is a joint winning strategy for the system processes. First, notice that any local strategy $\sigma_q \colon (\Sigma^q_{\mathrm{in}})^* \to \Sigma^q_{\mathrm{out}}$ for process q uniquely determines a joint strategy for all the system processes, since for any other system processes $p \in P_{\mathsf{sys}}$ with $p \neq q$, the output alphabet Σ^p_{out} is a singleton set. Assume, there is a solution $s = i_k \cdots i_1$ of $\mathcal{I}$. Then, the local strategy σ_q of process q is just to write the prefix $\sharp\, i_k \cdots i_1\, \sharp\, u_{i_1} \cdots u_{i_k}\, \sharp$ to the channel c_O. Since $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$, any global behavior consistent with this strategy is contained in L, regardless of which letter is written to c_I in the first round. Hence, σ_q determines a joint winning strategy. On the other hand, since process q is not informed about the letters written to c_I, it has to produce a prefix $\sharp\, i_k \cdots i_1\, \sharp\, w\, \sharp$ suitable for both letters from Σ_{c_I}, U and V, which can be written to c_I by p_{env} in the first round. Hence, if the system processes have a joint winning strategy then process q produces a prefix $\sharp\, i_k \cdots i_1\, \sharp\, w\, \sharp$ such that $w = u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$ which implies that $s = i_1 \cdots i_k$ is a solution of $\mathcal{I}$.

Finally, assume that $|P_{\mathsf{sys}}| = 1$ but $H_{p_{\mathsf{env}}} \neq \emptyset$. To show undecidability for this case, we use the above specification language. For this, let now $P_{\mathsf{sys}} = \{q\}$, $c_I \in H_{p_{\mathsf{env}}}$ and $c_O \in C_q$. Given an instance $\mathcal{I}$ of PCP over some alphabet, the labeling $(\Sigma_c)_{c \in C}$ of $\mathcal{A}$ and the specification L are defined exactly as above. Since the information written to c_I cannot be read by process q, by the same reasoning as above, an architecture with only one system process and a hidden external input channel is also shown to be undecidable for deterministic contextfree specifications. □

Thus, it turns out that the existence of an external input channel which cannot be read by some system process makes an architecture undecidable for global deterministic contextfree specifications. Hence, the only decidable cases (depicted in Figure 5.4) are architectures without external input channels and single-process architectures without hidden external input channels.

Notice that Theorem 5.3 holds even for specification languages recognized

by A-DV1CA. This can be shown by a reduction from the halting problem for 2-register machines by combining the ideas from Section 4.2 and those of the above proof. Given a 2-register machine $\mathcal{R}$, consider the following specification which we describe informally. Process q which writes to channel c_O is required to produce an infinite sequence of encodings of configurations $\sharp c_0 \sharp c_1 \sharp c_2 \cdots$ where $c_i \in \mathsf{Conf}$, for $i \in \mathbb{N}$, and $c_0 = \mathsf{Conf}_{\text{in}}$. Process p_{env} can claim a violation of the successor relation $\vdash_{\mathcal{R}}$ once by writing a letter E_0, E_1 or L to channel c_I which cannot be read by process q. Furthermore, to check the subsequent two configurations produced by process q according to the claimed violation by means of a visibly one-counter, process p_{env} has to write an appropriate sequence of letters $\mathsf{Push}, \mathsf{Pop}, N$ into c_I to enable the automaton to perform the corresponding test. A global system behavior $\alpha_{\mathcal{A}} \in (\Sigma^{\mathcal{A}})^{\omega}$ is contained in the specification language if process q writes a sequence $\sharp c_0 \sharp c_1 \sharp c_2 \cdots$ with $c_i \in \mathsf{Conf}$, for $i \in \mathbb{N}$, and $c_0 = \mathsf{Conf}_{\text{in}}$ into c_O and process p_{env} does not manage to detect a violation of the successor relation $\vdash_{\mathcal{R}}$. Clearly, since process q cannot observe the letters written to c_I by p_{env} there is a joint winning strategy for the system processes if and only if $\mathcal{R}$ does not halt.

5.4 Local Specifications

In the previous section we showed that deterministic contextfree global system specifications yield undecidability for almost all architectures. Only very special cases which basically correspond to the nondistributed setting or where the antagonistic environment process is excluded are decidable. In this section we concentrate on the realizability problem for global specifications given by a list of local specifications, one for each system process. We extend Theorem 5.2 in two directions. First, we consider architectures where cycles are allowed, and second, the local specifications may also be deterministic contextfree. The complete characterization of decidable architectures will be given in Section 5.5. For this, decidability and undecidability for several cases are shown in this section. In particular, we point out decidable cases of pipelines with backward-channels and of two-flanked pipelines with backward-channels. Furthermore, basic criteria yielding undecidability are presented.

5.4.1 Decidable Cases

In this subsection we consider pipelines with backward-channels and two-flanked pipelines with backward-channels. For both kinds of architectures we will point out decidable cases. The parameters which we adjust are the number of processes, and for each system process, the class of the local

specification as well as the presence of backward-channels. In particular, we show that any pipeline with backward-channels, where every process is allowed to have backward-channels and all local specifications are regular except the local specification of the worst informed process which may also be deterministic contextfree, is decidable. For two-flanked pipelines with backward-channels, we show that any two-flanked pipeline with backward-channels consisting of only two system processes with regular local specifications is decidable. Moreover, we show that two-flanked pipelines with backward-channels consisting of more than two system processes where the last process has no backward-channels and all local specifications are restricted to regular ones, are also decidable.

Pipelines with Backward-Channels

Consider a pipeline with backward-channels $\mathcal{A} = (P, C, r)$ with a labeling $(\Sigma_c)_{c \in C}$. We define a partition on the set of channels $C = C_{\mathsf{f}} \cup C_{\mathsf{b}} \cup C_{\mathsf{env}}$ by

$$C_{\mathsf{f}} = \bigcup_{i=1}^{n-1} \{c \in C_{p_i} \mid r(c) = p_{i+1}\}, \text{ and}$$
$$C_{\mathsf{b}} = \bigcup_{i=1}^{n} \{c \in C_{p_i} \mid r(c) = p_j \text{ for } j \leq i\}.$$

Channels from C_{f} and external input channels are called forward-channels of $\mathcal{A}$, and the set C_{b} consists of backward-channels and external output-channels of $\mathcal{A}$. Moreover, for every system process $p_i \in P_{\mathsf{sys}}$, for $1 \leq i \leq n$, we define the accumulated output alphabet of process p_i as

$$\Sigma_{\text{out}}^{\geq i} = \prod_{j=i}^{n} \Sigma_{\text{out}}^{p_j}$$

which comprises the output alphabet of process p_i and the output alphabets of all worse informed processes. Furthermore, we denote the alphabet which labels the backward-channels and external output channels of process p_i by

$$\Sigma_{\text{out}}^{\mathsf{b},p_i} = \prod_{c \in C_{\mathsf{b}} \cap C_{p_i}} \Sigma_c \ .$$

Moreover, for every process $p_i \in P$, for $0 \leq i < n$, let the alphabet labeling the forward-channels of process p_i be denoted by Σ_i, i.e.,

$$\Sigma_i = \prod_{c \in C_{\mathsf{f}} \cap C_{p_i}} \Sigma_c \ , \text{ for } 0 < i < n, \text{ and}$$
$$\Sigma_0 = \prod_{\substack{c \in C_{\mathsf{env}}, \\ r(c) = p_1}} \Sigma_c \ .$$

First, we show that for every system process $p_i \in P_{\mathsf{sys}}$ a local strategy of p_i needs not to depend on the inputs received via channels from C_{b} but only on inputs received from process p_{i-1} via forward-channels, i.e., if there is a joint winning strategy for the system processes, then there is also a joint winning strategy such that every local strategy can ignore the inputs received via channels from C_{b}.

Lemma 5.4. *Let L be a global system specification. There are local strategies $\sigma_{p_i} \colon (\Sigma_{\text{in}}^{p_i})^* \to \Sigma_{\text{out}}^{p_i}$ for the system processes, for $1 \leq i \leq n$, such that the joint strategy $\sigma_{\mathsf{sys}} = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ is winning if and only if there are strategies $\tau_{p_i} \colon \Sigma_{i-1}^* \to \Sigma_{\text{out}}^{p_i}$, for $1 \leq i \leq n$, such that $\tau_{\mathsf{sys}} = (\tau_{p_1}, \ldots, \tau_{p_n})$ is winning.*

Proof. Assume, that there are strategies $\tau_{p_i} \colon \Sigma_{i-1}^* \to \Sigma_{\text{out}}^{p_i}$, $1 \leq i \leq n$, such that $\tau_{\mathsf{sys}} = (\tau_{p_1}, \ldots, \tau_{p_n})$ is winning. Then, define local strategies $\sigma_{p_i} \colon (\Sigma_{\text{in}}^{p_i})^* \to \Sigma_{\text{out}}^{p_i}$, for $1 \leq i \leq n$, by

$$\sigma_{p_i}(u) = \tau_{p_i}(\text{Pr}_{\Sigma_{i-1}}(u))$$

for $u \in (\Sigma_{\text{in}}^{p_i})^*$, i.e., every local strategy σ_{p_i} ignores all the inputs received via channels from C_{b} and reacts exactly like τ_{p_i} does on the inputs received via forward-channels from process p_{i-1}. Obviously, any global system behavior which is consistent with σ_{sys} is also consistent with τ_{sys} and is therefore contained in the global system specification L, since τ_{sys} is winning. Hence, σ_{sys} is also winning for the system processes.

Conversely, assume that there are local strategies $\sigma_{p_i} \colon (\Sigma_{\text{in}}^{p_i})^* \to \Sigma_{\text{out}}^{p_i}$, for $1 \leq i \leq n$, such that the joint strategy $\sigma_{\mathsf{sys}} = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ is winning for the system processes. First, notice that for every system process p_i and every word $u \in \Sigma_{i-1}^*$, there is exactly one word $v \in (\Sigma_{\text{out}}^{\geq i})^*$ over the accumulated output alphabet of p_i with $|v| = |u|$ such that $u^\frown v$ is consistent with $(\sigma_{p_i}, \ldots, \sigma_{p_n})$. This is shown by induction on $|u|$. For the induction start, we have $u = \varepsilon$ and the statement holds for $v = \varepsilon$. So, for the induction step assume $u = u'a$ for some $u' \in \Sigma_{i-1}^*$ and $a \in \Sigma_{i-1}$. Applying the induction hypothesis, there is exactly one word $v' \in (\Sigma_{\text{out}}^{\geq i})^*$ with $|v'| = |u'|$ such that $u'^\frown v'$ is consistent with $(\sigma_{p_i}, \ldots, \sigma_{p_n})$. For every process p_j with $j \geq i$, let v_j denote the output sequence produced by the local strategy σ_{p_j} on the appropriate inputs from $u'^\frown v'$, i.e.,

$$v_j = \sigma_{p_j}^{(*)}(\text{Pr}_{\Sigma_{\text{in}}^{p_j}}(u'^\frown v'))$$

for $j \geq i$. Let $v = v_i{}^\frown \ldots {}^\frown v_n$. Obviously, $v \in (\Sigma_{\text{out}}^{\geq i})^*$ with $|v| = |u|$ and $u^\frown v$ is consistent with $(\sigma_{p_i}, \ldots, \sigma_{p_n})$. To show that v is the only word satisfying the above requirements, consider a word $w \in (\Sigma_{\text{out}}^{\geq i})^*$ with $|w| = |u|$ and $u^\frown w$ is consistent with $(\sigma_{p_i}, \ldots, \sigma_{p_n})$. Let $\overline{w}, \overline{v} \in (\Sigma_{\text{out}}^{\geq i})^*$ such that

$w = \overline{w}x$ and $v = \overline{v}y$ for some $x, y \in \Sigma_{\text{out}}^{\geq i}$. Notice, that since $u'^\frown\overline{v}$ and $u'^\frown\overline{w}$ are both consistent with $(\sigma_{p_i}, \ldots, \sigma_{p_n})$, by the induction hypothesis we have $\overline{w} = \overline{v} = v'$. Moreover, since $u^\frown w$ is consistent with $(\sigma_{p_i}, \ldots, \sigma_{p_n})$, we have

$$\Pr_{\Sigma_{\text{out}}^{p_j}}(u^\frown w) = \sigma_{p_j}^{(*)}(\Pr_{\Sigma_{\text{in}}^{p_j}}(u'^\frown\overline{w}) = \sigma_{p_j}^{(*)}(\Pr_{\Sigma_{\text{in}}^{p_j}}(u'^\frown\overline{v}) = \Pr_{\Sigma_{\text{out}}^{p_j}}(u^\frown v)$$

for every p_j with $j \geq i$. Thus, we have $w = v$.

Now, let the strategies $\tau_{p_i} \colon \Sigma_{i-1}^* \to \Sigma_{\text{out}}^{p_i}$, for $1 \leq i \leq n$, be defined by

$$\tau_{p_i}(u) = \sigma_{p_i}(u^\frown \Pr_{\Sigma_{\text{in}}^{p_i}}(v))$$

for $u \in \Sigma_{i-1}^*$, where $v \in (\Sigma_{\text{out}}^{\geq i})^*$ is the unique word with $|v| = |u|$ such that $u^\frown v$ is consistent with $(\sigma_{p_i}, \ldots, \sigma_{p_n})$. To prove that every global system behavior $\alpha_{\mathcal{A}} \in (\Sigma^{\mathcal{A}})^\omega$ which is consistent with τ_{sys} is also consistent with σ_{sys} we show by induction that every finite prefix $\mathsf{pref}_k(\alpha_{\mathcal{A}})$ of $\alpha_{\mathcal{A}}$ is consistent with σ_{sys}. Clearly, $\mathsf{pref}_0(\alpha_{\mathcal{A}})$ is consistent with σ_{sys}. So, for the induction step, let $k > 0$. Then, for every system process p_i, for $1 \leq i \leq n$, we have

$$\Pr_{\Sigma_{i-1}}(\mathsf{pref}_{k-1}(\alpha_{\mathcal{A}}))^\frown \Pr_{\Sigma_{\text{out}}^{\geq i}}(\mathsf{pref}_{k-1}(\alpha_{\mathcal{A}}))$$

is consistent with $(\sigma_{p_i}, \ldots, \sigma_{p_n})$, since $\mathsf{pref}_{k-1}(\alpha_{\mathcal{A}})$ is consistent with σ_{sys}. Moreover, we have

$$\begin{aligned}\Pr_{\Sigma_{\text{out}}^{p_i}}(\mathsf{pref}_k(\alpha_{\mathcal{A}})) &= \tau_{p_i}(\Pr_{\Sigma_{i-1}}(\mathsf{pref}_{k-1}(\alpha_{\mathcal{A}}))) \\ &= \sigma_{p_i}(\Pr_{\Sigma_{\text{in}}^{p_i}}(\mathsf{pref}_{k-1}(\alpha_{\mathcal{A}}))) \ .\end{aligned}$$

The first equality holds, since $\alpha_{\mathcal{A}}$ is consistent with τ_{sys} and the second one is due to the definition of τ_{p_i}. Thus, $\mathsf{pref}_k(\alpha_{\mathcal{A}}))$ is consistent with every local strategy σ_{p_i}, for $1 \leq i \leq n$, hence, it is consistent with σ_{sys}. This means, that a global system behavior $\alpha_{\mathcal{A}}$ which is consistent with τ_{sys} is also consistent with σ_{sys} which implies that τ_{sys} is winning for the system processes. □

From now on, we consider global system specifications $L(\ell)$ which are given by a list $\ell = (L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes $p_1, \ldots, p_n$ such that $L_{p_i} \in \mathsf{REG}_\omega$, for all $1 \leq i < n$, and $L_{p_n} \in \mathsf{DCFL}_\omega$. To prove that it can be decided whether $L(\ell)$ is realizable in $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$ we will use communication trees as defined in Section 5.1. Furthermore, the following definitions will be needed.

Notice, that according to Lemma 5.4 any system process can ignore the inputs it receives via channels from C_{b} to determine the output to be produced next. Furthermore, every better informed system process has enough information to make a decision for any worse informed system process on its outputs. Hence, for $1 \leq i \leq n$, we define an extended local strategy for process p_i by $\sigma_{\geq i} = (\sigma_i, \ldots, \sigma_n)$ where

$$\sigma_j \colon \Sigma_{i-1}^* \to \Sigma_{\text{out}}^{p_j}, \text{ for } i \leq j \leq n,$$

i.e., $\sigma_{\geq i}$ determines the next output symbol for each process p_j, for $i \leq j \leq n$ based on the inputs sent from p_{i-1} to p_i. For a language $L_{\text{in}} \subseteq \Sigma_{i-1}^{\omega}$, we say that an extended local strategy $\sigma_{\geq i} = (\sigma_i, \ldots, \sigma_n)$ for process p_i is locally winning on L_{in} if for each global system behavior $\alpha_{\mathcal{A}} \in (\Sigma^{\mathcal{A}})^{\omega}$ of $\mathcal{A}$ with $\text{Pr}_{\Sigma_{i-1}}(\alpha_{\mathcal{A}}) \in L_{\text{in}}$ which is consistent with $\sigma_{\geq i}$, we have $\text{Pr}_{\Sigma^{p_i}}(\alpha_{\mathcal{A}}) \in L_{p_i}$. Moreover, if $\sigma_{\geq i}$ is an extended local strategy for process p_i and $\sigma_1, \ldots, \sigma_{i-1}$ are local strategies for processes $p_1, \ldots, p_{i-1}$, then we call $(\sigma_1, \ldots, \sigma_{i-1}, \sigma_{\geq i})$ winning for processes $p_1, \ldots, p_i$ if any global system behavior $\alpha_{\mathcal{A}}$ which is consistent with $(\sigma_1, \ldots, \sigma_{i-1}, \sigma_{\geq i})$ fulfills $\text{Pr}_{\Sigma^{p_j}}(\alpha_{\mathcal{A}}) \in L_{p_j}$, for $1 \leq j \leq i$.

Recall, that communication trees are used to represent communication languages, i.e., sequences which can be sent along certain channels in $\mathcal{A}$. Now, we define a binary operation which combines two communication trees, one for the input sequences a system process can read and one for the output sequences the process can write, to a set of communication trees which now represent strategies for the process. Let Σ and Σ' be alphabets. We define the strategy product

$$\hookrightarrow\colon \mathbb{T}_{\mathsf{com}}(\Sigma) \times \mathbb{T}_{\mathsf{com}}(\Sigma') \to \mathcal{P}(\mathbb{T}_{\mathsf{com}}(\Sigma \times \Sigma'))$$

such that for $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma)$ and $t_{\text{out}} \in \mathbb{T}_{\mathsf{com}}(\Sigma')$, we have $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ if all of the following conditions are satisfied.

1. For every $u \in \Sigma^*$ and every $v \in (\Sigma')^*$ with $|u| = |v|$, if $t_{\text{in}}(u) = \bot$ or $t_{\text{out}}(v) = \bot$ then $t(u^\frown v) = \bot$.

2. For every $u \in \Sigma^*$, if $t_{\text{in}}(u) = \top$ then there is exactly one $v \in (\Sigma')^*$ with $|u| = |v|$ such that $t(u^\frown v) = \top$.

3. For every $u^\frown v \in (\Sigma \times \Sigma')^*$, if $t(u^\frown v) = \top$ then there is some $b \in \Sigma'$ such that for all $a \in \Sigma$ with $t_{\text{in}}(ua) = \top$ we have $t(ua^\frown vb) = \top$ and $t(ua^\frown vc) = \bot$ for $c \in \Sigma' \setminus \{b\}$.

Clearly, this definition ensures that for every $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$, we have for the communication language represented by t

$$\text{Pr}_{\Sigma}\big(L^{(\omega)}(t)\big) = L^{(\omega)}(t_{\text{in}}) \text{ and } \text{Pr}_{\Sigma'}\big(L^{(\omega)}(t)\big) \subseteq L^{(\omega)}(t_{\text{out}}) \ .$$

We define the strategy $\sigma(t)\colon \Sigma^* \to \Sigma'$ represented by $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ as follows. For $u \in \Sigma^*$, if $t_{\text{in}}(u) = \top$ then let $v \in (\Sigma')^*$ be the unique word with $t(u^\frown v) = \top$ which exists according to the second condition of the definition of the strategy product. Furthermore, let $b \in \Sigma'$ be the unique symbol with $t(ua^\frown vb) = \top$ for all $a \in \Sigma$ with $t_{\text{in}}(ua) = \top$ according to the third condition of the definition. Then, we define $\sigma(t)(u) = b$. Otherwise, if $t_{\text{in}}(u) = \bot$ then $\sigma(t)$ can provide an arbitrary symbol, so define $\sigma(t)(u) = a$ for some $a \in \Sigma'$.

Now, all necessary preparations are completed to prove the decidability of the realizability problem for $\mathcal{A}$ with $(\Sigma_c)_{c\in C}$ and $L(\ell)$. To do so, we will use extended strategies and communication trees. We will proceed as follows. In the following lemma, we deal with the system processes up to the worst informed process which is then handled separately in the subsequent lemma as its corresponding local specification may also be deterministic contextfree. Finally, the result is established by putting these lemmas together.

Lemma 5.5. *For any $1 \leq i < n$, there is a parity-NTA $\mathcal{N}_i$ over $\mathbb{B}$-labeled $\Sigma_{\text{out}}^{\geq i}$-trees which accepts a tree $t_{\text{out}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{\text{out}}^{\geq i})$ if and only if there exist*

- *a tree $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{i-1})$,*
- *a tree $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ and*
- *strategies $\sigma_{p_j} \colon \Sigma_{j-1}^* \to \Sigma_{\text{out}}^{p_j}$, for $1 \leq j < i$, for processes $p_1, \ldots, p_{i-1}$*

such that the following conditions are satisfied

- *$\sigma(t)$ is locally winning on $L^{(\omega)}(t_{\text{in}})$,*
- *$\sigma_{p_1}^{(*)} \circ \text{Pr}_{\Sigma_1} \circ \ldots \circ \sigma_{p_{i-2}}^{(*)} \circ \text{Pr}_{\Sigma_{i-2}} \circ \sigma_{p_{i-1}} \circ \text{Pr}_{\Sigma_{i-1}}$ generates a language $L \subseteq L^{\omega}(t_{\text{in}})$ over Σ_0^{ω} and*
- *$(\sigma_{p_1}, \ldots, \sigma_{p_{i-1}}, \sigma(t))$ is winning for processes $p_1, \ldots, p_i$.*

Proof. For $1 \leq i < n$, let $\mathcal{S}_i = (Q^{\mathcal{S}_i}, \Sigma_{i-1} \times \Sigma_{\text{out}}^{\geq i}, \delta^{\mathcal{S}_i}, q_{\text{in}}^{\mathcal{S}_i}, \text{col})$ be a parity-DFA which accepts a word $\alpha \in (\Sigma_{i-1} \times \Sigma_{\text{out}}^{\geq i})^{\omega}$ if and only if the local specification L_{p_i} is satisfied in the Σ_{p_i}-component of α, i.e.,

$$L(\mathcal{S}_i) = \left\{\alpha \in (\Sigma_{i-1} \times \Sigma_{\text{out}}^{\geq i})^{\omega} \mid \text{Pr}_{\Sigma_{p_i}}(\alpha) \in L_{p_i}\right\}.$$

We show the lemma inductively on i. For the base case, we construct a parity-NTA $\mathcal{N}_1$ over $\mathbb{B}$-labeled $\Sigma_{\text{out}}^{\geq 1}$-trees which accepts a tree $t_{\text{out}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{\text{out}}^{\geq 1})$ if and only if there is a tree $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ where $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_0)$ with $t_{\text{in}}(u) = \top$, for all $u \in \Sigma_0^*$, such that $\mathsf{\sigma}(t)$ is locally winning on $L^{(\omega)}(t_{\text{in}}) = \Sigma_0^{\omega}$. For this we define a parity-ATA $\mathcal{A}_1$ which is then translated into $\mathcal{N}_1$. The idea for the construction is the following. While processing t_{out}, $\mathcal{A}_1$ guesses a tree $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ and verifies that $\mathsf{\sigma}(t)$ is winning for process p_1 by simulating $\mathcal{S}_1$ on all infinite paths of t labeled by $\top$. Formally, $\mathcal{A}_1 = (Q^{\mathcal{A}_1}, \mathbb{B}, \delta^{\mathcal{A}_1}, q_{\text{in}}^{\mathcal{A}_1}, \text{col}^{\mathcal{A}_1})$ is defined as follows.

- $Q^{\mathcal{A}_1} = (Q^{\mathcal{S}_1} \times \mathbb{B}) \cup \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$,
- $q_{\text{in}}^{\mathcal{A}_1} = (q_{\text{in}}^{\mathcal{S}_1}, \top)$,

- $\mathrm{col}^{\mathcal{A}_1}(q) = \begin{cases} \mathrm{col}^{\mathcal{S}_1}(\mathrm{Pr}_0(q)) & \text{if } q \in Q^{\mathcal{S}_1} \times \mathbb{B}, \\ 0 & \text{if } q = q_{\mathsf{acc}}, \\ 1 & \text{if } q = q_{\mathsf{rej}}, \end{cases}$
- for $q \in Q^{\mathcal{S}_1}$ and $\zeta \in \mathbb{B}$,

$$\delta^{\mathcal{A}_1}\big((q, \top), \top\big) = \bigvee_{[b \,\in\, \Sigma_{\mathrm{out}}^{\geq 1}]} \; \bigwedge_{[(x,y) \,\in\, \Sigma_0 \times \Sigma_{\mathrm{out}}^{\geq 1}]} \big(\downarrow_y, q^{(b,x,y)} \big)$$

$$\text{where } q^{(b,x,y)} = \begin{cases} \big(\delta^{\mathcal{S}_1}\big(q, (x,y)\big), \top\big) & \text{if } y = b, \\ q_{\mathsf{acc}} & \text{if } y \neq b, \end{cases}$$

$$\delta^{\mathcal{A}_1}\big((q, \top), \bot\big) = \bigwedge_{y \,\in\, \Sigma_{\mathrm{out}}^{\geq 1}} (\downarrow_y, q_{\mathsf{rej}}),$$

$$\delta^{\mathcal{A}_1}(q_{\mathsf{acc}}, \zeta) = \bigwedge_{y \,\in\, \Sigma_{\mathrm{out}}^{\geq 1}} (\downarrow_y, q_{\mathsf{acc}}),$$

$$\delta^{\mathcal{A}_1}(q_{\mathsf{rej}}, \zeta) = \bigwedge_{y \,\in\, \Sigma_{\mathrm{out}}^{\geq 1}} (\downarrow_y, q_{\mathsf{rej}}).$$

The second component of a state is used to keep track of the labelings of the guessed tree t. Notice, that in the definition of the transition function the $\bot$ symbol in the second component of a state is omitted, since $t_{\mathrm{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_0)$ with $t_{\mathrm{in}}(u) = \top$, for all $u \in \Sigma_0^*$, and if the guessed tree t is in fact from $t_{\mathrm{in}} \hookrightarrow t_{\mathrm{out}}$ then we have $t(u^\frown v) = \bot$ implies $t_{\mathrm{out}}(v) = \bot$ for all $u^\frown v \in (\Sigma_0 \times \Sigma_{\mathrm{out}}^{\geq 1})^*$. Hence, if $\bot$ is guessed for a node of t then the automaton initiates an accepting branch by sending a copy with state q_{acc} in the appropriate direction of t_{out} where it is no longer needed to keep track of the guessed symbols. The automaton $\mathcal{A}_1$ processes a tree t_{out} as follows. Being in node $v \in \Sigma_{\mathrm{out}}^{\geq 1}$ in some state $(q, \top)$ it guesses a symbol $b \in \Sigma_{\mathrm{out}}^{\geq 1}$ which corresponds to the joint output of all system processes. Furthermore, for every possible input symbol $x \in \Sigma_0$, the automaton sends a $\top$-copy in direction b while updating state q according to the transition function of $\mathcal{S}_1$ on the symbol (x, b). For all other directions $y \neq b$, a $\bot$-copy is sent

initiating an accepting branch of the run of $\mathcal{A}_1$. Moreover, if symbol $\bot$ is encountered in t_{out} when being in a $\top$-copy which means that the output symbol $b \in \Sigma_{\text{out}}^{\geq 1}$ should not have been chosen in the previous step according to t_{out}, then t_{out} is rejected. In this way, $\mathcal{A}_1$ guesses a tree $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ where $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_0)$ with $t_{\text{in}}(u) = \top$, for all $u \in \Sigma_0^*$, and simulates $\mathcal{S}_1$ on every $\alpha \in L^{(\omega)}(t)$. Thus, if t_{out} is accepted by $\mathcal{A}_1$ then for the tree t guessed by $\mathcal{A}_1$ we have $\sigma(t)$ is locally winning on Σ_0^ω.

Now, for the induction step, let $1 < i < n$. Let $\mathcal{N}_{i-1}$ be a parity-NTA over $\mathbb{B}$-labeled $\Sigma_{\text{out}}^{\geq i-1}$-trees satisfying the above requirements according to the induction hypothesis. By Remark 2.8 a parity-NTA

$$\mathcal{N}'_{i-1} = (Q^{\mathcal{N}'_{i-1}}, \mathbb{B}, \delta^{\mathcal{N}'_{i-1}}, q_{\text{in}}^{\mathcal{N}'_{i-1}}, \text{col}^{\mathcal{N}'_{i-1}})$$

over $\mathbb{B}$-labeled $\Sigma_{i-1} \times \Sigma_{\text{out}}^{\geq i}$-trees can be constructed which accepts a tree t if and only if $\mathsf{wide}_Y(t) \in L(\mathcal{N}_{i-1})$ where $Y = \Sigma_{\text{out}}^{\mathsf{b},p_{i-1}}$. We show how a parity-NTA $\mathcal{N}_i$ over $\mathbb{B}$-labeled $\Sigma_{\text{out}}^{\geq i}$-trees can be constructed using $\mathcal{N}'_{i-1}$ and $\mathcal{S}_i$ such that $\mathcal{N}_i$ accepts a tree $t_{\text{out}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{\text{out}}^{\geq i})$ if and only if there exists a tree $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{i-1})$ and a tree $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ such that $t \in L(N'_{i-1})$ and $\sigma(t)$ is locally winning on $L^{(\omega)}(t_{\text{in}})$. Again, we define parity-NTA $\mathcal{N}_i$ by first constructing an alternating tree automaton, this time a Muller-ATA $\mathcal{A}_i$, over $\mathbb{B}$-labeled $\Sigma_{\text{out}}^{\geq i}$-trees which is then translated into $\mathcal{N}_i$. The idea here is to guess both trees $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{i-1})$ and $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ and to verify that $t \in L(\mathcal{N}'_{i-1})$ as well as that $\sigma(t)$ is locally winning on $L^{(\omega)}(t_{\text{in}})$ by simulating $\mathcal{N}'_{i-1}$ on t and simulating $\mathcal{S}_i$ on every $\alpha \in L^{(\omega)}(t_{\text{in}})$. Formally, $\mathcal{A}_i = (Q^{\mathcal{A}_i}, \mathbb{B}, \delta^{\mathcal{A}_i}, q_{\text{in}}^{\mathcal{A}_i}, \mathcal{F}^{\mathcal{A}_i})$ is defined as follows.

- $Q^{\mathcal{A}_i} = ((Q^{\mathcal{S}_i} \cup \{q_{\mathsf{acc}}\}) \times Q^{\mathcal{N}'_{i-1}} \times \mathbb{B}) \cup \{q_{\mathsf{rej}}\}$,
- $q_{\text{in}}^{\mathcal{A}_i} = (q_{\text{in}}^{\mathcal{S}_i}, q_{\text{in}}^{\mathcal{N}'_{i-1}}, \top)$,
- a subset $Q \subseteq Q^{\mathcal{A}_i}$ is contained in $\mathcal{F}^{\mathcal{A}_i}$ if and only if $q_{\mathsf{rej}} \notin Q$ and $\min\{\text{col}^{\mathcal{S}_i}(\text{Pr}_0(q)) \mid q \in Q\}$ is even and $\min\{\text{col}^{\mathcal{N}'_{i-1}}(\text{Pr}_1(q)) \mid q \in Q\}$ is even where we define $\text{col}^{\mathcal{S}_i}(q_{\mathsf{acc}}) = 0$,
- for $q \in Q^{\mathcal{S}_i}$, $p \in Q^{\mathcal{N}'_{i-1}}$ and $\zeta \in \mathbb{B}$,

  $\delta^{\mathcal{A}_i}\big((q,p,\top),\top\big) =$

  $$\bigvee_{[\emptyset \neq X \subseteq \Sigma_{i-1}]} \bigvee_{[b \in \Sigma_{\text{out}}^{\geq i}]} \bigvee_{[\varphi \in \delta^{\mathcal{N}'_{i-1}}(p,\top)]} \bigwedge_{[(x,y) \in \Sigma_{i-1} \times \Sigma_{\text{out}}^{\geq i}]} (\downarrow_y, q^{(X,b,\varphi,x,y)})$$

$$\text{where } q^{(X,b,\varphi,x,y)} = \begin{cases} \big(\delta^{\mathcal{S}_i}\big(q,(x,y)\big),r,\top\big) & \text{if } x \in X, y = b \text{ and} \\ & (\downarrow_{(x,y)},r) \in \varphi, \\ (q_{\mathsf{acc}},r,\bot) & \text{if } x \notin X \text{ or } y \neq b \text{ and} \\ & (\downarrow_{(x,y)},r) \in \varphi, \end{cases}$$

$$\delta^{\mathcal{A}_i}\big((q,p,\top),\bot\big) = \bigwedge_{y \,\in\, \Sigma_{\text{out}}^{\geq i}} (\downarrow_y, q_{\mathsf{rej}}),$$

$$\delta^{\mathcal{A}_i}\big((q_{\mathsf{acc}},p,\bot),\zeta\big) = \bigvee_{[\varphi \in \delta^{\mathcal{N}'_{i-1}}(p,\bot)]} \; \bigwedge_{[(x,y)\in\Sigma_{i-1}\times\Sigma_{\text{out}}^{\geq i}]} \big(\downarrow_y, (q_{\mathsf{acc}}, p^{(\varphi,x,y)},\bot)\big)$$

$$\text{where } (\downarrow_{(x,y)}, p^{(\varphi,x,y)}) \in \varphi,$$

$$\delta^{\mathcal{A}_i}(q_{\mathsf{rej}},\zeta) = \bigwedge_{y \,\in\, \Sigma_{\text{out}}^{\geq i}} (\downarrow_y, q_{\mathsf{rej}}).$$

Due to this construction, in each step $\mathcal{A}_i$ guesses a symbol $b \in \Sigma_{\text{out}}^{\geq i}$ corresponding to the joint output of all system processes p_j, for $i \leq j \leq n$, as well as a set $\emptyset \neq X \subseteq \Sigma_{i-1}$ of possible inputs which can be received via the forward-channels from process p_{i-1}. For every symbol $(x,b) \in \Sigma_{i-1} \times \Sigma_{\text{out}}^{\geq i}$ with $x \in X$, a $\top$-copy is sent in direction b, otherwise a $\bot$-copy is sent. Similar to the base case, if symbol $\bot$ is encountered in t_{out} when being in a $\top$-copy which means that the output symbol $b \in \Sigma_{\text{out}}^{\geq i}$ should not have been chosen in the previous step according to t_{out}, then t_{out} is rejected. By doing so, trees $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{i-1})$ and $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ are guessed by $\mathcal{A}_i$. Moreover, $\mathcal{A}_i$ simulates $\mathcal{N}'_{i-1}$ on t and it simulates $\mathcal{S}_i$ on every $\alpha \in L^{(\omega)}(t)$. Thus, $\mathcal{A}_i$ accepts a tree t_{out} if and only if for the trees $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{i-1})$ and $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ guessed by $\mathcal{A}_i$ we have $t \in L(\mathcal{N}'_{i-1})$ and $\sigma(t)$ is locally winning on $L^{(\omega)}(t_{\text{in}})$. Using the induction hypothesis, one verifies that the parity-NTA $\mathcal{N}_i$ equivalent to $\mathcal{A}_i$ fulfills all conditions of the lemma. □

Now, we tackle the worst informed process of a pipeline with backward-channels. Recall, that we consider system specifications given by a list of local specifications which are regular for system processes $p_1, \ldots, p_{n-1}$ and deterministic contextfree for the worst informed process p_n.

Lemma 5.6. *There is a parity-NPDTA $\mathcal{N}_n$ over $\mathbb{B}$-labeled $\Sigma_{n-1} \times \Sigma_{\text{out}}^{p_n}$-trees which accepts a tree t if and only if there exist*

- *a tree $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{n-1})$ and*

- *a tree* $t_{\text{out}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{\text{out}}^{p_n})$

such that $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ *and* $\sigma(t)$ *is locally winning on* $L^{(\omega)}(t_{\text{in}})$.

Proof. Let $\mathcal{S}_n = (Q^{\mathcal{S}_n}, \Sigma_{n-1} \times \Sigma_{\text{out}}^{p_n}, \Gamma^{\mathcal{S}_n}, \delta^{\mathcal{S}_n}, q_{\text{in}}^{\mathcal{S}_n}, \text{col}^{\mathcal{S}_n})$ be a parity-DPDA recognizing the local specification for process p_n, i.e., $L(\mathcal{S}_n) = L_{p_n}$. We construct a parity-NPDTA $\mathcal{N}_n$ satisfying the requirements of the lemma using the following idea. Given a $\mathbb{B}$-labeled $\Sigma_{n-1} \times \Sigma_{\text{out}}^{p_n}$-tree t, the pushdown tree automaton $\mathcal{N}_n$ guesses a strategy $\sigma\colon \Sigma_{n-1}^* \to \Sigma_{\text{out}}^{p_n}$ and checks whether $\sigma = \sigma(t)$, i.e., whether the guessed strategy corresponds to the strategy represented by t. For this, define $\mathcal{N}_n = (Q^{\mathcal{N}_n}, \mathbb{B}, \Gamma^{\mathcal{N}_n}, \delta^{\mathcal{N}_n}, q_{\text{in}}^{\mathcal{N}_n}, \text{col}^{\mathcal{N}_n})$ as follows.

- $Q^{\mathcal{N}_n} = Q^{\mathcal{S}_n} \cup \{q_{\mathsf{rej}}, q_\perp\}$,
- $q_{\text{in}}^{\mathcal{N}_n} = q_{\text{in}}^{\mathcal{S}_n}$,
- $\Gamma^{\mathcal{N}_n} = \Gamma^{\mathcal{S}_n}$,
- $\text{col}^{\mathcal{N}_n}(q) = \begin{cases} \text{col}^{\mathcal{S}_n}(q) & \text{if } q \in Q^{\mathcal{S}_n}, \\ 1 & \text{if } q = q_{\mathsf{rej}}, \\ 0 & \text{if } q = q_\perp, \end{cases}$
- for $q \in Q^{\mathcal{S}_n}$, $A \in \Gamma_\perp^{\mathcal{N}_n}$ and $\zeta, \zeta' \in \mathbb{B}$,

  if $\delta^{\mathcal{S}_n}(q, \varepsilon, A) \neq \emptyset$ then

  $\delta^{\mathcal{N}_n}(q, \top, A) = (\circlearrowleft_\varepsilon, \delta^{\mathcal{S}_n}(q, \varepsilon, A))$,

  if $\delta^{\mathcal{S}_n}(q, \varepsilon, A) = \emptyset$ then

  $$\delta^{\mathcal{N}_n}(q, \top, A) = \bigvee_{[\emptyset \neq X \subseteq \Sigma_{n-1}]} \bigvee_{[b \in \Sigma_{\text{out}}^{p_n}]} \bigwedge_{[(x,y) \in \Sigma_{n-1} \times \Sigma_{\text{out}}^{p_n}]} (\downarrow_{(x,y)}, (q, \gamma)^{(X,b,x,y)})$$

  $$\text{where } (q, \gamma)^{(X,b,x,y)} = \begin{cases} \delta^{\mathcal{S}_n}(q, (x, y), A) & \text{if } x \in X \text{ and } y = b, \\ (q_\perp, A) & \text{else,} \end{cases}$$

  $\delta^{\mathcal{N}_n}(q_\perp, \perp, A) = \bigwedge_{z \in \Sigma} (\downarrow_z, q_\perp, A)$, and

  $\delta^{\mathcal{N}_n}(q, \perp, A) = \delta^{\mathcal{N}_n}(q_\perp, \top, A) = \delta^{\mathcal{N}_n}(q_{\mathsf{rej}}, \zeta, A) = \bigwedge_{z \in \Sigma} (\downarrow_z, q_{\mathsf{rej}}, A)$

  where $\Sigma = \Sigma_{n-1} \times \Sigma_{\text{out}}^{p_n}$.

According to this definition, in each step when being in a node labeled by $\top$, $\mathcal{N}_n$ chooses an output symbol $b \in \Sigma_{\text{out}}^{p_n}$ and guesses a nonempty set of possible inputs $X \subseteq \Sigma_{n-1}$ which process p_n may receive in the next step. For every $x \in X$, a $\top$-copy is sent in direction (x, b), since exactly those successors should be labeled by $\top$. Otherwise, a $\bot$-copy is sent signalizing that a node should be labeled by $\bot$. On the other hand, when being in a node labeled by $\top$, $\mathcal{N}_n$ sends a $\bot$-copy to all successors, as they have to be labeled by $\bot$. The automaton $\mathcal{N}_n$ checks whether the output symbols as well as the sets of input symbols are guessed correctly. Moreover, $\mathcal{N}_n$ simulates $\mathcal{S}_n$ on all paths labeled by $\top$. □

Now, by combining Lemma 5.5 and Lemma 5.6 we show that the realizability problem for pipelines with backward-channels is decidable for system specifications given by a list of local specifications where the local specification of the worst informed process is deterministic contextfree and all other local specifications are regular.

Theorem 5.7. *Let $\mathcal{A} = (P, C, r)$ be a pipeline with backward-channels with a labeling $(\Sigma_c)_{c \in C}$ and a list $\ell = (L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes. The realizability problem for $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$ and $L(\ell)$ is decidable if $L_{p_i} \in \mathsf{REG}_\omega$, for all $1 \leq i < n$, and $L_{p_n} \in \mathsf{DCFL}_\omega$.*

Proof. If $n = 1$, then the result follows directly from Theorem 3.18, since this case corresponds to the nondistributed setting, i.e., one has to solve a deterministic contextfree Gale-Stewart game. So consider $n > 1$. Let $\mathcal{N}_{n-1}$ be the parity-NTA over $\mathbb{B}$-labeled $\Sigma_{\text{out}}^{\geq n-1}$-trees according to Lemma 5.5 and $\mathcal{N}_n$ be the parity-NPDTA over $\mathbb{B}$-labeled $\Sigma_{n-1} \times \Sigma_{\text{out}}^{p_n}$-trees according to Lemma 5.6. Similar as in the proof of Lemma 5.5, let $\mathcal{N}'_{n-1}$ denote a parity-NTA over $\mathbb{B}$-labeled $\Sigma_{n-1} \times \Sigma_{\text{out}}^{\geq n}$-trees which accepts a tree t if and only if $\mathsf{wide}_Y(t) \in L(\mathcal{N}_{n-1})$ where $Y = \Sigma_{\text{out}}^{\mathsf{b}, p_{n-1}}$. Notice that $\mathcal{N}'_{n-1}$ can be constructed by Remark 2.8. Finally, let $\mathcal{N}$ be a parity-NPDTA over $\mathbb{B}$-labeled $\Sigma_{n-1} \times \Sigma_{\text{out}}^{p_n}$-trees such that $L(\mathcal{N}) = L(\mathcal{N}'_{n-1}) \cap L(\mathcal{N}_n)$. We show that $L(\ell)$ is realizable in $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$ if and only if $L(\mathcal{N}) \neq \emptyset$. As nonemptiness for parity-NPDTA is decidable by Theorem 2.9, the theorem follows.

Assume that there is a tree $t \in L(\mathcal{N})$. Then, since $t \in L(\mathcal{N}'_{n-1})$, we have $s_{\text{out}} = \mathsf{wide}_Y(t) \in L(\mathcal{N}_{n-1})$ where $Y = \Sigma_{\text{out}}^{\mathsf{b}, p_{n-1}}$, and by Lemma 5.5 there are a tree $s_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{n-2})$, a tree $s \in s_{\text{in}} \hookrightarrow s_{\text{out}}$ and strategies $\sigma_{p_j} \colon \Sigma_{j-1}^* \to \Sigma_{\text{out}}^{p_j}$, for $1 \leq j < n-1$, for processes $p_1, \ldots, p_{n-2}$ such that $(\sigma_{p_1}, \ldots, \sigma_{p_{n-2}}, \sigma(s))$ is winning for processes $p_1, \ldots, p_{n-1}$. Moreover, let $\sigma_{p_{n-1}} \colon \Sigma_{n-2}^* \to \Sigma_{\text{out}}^{p_{n-1}}$ be defined by

$$\sigma_{p_{n-1}}(u) = \Pr_{\Sigma_{\text{out}}^{p_{n-1}}}\big((\sigma(s))(u)\big),$$

for $u \in \Sigma_{n-2}^*$. Furthermore, since $t \in L(\mathcal{N}_n)$, by Lemma 5.6 there are trees $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{n-1})$ and $t_{\text{out}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{\text{out}}^{p_n})$ such that $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$ and $\sigma(t)$ is locally winning on $L^{(\omega)}(t_{\text{in}})$. Moreover, notice that by definition of the strategy product we have

$$\Pr_{\Sigma_{\text{out}}^{\geq n-1}}\big(L^{(\omega)}(s)\big) \subseteq L^{(\omega)}(s_{\text{out}}) \text{ and } \Pr_{\Sigma_{n-1}}\big(L^{(\omega)}(t)\big) = L^{(\omega)}(t_{\text{in}}).$$

Since $\Pr_{\Sigma_{n-1} \times \Sigma_{\text{out}}^{p_n}}(L^{(\omega)}(s_{\text{out}})) = L^{(\omega)}(t)$, the language generated by

$$\sigma_{p_1}^{(*)} \circ \Pr_{\Sigma_1} \circ \ldots \circ \sigma_{p_{n-2}}^{(*)} \circ \Pr_{\Sigma_{n-2}} \circ \sigma_{p_{n-1}} \circ \Pr_{\Sigma_{n-1}}$$

over Σ_0^ω is a subset of $L^{(\omega)}(t_{\text{in}})$. Hence, $\sigma_{\mathsf{sys}} = (\sigma_{p_1}, \ldots, \sigma_{p_{n-1}}, \sigma_{p_n})$ with $\sigma_{p_n} = \sigma(t)$ is winning for the system processes $p_1, \ldots, p_n$. Notice that by Lemma 5.4, the strategies $\sigma_{p_i} \colon \Sigma_{i-1}^* \to \Sigma_{\text{out}}^{p_i}$ can be translated into local strategies $\sigma'_{p_i} \colon (\Sigma_{\text{in}}^{p_i})^* \to \Sigma_{\text{out}}^{p_i}$, for $1 \leq i \leq n$.

Now, assume $L(\mathcal{N}) = \emptyset$. If $L(\mathcal{N}_n) = \emptyset$, then for all $t_{\text{in}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{n-1})$ and all $t_{\text{out}} \in \mathbb{T}_{\mathsf{com}}(\Sigma_{\text{out}}^{p_n})$ we have $\sigma(t)$ is not locally winning on $L^{(\omega)}(t_{\text{in}})$, for all $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$, i.e., the local specification L_{p_n} is not satisfiable. Otherwise, for every $t \in L(\mathcal{N}_n)$ we have $t \notin L(\mathcal{N}'_{n-1})$ which yields that processes $p_1, \ldots, p_{n-1}$ cannot generate any language $L \subseteq \Pr_{\Sigma_{n-1}}\big(L^{(\omega)}(t)\big)$ over Σ_0^ω. □

Two-Flanked Pipelines with Backward-Channels

Now, we consider two-flanked pipelines with backward-channels. Moreover, we consider only regular local specifications. We will prove decidability for the following two cases. First, we will show that a pipeline with backward-channels is decidable, if there are no backward-channels from the last process. Then, deal with the special case of two-flanked pipelines with backward-channels comprising just two system processes where we however allow backward-channels from the last process.

So, first let $\mathcal{A} = (P, C, r)$ be a two-flanked pipeline with backward-channels such that there are no backward-channels from the last process p_n, i.e., $r(C_{p_n}) = \{p_n\}$. Furthermore, let $(\Sigma_c)_{c \in C}$ be a labeling of $\mathcal{A}$ and $\ell = (L_{p_1}, \ldots, L_{p_n})$ be a list of local specifications for the system processes $p_1, \ldots, p_n$ with $L_{p_i} \in \mathsf{REG}_\omega$, for all $1 \leq i \leq n$.

Notice, that in this case, for all $0 < i < j < n$, process p_i is better informed than process p_j, however, process p_n is incomparably informed with any other system process. This means, that for all $0 < i \leq j < n$, process p_i can determine the decisions of process p_j, however not the decisions of process p_n. We exploit the fact that there is a linear informedness order on $P_{\mathsf{sys}} \setminus \{p_n\}$ which allows us to use the ideas and constructions developed above for pipelines with backward-channels also for this case. For this, we adapt the definitions used for pipelines with backward-channels. The partition of

the set of channels $C = C_{\mathsf{f}} \cup C_{\mathsf{b}} \cup C_{\mathsf{env}}$ is defined exactly as before. For every system process $p_i \in P_{\mathsf{sys}} \setminus \{p_n\}$ we redefine the accumulated output alphabet $\Sigma_{\text{out}}^{\geq i}$ of process p_i which comprises the output alphabets of all worse informed processes as

$$\Sigma_{\text{out}}^{\geq i} = \prod_{j=i}^{n-1} \Sigma_{\text{out}}^{p_j},$$

i.e., we exclude the output alphabet of the last process p_n. The alphabets $\Sigma_{\text{out}}^{\mathsf{b},p_i}$, for $1 \leq i \leq n$, and Σ_i, for $0 < i < n$ are defined as before. However, we have to account for the external input channels via which now information is sent to the first process p_1 and to the last process p_n. Hence, we define

$$\Sigma_{01} = \Sigma_0 = \prod_{\substack{c \in C_{\mathsf{env}}, \\ r(c)=p_1}} \Sigma_c, \text{ and } \Sigma_{0n} = \prod_{\substack{c \in C_{\mathsf{env}}, \\ r(c)=p_n}} \Sigma_c \ .$$

As for pipelines with backward-channels, also for this architecture, one can show that a local strategy of a system process $p_i \in P_{\mathsf{sys}} \setminus \{p_n\}$ needs not to depend on the inputs received via channels from C_{b} but only on those inputs received from process p_{i-1}. Thus, Lemma 5.4 is reformulated as follows.

Lemma 5.8. *Let L be a global system specification. There are local strategies $\sigma_{p_i} \colon (\Sigma_{\text{in}}^{p_i})^* \to \Sigma_{\text{out}}^{p_i}$ for the system processes, for $1 \leq i \leq n$, such that the joint strategy $\sigma_{\mathsf{sys}} = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ is winning if and only if there are strategies $\tau_{p_i} \colon \Sigma_{i-1}^* \to \Sigma_{\text{out}}^{p_i}$, for $1 \leq i \leq n-1$, and $\tau_{p_n} \colon (\Sigma_{\text{in}}^{p_n})^* \to \Sigma_{\text{out}}^{p_n}$ such that $\tau_{\mathsf{sys}} = (\tau_{p_1}, \ldots, \tau_{p_n})$ is winning.*

So, analogously to the case of pipelines with backward-channels, we can define extended local strategies. For $1 \leq i \leq n-1$, an extended local strategy for process p_i is now defined as $\sigma_{\geq i} = (\sigma_i, \ldots, \sigma_{n-1})$ where

$$\sigma_j \colon \Sigma_{i-1}^* \to \Sigma_{\text{out}}^{p_j}, \text{ for } i \leq j \leq n-1,$$

i.e., $\sigma_{\geq i}$ now determines the next output symbol for each process which is worse informed than p_i based on the inputs sent from p_{i-1} to p_i. Accordingly, for a language $L_{\text{in}} \subseteq \Sigma_{i-1}^{\omega}$, we say that an extended local strategy $\sigma_{\geq i} = (\sigma_i, \ldots, \sigma_{n-1})$ for process p_i is locally winning on L_{in} if for each global system behavior $\alpha_{\mathcal{A}} \in (\Sigma^{\mathcal{A}})^{\omega}$ of $\mathcal{A}$ with $\Pr_{\Sigma_{i-1}}(\alpha_{\mathcal{A}}) \in L_{\text{in}}$ which is consistent with $\sigma_{\geq i}$, we have $\Pr_{\Sigma^{p_i}}(\alpha_{\mathcal{A}}) \in L_{p_i}$. Moreover, if $\sigma_{\geq i}$ is an extended local strategy for process p_i and $\sigma_1, \ldots, \sigma_{i-1}$ are local strategies for processes $p_1, \ldots, p_{i-1}$, then we call $(\sigma_1, \ldots, \sigma_{i-1}, \sigma_{\geq i})$ winning for processes $p_1, \ldots, p_i$ if any global system behavior $\alpha_{\mathcal{A}}$ which is consistent with $(\sigma_1, \ldots, \sigma_{i-1}, \sigma_{\geq i})$ fulfills $\Pr_{\Sigma^{p_j}}(\alpha_{\mathcal{A}}) \in L_{p_j}$, for $1 \leq j \leq i$.

Clearly, with the new definitions of accumulated output alphabets and extended local strategies Lemma 5.5 holds for this case just as before. Now, in contrast to pipelines with backward-channels which we treated previously, in this case the last process p_n has additionally some input channels from the environment, however we consider only regular local specifications also for this process. So, Lemma 5.6 has to be reformulated as follows.

Lemma 5.9. *There is a parity-NTA $\mathcal{N}_n$ over $\mathbb{B}$-labeled Σ_{n-1}-trees which accepts a tree $t \in \mathbb{T}_{\mathsf{com}}(\Sigma_{n-1})$ if and only if there is a local strategy for process p_n which is locally winning on $\{\alpha \in (\Sigma_{\text{in}}^{p_n})^\omega \mid \Pr_{\Sigma_{n-1}}(\alpha) \in L^\omega(t)\}$.*

Proof. Let $\mathcal{S}_n = (Q^{\mathcal{S}_n}, \Sigma^{p_n}, \delta^{\mathcal{S}_n}, q_{\text{in}}^{\mathcal{S}_n}, \text{col}^{\mathcal{S}_n})$ be a parity-DFA recognizing the local specification for process p_n, i.e., $L(\mathcal{S}_n) = L_{p_n}$. The idea is to construct a parity-ATA $\mathcal{A}_n$ which guesses a strategy for process p_n when running on a tree $t \in \mathbb{T}_{\mathsf{com}}(\Sigma_{n-1})$, and verifies that this strategy is locally winning on $\{\alpha \in (\Sigma_{\text{in}}^{p_n})^\omega \mid \Pr_{\Sigma_{n-1}}(\alpha) \in L^\omega(t)\}$ by simulating $\mathcal{S}_n$ on all infinite paths of t labeled by $\top$. Formally, the parity-ATA $\mathcal{A}_n = (Q^{\mathcal{A}_n}, \mathbb{B}, \delta^{\mathcal{A}_n}, q_{\text{in}}^{\mathcal{A}_n}, \text{col}^{\mathcal{A}_n})$ is defined as follows.

- $Q^{\mathcal{A}_n} = Q^{\mathcal{S}_n} \cup \{q_{\mathsf{acc}}\}$,
- $q_{\text{in}} = q_{\text{in}}^{\mathcal{S}_n}$,
- $\text{col}^{\mathcal{A}_n}(q) = \begin{cases} \text{col}^{\mathcal{S}_n}(q) & \text{if } q \in Q^{\mathcal{S}_n}, \\ 0 & \text{if } q = q_{\mathsf{acc}}, \end{cases}$
- for $q \in Q^{\mathcal{S}_n}$,
$$\delta^{\mathcal{A}_n}(q, \top) = \bigvee_{[b \in \Sigma_{\text{out}}^{p_n}]} \bigwedge_{[(x,y) \in \Sigma_{n-1} \times \Sigma_{0n}]} \big(\downarrow_x, \delta^{\mathcal{S}_n}(q, (x, y, b)) \big),$$
- for $q \in Q^{\mathcal{A}_n}$,
$$\delta^{\mathcal{A}_n}(q, \bot) = \bigvee_{[b \in \Sigma_{\text{out}}^{p_n}]} \bigwedge_{[(x,y) \in \Sigma_{n-1} \times \Sigma_{0n}]} \big(\downarrow_x, q_{\mathsf{acc}} \big).$$

By this construction, in each step when being in a node labeled by $\top$, $\mathcal{A}_n$ chooses an output symbol $b \in \Sigma_{\text{out}}^{p_n}$ and for every input symbol $(x, y) \in \Sigma_{n-1} \times \Sigma_{0n}$ a copy is sent in direction x leading to the appropriate state according to the transition function $\delta^{\mathcal{S}_n}$. On the other hand when being in a node labeled by $\bot$, the accepting state q_{acc} is sent. Clearly, in this way, the automaton guesses a strategy for process p_n and by simulating $\mathcal{S}_n$ on all infinite paths labeled by $\top$ it checks whether the guessed strategy is winning on $\{\alpha \in (\Sigma_{\text{in}}^{p_n})^\omega \mid \Pr_{\Sigma_{n-1}}(\alpha) \in L^\omega(t)\}$. Moreover, $\mathcal{A}_n$ can be translated into an equivalent parity-NTA $\mathcal{N}_n$. □

Along the lines of Theorem 5.7, now Lemma 5.5 and Lemma 5.9 are combined to show the decidability of the realizability problem for two-flanked pipelines with backward-channels with the restriction on the last process which is not admitted to have backward-channels for specifications given by a list of regular local specifications.

Theorem 5.10. *Let $\mathcal{A} = (P, C, r)$ be a two-flanked pipeline with backward-channels with a labeling $(\Sigma_c)_{c\in C}$ and a list $\ell = (L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes. The realizability problem for $\mathcal{A}$ with $(\Sigma_c)_{c\in C}$ and $L(\ell)$ is decidable if $r(C_{p_n}) = p_n$ and $L_{p_i} \in \mathsf{REG}_\omega$, for all $1 \leq i \leq n$.*

Proof. If $n = 1$, then the result trivially holds as in this case $\mathcal{A}$ is a pipeline comprising just one system process. So consider $n > 1$. Let $\mathcal{N}_{n-1}$ be the parity-NTA over $\mathbb{B}$-labeled $\Sigma_{\text{out}}^{p_{n-1}}$-trees according to Lemma 5.5 and $\mathcal{N}'_{n-1}$ be a corresponding parity-NTA over $\mathbb{B}$-labeled Σ_{n-1}-trees which accepts a tree t if and only if $\mathsf{wide}_Y(t) \in L(\mathcal{N}_{n-1})$ where $Y = \Sigma_{\text{out}}^{\mathsf{b},p_{n-1}}$. Moreover, let $\mathcal{N}_n$ be the parity-NTA over $\mathbb{B}$-labeled Σ_{n-1}-trees according to Lemma 5.9. Furthermore, let $\mathcal{N}$ be a parity-NTA over $\mathbb{B}$-labeled Σ_{n-1}-trees such that $L(\mathcal{N}) = L(\mathcal{N}'_{n-1}) \cap L(\mathcal{N}_n)$. Clearly, analogously as in the proof of Theorem 5.10, $L(\ell)$ is realizable in $\mathcal{A}$ with $(\Sigma_c)_{c\in C}$ if and only if $L(\mathcal{N}) \neq \emptyset$. As nonemptiness for parity-NTA is decidable, the theorem follows. □

Now we consider two-flanked pipelines with backward-channels where backward-channels from the last process are allowed, but the number of system processes is restricted to two. To show that the realizability problem for such architectures is decidable for specifications given by a pair of regular local specifications, we will again use alternating parity tree automata which guess local strategies, each for the respective system process. The difficulty here lies in the treatment of internal communication channels which are at the same time output channels (for one process) and input channels (for the other process). The idea is to extend communication trees to be able to represent a joint output language produced on the internal communication channels by the system processes.

For two alphabets Σ_0 and Σ_1, we say that a $(\mathbb{B}\times\mathbb{B})$-labeled full $(\Sigma_0\times\Sigma_1)$-tree t is an extended communication tree over $\Sigma_0 \times \Sigma_1$ if for every node $u^\frown v \in (\Sigma_0 \times \Sigma_1)^*$ the following is satisfied, for $i \in \{0, 1\}$,

- $t(\varepsilon) = (\top, \top)$,
- if $\mathrm{Pr}_i(t(u^\frown v)) = \bot$ then $\mathrm{Pr}_i(t(ua_0{}^\frown va_1)) = \bot$ for all $a_0 \in \Sigma_0$, $a_1 \in \Sigma_1$,
- if $\mathrm{Pr}_i(t(u^\frown v)) = \top$ then $\mathrm{Pr}_i(t(ua_0{}^\frown va_1)) = \top$ for some $a_i \in \Sigma_i$ and all $a_{1-i} \in \Sigma_{1-i}$.

We denote the set of all extended communication trees over $\Sigma_0 \times \Sigma_1$ by $\mathbb{T}^{\mathsf{ext}}_{\mathsf{com}}(\Sigma_0 \times \Sigma_1)$. An extended communication tree $t \in \mathbb{T}^{\mathsf{ext}}_{\mathsf{com}}(\Sigma_0 \times \Sigma_1)$ represents the joint communication language

$$L^{(\omega)}(t) = \{\alpha \in (\Sigma_0 \times \Sigma_1)^\omega \mid t(\mathsf{pref}_k(\alpha)) = (\top, \top) \text{ for all } k \in \mathbb{N}\}.$$

We will use extended communication trees over the alphabet $\Sigma_1 \times \Sigma_2$ to represent the joint output language produced on the internal communication channels by the system processes p_1 and p_2, where Σ_i is the alphabet containing the symbols which can be sent along the internal communication channels from process p_i to the other system process, for $i \in \{1, 2\}$. Notice, that with regard to the strategies for the system processes even more information can be deduced from an extended communication tree than the joint language represented by the tree. If a node $u^\frown v \in (\Sigma_1 \times \Sigma_2)^*$ is labeled by $(\top, \bot)$, then it means that process p_1 may answer the prefix $\mathsf{pref}_{|v|-1}(v)$ by u, but process p_2 may not answer the prefix $\mathsf{pref}_{|u|-1}(u)$ by v and the other way around for the labeling $(\bot, \top)$. This assertion is, of course, different from saying that $u^\frown v$ will not occur or should not be produced.

Theorem 5.11. *Let $\mathcal{A} = (P, C, r)$ be a two-flanked pipeline with backward-channels with a labeling $(\Sigma_c)_{c \in C}$ and a list $\ell = (L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes. The realizability problem for $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$ and $L(\ell)$ is decidable if $|P_{\mathsf{sys}}| = 2$ and $L_{p_i} \in \mathsf{REG}_\omega$, for all $1 \leq i \leq n$.*

Proof. For $i \in \{1, 2\}$, let Γ_i denote the alphabet containing the symbols which process p_i can receive via external input channels from the environment. Moreover, let Σ_i denote the alphabet containing the symbols which can be sent along the internal communication channels of process p_i and let alphabet Δ_i contain the symbols which can be sent via external output channels of process p_i, i.e., for $i \in \{1, 2\}$,

$$\Gamma_i = \prod_{\substack{c \in C_{\mathsf{env}}, \\ r(c) = p_i}} \Sigma_c, \qquad \Sigma_i = \prod_{c \in C_i \setminus H_i} \Sigma_c, \quad \text{and} \quad \Delta_i = \prod_{c \in H_i} \Sigma_c.$$

Figure 5.5 depicts the induced graph $G_{\mathcal{A}}$ of $\mathcal{A}$ where the edges are labeled by the corresponding alphabets. Furthermore, let $\mathcal{S}_i = (Q^{\mathcal{S}_i}, \Sigma^{p_i}, \delta^{\mathcal{S}_i}, q_{\text{in}}^{\mathcal{S}_i}, \text{col}^{\mathcal{S}_i})$ be parity-DFA recognizing the local specification of process p_i, i.e., $L(\mathcal{S}_i) = L_{p_i}$, for $i \in \{1, 2\}$. The idea is to construct two parity-ATA $\mathcal{A}_1$ and $\mathcal{A}_2$ over $(\mathbb{B} \times \mathbb{B})$-labeled $(\Sigma_1 \times \Sigma_2)$-trees such that, when running on a tree $t \in \mathbb{T}^{\mathsf{ext}}_{\mathsf{com}}(\Sigma_1 \times \Sigma_2)$, $\mathcal{A}_1$ guesses a strategy for process p_1 and $\mathcal{A}_2$ for process p_2, respectively. Furthermore, $\mathcal{A}_1$ verifies that the guessed strategy is locally winning on $\{\alpha \in (\Sigma_{\text{in}}^{p_1})^\omega \mid \text{Pr}_{\Sigma_2}(\alpha) \in \text{Pr}_{\Sigma_2}(L^\omega(t))\}$ by simulating $\mathcal{S}_1$ on all infinite paths of t labeled by $(\top, \top)$. Accordingly, $\mathcal{A}_2$ verifies that its

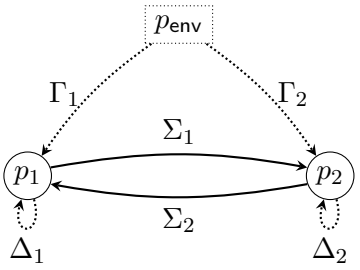


Figure 5.5: Two-flanked pipeline with two system processes

guessed strategy is locally winning on $\{\alpha \in (\Sigma_{\text{in}}^{p_2})^\omega \mid \Pr_{\Sigma_1}(\alpha) \in \Pr_{\Sigma_1}(L^\omega(t))\}$ using $\mathcal{S}_2$. Formally, the parity-ATA $\mathcal{A}_1 = (Q^{\mathcal{A}_1}, \mathbb{B} \times \mathbb{B}, \delta^{\mathcal{A}_1}, q_{\text{in}}^{\mathcal{A}_1}, \text{col}^{\mathcal{A}_1})$ is defined as follows.

- $Q^{\mathcal{A}_1} = Q^{\mathcal{S}_1} \cup \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$,
- $q_{\text{in}}^{\mathcal{A}_1} = q_{\text{in}}^{\mathcal{S}_1}$,
- $\text{col}^{\mathcal{A}_1}(q) = \begin{cases} \text{col}^{\mathcal{S}_1}(q) & \text{if } q \in Q^{\mathcal{S}_1}, \\ 0 & \text{if } q = q_{\mathsf{acc}}, \\ 1 & \text{if } q = q_{\mathsf{rej}}, \end{cases}$
- for $q \in Q^{\mathcal{S}_1}$ and $\zeta_1, \zeta_2, \zeta_3 \in \mathbb{B}$,

$$\delta^{\mathcal{A}_1}(q, (\top, \top)) = \bigvee_{[(a,b)\in\Sigma_1\times\Delta_1]} \bigwedge_{[(x,y)\in\Gamma_1\times\Sigma_2]} (\downarrow_{(a,y)}, (\delta^{\mathcal{S}_1}(q, (a, b, x, y)),$$

$$\delta^{\mathcal{A}_1}(q, (\bot, \zeta_1)) = \delta^{\mathcal{A}_1}(q_{\mathsf{rej}}, (\zeta_2, \zeta_3)) = \bigwedge_{z\in\Sigma_1\times\Sigma_2} (\downarrow_z, q_{\mathsf{rej}}),$$

$$\delta^{\mathcal{A}_1}(q, (\top, \bot)) = \delta^{\mathcal{A}_1}(q_{\mathsf{acc}}, (\zeta_1, \zeta_2)) = \bigwedge_{z\in\Sigma_1\times\Sigma_2} (\downarrow_z, q_{\mathsf{acc}}).$$

Analogously, parity-ATA $\mathcal{A}_2 = (Q^{\mathcal{A}_2}, \mathbb{B} \times \mathbb{B}, \delta^{\mathcal{A}_2}, q_{\text{in}}^{\mathcal{A}_2}, \text{col}^{\mathcal{A}_2})$ is defined by

- $Q^{\mathcal{A}_2} = Q^{\mathcal{S}_2} \cup \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$,
- $q_{\text{in}}^{\mathcal{A}_2} = q_{\text{in}}^{\mathcal{S}_2}$,
- $\text{col}^{\mathcal{A}_2}(q) = \begin{cases} \text{col}^{\mathcal{S}_2}(q) & \text{if } q \in Q^{\mathcal{S}_2}, \\ 0 & \text{if } q = q_{\mathsf{acc}}, \\ 1 & \text{if } q = q_{\mathsf{rej}}, \end{cases}$

- for $q \in Q^{\mathcal{S}_2}$ and $\zeta_1, \zeta_2, \zeta_3 \in \mathbb{B}$,

$$\delta^{\mathcal{A}_2}(q, (\top, \top)) = \bigvee_{[(a,b) \in \Sigma_2 \times \Delta_2]} \bigwedge_{[(x,y) \in \Gamma_2 \times \Sigma_1]} (\downarrow_{(y,a)}, (\delta^{\mathcal{S}_2}(q, (a, b, x, y)),$$

$$\delta^{\mathcal{A}_2}(q, (\zeta_1, \bot)) = \delta^{\mathcal{A}_2}(q_{\mathsf{rej}}, (\zeta_2, \zeta_3)) = \bigwedge_{z \in \Sigma_1 \times \Sigma_2} (\downarrow_z, q_{\mathsf{rej}}),$$

$$\delta^{\mathcal{A}_2}(q, (\bot, \top)) = \delta^{\mathcal{A}_2}(q_{\mathsf{acc}}, (\zeta_1, \zeta_2)) = \bigwedge_{z \in \Sigma_1 \times \Sigma_2} (\downarrow_z, q_{\mathsf{acc}}).$$

By the above construction, when running on a tree t, in each step, $\mathcal{A}_1$ guesses an output symbol $(a, b) \in \Sigma_1 \times \Delta_1$ and sends, for any possible input signal $(x, y) \in \Gamma_1 \times \Sigma_2$, a copy into direction (a, y). If a labeling $(\bot, \zeta)$, for some $\zeta \in \mathbb{B}$, is encountered when being in a state $q \in Q^{\mathcal{S}_1}$ which means, that the component $a \in \Sigma_1$ of the output should not have been chosen in previous step according to t, then $\mathcal{A}_1$ rejects t by proceeding into the rejecting state q_{rej}. On the other hand, if the labeling $(\top, \bot)$ is encountered when being in a state $q \in Q^{\mathcal{S}_1}$ which means, that the input $y \in \Sigma_2$ will not occur in this situation according to t, then $\mathcal{A}_1$ goes into the accepting state q_{acc}. By doing so, $\mathcal{A}_1$ guesses a local strategy for p_1 on all inputs from Γ_1 and those inputs from Σ_2 which may occur according to t. Moreover, $\mathcal{A}_1$ simulates $\mathcal{S}_1$ on all paths consistent with this strategy. Analogously, $\mathcal{A}_2$ guesses an output symbol $(a, b) \in \Sigma_2 \times \Delta_2$ in each step and sends, for any possible input signal $(x, y) \in \Gamma_2 \times \Sigma_1$, a copy into direction (y, a). If a labeling $(\zeta, \bot)$, for some $\zeta \in \mathbb{B}$, is encountered when being in a state $q \in Q^{\mathcal{S}_2}$ which means, that output $a \in \Sigma_2$ should not have been chosen in the previous step according to t, then t is rejected. If $(\bot, \top)$ is encountered which means, that the input $y \in \Sigma_1$ will not occur in this situation according to t, then $\mathcal{A}_2$ proceeds to q_{acc}. This way, $\mathcal{A}_2$ guesses a local strategy for p_2 on all inputs from Γ_2 and those inputs from Σ_1 which may occur according to t. Furthermore, now $\mathcal{S}_2$ is simulated on all paths consistent with this strategy.

Now, let $\mathcal{N}$ be a parity-NTA such that $L(\mathcal{N}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. Clearly, if $t \in L(\mathcal{N})$ then the accepting run of $\mathcal{A}_1$ on t yields a local winning strategy for process p_1 on all inputs coming from the environment and those inputs from p_2 which may occur according to t. Furthermore, the accepting run of $\mathcal{A}_2$ on t yields a local winning strategy for process p_2 on all inputs from the environment and inputs from p_1 which may occur according to t. On the other hand, for a joint winning strategy $\sigma_{\mathsf{sys}} = (\sigma_1, \sigma_2)$ for the system processes p_1 and p_2, we can define a $(\mathbb{B} \times \mathbb{B})$-labeled $(\Sigma_1 \times \Sigma_2)$-tree t such that $t \in L(\mathcal{N})$ as follows. For any node $u^\frown v \in (\Sigma_1 \times \Sigma_2)^*$, define its labeling by $\mathrm{Pr}_{\Sigma_1}(t(u^\frown v)) = \top$ if and only if there exist $w \in \Gamma_1^*$ and $x \in \Delta_1^*$ such that

- $|u| = |w| = |x|$ and
- $w^\frown v^\frown u^\frown x$ is consistent with σ_1,

and $\mathrm{Pr}_{\Sigma_2}(t(u^\frown v)) = \top$ if and only if there are $w \in \Gamma_2^*$ and $x \in \Delta_2^*$ such that

- $|v| = |w| = |x|$ and
- $w^\frown u^\frown v^\frown x$ is consistent with σ_2.

Hence, we have $L(\mathcal{N}) \neq \emptyset$ if and only if there is a joint winning strategy for the system processes p_1 and p_2. As nonemptiness for parity-NTA is decidable, the theorem follows. □

5.4.2 Undecidable Cases

In this subsection we will point out undecidable cases of architectures for system specifications given by a list of local specifications. Recall, that in the previous subsection the realizability problem for pipelines with backward-channels and specifications given by a list of local specifications is shown to be decidable if the local specification of the worst informed process is deterministic contextfree and all other local specifications are regular (see Theorem 5.7). The first question we investigate here is how a relaxation of this restrictions effects the decidability. So, first we consider architectures with specifications given by a list of local specifications which are now allowed to contain at least two deterministic contextfree local specifications. Then, we consider architectures which again have just one deterministic contextfree local specifications, however, now the corresponding system process is not the worst informed one. We show both cases to be undecidable even for deterministic one-counter specifications. Finally, we deal with two further special cases of architectures which are undecidable even if all local specifications are regular.

Theorem 5.12. *Let $\mathcal{A} = (P, C, r)$ be a connected architecture with a labeling $(\Sigma_c)_{c\in C}$ and a list $\ell = (L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes. The realizability problem for $\mathcal{A}$ with $(\Sigma_c)_{c\in C}$ and $L(\ell)$ is undecidable if $L_{p_i}, L_{p_j} \in \mathsf{D1CL}_\omega$, for some $i, j \in \{1, \ldots, n\}$ with $i \neq j$.*

Proof. We proceed by a reduction from the halting problem for 2-register machines. For this, let $\mathcal{R} = \big((0\colon I_0), \ldots, (k-1\colon I_{k-1}), (k\colon \texttt{HALT})\big)$ be a 2-register machine. Define the labeling for the architecture $\mathcal{A}$ by $\Sigma_c = [k+1]$, for every channel $c \in C \setminus C_{p_{\mathsf{env}}}$, and $\Sigma_c = \{\sharp\}$, for every external input channel $c \in C_{p_{\mathsf{env}}}$. Moreover, for every system process $p_m \in P_{\mathsf{sys}}$, let the language L_m contain exactly the words $\alpha \in (\Sigma^{p_m})^\omega$ which satisfy the following condition for all $d, d' \in [\mathsf{dim}(\Sigma^{p_m})]$,

$$\text{if } \mathrm{Pr}_d(\alpha) \neq \sharp^\omega \text{ and } \mathrm{Pr}_{d'}(\alpha) \neq \sharp^\omega \text{ then } \mathrm{Pr}_d(\alpha) = \mathrm{Pr}_{d'}(\alpha).$$

This means, that a local process behavior of a system process p_m is contained in L_m if and only if the symbols sent to the output channels by process p_m and those process p_m receives via internal communication channels are identical in each step.

Now, consider the following deterministic one-counter automata $\mathcal{P}_0$ and $\mathcal{P}_1$. Let parity-D1CA $\mathcal{P}_0 = (Q^{\mathcal{P}_0}, [k+1], \{A\}, q_{\text{in}}^{\mathcal{P}_0}, \delta^{\mathcal{P}_0}, \text{col}^{\mathcal{P}_0})$ be defined by

- $Q^{\mathcal{P}_0} = \{q_x \mid x \in [k+1]\} \cup \{q_{\mathsf{acc}}, q_{\mathsf{rej}}, q_?\}$,
- $q_{\text{in}}^{\mathcal{P}_0} = q_0$,
- $\text{col}^{\mathcal{P}_0}(q) = \begin{cases} 1 & \text{if } q \in Q^{\mathcal{P}_0} \setminus \{q_{\mathsf{acc}}\}, \\ 0 & \text{if } q = q_{\mathsf{acc}}, \end{cases}$
- for $x, y \in [k+1]$ and $Z \in \{A, \bot\}$,

$$\delta^{\mathcal{P}_0}(q_x, x, Z) = \begin{cases} (q_{x+1}, Z) & \text{if } I_x \in \{\texttt{INC(X}_1\texttt{)}, \texttt{DEC(X}_1\texttt{)}\}, \\ (q_?, Z) & \text{if } I_x = \texttt{IF X}_1\texttt{=0 GOTO y}, \\ (q_{x+1}, AZ) & \text{if } I_x = \texttt{INC(X}_0\texttt{)}, \\ (q_{x+1}, \varepsilon) & \text{if } I_x = \texttt{DEC(X}_0\texttt{)} \text{ and } Z = A, \\ (q_{x+1}, Z) & \text{if } I_x = \texttt{IF X}_0\texttt{=0 GOTO y} \text{ and } Z = A, \\ (q_y, Z) & \text{if } I_x = \texttt{IF X}_0\texttt{=0 GOTO y} \text{ and } Z = \bot, \\ (q_{\mathsf{acc}}, Z) & \text{if } I_x = \texttt{HALT}, \end{cases}$$

$$\begin{aligned} \delta^{\mathcal{P}_0}(q_?, x, Z) &= (q_x, x, Z), \\ \delta^{\mathcal{P}_0}(q_x, y, Z) &= (q_{\mathsf{rej}}, Z) \text{ if } x \neq y, \\ \delta^{\mathcal{P}_0}(q_{\mathsf{acc}}, x, Z) &= (q_{\mathsf{acc}}, Z), \\ \delta^{\mathcal{P}_0}(q_{\mathsf{rej}}, x, Z) &= (q_{\mathsf{rej}}, Z). \end{aligned}$$

We define the parity-D1CA $\mathcal{P}_1 = (Q^{\mathcal{P}_1}, [k+1], \{A\}, q_{\text{in}}^{\mathcal{P}_1}, \delta^{\mathcal{P}_1}, \text{col}^{\mathcal{P}_1})$ analogously as $\mathcal{P}_0$ by swapping the registers. Hence,

- $Q^{\mathcal{P}_1} = Q^{\mathcal{P}_0}$,
- $q_{\text{in}}^{\mathcal{P}_1} = q_0$,
- $\text{col}^{\mathcal{P}_1}(q) = \begin{cases} 1 & \text{if } q \in Q^{\mathcal{P}_1} \setminus \{q_{\mathsf{acc}}\}, \\ 0 & \text{if } q = q_{\mathsf{acc}}, \end{cases}$

- for $x, y \in [k+1]$ and $Z \in \{A, \bot\}$,

$$\delta^{\mathcal{P}_1}(q_x, x, Z) = \begin{cases} (q_{x+1}, Z) & \text{if } I_x \in \{\texttt{INC(X}_0\texttt{)}, \texttt{DEC(X}_0\texttt{)}\}, \\ (q_?, Z) & \text{if } I_x = \texttt{IF X}_0\texttt{=0 GOTO y}, \\ (q_{x+1}, AZ) & \text{if } I_x = \texttt{INC(X}_1\texttt{)}, \\ (q_{x+1}, \varepsilon) & \text{if } I_x = \texttt{DEC(X}_1\texttt{)} \text{ and } Z = A, \\ (q_{x+1}, Z) & \text{if } I_x = \texttt{IF X}_1\texttt{=0 GOTO y} \text{ and } Z = A, \\ (q_y, Z) & \text{if } I_x = \texttt{IF X}_1\texttt{=0 GOTO y} \text{ and } Z = \bot, \\ (q_{\mathsf{acc}}, Z) & \text{if } I_x = \texttt{HALT}, \end{cases}$$

$$\begin{aligned} \delta^{\mathcal{P}_1}(q_?, x, Z) &= (q_x, x, Z), \\ \delta^{\mathcal{P}_1}(q_x, y, Z) &= (q_{\mathsf{rej}}, Z) \text{ if } x \neq y, \\ \delta^{\mathcal{P}_1}(q_{\mathsf{acc}}, x, Z) &= (q_{\mathsf{acc}}, Z), \\ \delta^{\mathcal{P}_1}(q_{\mathsf{rej}}, x, Z) &= (q_{\mathsf{rej}}, Z). \end{aligned}$$

For $r \in \{0, 1\}$, the automaton $\mathcal{P}_r$ works a follows. While reading a sequence of line numbers $\alpha \in [k+1]^\omega$ it simulates register r of $\mathcal{R}$ using its stack. For this, being in some state q_x, for $x \in [k+1]$, which means that the automaton expects line number x for the next input symbol, if $\mathcal{P}_r$ reads a symbol $y \neq x$ then it proceeds to the rejecting state q_{rej}. Otherwise, $\mathcal{P}_r$ performs a push-, pop- or a skip-transition while updating its state according to instruction I_x. If I_x concerns the other register $1-r$ then the stack remains untouched, since it holds the value of register r which is not modified. Moreover, if register $1-r$ is required to be incremented or decremented then the next line number is certain and the state is updated to q_{x+1}. On the other hand, if I_x is a goto-instruction then $\mathcal{P}_r$ proceeds to the special state $q_?$ which means that the next line number is not known as the value of register $1-r$ is unknown, hence, no line number can be expected for the next input symbol. Otherwise, if I_x concerns register r then $\mathcal{P}_r$ proceeds according to I_x and the current stack content. A sequence of line numbers is accepted if line number k is read from state q_k.

Now, let $\ell = (L_{p_1}, \ldots, L_{p_n})$ be defined by $L_{p_m} = L_m$, for all $m \in \{1, \ldots, n\} \setminus \{i, j\}$, and $L_{p_i} = L_i \cap L(\mathcal{P}_0)$ and $L_{p_j} = L_j \cap L(\mathcal{P}_1)$. Clearly, in order to satisfy the specification $L(\ell)$ any joint winning strategy for the system processes has to ensure that the sequences written to the output channels of the processes are all identical, since $\mathcal{A}$ is connected and the local behavior of each process p_m has to satisfy L_m. Furthermore, this unique sequence produced by the joint strategy has to be contained in $L(\mathcal{P}_0) \cap L(\mathcal{P}_1)$, i.e., it has to correspond to the sequence of the line numbers of the run of $\mathcal{R}$

reaching the halting configuration. Thus, there is a joint winning strategy for the system processes if and only if $\mathcal{R}$ halts. □

Now, we consider architectures and specifications given by a list of local specifications with just one deterministic contextfree local specification. For this case, the following theorem gives a condition which yields undecidability.

Theorem 5.13. *Let $\mathcal{A} = (P, C, r)$ be an architecture with a labeling $(\Sigma_c)_{c\in C}$ and a list $\ell = (L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes. The realizability problem for $\mathcal{A}$ with $(\Sigma_c)_{c\in C}$ and $L(\ell)$ is undecidable if there are two system processes p_i and p_j with $i \neq j$ such that*

- *p_i and p_j are connected,*
- *$p_i \in P_{\mathsf{reach}}$,*
- *$L_{p_i} \in \mathsf{D1CL}_\omega$, and*
- *p_j is not better informed than p_i.*

Proof. We proceed by a reduction from the halting problem for 2-register machines. For this, given a 2-register machine

$$\mathcal{R} = \big((0\colon I_0), \ldots, (k-1\colon I_{k-1}), (k\colon \texttt{HALT})\big),$$

let Conf and $\mathsf{Conf}_{\mathrm{in}}$ be defined as in the proof of Theorem 4.4. The idea is to require that p_i and p_j simultaneously produce identical sequences of encodings of configurations of $\mathcal{R}$ starting with the initial configuration $\mathsf{Conf}_{\mathrm{in}}$. Furthermore, for every pair of consecutive encodings of configurations produced by the processes the environment should be able to trigger process p_i which is reachable and has a deterministic one-counter specification to check whether the successor relation $\vdash_{\mathcal{R}}$ is violated by indicating the claimed violation by one of the letters E_0, E_1 or L. Since p_j is not better informed than p_i this information can be hidden from process p_j. Obviously, the system processes have a joint winning strategy if and only if processes p_i and p_j manage to produce a sequence encoding the infinite run of $\mathcal{R}$ which exists if and only if $\mathcal{R}$ does not halt.

To be more formal, let processes $p_{x_0}, \ldots, p_{x_m}$ be such that $p_{x_e} \neq p_{x_{e'}}$, for all $e, e' \in [m+1]$ with $e \neq e'$, $p_{x_0} = p_i$, $p_{x_m} = p_j$, and for all $e \in [m]$, p_{x_e} sends information to $p_{x_{e+1}}$ or vice versa. Clearly, such processes exist, since p_i and p_j are connected. Furthermore, for $e \in [m]$, let $c_{x_e} \in C$ be such that either $c_{x_e} \in C_{p_{x_e}}$ and $r(c_{x_e}) = p_{x_{e+1}}$ or $c_{x_e} \in C_{p_{x_{e+1}}}$ and $r(c_{x_e}) = p_{x_e}$, and let $C_x = \{c_{x_e} \mid e \in [m]\}$. Moreover, due to the fact that p_i is reachable, there is a directed path from p_{env} to p_i, i.e., there are processes $p_{y_0}, \ldots, p_{y_l}$ such that $p_{y_e} \neq p_{y_{e'}}$, for all $e, e' \in [l+1]$ with $e \neq e'$, $p_{y_0} = p_{\mathsf{env}}$, $p_{y_l} = p_i$, and for all

$e \in [l]$, p_{y_e} sends information to $p_{y_{e+1}}$. Furthermore, for $e \in [l]$, let $c_{y_e} \in C$ be such that $c_{y_e} \in C_{p_{y_e}}$ and $r(c_{y_e}) = p_{y_{e+1}}$, and let $C_y = \{c_{y_e} \mid e \in [l]\}$. First, assume that $C_x \cap C_y = \emptyset$.

We define the labeling for $\mathcal{A}$ by $\Sigma_c = \{N, E_0, E_1, L\} = \Sigma_y$, for all $c \in C_y$, and $\Sigma_c = \{\sharp, r_0, r_1\} \cup [k+1] = \Sigma_x$, for all $c \in C_x \cup C_{p_i} \cup C_{p_j}$. Furthermore, let $\Sigma_c = \{\flat\}$, for all other channels $c \in C \setminus (C_x \cup C_{p_i} \cup C_{p_j} \cup C_y)$. Now, we define the local specifications as follows. Analogously, as in the proof of Theorem 5.12, for every process $p \in \{p_{x_1}, \ldots, p_{x_m}\}$, let the local specification L_p contain exactly the words $\alpha \in (\Sigma^p)^\omega$ satisfying the following condition for all $d, d' \in [\mathsf{dim}(\Sigma^p)]$

$$\text{if } \mathrm{Pr}_d(\alpha) \in \Sigma_x^\omega \text{ and } \mathrm{Pr}_{d'}(\alpha) \in \Sigma_x^\omega \text{ then } \mathrm{Pr}_d(\alpha) = \mathrm{Pr}_{d'}(\alpha).$$

Moreover, for every process $p \in \{p_{y_1}, \ldots, p_{y_{l-1}}\}$, define

$$L_p = \{\alpha^\frown\beta \in (\Sigma^p_{\text{in}} \times \Sigma^p_{\text{out}})^\omega \mid \mathrm{Pr}_{\Sigma_c}(\beta) = N \cdot \mathrm{Pr}_{\Sigma_{c'}}(\alpha), \text{ for } c, c' \in C_y\},$$

i.e., a local behavior of a process p_{y_r}, for $r \in \{1, \ldots, l-1\}$, is contained in $L_{p_{y_r}}$ if in each step the input symbols received via $c_{y_{r-1}}$ in the previous step are just forwarded to the output channels c_{y_r}, whereas in the first step where no input is read yet process p_{y_r} sends letter N.

Now, consider the following language $L_{\mathcal{R}}$ over $\Sigma_y \times \Sigma_x$. A word $\alpha^\frown\beta \in (\Sigma_y \times \Sigma_x)^\omega$ is contained in $L_{\mathcal{R}}$ if

- $\beta = \sharp^l \sharp c_0 \sharp c_1 \sharp c_2 \cdots$ where $c_e \in \mathsf{Conf}$, for $e \in \mathbb{N}$, and $c_0 = \mathsf{Conf}_{\text{in}}$, and
- if $\alpha = uXw$ with $u \in \{N\}^*$, $X \in \{L, E_0, E_1\}$ and $w \in (\Sigma_y)^\omega$ then for the smallest $s \geq |u|$ such that $\beta(s) = \sharp$ and $\beta(s+1) \in [k+1]$ the following holds. If

  $$\beta = \mathsf{pref}_s(\beta) \sharp\, \ell\, r_0^{n_0}\, r_1^{n_1} \sharp\, \ell'\, r_0^{n_0'}\, r_1^{n_1'} \sharp\, \beta',$$

  for some $n_0, n_1, n_0', n_1' \in \mathbb{N}$ and $\beta' \in \Sigma_y^\omega$, and

  $$\ell\, r_0^{n_0}\, r_1^{n_1} \vdash_{\mathcal{R}} \ell''\, r_0^{n_0''}\, r_1^{n_1''}$$

  then $n_0' = n_0''$ if $X = E_0$, $n_1' = n_1''$ if $X = E_1$, an $\ell' = \ell''$ if $X = L$.

The first condition requires that the Σ_x-component of a word in $L_{\mathcal{R}}$ starts by exactly as many symbols $\sharp$ as there are processes on the directed path leading from the environment to process p_i, followed by an infinite sequence of encodings of configurations of $\mathcal{R}$ which are separated by symbol $\sharp$, where the first encoded configuration is the initial configuration. The second condition requires that immediately after the first occurrence of a letter $X \neq N$ in the Σ_y-component for the following two encodings of configurations in the

Σ_x-component the line number is updated correctly if $X = L$ or register r is updated correctly according to $\mathcal{R}$ if $X = E_r$, for $r \in \{0, 1\}$. Clearly, using ideas as in the proof of Theorem 4.4, one shows that $L_{\mathcal{R}} \in \mathsf{D1CL}_\omega$.

Using the language $L_{\mathcal{R}}$, we define the local specification of process p_i. Let L_{p_i} contain exactly those words $\alpha \in (\Sigma^{p_i})^\omega$ satisfying the following conditions for all $d, d' \in [\mathsf{dim}(\Sigma^p)]$

- if $\mathrm{Pr}_d(\alpha) \in \Sigma_x^\omega$ and $\mathrm{Pr}_{d'}(\alpha) \in \Sigma_x^\omega$ then $\mathrm{Pr}_d(\alpha) = \mathrm{Pr}_{d'}(\alpha)$, and
- if $\mathrm{Pr}_d(\alpha) \in \Sigma_y^\omega$ and $\mathrm{Pr}_{d'}(\alpha) \in \Sigma_x^\omega$ then $\mathrm{Pr}_d(\alpha)^\frown\mathrm{Pr}_{d'}(\alpha) \in L_{\mathcal{R}}$.

Clearly, $L_{p_i} \in \mathsf{D1CL}_\omega$. It remains to define the local specifications of all other system processes $p \in P_{\mathsf{sys}} \setminus \{p_{y_1}, \ldots, p_{y_l}, p_{x_1}, \ldots, p_{x_m}\}$ which we define as $L_p = (\Sigma^p)^\omega$.

Now, suppose $\mathcal{R}$ does not halt. Then, consider the following joint strategy σ_{sys}. Processes $p_{x_0}, \ldots, p_{x_m}$ simultaneously send the sequence $\sharp^l \sharp c_0 \sharp c_1 \sharp c_2 \cdots$ where $c_e \in \mathsf{Conf}$ and $c_e \vdash_{\mathcal{R}} c_{e+1}$, for all $e \in \mathbb{N}$, and $c_0 = \mathsf{Conf}_{\mathrm{in}}$, which corresponds to the encoding of the infinite run of $\mathcal{R}$, to the appropriate output channels. Processes $p_{y_1}, \ldots, p_{y_{l-1}}$ transmit the input sequence coming via c_{y_0} from the environment to process p_i. Obviously, by doing so, all local specifications of the system processes are fulfilled regardless of which sequence is produced by the environment, as the sequence produced by the processes $p_{x_0}, \ldots, p_{x_m}$ never violates the successor relation. Hence, σ_{sys} is a joint winning strategy for the system processes.

On the other hand, if $\mathcal{R}$ halts, then any joint strategy for the system processes can be spoiled by the environment, since any sequence of encodings of configurations produced by processes $p_{x_0}, \ldots, p_{x_m}$ which starts with $\mathsf{Conf}_{\mathrm{in}}$ eventually has to violate the successor relation $\vdash_{\mathcal{R}}$ as the halting configuration has no successor configuration. This violation can be claimed by the environment by sending an appropriate letter L, E_0 or E_1, however, since process p_j is not informed about the claim of the environment, it cannot react on this letter which implies that also none of the processes $p_{x_0}, \ldots, p_{x_m}$ can. Thus, there is a joint winning strategy for the system processes if and only if $\mathcal{R}$ does not halt.

Finally, notice that to account for the case where $C_x \cap C_y \neq \emptyset$ the proof can be adapted by extending the alphabets for the channels from $C_x \cap C_y$ such that the inputs produced by the environment are transmitted to process p_i as well as such that it is guaranteed that the sequences of encodings of configurations produced by p_i and by p_j coincide. This is easily established by defining $\Sigma_c = \Sigma_x \times \Sigma_y$, for $c \in C_x \cap C_y$ and appropriately adjusting the local specifications of the concerned processes. □

We conclude this subsection by pointing out two certain patterns such that the occurrence of at least one of this patterns in an architecture yields

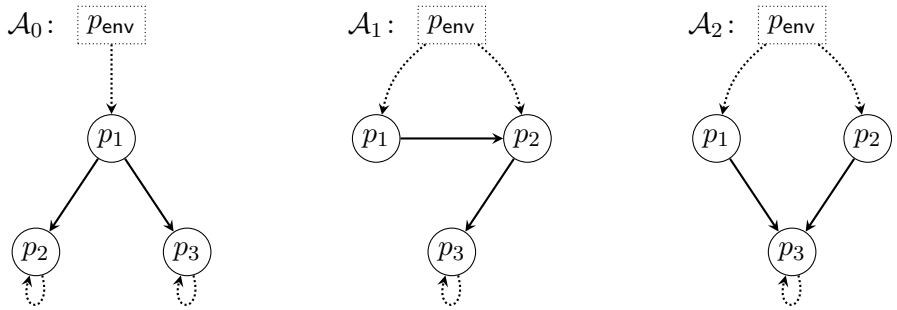


Figure 5.6: Undecidable architectures for local regular specifications

undecidability for this architecture even if all local specifications for the system processes are regular. Notice, that for architectures $\mathcal{A}_0$, $\mathcal{A}_1$ and $\mathcal{A}_2$ depicted in Figure 5.6 the realizability problem is shown to be undecidable for specifications given by a list of regular local specifications in [MT01] by adapting the reduction from the halting problem for deterministic Turing machines presented in [PR90]. Moreover, in [FS05] the idea of encryption is used where the environment is provided with the possibility to send encryption functions such that certain processes are requested to encrypt their output sequences according to the received encryption function. By doing so, those sequences become incomprehensible to processes which are not informed about the corresponding encryption function. By combining these ideas one can easily show that each of the architectures $\mathcal{A}_0$, $\mathcal{A}_1$ and $\mathcal{A}_2$ depicted in Figure 5.6 remains undecidable even if they are modified by replacing some external output channels by any internal communication channels.

The first pattern which yields undecidability of an architecture is the occurrence of some reachable process which sends information to two further not better informed processes. Clearly, if such a pattern is present in an architecture then undecidability directly follows from the undecidability of a corresponding modified version of $\mathcal{A}_0$.

Theorem 5.14. *Let $\mathcal{A} = (P, C, r)$ be an architecture with a labeling $(\Sigma_c)_{c\in C}$ and a list $\ell = (L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes. The realizability problem for $\mathcal{A}$ with $(\Sigma_c)_{c\in C}$ and $L(\ell)$ is undecidable if there are three system processes $p_i, p_j, p_k \in P_{\mathsf{reach}}$ with $i \neq j$, $i \neq k$, $j \neq k$ such that*

- *p_i sends information to processes p_j and p_k,*
- *p_j and p_k are not better informed than p_i, and*
- *$L_{p_m} \in \mathsf{REG}_\omega$, for all $m \in \{1, \ldots, n\}$.*

The second pattern is deduced from the undecidability of the modified versions of $\mathcal{A}_1$ and $\mathcal{A}_2$. Any architecture containing at least two reachable and incomparably informed system processes such that from both processes there is a directed path leading to another system process is undecidable.

Theorem 5.15. *Let $\mathcal{A} = (P, C, r)$ be an architecture with a labeling $(\Sigma_c)_{c \in C}$ and a list $\ell = (L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes. The realizability problem for $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$ and $L(\ell)$ is undecidable if there are three system processes $p_i, p_j, p_k \in P_{\mathsf{reach}}$ with $i \neq j$, $i \neq k$, $j \neq k$ such that*

- *p_i and p_j are incomparably informed,*
- *there are directed paths from p_i to p_k and from p_j to p_k, and*
- *$L_{p_m} \in \mathsf{REG}_\omega$, for all $m \in \{1, \ldots, n\}$.*

5.5 Characterization

Using the results shown in the previous section we now derive our main result of this chapter, a characterization of decidable architectures with specifications given by a list of regular and deterministic contextfree local specifications. First notice, that for any architecture $\mathcal{A}$, if the global specification is given by a list of local specifications, then the realizability problem is decidable if and only if it is decidable for every connected subarchitecture of $\mathcal{A}$. Hence, for unconnected architectures, the individual connected subarchitectures can be treated separately. For this reason, it suffices to consider connected architectures only. The characterization is formulated in the following theorem.

Theorem 5.16. *Let $\mathcal{A} = (P, C, r)$ be a connected architecture with some labeling $(\Sigma_c)_{c \in C}$ and a list $\ell = (L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes. The realizability problem for $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$ and $L(\ell)$ is decidable if and only if*

- *$\mathcal{A}$ is a pipeline with backward-channels and $L_p \in \mathsf{DCFL}_\omega$, for the worst informed system process p, and $L_{p'} \in \mathsf{REG}_\omega$, for all $p' \in P_{\mathsf{sys}} \setminus \{p\}$, or*
- *any connected subarchitecture $\mathcal{A}[S]$ of $\mathcal{A}$ with $S \subseteq P_{\mathsf{reach}}$ is*
  - *a pipeline with backward-channels, or*
  - *a two-flanked pipeline with backward-channels with $r_S(C_q^S) = q$ where q is the last process of $\mathcal{A}[S]$, or*
  - *a two-flanked pipeline with backward-channels if $|S| = 2$,*

and $L_p \in \mathsf{DCFL}_\omega$ *for some unreachable system process* $p \in P_{\mathsf{sys}} \setminus P_{\mathsf{reach}}$*, and* $L_{p'} \in \mathsf{REG}_\omega$*, for all* $p' \in P_{\mathsf{sys}} \setminus \{p\}$*.*

Proof. If the first condition is fulfilled, i.e., $\mathcal{A}$ is a pipeline with backward-channels and the local specification of the worst informed system process is deterministic contextfree, and all other system processes have regular local specifications, then, clearly, $\mathcal{A}$ is decidable by Theorem 5.7.

For the second condition, first consider the case where there are no unreachable system processes, i.e., $P_{\mathsf{sys}} = P_{\mathsf{reach}}$. Then, all local specifications are regular and decidability directly follows by Theorem 5.7, Theorem 5.10 and Theorem 5.11, since in this case $\mathcal{A}$ is either a pipeline with backward-channels or it is a two-flanked pipeline with backward-channels such that either there are no backward-channels from the last process or it contains only two system processes.

Now, assume that $\mathcal{A}$ also contains unreachable system processes. Let the unreachable process $\overline{p} \in P_{\mathsf{sys}} \setminus P_{\mathsf{reach}}$ be the only process having a deterministic contextfree local specification $L_{\overline{p}} \in \mathsf{DCFL}_\omega$. Moreover, let

$$\Sigma_U = \prod_{p \in P_{\mathsf{sys}} \setminus P_{\mathsf{reach}}} \Sigma^p_{\mathrm{out}}$$

denote the joint alphabet of the output channels of all unreachable processes. Furthermore, for any unreachable process $p \in P_{\mathsf{sys}} \setminus P_{\mathsf{reach}}$, define the language $L'_p = \{\alpha \in \Sigma_U^\omega \mid \mathrm{Pr}_{\Sigma^p}(\alpha) \in L_p\}$ and let

$$L_U = \bigcap_{p \in P_{\mathsf{sys}} \setminus P_{\mathsf{reach}}} L'_p.$$

Clearly, $L_U \in \mathsf{DCFL}_\omega$, since $L'_{\overline{p}}$ is deterministic contextfree and all other L'_p are regular, for $p \neq \overline{p}$.

Now, we show for the case where $P_{\mathsf{sys}} \setminus P_{\mathsf{reach}} \neq \emptyset$ that $\mathcal{A}$ is decidable if the second condition of the theorem is satisfied. A generic architecture for this case is sketched in Figure 5.7. Now, consider any connected subarchitecture $\mathcal{A}[S] = (S \cup \{p_{\mathsf{env}}\}, C[S], r_S)$ of $\mathcal{A}$ with $S \subseteq P_{\mathsf{reach}}$. Let

$$C_{U \to S} = \{c \in C_p \mid p \in P_{\mathsf{sys}} \setminus P_{\mathsf{reach}}\} \cap \{c \in C \mid r(c) \in S\},$$

i.e., $C_{U \to S}$ contains all channels via which information is sent from some unreachable process to a process from S in $\mathcal{A}$. The idea is to extend the subarchitecture $\mathcal{A}[S]$ for being able to simulate channels from $C_{U \to S}$. Therefor, we reroute the channels from $C_{U \to S}$ such that the respective recipients remain unchanged, however, now particular processes from S are writing to those channels instead of the original unreachable processes. Then, if a joint winning strategy exists for the system processes of the new architecture which produces output sequences along the new channels such that they

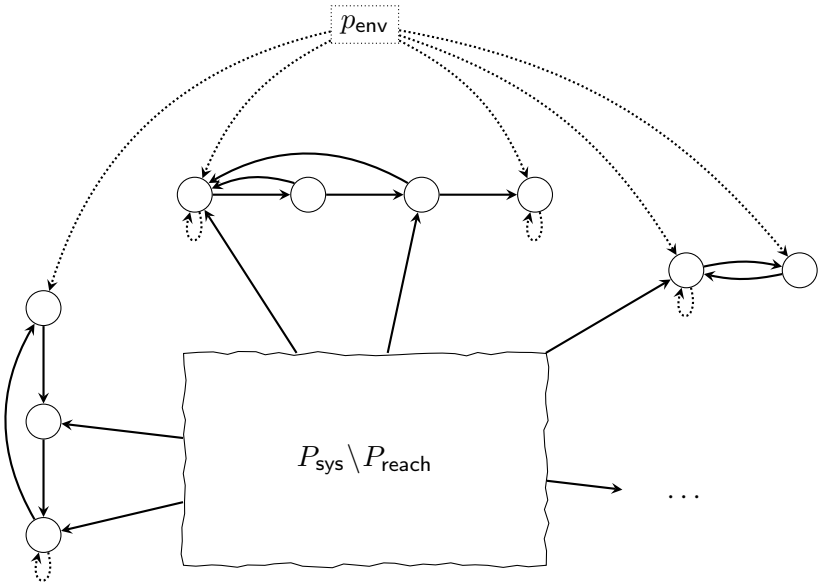


Figure 5.7: Generic decidable architecture

don't depend on the inputs coming from the environment, clearly, this could also be accomplished by the respective original unreachable processes in $\mathcal{A}$ along the corresponding original channels in $C_{U \to S}$.

We first consider the case where $\mathcal{A}[S] = (\{s_0, \ldots, s_m\}, C[S], r_S)$ is a two-flanked pipeline with backward-channels, where $m = |S|$ and $s_0 = p_{\mathsf{env}}$, such that there are no backward-channels from the last process s_m. Define the architecture $\mathcal{B}_S = (P^{\mathcal{B}}, C^{\mathcal{B}}, r^{\mathcal{B}})$ as follows,

- $P^{\mathcal{B}} = \{s_0, \ldots, s_m\}$,
- $C^{\mathcal{B}}_p = C^S_p$ for all $p \in \{s_0, \ldots, s_m\} \setminus \{s_{m-1}\}$,
- $C^{\mathcal{B}}_{s_{m-1}} = C^S_{s_{m-1}} \cup C_0 \cup C_1$ where $C_i = \{c^{(i)} \mid c \in C_{U \to S}\}$ for $i \in \{0, 1\}$,
- $r^{\mathcal{B}}(c) = r_S(c)$, for $c \in C^{\mathcal{B}} \setminus (C_0 \cup C_1)$,
- $r^{\mathcal{B}}(c^{(0)}) = r(c)$, for $c^{(0)} \in C_0$ and the corresponding channel $c \in C_{U \to S}$,
- $r^{\mathcal{B}}(c^{(1)}) = s_m$, for $c^{(1)} \in C_1$.

Thus, $\mathcal{B}_S$ evolves from $\mathcal{A}[S]$ by adding two copies C_0 and C_1 of channels $C_{U \to S}$ to the last but one process s_{m-1} of the two-flanked pipeline. Via channels from C_0 process s_{m-1} sends information to the respective recipients of the corresponding channels from $C_{U \to S}$, whereas channels from C_1 are all read by process s_m. Furthermore, let the labeling $(\widehat{\Sigma}_c)_{c \in C^{\mathcal{B}}}$ be defined by

$\widehat{\Sigma}_c = \Sigma_c$, for $c \in C^{\mathcal{B}} \setminus (C_0 \cup C_1)$, and for $i \in \{0,1\}$, let $\widehat{\Sigma}_{c^{(i)}} = \Sigma_c$ for every $c^{(i)} \in C_i$ where c is the corresponding channel in $C_{U \to S}$, i.e., the alphabets assigned to the channels from C_0 and C_1 are exactly the same as for the original channels from $C_{U \to S}$. To be able to distinguish between this two copies of channels C_0 and C_1 we denote the corresponding joint alphabets by

$$\widehat{\Sigma}_{C_i} = \prod_{c \in C_i} \widehat{\Sigma}_c, \text{ for } i \in \{0,1\}.$$

Now, we define the list $\widehat{\ell} = (\widehat{L}_{s_1}, \dots, \widehat{L}_{s_m})$ of local specification for the system processes of $\mathcal{B}_S$. For every $p \in \{s_1, \dots, s_{m-2}\}$ the local specifications remain as in ℓ, i.e., $\widehat{L}_p = L_p$. The local specification of the last process s_m is defined by $\widehat{L}_{s_m} = \{\alpha \in (\widehat{\Sigma}^{s_m})^\omega \mid \mathrm{Pr}_{\Sigma^{s_m}}(\alpha) \in L_{s_m}\}$, i.e., there are no conditions imposed on the information sent along the new channels. Finally, let the local specification of process s_{m-1} be defined by $\widehat{L}_{s_{m-1}} = \{\alpha \in (\widehat{\Sigma}^{s_{m-1}})^\omega \mid \mathrm{Pr}_{\widehat{\Sigma}_{C_0}}(\alpha) = \mathrm{Pr}_{\widehat{\Sigma}_{C_1}}(\alpha) \text{ and } \mathrm{Pr}_{\Sigma^{s_{m-1}}}(\alpha) \in L_{s_{m-1}}\}$ Hence, for process s_{m-1} we additionally require that the sequences produced on channels from C_0 coincide with the corresponding sequences sent along the channels from C_1.

Notice that the architecture $\mathcal{B}_S$ is a pipeline with backward-channels which has no backward-channels from the last process, i.e., $r^{\mathcal{B}}(C^{\mathcal{B}}_{s_m}) = s_m$. Moreover, as all local specifications in $\widehat{\ell}$ are regular, by Theorem 5.10 the realizability problem for $\mathcal{B}_S$ with $(\widehat{\Sigma}_c)_{c \in C^{\mathcal{B}}}$ and $L(\widehat{\ell})$ can be solved. Let $\mathcal{N}_{m-1}$ be the parity-NTA over $\mathbb{B}$-labeled $\widehat{\Sigma}^{s_{m-1}}_{\mathrm{out}}$-trees according to Lemma 5.5 and let $\mathcal{N}'_{m-1}$ be a corresponding parity-NTA over $\mathbb{B}$-labeled $\widehat{\Sigma}_{m-1}$-trees which accepts a tree t if and only if $\mathsf{wide}_Y(t) \in L(\mathcal{N}_{m-1})$ where $Y = \widehat{\Sigma}^{\mathsf{b},s_{m-1}}_{\mathrm{out}}$. Moreover, let $\mathcal{N}_m$ be the parity-NTA over $\mathbb{B}$-labeled $\widehat{\Sigma}_{m-1}$-trees according to Lemma 5.9. Furthermore, let $\mathcal{N}$ be a parity-NTA over $\mathbb{B}$-labeled $\widehat{\Sigma}_{m-1}$-trees such that $L(\mathcal{N}) = L(\mathcal{N}'_{m-1}) \cap L(\mathcal{N}_m)$. Clearly, if $L(\mathcal{N}) = \emptyset$, then $L(\ell)$ is not realizable in $\mathcal{A}$ with $(\Sigma_c)_{c \in C}$. So, assume $L(\mathcal{N}) \neq \emptyset$.

Now, we construct an alternating finite automaton $\mathcal{C}_S$ over $\widehat{\Sigma}_{C_1}$ which accepts a word $\nu \in (\widehat{\Sigma}_{C_1})^\omega$ if and only if there is a tree $t \in \mathbb{T}_{\mathsf{com}}(\widehat{\Sigma}_{m-1})$ such that $\mathrm{Pr}_{\widehat{\Sigma}_{C_1}}(L^{(\omega)}(t)) = \{\nu\}$ and $t \in L(\mathcal{N})$, i.e., there is a joint winning strategy for the system processes such that process s_{m-1} produces the sequence ν along the channels from C_1 (and consequently also along the channels from C_0 due to $\widehat{L}_{s_{m-1}}$) for any external input sequence produced by the environment. The idea for the construction is the following. While processing a word $\nu \in (\widehat{\Sigma}_{C_1})^\omega$ the automaton successively guesses a tree $t \in \mathbb{T}_{\mathsf{com}}(\widehat{\Sigma}_{m-1})$ which satisfies $\mathrm{Pr}_{\widehat{\Sigma}_{C_1}}(L^{(\omega)}(t)) = \{\nu\}$. Moreover, the automaton verifies that t is accepted by $\mathcal{N}$ by furthermore guessing an accepting run of $\mathcal{N}$ on t. Formally, define the parity-AFA $\mathcal{C}_S = (Q^{\mathcal{C}}, \widehat{\Sigma}_{C_1}, \delta^{\mathcal{C}}, q^{\mathcal{C}}_{\mathrm{in}}, \mathrm{col}^{\mathcal{C}})$ as follows.

- $Q^{\mathcal{C}} = Q^{\mathcal{N}} \times \mathbb{B}$
- $q_{\text{in}}^{\mathcal{C}} = (q_{\text{in}}^{\mathcal{N}}, \top)$
- $\text{col}^{\mathcal{C}}(q, \zeta) = \text{col}^{\mathcal{N}}(q)$ for all $q \in Q^{\mathcal{N}}$ and all $\zeta \in \mathbb{B}$
- for $q \in Q^{\mathcal{N}}$ and $b \in \widehat{\Sigma}_{C_1}$

$$\delta^{\mathcal{C}}\big((q, \top), b\big) = \bigvee_{[\emptyset \neq X \subseteq \widehat{\Sigma}_{m-1}]} \bigvee_{[\varphi \in \delta^{\mathcal{N}}(q, \top)]} \bigwedge_{[a \in \widehat{\Sigma}_{m-1}]} q^{(X, \varphi, a, b)}$$

$$\text{where } q^{(X,\varphi,a,b)} = \begin{cases} (p, \top) & \text{if } a \in X \text{ and } \Pr_{\Sigma_{C_1}}(a) = b \\ & \text{and} (\downarrow_a, p) \in \varphi, \\ (p, \bot) & \text{if } \big(a \notin X \text{ or } \Pr_{\Sigma_{C_1}}(a) \neq b\big) \\ & \text{and} (\downarrow_a, p) \in \varphi, \end{cases}$$

$$\delta^{\mathcal{C}}\big((q, \bot), b\big) = \bigvee_{[\varphi \in \delta^{\mathcal{N}}(q, \bot)]} \bigwedge_{[a \in \widehat{\Sigma}_{m-1}]} q^{(\varphi, a)}$$

$$\text{where } q^{(\varphi,a)} = (p, \bot) \text{ if } (\downarrow_a, p) \in \varphi.$$

Clearly, if $L(\mathcal{C}_S) \neq \emptyset$ then there is a strategy σ_{sys} for the system processes P_{sys} such that any global system behavior $\alpha_{\mathcal{A}} \in (\Sigma^{\mathcal{A}})^{\omega}$ consistent with σ_{sys} fulfills all local specifications of the processes in S. For this, let $\nu \in L(\mathcal{C}_S)$. Then, there is a tree $t \in L(\mathcal{N})$ such that $\Pr_{\widehat{\Sigma}_{C_1}}(L^{(\omega)}(t)) = \{\nu\}$ which means that there is a joint winning strategy $(\widehat{\sigma}_{s_1}, \ldots, \widehat{\sigma}_{s_m})$ for the system processes in $\mathcal{B}_S$ such that the language generated by

$$\widehat{\sigma}_{s_1}^{(*)} \circ \Pr_{\widehat{\Sigma}_1} \circ \ldots \circ \widehat{\sigma}_{s_{m-2}}^{(*)} \circ \Pr_{\widehat{\Sigma}_{m-2}} \circ \widehat{\sigma}_{s_{m-1}} \circ \Pr_{\widehat{\Sigma}_{m-1}}$$

over $\widehat{\Sigma}_{p_{\mathsf{env}}}^{\omega}$ is a subset of $L^{(\omega)}(t)$. Hence, we can define $\sigma_{\mathsf{sys}} = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ any global system behavior which is consistent with σ_{sys} satisfies all local specifications L_p of the processes $p \in S$ as follows. For $p \in \{s_1, \ldots, s_{m-2}\}$, define $\sigma_p = \widehat{\sigma}_p$. For processes s_{m-1} and s_m, we define corresponding strategies by $\sigma_{s_{m-1}} = \widehat{\sigma}_{s_{m-1}} \circ \Pr_{\Sigma_{\text{out}}^{s_{m-1}}}$ and $\sigma_{s_m}(u) = \widehat{\sigma}_{s_m}(u^{\frown} \mathsf{pref}_{|u|}(\nu))$ for all $u \in \Sigma_{\text{in}}^{s_m}$. Furthermore, for every unreachable process $p \in P_{\mathsf{sys}} \setminus P_{\mathsf{reach}}$, define σ_p such that it produces a sequence $\alpha_p \in (\Sigma_{\text{out}}^{p})$ which satisfies $\Pr_{\Sigma_c}(\alpha_p) = \Pr_{\Sigma_c}(\nu)$ for every $c \in C_p \cap C_{U \to S}$.

Now, let the subarchitecture $\mathcal{A}[S] = (\{s_0, \ldots, s_m\}, C[S], r_S)$ be a pipeline with backward-channels. This case can be reduced to the previous case

of two-flanked pipelines with backward-channels which have no backward-channels from the last process. For this, we first augment $\mathcal{A}[S]$ by an auxiliary process s_{m+1}, since the last process s_m of $\mathcal{A}[S]$ may have backward-channels. By doing so, we obtain a two-flanked pipelines with backward-channels which have no backward-channels from the last process which is the now process s_{m+1}. Then, this architecture is handled exactly as above.

Finally, we consider the case where $\mathcal{A}[S] = (\{s_0, s_1, s_2\}, C[S], r_S)$ is a two-flanked pipeline with backward-channels containing exactly two system processes s_1 and s_2. For this case, we define the architecture $\mathcal{B}_S = (P^{\mathcal{B}}, C^{\mathcal{B}}, r^{\mathcal{B}})$ as follows.

- $P^{\mathcal{B}} = \{s_0, s_1, s_2\}$,
- $C^{\mathcal{B}}_{s_i} = C^S_{s_i} \cup C_i$ where $C_i = \{c^{(0)} \mid c \in C_{U \to S}, r(c) \neq s_i\}$, for $i \in \{1, 2\}$,
- $r^{\mathcal{B}}(c) = r_S(c)$, for $c \in C^{\mathcal{B}} \setminus (C_1 \cup C_2)$,
- $r^{\mathcal{B}}(C_1) = s_2$ and $r^{\mathcal{B}}(C_2) = s_1$.

Thus, in this case, $\mathcal{B}_S$ is obtained from $\mathcal{A}[S]$ by appending a copy

$$C_0 = C_1 \cup C_2$$

of channels from $C_{U \to S}$ to the subarchitecture. Now, each process take the role of the unreachable process sending information to the other process, i.e., channels from $C_{U \to S}$ which are read by process s_1 are simulated by process s_2 by sends information via C_2 and, vice versa, channels from $C_{U \to S}$ which are read by process s_2 are simulated by process s_1 by sends information via C_1. We define the labeling $(\widehat{\Sigma}_c)_{c \in C^{\mathcal{B}}}$ as above. Moreover, the local specifications are adapted accordingly, $\widehat{L}_{s_i} = \{\alpha \in (\widehat{\Sigma}^{s_i})^\omega \mid \Pr_{\Sigma^{s_i}}(\alpha) \in L_{s_i}\}$, for $i \in \{1, 2\}$. Notice that since $\mathcal{B}_S$ is a two-flanked pipeline containing only two system processes and as both local specifications $\widehat{L}_{s_i}$, for $i \in \{1, 2\}$, are regular, the realizability problem can be solved by Theorem 5.11. So, let now $\mathcal{N}$ be a parity-NTA over $(\mathbb{B} \times \mathbb{B})$-labeled $(\widehat{\Sigma}_1 \times \widehat{\Sigma}_2)$-trees as in Theorem 5.11 where

$$\widehat{\Sigma}_i = \prod_{c \in C^{\mathcal{B}}_i \setminus H^{\mathcal{B}}_i} \widehat{\Sigma}_c, \text{ for } i \in \{1, 2\}.$$

Analogously, as for two-flanked pipelines with backward-channels with no backward-channels from the last process, one can construct a parity-AFA $\mathcal{C}_S$ which now accepts a word $\nu \in (\widehat{\Sigma}_{C_0})^\omega$ if and only if there is a tree $t \in \mathbb{T}^{\mathsf{ext}}_{\mathsf{com}}(\widehat{\Sigma}_1 \times \widehat{\Sigma}_2)$ such that $\Pr_{\widehat{\Sigma}_{C_0}}(L^{(\omega)}(t)) = \{\nu\}$ and $t \in L(\mathcal{N})$. Formally, parity-AFA $\mathcal{C}_S = (Q^{\mathcal{C}}, \widehat{\Sigma}_{C_1}, \delta^{\mathcal{C}}, q^{\mathcal{C}}_{\text{in}}, \text{col}^{\mathcal{C}})$ is defined as follows.

- $Q^{\mathcal{C}} = Q^{\mathcal{N}} \times (\mathbb{B} \times \mathbb{B})$

- $q_{\text{in}}^{\mathcal{C}} = (q_{\text{in}}^{\mathcal{N}}, (\top, \top))$
- $\text{col}^{\mathcal{C}}(q, \zeta) = \text{col}^{\mathcal{N}}(q)$ for all $q \in Q^{\mathcal{N}}$ and all $\zeta \in \mathbb{B} \times \mathbb{B}$
- for $q \in Q^{\mathcal{N}}$ and $b \in \widehat{\Sigma}_{C_0}$ and $\zeta \in (\mathbb{B} \times \mathbb{B}) \setminus \{(\top, \top)\}$

$$\delta^{\mathcal{C}}\big((q, (\top, \top)), b\big) = \bigvee_{[\emptyset \neq X \subseteq \widehat{\Sigma}_1 \times \widehat{\Sigma}_2]} \bigvee_{[\varphi \in \delta^{\mathcal{N}}(q, (\top, \top))]} \bigwedge_{[a \in \widehat{\Sigma}_1 \times \widehat{\Sigma}_2]} q^{(X, \varphi, a, b)}$$

$$\text{where } q^{(X,\varphi,a,b)} = \begin{cases} (p, (\top, \top)) & \text{if } a \in X \text{ and } \text{Pr}_{\Sigma_{C_0}}(a) = b \\ & \text{and} (\downarrow_a, p) \in \varphi, \\ \bigvee\limits_{\substack{\zeta' \in \mathbb{B} \times \mathbb{B}, \\ \zeta' \neq (\top, \top)}} (p, \zeta') & \text{if } \big(a \notin X \text{ or } \text{Pr}_{\Sigma_{C_0}}(a) \neq b\big) \\ & \text{and} (\downarrow_a, p) \in \varphi, \end{cases}$$

$$\delta^{\mathcal{C}}\big((q, \zeta), b\big) = \bigvee_{[\varphi \in \delta^{\mathcal{N}}(q, \zeta)]} \bigwedge_{[a \in \widehat{\Sigma}_1 \times \widehat{\Sigma}_2]} q^{(\varphi, a)}$$

$$\text{where } q^{(\varphi,a)} = \bigvee_{\substack{\zeta' \in \mathbb{B} \times \mathbb{B}, \\ \zeta' \neq (\top, \top)}} (p, \zeta') \quad \text{if } (\downarrow_a, p) \in \varphi.$$

Clearly, if $\nu \in L(\mathcal{C}_S)$ then there is a joint winning strategy $(\widehat{\sigma}_{s_1}, \widehat{\sigma}_{s_2})$ for processes s_1 and s_2 such that any global system behavior of $\mathcal{B}_S$ which is consistent with this strategy fulfills both local specifications $\widehat{L}_{s_1}$ and $\widehat{L}_{s_2}$. Furthermore, since the sequence ν produced along the channels from $C_0 = C_1 \cup C_2$ by this strategy is completely independent of the external input produced by the environment, we can define a joint strategy σ_{sys} such that any global system behavior of $\mathcal{A}$ which is consistent with σ_{sys} fulfills L_{s_1} and L_{s_2}. For this, define $\sigma_{s_i} = \widehat{\sigma}_{s_i} \circ \text{Pr}_{\Sigma_{\text{out}}^{s_i}}$, for $i \in \{1, 2\}$, and for the unreachable processes p sending information to s_1 or s_2, the corresponding strategies σ_p are required to produce a sequence $\alpha_p \in (\Sigma_{\text{out}}^p)$ which satisfies $\text{Pr}_{\Sigma_c}(\alpha_p) = \text{Pr}_{\Sigma_c}(\nu)$ for every $c \in C_p \cap C_{U \to S}$.

Finally, for any connected subarchitecture $\mathcal{A}[S]$ with $S \subseteq P_{\mathsf{reach}}$ (which is either a pipeline with backward-channels, or a two-flanked pipeline with backward-channels which has no backward-channels from the last process or contains just two system processes), let

$$L'(\mathcal{C}_S) = \{\alpha \in \Sigma_U^\omega \mid \text{Pr}_{\Sigma_{U \to S}}(\alpha) \in L(\mathcal{C}_S)\}$$

where $\Sigma_{U \to S} = \prod_{c \in C_{U \to S}} \Sigma_c$. Clearly, there is a joint winning strategy σ_{sys} for the system processes in $\mathcal{A}$ if and only if

$$L \;=\; L_U \;\cap \bigcap_{\substack{S \subseteq P_{\mathsf{reach}}, \\ \mathcal{A}[S] \text{ is} \\ \text{connected}}} L'(\mathcal{C}_S) \;\neq\; \emptyset$$

which can be decided, since L is deterministic contextfree.

For the other direction, notice that, if both conditions of the theorem are not fulfilled, then either there are at least two system processes $p \neq p'$ with deterministic contextfree local specifications $L_p, L_{p'}$, or P_{sys} contains a reachable process p with a deterministic contextfree local specification L_p and also some not better informed process $p' \neq p$, or there are three reachable distinct system processes p, p' and p'' such that p' and p'' are incomparably informed and either p sends information to both, p' and p'', or there are directed paths from p' to p as well as from p'' to p. Hence, in this case, undecidability of $\mathcal{A}$ follows by Theorem 5.12, Theorem 5.13, Theorem 5.14 and Theorem 5.15. □

5.6 Summary of Results

We investigated the realizability problem for distributed systems which is to decide, given a distributed system represented by a labeled architecture and a global system specification, whether there is a joint winning strategy for the system processes. In particular, we considered regular and deterministic contextfree specifications.

First, we showed that in case of deterministic contextfree global system specifications the realizability problem becomes undecidable even for very simple architectures, namely, for architectures with at least two system processes or architectures containing hidden channels from the environment process, i.e., channels which are not read by any system process. So basically, for deterministic contextfree global system specifications the realizability problem turns out to be decidable only for nondistributed settings. Furthermore, we mentioned that this result holds even for the restricted class of deterministic contextfree global specifications recognizable by deterministic visibly one-counter with weak acceptance condition.

Then, we concentrated on system specifications given by a list of local system specifications, one for each system process. We extended the result of [MT01], where the class of system specifications given by a list of regular local system specifications, however, only for the case of acyclic architectures is studied. Our main result is a complete characterization of decidable architectures for general architectures where cycles are allowed and for the class of system specifications given by a list of local system specifications which may be regular or deterministic contextfree.

Chapter 6

Finite-Time Pushdown Games

In infinite games with strong winning conditions such as parity or Muller winning conditions, the winner of a play depends on events which happen infinitely often during the play and hence can be determined only after infinitely many rounds. In contrast, finite-duration variants of infinite-duration games are provided with a criterion which stops a play after a finite number of rounds and declares a winner based on the finite play prefix. This criterion is sound if, for $i \in \{0, 1\}$, Player i wins the infinite-duration game if and only if Player i wins the corresponding finite-duration game. Clearly, as a finite-duration game has a reachability objective, a sound termination criterion which stops a play after at most n rounds yields an algorithm to determine the winner of the corresponding infinite-duration game just by building the attractor on a finite tree of depth at most n.

McNaughton introduced a finite-duration variant for Muller games played on finite game graphs [McN00]. He defined the termination criterion by means of so-called scoring functions which rate finite play prefixes. A play is stopped when some scoring function reaches a given threshold score value. McNaughton proved soundness of this criterion for a factorial threshold score value, i.e., he showed equivalence between Muller games and corresponding finite-time Muller games with factorial threshold score. This implies that the winner of a Muller game on a finite game graph can be determined by solving a reachability game over a game graph which is doubly-exponential in the size of the game graph of the original Muller game. This result was improved by Fearnley and Zimmermann who showed that the constant threshold score value of three suffices for the equivalence of the corresponding games [FZ10]. Furthermore, a score-based reduction from Muller games to safety games evolved from this result [Zim12, NRZ12] which yields general non-deterministic winning strategies called permissive strategies. This extends the work of Bernet et al. [BJW02] on permissive strategies for parity games to Muller games.

In this chapter we extend these results to parity pushdown games. First, we introduce a new finite-duration variant for parity pushdown games. This is necessary since the known results on finite game graphs do not hold for infinite ones. By exploiting the intrinsic structure of pushdown graphs we define stair-scoring functions and prove equivalence between parity pushdown games and their corresponding finite-duration variants with a threshold stair-score which is exponential in the size of the underlying pushdown system. This yields a new reduction method which determines the winner of a parity pushdown game by solving a reachability game on a finite tree. Moreover, we establish an almost matching lower bound on the threshold stair-score such that the equivalence between the corresponding games still holds, which is exponential in the cube root of the size of the underlying pushdown system. We introduce finite-time games formally in Section 6.1 where we give the new notion of stair-scoring functions which we use to define finite-time parity pushdown games. Then, in Section 6.2 we recall and adapt Walukiewicz's construction as it is needed for the following section. The equivalence between parity pushdown games and their corresponding finite-duration variants is proved in Section 6.3. Finally, we present the lower bound in Section 6.4.

6.1 Finite-Time Games

In this section we introduce a finite-duration variant of parity pushdown games. For this, the concept of scoring functions, which were originally introduced in [McN00] for Muller games, is adapted to parity games. Scoring functions are used to rate finite play prefixes. In Muller games, for every set of vertices F, the score of a finite play prefix w with respect to F is defined as the number of times the entire set F is visited in the longest suffix of w consisting merely of vertices in F. This is straightforwardly transfered to the case of parity games.

In the following, let (G, col) be a parity game with $G = (V, V_0, V_1, E, v_{\mathrm{in}})$ and $\mathrm{col}\colon V \to [n]$, for some $n \in \mathbb{N}$. We define scoring functions for parity games as follows. For every color $c \in [n]$, define the scoring function $\mathsf{Sc}_c\colon V^* \to \mathbb{N}$ assigning to any finite sequence of vertices a natural number by

$$\mathsf{Sc}_c(\varepsilon) = 0$$

and for $w \in V^*$ and $v \in V$,

$$\mathsf{Sc}_c(wv) = \begin{cases} \mathsf{Sc}_c(w) & \text{if } \mathrm{col}(v) > c, \\ \mathsf{Sc}_c(w) + 1 & \text{if } \mathrm{col}(v) = c, \\ 0 & \text{if } \mathrm{col}(v) < c. \end{cases}$$

Furthermore, the maximum score function $\mathsf{MaxSc}_c\colon V^* \cup V^\omega \to \mathbb{N} \cup \{\infty\}$, for every $c \in [n]$, is defined by

$$\mathsf{MaxSc}_c(\rho) = \sup_{w \sqsubseteq \rho} \mathsf{Sc}_c(w),$$

for $\rho \in V^* \cup V^\omega$, i.e., the maximum score function assigns to any finite or infinite sequence ρ of vertices the highest value which is reached by the corresponding scoring function on finite prefixes $w \sqsubseteq \rho$, or ∞ is assigned if this value is unbounded.

Now, consider the case where the set of vertices V of the game graph is finite and let σ be a positional winning strategy for Player i in the parity game. Notice that, for every play $\rho \in V^\omega$, if ρ is consistent with σ, then there are no two positions $j < j' \in \mathbb{N}$ with $\rho(j) = \rho(j')$ such that

$$\mathrm{Par}(\mathrm{col}(\rho(j))) = 1 - i \text{ and } \mathrm{col}(\rho(k)) \geq \mathrm{col}(\rho(j)), \text{ for all } j \leq k \leq j',$$

i.e., any play consistent with a positional winning strategy for Player i does not visit a vertex $v \in V$ with $\mathrm{Par}(\mathrm{col}(v)) = 1 - i$ twice without visiting some vertex of strictly smaller color in between, since otherwise, the play

$$\overline{\rho} = \mathsf{pref}_j(\rho)\big(\rho(j) \cdots \rho(j'-1)\big)^\omega$$

is consistent with σ, however, as $\min\{\mathrm{Inf}(\mathrm{col}(\overline{\rho}))\} = 1 - i$ it is not winning. Hence, applying the pigeonhole principle, it follows that positional winning strategies in parity games on finite graphs bound the scores of the losing player. For $c \in [n]$, let

$$|V|_c = |\{v \in V \mid \mathrm{col}(v) = c\}|,$$

i.e., $|V|_c$ denotes the number of vertices colored by c.

Remark 6.1. Let $\mathcal{G} = (G, \mathrm{col})$ be a parity game with $G = (V, V_0, V_1, E, v_{\mathrm{in}})$ and $\mathrm{col}\colon V \to [n]$, for $n \in \mathbb{N}$, where the set V is finite. If σ is a positional winning strategy for Player i in $\mathcal{G}$, then, for every play ρ that is consistent with σ and every $c \in [n]$ with $\mathrm{Par}(c) = 1 - i$, we have $\mathsf{MaxSc}_c(\rho) \leq |V|_c$.

Due to this remark, for parity games on finite game graphs, if both players play according to respective positional strategies, then a play can be stopped as soon as a certain threshold value $|V|_c + 1$ is reached by some scoring function Sc_c, for $c \in [n]$, and Player $\mathrm{Par}(c)$ can be declared to be the winner of the play. This is the idea behind finite-time versions of infinite games.

Formally, a finite-time parity game $\mathcal{G} = (G, \mathrm{col}, k)$ consists of a game graph $G = (V, V_0, V_1, E, v_{\mathrm{in}})$, a parity condition $\mathrm{col}\colon V \to [n]$, for $n \in \mathbb{N}$, and a threshold value $k \in \mathbb{N} \setminus \{0\}$. A play in $\mathcal{G}$ is a finite path

$$w = w(0) \cdots w(r) \in V^*$$

with $w(0) = v_{\text{in}}$ and $(w(i), w(i+1)) \in E$, for all $i \in [r]$, such that the following conditions are satisfied,

- $\mathsf{MaxSc}_c(w) = k$ for some $c \in [n]$, and
- $\mathsf{MaxSc}_c(w(0) \cdots w(r-1)) < k$, for all $c \in [n]$.

The play w is winning for Player i if $\mathrm{Par}(c) = i$. A strategy for Player i is a function $\sigma\colon V^*V_i \to V$ such that for every sequence $u \in V^*V_i$ of vertices, we have $(\mathsf{last}(u), \sigma(u)) \in E$. A play $w \in V^*$ is consistent with a strategy σ for Player i if $w(j+1) = \sigma(\mathsf{pref}_{j+1}(w))$ for every $j \in [|w|-1]$ with $w(j) \in V_i$. A strategy σ is winning for Player i if every play w which is consistent with σ is winning for Player i. Player i wins the finite-time parity game $\mathcal{G}$ if there exists a winning strategy for Player i in $\mathcal{G}$.

First notice, that every threshold value $k \in \mathbb{N}$ is eventually reached by some scoring function, if the path in the game graph is sufficiently long. Thus, there are no draws due to infinite plays.

Lemma 6.2. *Let V be a set of vertices and $\mathrm{col}\colon V \to [n]$ a coloring function, for $n \in \mathbb{N}$. For every $w \in V^*$ and every $k \in \mathbb{N}$, if $|w| \geq k^n$, then there is some $c \in [n]$ such that $\mathsf{MaxSc}_c(w) \geq k$.*

Proof. We show the lemma by induction on the number n of colors. Clearly, the statement holds trivially for the base case $n = 1$. Hence, for the induction step, let $n > 1$. Consider $w \in V^*$ with $|w| \geq k^n$, for $k \in \mathbb{N}$. If there is an infix u of w with $|u| = k^{n-1}$ such that $|\mathrm{Occ}(\mathrm{col}(u))| \leq n-1$, i.e., the vertices occurring in u are colored by at most $n-1$ distinct colors, then by applying the induction hypothesis, there is some color $c \in [n]$ such that $\mathsf{MaxSc}_c(u) \geq k$ and, hence, also $\mathsf{MaxSc}_c(w) \geq k$. Otherwise, if every infix u of w of length $|u| = k^{n-1}$ contains at least one vertex colored by c, for every color $c \in [n]$, then, the smallest color 0 occurs at least k times in $\mathsf{pref}_{k^n}(w) = w_1 \cdots w_k$ with $|w_i| = k^{n-1}$, for $1 \leq i \leq k$, and, hence, $\mathsf{MaxSc}_c(w) \geq k$. □

Due to this lemma, a play in a finite-time parity game (G, col, k) continues for at most k^n rounds where $[n]$ is the codomain of the coloring function col. Moreover, it can also be shown that the bound in Lemma 6.2 is tight, i.e., there is a game graph $G = (V, V_0, V_1, E, v_{\text{in}})$ such that for every k, there is a path $w \in V^*$ with $|w| = k^n - 1$ such that $\mathsf{MaxSc}_c(w) < k$, for all colors $c \in [n]$ (cf. Lemma 4.9 in [Zim12]).

Furthermore, the following simple consequence of the definition of the scoring functions guarantees that every play in a finite-time parity game has a unique winner. By definition of the scoring functions, only the value of one function $\mathsf{Sc}_{\mathrm{col}(w(i))}$ is increased in round i of a play w, for all other functions Sc_c with $c \neq \mathrm{col}(w(i))$ the value either remains constant or is reseted to zero.

Lemma 6.3. *Let V be a set of vertices and* $\mathrm{col}\colon V \to [n]$ *a coloring function, for $n \in \mathbb{N}$. For $w \in V^*$, $v \in V$ and $c, c' \in [n]$, if $\mathsf{Sc}_c(wv) = \mathsf{Sc}_c(w) + 1$ and $\mathsf{Sc}_{c'}(wv) = \mathsf{Sc}_{c'}(w) + 1$, then $c = c'$.*

Hence, a play in a finite-time parity game (G, col, k) continues until one of the scoring functions Sc_c is increased to the threshold value k. As soon as this happens, a unique winner of the play can be declared.

In [FZ10], finite-time Muller games are studied where the original scoring functions for Muller games are used. For the case of finite game graphs, the equivalence between Muller games and corresponding finite-time Muller games is established for the constant threshold $k = 3$. This means that Player i wins a Muller game if and only if Player i wins the corresponding finite-time Muller game with threshold $k = 3$. Due to Remark 6.1 an analogous result holds for parity games on finite game graphs.

Theorem 6.4. *Let $G = (V, V_0, V_1, E, v_{\mathrm{in}})$ be a game graph where V is finite and* $\mathrm{col}\colon V \to [n]$ *be a coloring function, for $n \in \mathbb{N}$. For every threshold value $k > \max_{c \in [n]} |V|_c$, Player i wins (G, col) if and only if Player i wins (G, col, k).*

However, this result does not hold for infinite game graphs. To verify this, consider the infinite pushdown game graph $G(\mathcal{S})$ induced by the pushdown system $\mathcal{S} = (Q, \Gamma, \Delta, q_{\mathrm{in}})$ where $Q = \{q_{\mathrm{in}}, q_1, q_2\}$, $\Gamma = \{A\}$ and Δ contains the following transitions, for $X \in \{A, \bot\}$,

- $(q_{\mathrm{in}}, X, q_{\mathrm{in}}, AX)$, $(q_{\mathrm{in}}, X, q_1, AX)$,
- $(q_1, A, q_1, \varepsilon)$, $(q_1, \bot, q_2, \bot)$,
- $(q_2, A, q_2, \varepsilon)$, $(q_2, \bot, q_2, \bot)$,

with the partition $Q_0 = \{q_1, q_2\}$ and $Q_1 = \{q_{\mathrm{in}}\}$. The part of $G(\mathcal{S})$ which is reachable from the initial vertex $v_{\mathrm{in}} = (q_{\mathrm{in}}, \bot)$ is depicted in Figure 6.1. Furthermore, let the coloring function $\mathrm{col}\colon Q \to [2]$ be defined by $\mathrm{col}(q_1) = 1$ and $\mathrm{col}(q_{\mathrm{in}}) = \mathrm{col}(q_2) = 0$. Clearly, the parity game $(G(\mathcal{S}), \mathrm{col})$ is won by Player O, since for every play $\rho \in V^\omega$,

$$\mathrm{Pr}_Q(\rho) = q_{\mathrm{in}}^\omega \text{ or } \mathrm{Pr}_Q(\rho) = q_{\mathrm{in}}^m q_1^{m+1} q_2^\omega, \text{ for } m > 0,$$

which means that configurations in state q_1 are seen only finitely many times during the play ρ and hence $\mathrm{Inf}(\mathrm{col}(\rho)) = 0$. However, for every threshold value $k > 1$, Player I wins the corresponding finite-time game $(G(\mathcal{S}), \mathrm{col}, k)$. The positional winning strategy σ_I for Player I is to move the token to configuration $(q_1, A^{k-1}\bot)$, i.e., $\sigma_I((q_{\mathrm{in}}, A^m\bot)) = (q_1, A^{m+1}\bot)$ if $m = k - 2$ and $\sigma_I((q_{\mathrm{in}}, A^m\bot)) = (q_{\mathrm{in}}, A^{m+1}\bot)$, otherwise. Following this strategy,

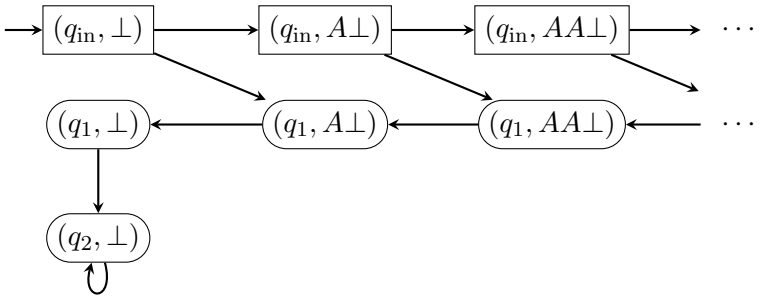


Figure 6.1: The pushdown game graph induced by $\mathcal{S}$

Player I wins since color 1 is the first to reach score k which happens when the token arrives at the configuration $(q_1, \bot)$.

Thus, to obtain an analogous result as in Theorem 6.4 for pushdown parity games, we have to introduce another version of finite-time parity games adjusted for pushdown parity games. For this, we adapt the scoring functions using the concept of stairs as defined in Definition 3.1, which we call stair-score functions.

Let (G, col) be a pushdown parity game with a pushdown game graph $G = (V, V_0, V_1, E, v_{\text{in}})$ and a coloring function $\text{col}\colon V \to [n]$, for $n \in \mathbb{N}$. To simplify our notation, we define the following functions

$$\begin{aligned}&\mathsf{MinCol}\colon V^+ \to [n],\\ &\mathsf{reset}\colon V^+ \to V^* \text{ and}\\ &\mathsf{lastBump}\colon V^+ \to V^+\end{aligned}$$

as follows. For $w \in V^*$, let

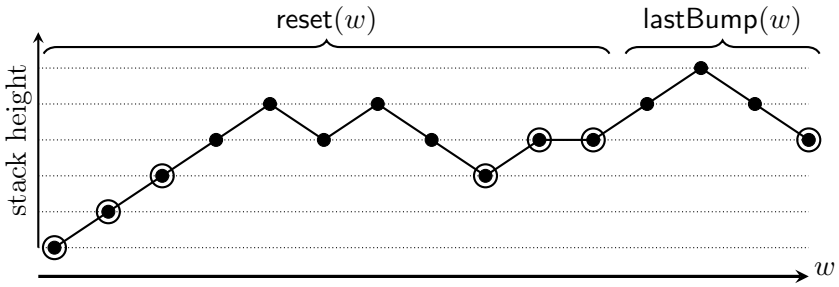
$$\mathsf{MinCol}(w) = \min\{\text{col}(w(i)) \mid 0 \leq i < |w|\},$$

and for $v \in V$ and $w = w(0) \cdots w(l) \cdots w(r) \in V^+$ where $r \geq 1$ and $0 \leq l < r$ is the greatest position such that $\mathsf{sh}(w(l)) \leq \mathsf{sh}(w(r))$, i.e., l is the second largest[3] stair position of w, define

$$\begin{aligned}&\mathsf{reset}(v) = \varepsilon \text{ and } \mathsf{lastBump}(v) = v,\\ &\mathsf{reset}(w) = w(0) \cdots w(l) \text{ and } \mathsf{lastBump}(w) = w(l+1) \cdots w(r).\end{aligned}$$

Notice, that for every $w \in V^+$, we have $\mathsf{reset}(w) \cdot \mathsf{lastBump}(w) = w$. Now, we give the definition of the stair-score functions.

[3] Notice that the last position of a finite path is always a stair position.



col:	0	2	1	0	2	1	0	0	0	1	1	1	1	2	1
$\mathsf{StairSc}_0$:	1	1	1	2	2	2	3	3	2	2	2	2	2	2	2
$\mathsf{StairSc}_1$:	0	0	1	0	0	1	0	0	0	1	2	3	4	4	3
$\mathsf{StairSc}_2$:	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 6.2: A finite path w, its stair positions and values of the stair-scoring functions

Definition 6.5 (Stair-scoring functions)**.** Let (G, col) be a parity pushdown game with $G = (V, V_0, V_1, E, v_{\mathrm{in}})$ and $\mathrm{col}\colon V \to [n]$, for $n \in \mathbb{N}$. For every color $c \in [n]$, define the function $\mathsf{StairSc}_c\colon V^* \to \mathbb{N}$ by

$$\mathsf{StairSc}_c(\varepsilon) = 0$$

and for $w \in V^+$,

$$\mathsf{StairSc}_c(w) = \begin{cases} \mathsf{StairSc}_c(\mathsf{reset}(w)) & \text{if } \mathsf{MinCol}(\mathsf{lastBump}(w)) > c, \\ \mathsf{StairSc}_c(\mathsf{reset}(w)) + 1 & \text{if } \mathsf{MinCol}(\mathsf{lastBump}(w)) = c, \\ 0 & \text{if } \mathsf{MinCol}(\mathsf{lastBump}(w)) < c. \end{cases}$$

Furthermore, for every color $c \in [n]$, the maximum stair-score function $\mathsf{MaxStairSc}_c\colon V^* \cup V^\omega \to \mathbb{N} \cup \{\infty\}$ is defined by

$$\mathsf{MaxStairSc}_c(\rho) = \sup_{w \sqsubseteq \rho} \mathsf{StairSc}_c(w),$$

for $\rho \in V^* \cup V^\omega$.

Figure 6.2 illustrates the above definitions, where an example path w is indicated with its corresponding stack heights and coloring. The positions from $\mathsf{StairPositions}(w)$ are indicated by the marked stack heights. Moreover, the decomposition of w in $\mathsf{reset}(w)$ and $\mathsf{lastBump}(w)$ is inscribed as well as the corresponding values of the stair-scoring functions for all prefixes of w.

Now, we define a new version of finite-time parity games played on pushdown game graphs using these new notions of stair-scoring functions. A

finite-time pushdown parity game $\mathcal{G} = (G, \mathrm{col}, k)$ consists of a pushdown game graph $G = (V, V_0, V_1, E, v_{\mathrm{in}})$, a parity condition $\mathrm{col}\colon V \to [n]$, for $n \in \mathbb{N}$, and a threshold value $k \in \mathbb{N} \setminus \{0\}$. A play in $\mathcal{G}$ is a finite path

$$w = w(0) \cdots w(r) \in V^*$$

with $w(0) = v_{\mathrm{in}}$ and $(w(i), w(i+1)) \in E$, for all $i \in [r]$, such that

- $\mathsf{MaxStairSc}_c(w) = k$ for some $c \in [n]$, and
- $\mathsf{MaxStairSc}_c(w(0) \cdots w(r-1)) < k$ for all $c \in [n]$.

The play w is winning for Player i if $\mathrm{Par}(c) = i$. The notions of strategies and winning strategies are defined as above (see page 126).

Again, we can show that every threshold value k is eventually reached by some stair-scoring function if the path in the pushdown game graph is sufficiently long. This bounds the length of the plays of finite-time parity pushdown games. For this, we need the following lemma.

Lemma 6.6. *Let $G = (V, E)$ be a pushdown graph and $w \in V^+$ be a path in G such that $0 \in \mathsf{StairPositions}(w)$. For any $m \in \mathbb{N}$, if $|w| \geq 2^m$ then there exists a prefix $w' \sqsubseteq w$ such that $|\mathsf{StairPositions}(w')| > m$.*

Proof. We show the claim by induction on m. For the base case, consider $m = 0$. Then, $|w| \geq 1$ implies $|\mathsf{StairPositions}(w)| > 0$, since the first configuration $w(0)$ of w is a stair position. For the induction step, we prove the contrapositive statement. Consider $w = w(0) \cdots w(r)$ such that for all $w' \sqsubseteq w$, $|\mathsf{StairPositions}(w')| \leq m+1$. Let $n_0, n_1 \in \mathsf{StairPositions}(w)$, where $n_0 = 0$, be the first two stair positions of w. First, consider the suffix $u = w(n_1) \cdots w(r)$. As $n_0, n_1 \in \mathsf{StairPositions}(w)$, it follows that $n_0, n_1 \in \mathsf{StairPositions}(w'')$, for every $w'' \sqsubseteq w$ with $|w''| \geq n_1 + 1$. From this, we conclude that for every $u' \sqsubseteq u$, we have $|\mathsf{StairPositions}(u')| = |\mathsf{StairPositions}(w(0) \cdots w(n_1 - 1)u')| - 1$ and $0 \in \mathsf{StairPositions}(u)$. This means, since $|\mathsf{StairPositions}(w')| \leq m + 1$, for all $w' \sqsubseteq w$, we have $|\mathsf{StairPositions}(u')| \leq m$, for all $u' \sqsubseteq u$. Thus, by applying the induction hypothesis, we have $|u| < 2^m$. Now, consider the prefix $v = w(0) \cdots w(n_1 - 1)$. If $n_1 = 1$, then we have $|v| = |w(0)| = 1 \leq 2^m$. Thus, let $n_1 \neq 1$, then $\mathsf{sh}(w(0)) = \mathsf{sh}(w(n_1))$, $\mathsf{sh}(w(1)) = \mathsf{sh}(w(n_1 - 1))$ and $\mathsf{sh}(w(i)) > 0$, for all $i \in \{1, \cdots, n_1 - 1\}$, Thus, for the same reason as for the suffix u we can apply the induction hypothesis and conclude that $|w(1) \cdots w(n_1 - 1)| = |v| - 1 < 2^m$, i.e., $|v| \leq 2^m$. Hence, $|w| = |v| + |u| < 2 \cdot 2^m = 2^{m+1}$. □

Notice, that the bound in Lemma 6.6 is tight, i.e., there is a pushdown graph $G = (V, E)$ such that for every $m \in \mathbb{N} \setminus \{0\}$ there exists a path $w_m \in V^+$ in G with $0 \in \mathsf{StairPositions}(w_m)$ such that $|w_m| = 2^m - 1$

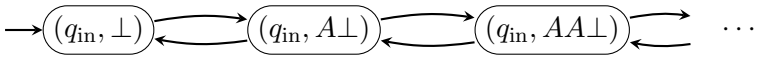


Figure 6.3: The pushdown game graph induced by $\mathcal{P}$

and $|\mathsf{StairPositions}(w')| \leq m$, for every prefix $w' \sqsubseteq w_m$. To verify this, consider the following PDS $\mathcal{P} = (Q, \Gamma, \Delta, q_{\text{in}})$ with $Q = \{q_{\text{in}}\}$, $\Gamma = \{A\}$, $\Delta = \{(q_{\text{in}}, \bot, q_{\text{in}}, A\bot), (q_{\text{in}}, A, q_{\text{in}}, AA), (q_{\text{in}}, A, q_{\text{in}}, \varepsilon)\}$ and its induced configuration graph $G(\mathcal{P}) = (V, E)$ depicted in Figure 6.3. For $v = (q_{\text{in}}, \gamma\bot) \in V$ and $\alpha \in \Gamma^*$, let $\mathsf{raise}(v, \alpha) = (q, \gamma\alpha\bot)$ and for $w = w(0) \cdots w(r) \in V^+$, let $\mathsf{raise}(w, \alpha) = \mathsf{raise}(w(0), \alpha) \cdots \mathsf{raise}(w(r), \alpha)$. We define sequences $w_m \in V^+$, for $m \in \mathbb{N} \setminus \{0\}$, inductively by $w_1 = v_{\text{in}}$ and for $m > 1$, $w_m = v_{\text{in}} \cdot \mathsf{raise}(w_{m-1}, A) \cdot w_{m-1}$. Now, we can show by induction on m that the statement holds for sequences w_m. First, since $|w_1| = 1$, we have $|\mathsf{StairPositions}(w_1)| = 1$. So, let $m > 1$, then $|w_m| = 1 + 2 \cdot |w_{m-1}|$ and for all prefixes $w' \sqsubseteq w_{m-1}$, we have $|\mathsf{StairPositions}(v_{\text{in}} \cdot \mathsf{raise}(w', A))| = |\mathsf{StairPositions}(v_{\text{in}} \cdot \mathsf{raise}(w_{n-1}, A) \cdot w')| = 1 + |\mathsf{StairPositions}(w')|$. Hence, applying the induction hypothesis yields $|w_n| = 1 + 2 \cdot (2^{n-1} - 1) = 2^n - 1$ and $|\mathsf{StairPositions}(w')| \leq m$ for every prefix $w' \sqsubseteq w_m$.

Now, due to the fact that for every play prefix $w' \sqsubseteq w$ of a play $w \in V^+$ in a finite-time pushdown parity game one can construct a sequence $u' \in Q^*$ of states of the corresponding PDS inducing the pushdown game graph with $|u'| = |\mathsf{Stairs}(w')|$ such that for every color $c \in [n]$, $\mathsf{StairSc}_c(w') = \mathsf{Sc}_c(u')$ and by combining Lemma 6.6 and Lemma 6.2 the desired upper bound on the length of the plays of finite-time parity pushdown games is obtained.

Lemma 6.7. *Let $G = (V, E)$ be a pushdown graph with a coloring function* $\mathrm{col}\colon V \to [n]$*, for $n \in \mathbb{N}$. For every $w \in V^*$ and every $k \in \mathbb{N}$, if $|w| \geq 2^{k^n}$, then there is some $c \in [n]$ such that $\mathsf{MaxStairSc}_c(w) \geq k$.*

Thus, a play in a finite-time parity pushdown game (G, col, k) continues for at most 2^{k^n} number of rounds where $[n]$ is the codomain of col. Again, the bound in Lemma 6.7 is tight, i.e., there is a game graph $G = (V, V_0, V_1, E, v_{\text{in}})$ such that for every k there is a path $w \in V^*$ with $|w| = k^n - 1$ such that $\mathsf{MaxStairSc}_c(w) < k$, for all colors $c \in [n]$. Moreover, Lemma 6.3 can directly be lifted to the new definition of stair-scoring functions which ensures a unique winner of a play. This is done in the following lemma.

Lemma 6.8. *Let $G = (V, E)$ be a pushdown graph with a coloring function* $\mathrm{col}\colon V \to [n]$*, for $n \in \mathbb{N}$. For $w \in V^*$, $v \in V$ and $c, c' \in [n]$, if $\mathsf{StairSc}_c(wv) = \mathsf{StairSc}_c(w) + 1$ and $\mathsf{StairSc}_{c'}(wv) = \mathsf{StairSc}_{c'}(w) + 1$, then $c = c'$.*

We prove our main result, the equivalence between parity pushdown

games and finite-time parity pushdown games in Section 6.3. To this end, Walukiewicz's reduction [Wal96] is needed.

6.2 Walukiewicz's Reduction

In this section, we recall the technique for solving parity pushdown games introduced in [Wal96] which comprises a reduction to parity games on finite game graphs. We slightly modify the original construction which is necessary for the proof of the main result in Section 6.3.

Consider a pushdown game $\mathcal{G} = (G(\mathcal{P}), \mathrm{col})$ with a game graph $G(\mathcal{P}) = (V, V_0, V_1, E, v_{\mathrm{in}})$ induced by a PDS $\mathcal{P} = (Q, \Gamma, \Delta, q_{\mathrm{in}})$ and a partition $Q_0 \cup Q_1$ of the set of states Q and a parity condition $\mathrm{col}\colon Q \to [n]$, for $n \in \mathbb{N}$. To simulate $\mathcal{G}$ by a game on a finite game graph the information stored on the stack is encoded by some finite memory structure. The essential component of this structure is the set $\mathsf{Pred} = (\mathcal{P}(Q))^n$ which we call the set of predictions. A prediction $P = (P_0, \ldots, P_{n-1}) \in \mathsf{Pred}$ contains for every color $c \in [n]$ a subset $P_c \subseteq Q$ of states.

The core idea of the game simulating the pushdown game is the following. The players are assigned different tasks, one of them makes predictions and the other one verifies them. Whenever a push-transition is to be simulated the predicting player has to make a prediction $P \in \mathsf{Pred}$ about the future round t when the same stack height as before performing the push-transition is reached again for the first time if it is reached at all. With this prediction, the predicting player claims that if the current push-transition is performed, then in round t some state $q \in P_c$ will be reached if $c \in [n]$ is the minimal color seen in between. Once a prediction P is proposed, the verifying player has two ways of reacting, either believing that P is correct or not. In the first case, he is not interested in verifying P, so the push-transition is not performed and the verifying player chooses a color $c \in [n]$ and a state $q \in P_c$, for some $P_c \neq \emptyset$, and skips a part of the simulated play by jumping to an appropriate position in the play. In the other case, he wants to verify the correctness of P, so the push-transition is performed and when the top of the stack is eventually popped it will turn out whether or not P is correct. The predicting player wins if P turns out to be correct and otherwise the verifying player wins. Thus, after a pop-transition the winner is certain. For the case where no pop-transition is performed at all, the parity condition determines the winner.

In the following, let Player i take the role of the predicting player and Player $1 - i$ the role of the verifying one. The game $\mathcal{G}'_i = (G', \mathrm{col}')$ which depends on $i \in \{0, 1\}$, with $G' = (V', V'_0, V'_1, E', v'_{\mathrm{in}})$ is defined as follows. For all states $q \in Q$, stack symbols $A, B \in \Gamma_\perp$, colors $c, d \in [n]$ and predictions

$P, R \in \mathsf{Pred}$, the set V' contains the vertices

$\mathsf{Check}[q, A, P, c, d]$,
$\mathsf{Push}[P, c, q, AB]$,
$\mathsf{Claim}[P, c, q, AB, R]$,
$\mathsf{Jump}[q, A, P, c, d]$,
$\mathsf{Win}_i[q]$,
$\mathsf{Win}_{1-i}[q]$.

Vertices of the form $\mathsf{Check}[q, A, P, c, d]$ correspond to the configurations of $\mathcal{G}$, vertices $\mathsf{Win}_i[q]$ and $\mathsf{Win}_{1-i}[q]$ are sink vertices. All other vertices are auxiliary vertices and serve as intermediates. Vertices $\mathsf{Push}[P, c, q, AB]$ are used to signalize the intention to perform a push-transition, vertices $\mathsf{Claim}[P, c, q, AB, R]$ are used to make new predictions and parts of a simulated play are skipped via vertices $\mathsf{Jump}[q, A, P, c, d]$.

The set E' consists of the following edges, where for the sake of readability, we denote an edge $(v_1, v_2) \in E'$ by $v_1 \to v_2$. For every skip-transition $\delta = (q, A, p, B) \in \Delta$ there are edges

$$\mathsf{Check}[q, A, P, c, d] \to \mathsf{Check}[p, B, P, \min\{c, \mathrm{col}(p)\}, \mathrm{col}(p)]\,,$$

for $P \in \mathsf{Pred}$ and $c, d \in [n]$. Thus, the first two components of the Check-vertices are updated according to δ, the prediction P remains untouched, the last but one component is used to keep track of the minimal color for being able to check the prediction for correctness and the last component determines the color of the current Check-vertex. For every push-transition $\delta = (q, A, p, BC) \in \Delta$ there are edges

$$\mathsf{Check}[q, A, P, c, d] \to \mathsf{Push}[P, c, p, BC]\,,$$

for all $P \in \mathsf{Pred}$ and $c, d \in [n]$. Here, a player states that a push-transition is to be performed such that the current state q has to be changed to p and the top of the stack A has to be replaced by BC. The information containing the current prediction P and the minimal color c is carried over, as this is needed in the case where the verifying player decides to skip. Moreover, to make a new prediction R, all edges

$$\mathsf{Push}[P, c, p, BC] \to \mathsf{Claim}[P, c, p, BC, R]$$

for every $R \in \mathsf{Pred}$ are needed. In case a new prediction is to be verified, a push-transition is finally performed using edges of the form

$$\mathsf{Claim}[P, c, p, BC, R] \to \mathsf{Check}[p, B, R, \mathrm{col}(p), \mathrm{col}(p)]$$

where the prediction P, the color c and the lower stack symbol C are discarded since they are no longer needed. For the case where the verifying player intends to skip a part of a play, all edges

$$\mathsf{Claim}[P, c, p, BC, R] \to \mathsf{Jump}[q, C, P, c, e]$$

with $q \in R_e$ are contained in E'. Here, the verifying player chooses a color $e \in [n]$ for the minimal color of the skipped part and a state q from the corresponding component R_e of the prediction R. Now, the lower stack symbol C, the prediction P and the color c additionally have to be carried over, whereas B and R are discarded. Then, all edges

$$\mathsf{Jump}[q, C, P, c, e] \to \mathsf{Check}[q, C, P, \min\{c, e, \mathrm{col}(q)\}, \min\{e, \mathrm{col}(q)\}]$$

are contained in E' where the last component of the Check-vertex is set to be the minimum of the color of the current state q and the minimal color of the part just skipped. For the last but one component, we also have to account for the color c, which is necessary for eventually checking P for correctness. Finally, we have for every pop-transition $(q, A, p, \varepsilon) \in \Delta$, the edges

$$\mathsf{Check}[q, A, P, c, d] \to \mathsf{Win}_i[p] \quad \text{if } p \in P_c\,, \text{ and}$$
$$\mathsf{Check}[q, A, P, c, d] \to \mathsf{Win}_{1-i}[p] \quad \text{if } p \notin P_c\,,$$

for $P \in \mathsf{Pred}$ and $c, d \in [n]$, which lead to the sink vertex of the predicting player $\mathsf{Win}_i[p]$ if the prediction P turns out to be correct or to the sink vertex of the verifying player $\mathsf{Win}_{1-i}[p]$ otherwise. Moreover, the following edges are contained in E', for $j \in \{0, 1\}$ and $q \in Q$,

$$\mathsf{Win}_j[q] \to \mathsf{Win}_j[q].$$

The initial vertex v'_{in} has to correspond to the initial configuration of the parity pushdown game $v_{\mathrm{in}} = (q_{\mathrm{in}}, \bot)$, hence, it is defined to be

$$v'_{\mathrm{in}} = \mathsf{Check}[q_{\mathrm{in}}, \bot, P^{\mathrm{in}}, \mathrm{col}(q_{\mathrm{in}}), \mathrm{col}(q_{\mathrm{in}})]$$

where prediction P^{in} is such that $P^{\mathrm{in}}_c = \emptyset$ for every $c \in [n]$, since the symbol $\bot$ cannot be deleted from the empty stack. The set of vertices V'_i of the predicting Player i is defined to consist of all Push-vertices, as there Player i has to make a new prediction, and of those $\mathsf{Check}[p, A, P, c, d]$ vertices where $p \in Q_i$. Accordingly, all other vertices belong to Player $1 - i$. Finally, the coloring function $\mathrm{col}' \colon V' \to [n + 1]$ is defined as follows. For $v \in V'$,

$$\mathrm{col}'(v) = \begin{cases} d & \text{if } v = \mathsf{Check}[p, A, P, c, d], \\ j & \text{if } v = \mathsf{Win}_j[q], \\ n & \text{otherwise,} \end{cases}$$

Notice, that auxiliary vertices are colored by the maximal color n which does not appear in $\mathcal{G}$, since they should have no influence on the minimal color seen infinitely often. This is guaranteed by the structure of G', as there are no loops consisting only of auxiliary vertices. Moreover, notice that in the original construction, Jump-vertices are colored by the minimal color of the skipped part of the play which is chosen by the verifying player. This is avoided here by shifting the color of a Jump-vertex to the successive Check-vertex. For this purpose, the last component of the Check-vertices is introduced.

Theorem 6.9 ([Wal96]). *Let $\mathcal{G}$ be a parity pushdown game. For $i \in \{0,1\}$, Player i wins $\mathcal{G}$ if and only if Player i wins $\mathcal{G}'_i$.*

Now, we describe how a pushdown winning strategy σ for Player i in $\mathcal{G}$ can be constructed from a positional winning strategy σ'_i for Player i in $\mathcal{G}'_i$. The idea is to simulate σ'_i in $\mathcal{G}$. As long as only skip- and push-transitions are involved, the decisions made by σ'_i can simply be transfered to σ. As soon as the first pop-transition is used, σ'_i leads to some sink Win_i-vertex at which the future moves of σ'_i are no longer useful for playing in the original game $\mathcal{G}$. To overcome this, the pushdown strategy σ uses its stack to store Claim-vertices visited during the simulated play. This allows to reset the simulated play and to continue from the appropriate successor Jump-vertex of the Claim-vertex stored on the stack.

Formally, let $G'|_{\sigma'_i} = (V'|_{\sigma'_i}, V'_0|_{\sigma'_i}, V'_1|_{\sigma'_i}, E'|_{\sigma'_i}, v'_{\text{in}})$ be the game graph obtained from $\mathcal{G}'_i$ by restricting it to the vertices and edges visited by σ'_i. Notice, that this implies that every vertex from $V'_i|_{\sigma'_i}$ has a unique successor in $G'|_{\sigma'_i}$ and that Win_{1-i}-vertices are not contained in $V'_i|_{\sigma'_i}$. Furthermore, let

$$\ell \colon E'|_{\sigma'_i} \to \Delta \cup \{\varepsilon\}$$

be the labeling which assigns to every edge in $E'|_{\sigma'_i}$ its corresponding transition $\delta \in \Delta$, i.e., for $(v, v') \in E'|_{\sigma'_i}$,

$$\ell(v,v') = \begin{cases} (q, A, p, B) & \text{if } v = \mathsf{Check}[q, A, P, c, d] \text{ and} \\ & v' = \mathsf{Check}[p, B, P, c', d'] \\ (q, A, p, BC) & \text{if } v = \mathsf{Check}[q, A, P, c, d] \text{ and} \\ & v' = \mathsf{Push}[P, c, p, BC] \\ (q, A, p, \varepsilon) & \text{if } v = \mathsf{Check}[q, A, P, c, d] \text{ and } v' = \mathsf{Win}_i[p] \\ \varepsilon & \text{otherwise.} \end{cases}$$

We construct a deterministic pushdown transducer $\mathcal{T}_\sigma$ implementing the pushdown strategy σ from σ'_i by exploiting the game graph $G'|_{\sigma'_i}$ for its finite

control and the Claim-vertices as its stack symbols. Formally, the DPDT implementing σ is defined by $\mathcal{T}_\sigma = (Q^\sigma, \Gamma^\sigma, \Delta^\sigma, q^\sigma_{\text{in}}, \Sigma^\sigma_I, \Sigma^\sigma_O, \lambda^\sigma)$ where

- $Q^\sigma = V'|_{\sigma'_i}$,
- $\Gamma^\sigma = \{v \in V'|_{\sigma'_i} \mid v \text{ is a } \mathsf{Claim}\text{-vertex}\}$,
- $q^\sigma_{\text{in}} = v'_{\text{in}}$,
- $\Sigma^\sigma_I = \Sigma^\sigma_O = \Delta$,
- for every $(v, v') \in E'|_{\sigma'_i}$,
  - if v is not a Claim-vertex *and* v' is not a Win_i-vertex then
    $$(v, Z, \ell(v, v'), v', Z) \in \Delta^\sigma, \text{ for every } Z \in \Gamma^\sigma_\perp,$$
  - if v is a Claim-vertex and v' is a Check-vertex, then
    $$(v, Z, \ell(v, v'), v', vZ) \in \Delta^\sigma, \text{ for every } Z \in \Gamma^\sigma_\perp,$$
  - if $v = \mathsf{Check}[q, A, P, c, d]$ and $v' = \mathsf{Win}_i[p]$, for $q, p \in Q$, $A \in \Gamma$, $P \in \mathsf{Pred}$ and $c, d \in [n]$, then
    $$(v, Z, \ell(v, v'), \mathsf{Jump}[p, C, R, e, c], \varepsilon) \in \Delta^\sigma,$$
    for every $Z = \mathsf{Claim}[R, e, q', BC, R'] \in \Gamma^\sigma$.
- for Check-vertices $v \in V'|_{\sigma'_i}$,
  $$\lambda^\sigma(v) = \ell(v, v') \text{ where } (v, v') \in E'|_{\sigma'_i}.$$

Due to this construction, $\mathcal{T}_\sigma$ updates its states according to the transition structure of $G'|_{\sigma'_i}$ as long as no Claim-vertices and Win_i-vertices are involved. As soon as a Claim-vertex v is reached, it is pushed onto the stack while the state is updated to the successive Check-vertex v' according to the current transition $\ell(v, v')$ which is read. Moreover, each time some Win_i-vertex $v' = \mathsf{Win}_i[p]$ is reached from a vertex $v = \mathsf{Check}[q, A, P, c, d]$, the topmost symbol $Z = \mathsf{Claim}[R, e, q', BC, R']$ is popped from the stack and the pushdown transducer proceeds to the state $\mathsf{Jump}[p, C, R, e, c]$ which would be reached in $G'_i|_{\sigma'_i}$ if Player $1 - i$ had chosen color c and state $p \in R_c$ to determine the successor of $\mathsf{Claim}[R, e, q', BC, R']$. Furthermore, the corresponding transition $\ell(v, v')$ of the edge (v, v') chosen by σ'_i determines the output of $\mathcal{T}_\sigma$.

The correctness of this construction emerges from Lemma 6.11 in the following section, which establishes a connection between the values of the scoring functions of the plays in $\mathcal{G}'_i$ and the values of the stair-scoring functions of the corresponding plays in $\mathcal{G}$.

6.3 Equivalence: Infinite-Time and Finite-Time

In this section, we prove the equivalence between parity pushdown games and the corresponding finite-time parity pushdown games for a certain threshold which is exponential in the size of the underlying pushdown system. For any parity pushdown game $\mathcal{G} = (G(\mathcal{P}), \mathrm{col})$ with the pushdown game graph $G(\mathcal{P}) = (V, V_0, V_1, E, v_{\mathrm{in}})$ induced by a PDS $\mathcal{P} = (Q, \Gamma, \Delta, q_{\mathrm{in}})$ and a parity condition $\mathrm{col}\colon Q \to [n]$, for $n \in \mathbb{N}$, define $k_{\mathcal{G}}$ by

$$k_{\mathcal{G}} = |Q| \cdot |\Gamma| \cdot 2^{|Q| \cdot n} \cdot n.$$

Clearly, $k_{\mathcal{G}}$ is an upper bound on the number of Check-vertices in $\mathcal{G}'_i$ of the same color. Furthermore, to simplify our notation, we define the function

$$\mathsf{lastStrictBump}\colon \{v_{\mathrm{in}}\} \cdot V^* \to V^+$$

as follows. For $w \in V^+$ with $\mathsf{sh}(\mathsf{last}(w)) = 0$, let

$$\mathsf{lastStrictBump}(w) = w,$$

and for $w(0) \cdots w(l) \cdots w(r) \in V^+$ where l is the greatest position such that $\mathsf{sh}(w(l)) < \mathsf{sh}(w(r))$, define

$$\mathsf{lastStrictBump}(w) = w(l+1) \cdots w(r).$$

Now, we can formulate our main theorem.

Theorem 6.10. *Let $\mathcal{G} = (G, \mathrm{col})$ be a parity pushdown game and let $\mathcal{G}_k = (G, \mathrm{col}, k)$ be the corresponding finite-time parity pushdown game with threshold value k. For every $k > k_{\mathcal{G}}$, Player i wins $\mathcal{G}$ if and only if Player i wins $\mathcal{G}_k$.*

To prove this theorem, we need the following lemma which establishes a relation between the values of the scoring functions of the plays in $\mathcal{G}'_i$ and the values of the stair-scoring functions of the corresponding plays in $\mathcal{G}$. Let σ'_i be a positional winning strategy for Player i in $\mathcal{G}'_i$ and $\mathcal{T}_\sigma$ be the DPDT implementing the corresponding pushdown winning strategy σ for Player i in $\mathcal{G}$ as defined in the previous section.

Lemma 6.11. *For every play prefix w in $\mathcal{G}$ consistent with σ, there is a play prefix w' in $\mathcal{G}'_i$ that is consistent with σ'_i such that $\mathsf{StairSc}_c(w) = \mathsf{Sc}_c(w')$, for every $c \in [n]$.*

Proof. We show the lemma by induction over $|w|$. To prove our claim, we strengthen the induction hypothesis as follows. For every play prefix w in $\mathcal{G}$ that is consistent with σ, there is a play prefix w' in $\mathcal{G}'_i|_{\sigma'_i}$ that is consistent with σ'_i by construction, such that the following requirements are satisfied, let $\mathsf{last}(w) = (q, A\gamma)$,

(i) $\mathsf{StairSc}_c(w) = \mathsf{Sc}_c(w')$, for every $c \in [n]$.

(ii) $\mathsf{last}(w') = \mathsf{Check}[q, A, P, c, d]$, for some $P \in \mathsf{Pred}$, $d \in [n]$ and for $c = \mathsf{MinCol}(\mathsf{lastStrictBump}(w))$.

(iii) Let (v, γ_σ) be the last configuration of the run of $\mathcal{T}_\sigma$ on the sequence of transitions induced by w. Furthermore, if $\gamma_\sigma \neq \bot$, let $\gamma_\sigma(j) = \mathsf{Claim}[P_j, c_j, p_j, B_jC_j, R_j]$ for every $0 \leq j \leq |\gamma_\sigma| - 2$. We require

- $v = \mathsf{last}(w')$,
- $C_0 \cdots C_k = \gamma$ where $k = |\gamma_\sigma| - 2$ and
- $R_0 = P$.

For the induction start, we have $w = v_{\text{in}} = (q_{\text{in}}, \bot)$. Define w' by

$$w' = v'_{\text{in}} = \mathsf{Check}[q_{\text{in}}, \bot, P^{\text{in}}, \text{col}(q_{\text{in}}), \text{col}(q_{\text{in}})].$$

Since $\text{col}(v_{\text{in}}) = \text{col}'(v'_{\text{in}}) = \text{col}(q_{\text{in}})$, we have $\mathsf{StairSc}_c(w) = \mathsf{Sc}_c(w')$, for every $c \in [n]$. Moreover, requirements (ii) and (iii) are satisfied as well.

Now, let $w = w(0) \cdots w(r)$ with $r > 0$ and $w(r-1) = (q, A\gamma)$. Moreover, let $\mathsf{reset}(w) = w(0) \cdots w(s)$ and $w(s) = (q_s, A_s\gamma_s)$. The induction hypothesis yields play prefixes u' and u'_s in $\mathcal{G}'_i|_{\sigma'_i}$ such that by requirement (i) we have

$$\mathsf{StairSc}_c(w(0) \cdots w(r-1)) = \mathsf{Sc}_c(u') \text{ and}$$
$$\mathsf{StairSc}_c(w(0) \cdots w(s)) = \mathsf{Sc}_c(u'_s),$$

for every $c \in [n]$. Also, for some $P, P_s \in \mathsf{Pred}$ and $d, d_s \in [n]$, by requirement (ii) we have

$$\mathsf{last}(u') = \mathsf{Check}[q, A, P, c, d]$$
$$\text{with } c = \mathsf{MinCol}(\mathsf{lastStrictBump}(w(0) \cdots w(r-1))) \text{ and}$$

$$\mathsf{last}(u'_s) = \mathsf{Check}[q_s, A_s, P_s, c_s, d_s]$$
$$\text{with } c_s = \mathsf{MinCol}(\mathsf{lastStrictBump}(w(0) \cdots w(s))).$$

We distinguish three cases, whether the transition from $w(r-1)$ to $w(r)$ is a skip-, push-, or pop-transition.

In case of a skip-transition $\delta = (q, A, p, B)$, we have $w(r) = (p, B\gamma)$. By construction, there is also an edge from $\mathsf{last}(u') = \mathsf{Check}[q, A, P, c, d]$ to the vertex

$$v = \mathsf{Check}[p, B, P, \min\{c, \text{col}(p)\}, \text{col}(p)]$$

in $\mathcal{G}'_i|_{\sigma'_i}$ labeled by $\ell(\mathsf{last}(u'), v) = \delta$. Thus, let $w' = u'v$. This choice satisfies requirement (ii), as for a skip-transition from $w(r-1)$ to $w(r)$ it holds

$$\begin{aligned}&\mathsf{MinCol}(\mathsf{lastStrictBump}(w))\\ &= \min\{\mathsf{MinCol}(\mathsf{lastStrictBump}(w(0)\cdots w(r-1))), \mathrm{col}(w(r))\}\\ &= \min\{c, \mathrm{col}(p)\}\ .\end{aligned}$$

Furthermore, requirement (iii) is satisfied since when processing δ, the pushdown transducer $\mathcal{T}_\sigma$ changes its state $\mathsf{last}(u')$ to v while the stack is left unchanged. To prove the equality of the scores, let $e = \mathrm{col}(w(r))$, which is also the color of v in $\mathcal{G}'_i|_{\sigma'_i}$. Then, we have

$$\mathsf{StairSc}_e(w) = \mathsf{StairSc}_e(w(0)\cdots w(r-1)) + 1 = \mathsf{Sc}_e(u') + 1 = \mathsf{Sc}_e(w')$$

and for $e' < e$,

$$\mathsf{StairSc}_{e'}(w) = \mathsf{StairSc}_{e'}(w(0)\cdots w(r-1)) = \mathsf{Sc}_{e'}(u') = \mathsf{Sc}_{e'}(w').$$

Finally, for $e' > e$, we have

$$\mathsf{StairSc}_{e'}(w) = 0 = \mathsf{Sc}_{e'}(w').$$

In case of a push-transition $\delta = (q, A, p, BC)$, we have $w(r) = (p, BC\gamma)$. Consider the finite path

$$u'' = \mathsf{Push}[P, c, p, BC] \cdot \mathsf{Claim}[P, c, p, BC, R] \cdot \mathsf{Check}[p, B, R, \mathrm{col}(p), \mathrm{col}(p)]$$

in $\mathcal{G}'_i|_{\sigma'_i}$ where R is the prediction picked by σ'_i. Notice that there is indeed an edge from $\mathsf{last}(u')$ to $\mathsf{Push}[P, c, p, BC]$ in $E'|_{\sigma'_i}$. We claim that $w' = u'u''$ has the desired properties. Requirement (ii) is satisfied, as $\mathsf{lastStrictBump}(w) = w(r)$ in this case, and $\mathsf{MinCol}(w(r)) = \mathrm{col}(p)$. Furthermore, $\mathsf{Claim}[P, c, p, BC, R]$ is pushed onto the stack of $\mathcal{T}_\sigma$ when processing δ. Hence, requirement (iii) is satisfied.

The scores evolve as in the case of a skip-transition explained above since in both cases we have $\mathsf{lastBump}(w) = w(r)$, and u'' contains exactly one vertex with color in $[n]$, namely its last vertex, which has the same color as $w(r)$. The intermediate auxiliary vertices have color n and, therefore, do not influence the scores we are interested in.

Finally, the case of a pop-transition is the most involved one since a play in $\mathcal{G}'_i|_{\sigma'_i}$ ends in a sink Win_i-vertex, as soon as a pop-transition is simulated. In this case, $\mathcal{T}_\sigma$ uses the top Claim-vertex stored on its stack to determine the appropriate Check-vertex that allows to continue playing according to σ'_i. Suppose the transition is $\delta = (q, A, p, \varepsilon)$, i.e., we have $w(r) = (p, \gamma)$. Let $\delta_s = (q_s, A_s, q', BC)$ be the push-transition of the PDS underlying $\mathcal{G}$,

which induces the edge $(w(s), w(s+1)) \in E$. Note that $C\gamma_s = \gamma$, since the stack content $C\gamma_s$ remains untouched until δ is executed from $w(r-1)$ to $w(r)$. Hence, $w(r) = (p, C\gamma_s)$. By definition of σ, there is an edge from $\mathsf{last}(u') = \mathsf{Check}[q, A, P, c, d]$ to $\mathsf{Win}_i[p]$ in $E'|_{\sigma'_i}$ such that $p \in P_c$.

Now, consider the run of $\mathcal{T}_\sigma$ on w. By construction, the transducer pops the top Claim-vertex v from its stack while processing the transition δ. We show that $v = \mathsf{Claim}[P_s, c_s, q', BC, P]$. First, notice that v was pushed onto the stack while processing the transition from $w(s)$ to $w(s+1)$, which is induced by δ_s. Applying the induction hypothesis shows that the run of $\mathcal{T}_\sigma$ on the sequence of transitions induced by $w(0) \cdots w(s)$ ends in state $\mathsf{last}(u'_s) = \mathsf{Check}[q_s, A_s, P_s, c_s, d_s]$ with some stack content $\gamma_\sigma \in (\Gamma^\sigma)^+ \bot$ satisfying the above requirements. Since now δ_s is to be processed, the run of $\mathcal{T}_\sigma$ is continued as follows for some $R \in \mathsf{Pred}$,

$$
\begin{aligned}
&(\mathsf{last}(u'_s), \gamma_\sigma) \overset{\delta_s}{\vdash} (\mathsf{Push}[P_s, c_s, q', BC], \gamma_\sigma) \\
&\overset{\varepsilon}{\vdash} (\mathsf{Claim}[P_s, c_s, q', BC, R], \gamma_\sigma) \\
&\overset{\varepsilon}{\vdash} (\mathsf{Check}[q', B, R, \mathrm{col}(q'), \mathrm{col}(q')], \mathsf{Claim}[P_s, c_s, q', BC, R] \cdot \gamma_\sigma)
\end{aligned}
$$

It remains to show that $R = P$, which is done by applying the induction hypothesis to the run of $\mathcal{T}_\sigma$ on transitions induced by $w(0) \cdots w(r-1)$. The top symbol $\mathsf{Claim}[P_s, c_s, q', BC, R]$, which is pushed on the stack while processing $(w(s), w(s+1))$, remains untouched until $w(r-1)$ is reached and is again the top symbol after processing $(w(r-2), w(r-1))$. However, since $\mathsf{last}(u') = \mathsf{Check}[q, A, P, c, d]$ is the state reached by $\mathcal{T}_\sigma$ after processing $w(0) \cdots w(r-1)$ it follows from requirement (iii) that $R = P$.

Consider the following finite path $u'' = u''(0) \cdots u''(3)$ in $\mathcal{G}'_i|_{\sigma'_i}$ with

$$
\begin{aligned}
u''(0) &= \mathsf{Push}[P_s, c_s, q', BC], \\
u''(1) &= v, \\
u''(2) &= \mathsf{Jump}[p, C, P_s, c_s, c], \text{ and} \\
u''(3) &= \mathsf{Check}[p, C, P_s, \min\{c_s, c, \mathrm{col}(p)\}, \min\{c, \mathrm{col}(p)\}].
\end{aligned}
$$

Notice that there is an edge from $\mathsf{last}(u'_s)$ to $\mathsf{Push}[P_s, c_s, q', BC]$ in $E'|_{\sigma'_i}$. So, we can show that $w' = u'_s u''$ satisfies the above requirements. Requirement (ii) is satisfied, since

$$
\begin{aligned}
&\mathsf{MinCol}\big(\mathsf{lastStrictBump}(w)\big) \\
&= \min\big\{\mathsf{MinCol}\big(\mathsf{lastStrictBump}(w(0) \cdots w(s))\big), \\
&\qquad\qquad \mathsf{MinCol}\big(w(s+1) \cdots w(r-1)\big), \mathrm{col}(w(r))\big\} \\
&= \min\big\{c_s, \mathsf{MinCol}\big(\mathsf{lastStrictBump}(w(0) \cdots w(r-1))\big), \mathrm{col}(p)\big\} \\
&= \min\big\{c_s, c, \mathrm{col}(p)\big\} \ .
\end{aligned}
$$

Requirement (iii) is satisfied, since after processing δ by $\mathcal{T}_\sigma$, the top stack symbol v is popped from the stack and the state

$$\mathsf{Check}[p, C, P_s, \min\{c_s, c, \mathrm{col}(p)\}, \min\{c, \mathrm{col}(p)\}]$$

is reached. By doing so, the same stack content is reestablished as after the run of $\mathcal{T}_\sigma$ on $\mathsf{reset}(w)$. Hence, by applying the induction hypothesis, we have $C_0 \cdots C_k = \gamma_s$. Since we have $\gamma = C\gamma_s$, this suffices. To show requirement (i), let color $e \in [n]$ be

$$\begin{aligned} e &= \mathsf{MinCol}(\mathsf{lastBump}(w)) \\ &= \min\{\mathsf{MinCol}(\mathsf{lastStrictBump}(w(0) \cdots w(r-1))), \mathrm{col}(w(r))\} \\ &= \min\{c, \mathrm{col}(p)\} \ . \end{aligned}$$

Notice that e is also the color of the vertex $\mathsf{last}(w')$ in $\mathcal{G}_i'|_{\sigma_i'}$ which is

$$\mathsf{last}(w') = \mathsf{Check}[p, C, R, \min\{c_s, c, \mathrm{col}(p)\}, \min\{c, \mathrm{col}(p)\}].$$

Thus, $\mathsf{StairSc}_e(w) = \mathsf{StairSc}_e(w(0) \cdots w(s)) + 1 = \mathsf{Sc}_e(u_s') + 1 = \mathsf{Sc}_e(w')$ and for $e' < e$ $\mathsf{StairSc}_{e'}(w) = \mathsf{StairSc}_{e'}(w(0) \cdots w(s)) = \mathsf{Sc}_{e'}(u_s') = \mathsf{Sc}_{e'}(w')$. Finally, if $e' > e$, $\mathsf{StairSc}_{e'}(w) = 0 = \mathsf{Sc}_{e'}(w')$. □

Now, the proof of the main Theorem 6.10 is straightforward.

Proof of Theorem 6.10. Assume that Player i wins $\mathcal{G}$, then due to Theorem 6.9 Player i also wins $\mathcal{G}_i'$. For every color $c \in [n]$, there are at most $k_\mathcal{G}$ Check-vertices colored by c. Hence, by Remark 6.1 there is a positional winning strategy σ_i' in $\mathcal{G}_i'$ for Player i such that for every color $c \in [n]$ with $\mathrm{Par}(c) = 1 - i$, we have $\mathsf{MaxSc}_c(\rho') \leq k_\mathcal{G}$, for every play ρ' which is consistent with σ_i'. From Lemma 6.11 it follows that the pushdown strategy σ that is constructed from σ_i' bounds the values of the stair-scoring functions of Player $1 - i$ by $k_\mathcal{G}$. Thus, for every play ρ which is consistent with σ and every $k > k_\mathcal{G}$, there exists $w \sqsubset \rho$ such that w is winning for Player i in $\mathcal{G}_k$. Hence, using the same strategy σ Player i wins every finite-time game $\mathcal{G}_k$ for $k > k_\mathcal{G}$. The reverse direction follows by determinacy of parity games. □

6.4 Lower Bounds

In the previous section, we proved the equivalence between parity pushdown games and corresponding finite-time parity pushdown games with an exponential threshold. In this section, we present an almost matching lower bound on the threshold value that always yields equivalent games. To this end, we construct a parity pushdown game in which the winning player is forced to produce a play prefix leading to a configuration of high stack

height while only visiting states such that the stair-scoring functions of the opponent increase. Thereby, the opponent is the first player to reach high stair-scores, although he loses the play eventually.

Theorem 6.12. *There are a family of pushdown games $(G(\mathcal{P}_n), \mathrm{col}_n)$ and threshold values k_n exponential in the cube root of the size of $\mathcal{P}_n$ such that for every $n > 0$, Player O wins the pushdown game $(G(\mathcal{P}_n), \mathrm{col}_n)$, but for every $k \leq k_n$, Player I wins the finite-time pushdown game $(G(\mathcal{P}_n), \mathrm{col}_n, k)$.*

Proof. Let the i-th prime number be denoted by prim_i. For $n > 0$, define

$$k_n = \prod_{i=1}^{n} \mathsf{prim}_i$$

and let the PDS $\mathcal{P}_n = (Q_n, \Gamma, \Delta_n, q_{\mathrm{in}})$ be defined by

- $Q_n = \{q_{\mathrm{in}}, q_\Box\} \cup \bigcup_{i=1}^{n} M_i$, where $M_i = \{q^j_{\mathsf{prim}_i} \mid 0 \leq j < \mathsf{prim}_i\}$,
- $\Gamma = \{A\}$,
- Δ_n consists of the following transitions
  - $(q_{\mathrm{in}}, X, q_{\mathrm{in}}, AX)$ and $(q_{\mathrm{in}}, X, q_\Box, AX)$, for every $X \in \{A, \bot\}$,
  - $(q_\Box, A, q^0_{\mathsf{prim}_i}, A)$ for every $1 \leq i \leq n$,
  - $(q^j_{\mathsf{prim}_i}, A, q^\ell_{\mathsf{prim}_i}, \varepsilon)$, where $\ell = (j+1) \bmod \mathsf{prim}_i$, and
  - $(q, \bot, q, \bot)$, for every $q \in Q_n \setminus \{q_{\mathrm{in}}\}$.

Furthermore, define the partition of the set Q_n of states by $(Q_n)_1 = \{q_\Box\}$ and $(Q_n)_0 = Q_n \setminus (Q_n)_1$, i.e., Player I moves from configurations in state $q_\Box$ and all other configurations belong to Player O. Moreover, let the coloring function $\mathrm{col}_n \colon Q_n \to \{0, 1\}$ be given by

$$\mathrm{col}_n(q) = \begin{cases} 0 & \text{if } q = q^0_{\mathsf{prim}_i}, \text{ for } 1 \leq i \leq n \\ 1 & \text{otherwise,} \end{cases}$$

for $q \in Q_n$. For the threshold value, we have $k_n \geq 2^n$, and the number of states $|Q_n|$ can be bounded from above by $\mathcal{O}(n^2 \log(n))$ [BS96]. Hence, the threshold value k_n is exponential in the cube root of the size of $\mathcal{P}_n$. The pushdown game (G_2, col_2) for the case $n = 2$ is depicted in Figure 6.4. Double-lined vertices correspond to configurations colored by 0.

A play in the game (G_n, col_n) proceeds as follows. Player O picks a natural number $x > 0$ by moving the token to the configuration $(q_\Box, A^x \bot)$. If he fails to do so by staying in q_{in} ad infinitum, he loses since $\mathrm{col}_n(q_{\mathrm{in}}) = 1$. At configuration $(q_\Box, A^x \bot)$, Player I picks a modulus $\mathsf{prim}_i \in \{\mathsf{prim}_1, \ldots, \mathsf{prim}_n\}$

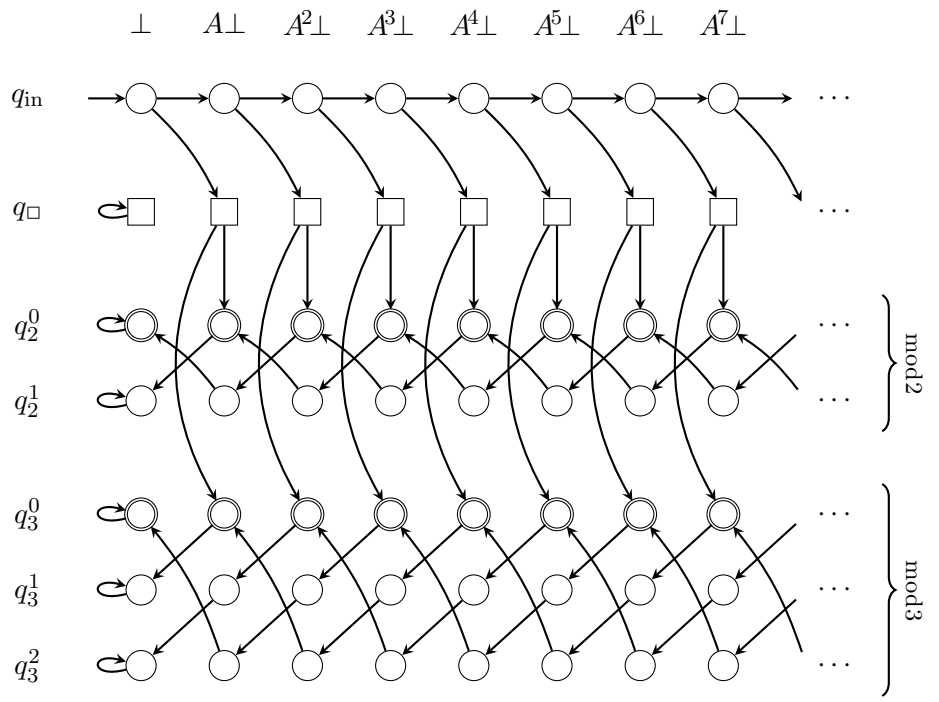


Figure 6.4: Pushdown game (G_2, col_2)

by moving the token to $(q^0_{\mathsf{prim}_i}, A^x\bot)$. From this configuration, a single path emanates, i.e., there is only one way to continue the play. Player O wins this play if and only if x mod $\mathsf{prim}_i = 0$. Hence, Player O has a winning strategy for this game by moving the token to some non-zero multiple of k_n. Thus, the parity pushdown game (G_n, col_n) is won by Player O, for every $n > 0$.

Now, let $k \leq k_n$. If Player O reaches $(q_{\text{in}}, A^{k-1}\bot)$, then he loses the finite-time pushdown game (G_n, col_n, k), since in this case we have

$$\mathsf{MaxStairSc}_1(w) = k \text{ and } \mathsf{MaxStairSc}_0(w) = 0,$$

for the play prefix w produced by Player O. On the other hand, if Player O moves the token to a configuration $(q_\Box, A^x\bot)$ for some $x \leq k-1$, then there is a $\mathsf{prim}_i \in \{\mathsf{prim}_1, \ldots, \mathsf{prim}_n\}$ such that x mod $\mathsf{prim}_i \neq 0$, as $x < k_n$. Hence, assume Player I moves the token to $(q^0_{\mathsf{prim}_i}, A^x\bot)$. Then, the play ends in a self-loop at a configuration $(q^m_{\mathsf{prim}_i}, \bot)$ for some $m \neq 0$. The path w from $(q_{\text{in}}, \bot)$ to $(q^m_{\mathsf{prim}_i}, \bot)$ via $(q_\Box, A^x)$ satisfies $\mathsf{MaxStairSc}_0(w) \leq x$. Since $q^m_{\mathsf{prim}_i}$ is colored by 1, the stair-scoring function $\mathsf{StairSc}_0$ is never increased while using the self-loop at $(q^m_{\mathsf{prim}_i}, \bot)$. Thus, his scores never reach the threshold value k. Hence, Player I is the first to reach this threshold value since by Lemma 6.7 there is some color that reaches the threshold value eventually.

Thus, Player I wins (G_n, col_n, k). □

6.5 Summary of Results

We investigated finite-duration variants of infinite-duration games for the case of parity pushdown games. It turned out that known results for games on finite game graphs which establish equivalence between finite-time games and the corresponding infinite-duration games don't hold for the case of infinite game graphs.

To overcome this problem and, nevertheless, to obtain an analogous connection between infinite-duration games and corresponding finite-duration games on pushdown graphs, we introduced a new finite-duration variant for parity pushdown games. For this, the notions of scoring functions were adapted to exploit the intrinsic structure of a pushdown game graph. Using this new definitions of stair-scoring functions, we obtained a finite-duration parity pushdown game that always has the same winner as the corresponding parity pushdown game, if a play terminates as soon as some stair-scoring function reaches a threshold value which is exponential in the size of the underlying pushdown system. This result yields that the winner of a parity pushdown game can be determined by solving a finite reachability game.

Furthermore, we established an almost matching lower bound on the threshold values which constitute the termination condition for the plays of finite-time pushdown games, which is exponential in the cube root of the size of the underlying pushdown system.

Chapter 7

Conclusion

The main object of investigation in this thesis are infinite games played on pushdown graphs as well as Gale-Stewart games with contextfree winning conditions. The study of such infinite-state games is motivated by non-terminating reactive systems with non-regular system specifications, i.e., specifications over infinite memory structures. Considering pushdown games in this context is the first step towards an effective theory of infinite-state games. Let us summarize the main results of the work in this chapter and point out some directions for further research.

7.1 Results

The main results of the thesis are the following.

- A tight connection between the *formats of game specifications and corresponding winning strategies* was established for a number of types of pushdown games, among them, pushdown games played on pushdown graphs and Gale-Stewart games defined by deterministic, deterministic visibly, deterministic realtime, and deterministic one-counter machines or automata, respectively, with parity and stair parity conditions. Furthermore, we exhibited special cases of pushdown games where this correspondence fails, namely those with deterministic visibly one-counter winning conditions. Moreover, we showed that deterministic blind one-counter strategies are not sufficient for solving pushdown games on pushdown graphs of deterministic blind one-counter machines.

- Solving deterministic contextfree *delay games* was shown to be undecidable, in general. Moreover, we gave a criterion which classifies sets of delay functions for which it is decidable, given a deterministic contextfree winning condition, whether there is a delay function in the

fixed set such that the player with lookahead wins the corresponding delay game. Furthermore, a non-elementary lower bound on delay functions was established, i.e., we showed that there is a deterministic contextfree winning condition such that for some delay function the player obtaining the lookahead wins the corresponding delay game, however, for every elementary delay function, he loses the corresponding delay game.

- The analysis of pushdown games in the context of *distributed synthesis problem* resulted in a complete characterization of decidable architectures for the class of specifications given by a list of local specifications, one for each process, which may be regular or deterministic contextfree. Moreover, for the case of global deterministic contextfree specifications, undecidability was proved for all de facto distributed architectures, i.e., only architectures corresponding to the nondistributed setting and the corner case where the environment doesn't send information at all are decidable for global deterministic contextfree specifications.

- A new *finite-duration* variant of pushdown games was introduced. We proved equivalence between infinite-duration and finite-duration pushdown games with an exponential threshold value determining the termination condition of a play. This yields a new reduction method to determine the winner of a pushdown game by solving a reachability game over a finite graph. Moreover, we established a lower bound on the threshold value, for which the equivalence holds, which is exponential in the cube root of the size of the underlying pushdown system.

7.2 Further Research

Let us mention some open questions and ideas for potential further research in this section.

The results of Chapter 3 raise the question concerning abstract reasons for the transfer of game specifications of certain types to corresponding winning strategies of the same type. Can we find criteria which precisely separate the classes of games played on pushdown graphs as well as contextfree Gale-Stewart games where solvability with winning strategies of the same format is guaranteed from those classes where winning strategies of the same format do not suffice? Incidentally, this question is open as well for regular games. Moreover, there is one more specific open question concerning blind one-counter games. We showed that deterministic blind one-counter strategies are not sufficient for solving pushdown games on pushdown graphs of deterministic blind one-counter machines (see Theorem 3.24). However, it is

open, whether this result also holds for corresponding Gale-Stewart games. Hence, are $\mathsf{DB1CL}_\omega$-games and $\mathsf{StDB1CL}_\omega$-games determined with $\mathsf{DB1CL}$ winning strategies?

Our results from Chapter 4 on contextfree delay games were shown also for restricted classes of deterministic contextfree winning conditions. Hence, we showed that undecidability and lower bounds on delay also hold for winning conditions recognized by deterministic visibly one-counter automata with weak acceptance conditions. Since in the proofs of Theorem 4.4 and Theorem 4.7, basically Player O is responsible for the behavior of the stack, we can restrict the class of visibly winning conditions even more by partitioning the alphabet $\Sigma_I \times \Sigma_O$ such that only the second component of a letter determines the membership in Σ_{push}, Σ_{pop} or Σ_{skip}, respectively. An interesting question is, whether our results also hold if we consider the class of visibly winning conditions where Player I controls the behavior of the stack, i.e., the alphabet $\Sigma_I \times \Sigma_O$ is partitioned such that now only the first component of a letter determines to which alphabet it belongs, Σ_{push}, Σ_{pop} or Σ_{skip}, respectively. The following example shows that linear lookahead is necessary for this case.

Example 7.1. Let $\Sigma_I = \{a, b\}$, $\Sigma_O = \{0, 1\}$. Furthermore, let $\Sigma_{\text{push}} = \{a\} \times \Sigma_O$ and $\Sigma_{\text{pop}} = \{b\} \times \Sigma_O$. Notice, that the first component of a letter from $\Sigma = \Sigma_I \times \Sigma_O$ determines to which alphabet it belongs. Consider the following winning condition L over Σ. A word $w = \alpha^\frown\beta \in \Sigma^\omega$ is contained in L if

- $\alpha(0) = b$, or
- $|\mathsf{pref}_n(\alpha)|_a > |\mathsf{pref}_n(\alpha)|_b$, for all $n > 0$, or
- for the minimal $n > 0$ with $|\mathsf{pref}_n(\alpha)|_a = |\mathsf{pref}_n(\alpha)|_b$ we have $\beta(m) = 1$ and $\beta(m') = 0$ for all $m' < m$, where $m < n$ is the maximal position such that $\alpha(m) = a$.

This means that in order to violate L Player I has to start with symbol a and eventually produce a prefix containing the same number of a's as b's, i.e., a configuration with empty stack is eventually reached. The task of Player O is to indicate the last position where Player I has produced a b before the stack is empty for the first time. ◇

Clearly, Player O wins $\Gamma_f(L)$ with the linear-delay function $f(n) = 2$, for $n \in \mathbb{N}$. This is due to the fact that the stack height is bounded by k after processing k letters. Hence, Player I can play at most k returns before the stack is empty. However, obviously for every bounded-delay function Player I has a winning strategy in the corresponding delay game. It would

be interesting to figure out whether linear delay is always sufficient for this restricted class of visibly winning conditions, where Player I controls the behavior of the stack, as well as whether the winner of such a game can be determined effectively.

In Chapter 6 we showed the equivalence between parity pushdown games and corresponding finite-time parity pushdown games for an exponential threshold value. Since every threshold value $k \in \mathbb{N}$ is eventually reached by some stair-scoring function if the play prefix is sufficiently long, according to Lemma 6.7, Player O wins if he manages to prevent his opponent from reaching an exponential stair-score which yields a safety game. An essential question is, whether and how a winning strategy of a finite-time parity pushdown game or a winning strategy of the corresponding safety game can be turned into a winning strategy for the original parity pushdown game. In a recent publication such results were established for Muller games on finite game graphs [NRZ12].

Bibliography

[AM04] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *STOC*, pages 202–211. ACM, 2004.

[BJW02] Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *ITA*, 36(3):261–275, 2002.

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

[BS96] Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory, Vol.1: Efficient Algorithms*. MIT, Cambridge MA, 1996.

[Büc60] Julius R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[Cac01] Thierry Cachat. Two-way tree automata solving pushdown games. In Grädel et al. [GTW02], pages 303–317.

[CB71] Rina S. Cohen and Janusz A. Brzozowski. Dot-depth of star-free events. *J. Comput. Syst. Sci.*, 5(1):1–16, 1971.

[CDT02] Thierry Cachat, Jacques Duparc, and Wolfgang Thomas. Solving pushdown games with a Σ_3 winning condition. In Julian C. Bradfield, editor, *CSL*, volume 2471 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2002.

[CG77a] Rina S. Cohen and Arie Y. Gold. Theory of omega-languages I. characterizations of omega-context-free languages. *J. Comput. Syst. Sci.*, 15(2):169–184, 1977.

[CG77b] Rina S. Cohen and Arie Y. Gold. Theory of omega-languages II. a study of various models of omega-type generation and recognition. *J. Comput. Syst. Sci.*, 15(2):185–208, 1977.

[CG78] Rina S. Cohen and Arie Y. Gold. Omega-computations on deterministic pushdown machines. *J. Comput. Syst. Sci.*, 16(3):275–300, 1978.

[Chu57] Alonzo Church. Application of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume 1, pages 3–50. Cornell University, 1957.

[Chu63] Alonzo Church. Logic, arithmetic and automata. In *Proceedings of the International Congress of Mathematicians 1962 (Djursholm, Sweden)*, pages 23–35. Institut Mittag-Leffler, 1963.

[CJH04] Krishnendu Chatterjee, Marcin Jurdzinski, and Thomas A. Henzinger. Quantitative stochastic parity games. In J. Ian Munro, editor, *SODA*, pages 121–130. SIAM, 2004.

[COT11] Namit Chaturvedi, Jörg Olschewski, and Wolfgang Thomas. Languages vs. ω-languages in regular infinite games. In Giancarlo Mauri and Alberto Leporati, editors, *Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 180–191. Springer, 2011.

[Dic13] Leonard E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. J. Math.*, 35:413–422, 1913.

[EC82] E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.

[EJ88] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science, FOCS*, pages 328–337. IEEE Computer Society, 1988.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377. IEEE Computer Society, 1991.

[Elg61] Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[Fin01] Olivier Finkel. Topological properties of omega context-free languages. *Theor. Comput. Sci.*, 262(1):669–697, 2001.

[Fin05] Olivier Finkel. On winning conditions of high borel complexity in pushdown games. *Fundam. Inform.*, 66(3):277–298, 2005.

[FLZ11] Wladimir Fridman, Christof Löding, and Martin Zimmermann. Degrees of lookahead in context-free infinite games. In Marc Bezem, editor, *CSL*, volume 12 of *LIPIcs*, pages 264–276. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[FP11] Wladimir Fridman and Bernd Puchala. Distributed synthesis for regular and contextfree specifications. In Filip Murlak and Piotr Sankowski, editors, *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 2011.

[Fri10] Wladimir Fridman. Formats of winning strategies for six types of pushdown games. In Montanari et al. [MNP10], pages 132–145.

[FS05] Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *LICS*, pages 321–330. IEEE Computer Society, 2005.

[FZ10] John Fearnley and Martin Zimmermann. Playing muller games in a hurry. In Montanari et al. [MNP10], pages 146–161.

[FZ12] Wladimir Fridman and Martin Zimmermann. Playing Pushdown Parity Games in a Hurry. In Marco Faella and Aniello Murano, editors, *Proceedings of the Third International Symposium on Games, Automata, Logic, and Formal Verification*, volume 96 of *EPTCScience*, pages 183–196, 2012.

[GG66] Seymour Ginsburg and Sheila A. Greibach. Deterministic context free languages. *Information and Control*, 9(6):620–648, 1966.

[GS53] David Gale and Frank M. Stewart. Infinite games with perfect information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the theory of games, vol. 2*, Annals of Mathematics Studies, vol. 28, pages 245–266. Princeton University Press, 1953.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[HKT10] Michael Holtmann, Łukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. In C.-H. Luke Ong, editor, *FOSSACS*, volume 6014 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2010.

[HL72] Frederick A. Hosch and Lawrence H. Landweber. Finite delay solutions for sequential conditions. In Maurice Nivat, editor, *ICALP*, pages 45–60. North-Holland, Amsterdam, 1972.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[KPV02] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Pushdown specifications. In Matthias Baaz and Andrei Voronkov, editors, *LPAR*, volume 2514 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2002.

[KV97] Orna Kupferman and Moshe Y. Vardi. Synthesis with incomplete information. In *Proceedings of the 2nd International Conference on Temporal Logic (ICTL 97)*, pages 91–106, Manchester, UK, 1997.

[KV99] Orna Kupferman and Moshe Y. Vardi. Church's problem revisited. *Bulletin of Symbolic Logic*, 5(2):245–263, 1999.

[KV00a] Orna Kupferman and Moshe Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2000.

[KV00b] Orna Kupferman and Moshe Y. Vardi. μ-calculus synthesis. In Mogens Nielsen and Branislav Rovan, editors, *MFCS*, volume 1893 of *Lecture Notes in Computer Science*, pages 497–507. Springer, 2000.

[KV01] Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In *LICS*, pages 389–398. IEEE Computer Society, 2001.

[LMS04] Christof Löding, P. Madhusudan, and Olivier Serre. Visibly pushdown games. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 408–420. Springer, 2004.

[Mar75] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, September 1975.

[McN65] Robert McNaughton. Finite-state infinite games. Project MAC Rep., Massachusetts Institute of Technology, Cambridge, September 1965.

[McN93] Robert McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993.

[McN00] Robert McNaughton. Playing infinite games in finite time. In Arto Salomaa, Derick Wood, and Sheng Yu, editors, *A Half-Century of Automata Theory*, pages 73–91. World Scientific, 2000.

[MNP10] Angelo Montanari, Margherita Napoli, and Mimmo Parente, editors. *Proceedings First Symposium on Games, Automata, Logic, and Formal Verification*, volume 25 of *EPTCS*, 2010.

[Mos91] Andrzej W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdansk, 1991.

[MS95] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of rabin, mcnaughton and safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.

[MT01] P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 396–407. Springer, 2001.

[MW81] Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. In Dexter Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 253–281. Springer, 1981.

[MW03] Swarup Mohalik and Igor Walukiewicz. Distributed games. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 338–351. Springer, 2003.

[NRZ12] D. Neider, R. Rabinovich, and M. Zimmermann. Down the Borel Hierarchy: Solving Muller Games via Safety Games. In M. Faella and A. Murano, editors, *Proceedings of the Third International Symposium on Games, Automata, Logic, and Formal Verification, GandALF 2012*, volume 96 of *Electronic Proceedings in Theoretical Computer Science*, pages 169–182, 2012.

[Pos46] E. L. Post. A variant of a recursively unsolvable problem. *Bulletion of the American Mathematical Society*, 52:264–268, 1946.

[PR79] Gary L. Peterson and John H. Reif. Multiple-person alternation. In *FOCS*, pages 348–363. IEEE Computer Society, 1979.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM Press, 1989.

[PR90] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *FOCS*, pages 746–757. IEEE Computer Society, 1990.

[RT07] Alexander Rabinovich and Wolfgang Thomas. Logical refinements of church's problem. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2007.

[Sch08] Sven Schewe. *Synthesis of Distributed Systems*. PhD thesis, Saarland University, 2008.

[Sel07] Victor L. Selivanov. Fine hierarchy of regular aperiodic *omega* -languages. In Tero Harju, Juhani Karhumäki, and Arto Lepistö, editors, *Developments in Language Theory*, volume 4588 of *Lecture Notes in Computer Science*, pages 399–410. Springer, 2007.

[Sel08] Victor L. Selivanov. Fine hierarchy of regular aperiodic omega-languages. *Int. J. Found. Comput. Sci.*, 19(3):649–675, 2008.

[Ser04] Olivier Serre. Games with winning conditions of high borel complexity. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 1150–1162. Springer, 2004.

[SS63] John C. Shepherdson and Howard E. Sturgis. Computability of recursive functions. *J. ACM*, 10(2):217–255, 1963.

[Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1995.

[Tho97] Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of formal languages*, volume Vol. 3, pages 389–455. Springer, 1997.

[Tra61] Boris A. Trakhtenbrot. Конечные автоматы и логика одноместных предикатов (Finite automata and the logic of monadic predicates). *Доклады академии Наук СССР (Proceedings of the USSR Academy of Sciences)*, 140:326–329, 1961.

[Var98] Moshe Y. Vardi. Reasoning about the past with two-way automata. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.

[Wal96] Igor Walukiewicz. Pushdown processes: Games and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV*,

volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1996.

[Wal01] Igor Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

[Zie04] Wieslaw Zielonka. Perfect-information stochastic parity games. In Igor Walukiewicz, editor, *FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 2004.

[Zim12] Martin Zimmermann. *Solving Infinite Games with Bounds*. PhD thesis, RWTH Aachen University, 2012.