Infinite Regular Games in the Higher-Order Pushdown and the Parametrized Setting

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

vorgelegt von

Diplom-Informatikerin
Michaela Slaats

aus Nettetal

Berichter: Universitätprofessor Dr. Dr.h.c. Wolfgang Thomas
Dr. habil. Didier Caucal
Universitätprofessor Dr. Erich Grädel

Tag der mündlichen Prüfung: 23. November 2011

Abstract

Higher-order pushdown systems extend the idea of pushdown systems by using a "higher-order stack" (which is a nested stack). More precisely on level 1 this is a standard stack, on level 2 it is a stack of stacks, and so on. We study the higher-order pushdown systems in the context of infinite regular games.

In the first part, we present a k-ExpTime algorithm to compute global positional winning strategies for parity games which are played on the configuration graph of a level-k higher-order pushdown system. To represent those winning strategies in a finite way we use a notion of regularity for sets of higher-order stacks that relies on certain ("symmetric") operations to build higher-order stacks. The construction of the strategies is based on automata theoretic techniques and uses the fact that the higher-order stacks constructed by symmetric operations can be arranged uniquely in a tree structure.

In the second part, we study the solution of games in the sense of Gale and Stewart where the winning condition is specified by an MSO-formula $\varphi(P)$ with a parameter $P \subseteq \mathbb{N}$. This corresponds to a three player game where the i-th round between the two original players is extended by the choice of the bit 1 or 0 depending on whether $i \in P$ or not. We consider the case that the parameter can be constructed by some deterministic machine, a "parameter generator". We solve the parametrized regular games for parameters P given by two kinds of such generators, namely: higher-order pushdown automata and collapsible pushdown automata.

In the third part, we study higher-order pushdown systems and higher-order counter systems (where the stack alphabet contains only one symbol), by comparing the language classes accepted by corresponding automata. For example, we show that level-k pushdown languages are level-$(k+1)$ counter languages.

Zusammenfassung

Pushdown-Systeme höherer Ordnung (auch Kellersysteme höherer Ordnung genannt) erweitern die Standard-Kellersysteme durch die Nutzung eines „Kellers höherer Ordnung“. Dies ist eine geschachtelte Kellerstruktur; so ist ein Level-1 Keller ein Standardkeller und ein Level-2 Keller ein Keller von Kellern. Wir untersuchen Kellersysteme höherer Ordnung im Kontext von unendlichen regulären Spielen.

Im ersten Teil geben wir einen k-ExpTime Algorithmus an, um globale positionelle Gewinnstrategien für Paritätsspiele zu berechnen, die auf dem Konfigurationsgraphen eines Level-k Kellersystems gespielt werden. Damit diese Strategien in einer endlichen Weise dargestellt werden können, bedarf es einer Definition von Regularität für Mengen von Kellern höherer Ordnung, welche auf der Nutzung von bestimmten („symmetrischen“) Operationen beruht, die gebraucht werden, um einen Keller aufzubauen. Die Konstruktion der Strategien basiert auf automatentheoretischen Techniken und benutzt die Tatsache, dass die Keller höherer Ordnung, welche mit symmetrischen Operationen konstruiert werden, in Form eines Baumes dargestellt werden können.

Im zweiten Teil befassen wir uns mit der Lösung von Spielen im Sinne von Gale und Stewart, bei denen die Gewinnbedingung durch eine MSO-Formel $\varphi(P)$ angegeben wird, wobei $P \subseteq \mathbb{N}$ ein Parameter ist. Dieses Szenarium entspricht einem Spiel mit drei Spielern, bei dem die i-te Runde zwischen den beiden ursprünglichen Spielern dadurch erweitert wird, dass ein Bit hinzugefügt wird, welches signalisiert, ob $i \in P$ oder nicht. Wir betrachten den Fall, dass der Parameter durch eine deterministische Maschine, den „Parametergenerator“, erzeugt wird. Wir lösen diese parameterisierten regulären Spiele für Paramerter P, die durch zwei verschiedene Arten von Generatoren erzeugt werden, und zwar Kellerautomaten höherer Ordnung und „Kollaps“-Kellerautomaten.

Im dritten Teil vergleichen wir Kellersysteme höherer Ordnung und Zählersysteme höherer Ordnung (bei diesen besteht das Kelleralphabet nur aus einem Symbol), durch Analyse der entsprechenden Sprachklassen. Beispielsweise zeigen wir, dass Level-k Kellersprachen auch Level-$(k+1)$ Zählersprachen sind.

Contents

1 Introduction

Two-player games have gained their importance in computer science from their ability to model the interaction between a program and an environment and to synthesize automatically a controller from this model. In this way they are used in the development of hardware and software, for example, in the area of control systems or communication protocols. The development of various tools has also lead to the application of theoretical results in the industrial practice. Another important property that makes two-player games so useful is that they can model nonterminating behavior.

We consider in this thesis infinite games over infinite graphs which are constructed by higher-order pushdown systems. We compute global positional winning strategies for both players in these games. Furthermore, we examine infinite regular games which are enriched by a parameter $P \subseteq \mathbb{N}$ and solve them using a deterministic machine to generate the parameter. Before we give some further insights into the contributions of this thesis we start with some basic knowledge about infinite games and higher-order pushdown systems.

Infinite Games. When talking about two-player games we first of all consider games played infinitely long on some *finite game arena* G. The game arena is given by some finite graph consisting of two kinds of vertices $V = V_0 \cup V_1$ which are connected by directed edges. We draw V_0 vertices as $\bigcirc$. They belong to Player 0. The V_1 vertices are drawn as $\Box$. They belong to Player 1. A *play* starting in some designated vertex v is an infinite path in the game arena, where in every step the player owning the current vertex decides which edge to follow to go to the next vertex. Both players have perfect information about the game arena, as well as of the current position in the game arena and the past of the current play. There is no randomness assumed in the game.

To decide which player wins a play we have to assign some *winning condition* φ to the game and Player 0 has to fulfill the winning condition

to win the play. Player 1 needs to prohibit that the winning condition is fulfilled to win. There are plenty of winning conditions for this kind of games. One of the simplest conditions is *reachability*. In this case a subset F of the vertices of the game graph is given and Player 0 has to reach from the starting vertex one of the vertices in F. This leads directly to another kind of games called *safety games*. There again a subset S of the vertices of the game arena is given and now Player 0 has to achieve that the whole play stays in the part of the game arena given by S. The reachability games are *dual* to the safety games since when Player 0 wants to reach F, Player 1 wants to stay in the region $V \setminus F$. When we consider a winning condition that demands recurring reachability we speak about *Büchi games*. In this case the set $F \subseteq V$ has to be visited again and again in the play for Player 0 to win. The games we consider later most of the time have a more involved winning condition. In this case the winning condition is not defined by a set of vertices but by a function that assigns to each vertex in V a natural number taken from some finite range. These numbers are also called colors. An infinite play is won by Player 0 if the smallest color which is seen infinitely often in the play is even, and by Player 1 if it is odd. If a play is finite then the player which cannot choose a next vertex loses. These games are called *parity games*. Another way to formulate a winning condition is to use a set of sets of vertices. For example in Muller games the winning condition is given by $\mathcal{F} \subseteq 2^V$. A play is won by Player 0 if the set of nodes which appears infinitely often in the play is in $\mathcal{F}$. There are many further winning conditions which we will neither introduce nor use here. We consider here only ω-regular winning conditions. A winning condition is *ω-regular* if the language defined by the winning plays is ω-regular.

We say that a player has a *winning strategy* in a game starting from some vertex if he has a specification for his behavior in the game, such that he wins if he follows this specification no matter what the other player does. Formally, a strategy for Player i ($i \in \{0, 1\}$) is a function that assigns to each play prefix ending in a vertex of Player i the vertex he is supposed to go to next. The *winning region* of a player consists of all vertices from which he has a winning strategy. A special kind of winning strategies are the *positional winning strategies*. In this case the function defining the strategy always depends only on the current vertex in the play and not on the whole play prefix. For example in reachability, safety and Büchi games there always exists a positional

winning strategy for the player who wins. This holds also for parity games [Zie98], which allow to define more complex winning conditions than the before mentioned. The Muller games on the other hand do not have positional winning strategies in general. A survey about infinite games can be found in [GTW01].

Higher-Order Pushdown Systems. Until now we have assumed that the game arena is finite but the whole setting can also be used in the case of an infinite game arena. To construct such an infinite game arena there are several possibilities. The main point is to get a finite representation of the infinite game arena. One common way is to use the configuration graph of a *pushdown system* as an infinite game arena. For the pushdown games it is sufficient to use the states of the pushdown system in the configurations to define the partitioning of the vertices into the ones for Player 0 and Player 1 as well as for the respectively definition of the winning condition [Cac01]. In [Wal96] it has been shown that there exists an ExpTime procedure for finding the winner in a parity pushdown game. Cachat has given in [Cac02] a uniform construction to compute the winning regions for reachability and Büchi pushdown games. Besides this he has shown that the winning regions, respectively, the configurations forming them, are regular. Serre has extended this result in [Ser03] to all kind of ω-regular winning conditions. A great benefit of pushdown graphs, which makes them so interesting as underlying graphs for games is the fact, that they have a decidable *MSO-theory*. This has been shown by Muller and Schupp in [MS85] and bases on Rabins Tree Theorem [Rab69].

The property of having a decidable MSO-theory is still present if we extend the pushdown graphs to *higher-order pushdown graphs* as shown in [CW03]. The higher-order pushdown systems which have been introduced by Maslov in [Mas76] use as storage higher-order stacks, which are nested stacks. For example a level-1 stack is the same as used in the standard pushdown systems, i.e., it contains a sequence of letters. A level-2 stack contains a sequence of level-1 stacks, so it is a stack of stacks. On level 3 we have a stack of stacks of stacks. The configuration graphs of higher-order pushdown systems form a hierarchy of graphs which is equivalent to the *Caucal hierarchy* [Cau02] which has been shown in [CW03, Wöh05, Car06]. The Caucal hierarchy consists of graphs which can be generated from the infinite binary tree by applying

the two processes of MSO-interpretations and unfoldings in alternation. A survey about these graphs can be found in [Tho03]. When working with higher-order stacks there are two different definitions of a set of operations over these stacks and for regularity over sets of stacks. The operations which are identical for both definitions are the *push* and *pop* of a stack symbol in the topmost level-1 stack and the operation $copy_\ell$ which copies the topmost level-ℓ stack of a level-k stack for $\ell < k$ and adds it at the top of the stack. The two different sets of operations differ mainly in the definition of the operation that deletes whole substacks. In the first definition the topmost level-ℓ stack of a level-k stack can be deleted for $\ell < k$ without further conditions except that there is at least one further level-ℓ stack below the deleted one. In the second definition the topmost level-ℓ stack of a level-k stack with $\ell < k$ can only be deleted if the two topmost level-ℓ stacks are equal. In this case we speak about *symmetric operations*. If the first kind of operations is used, then the definition of regularity for sets of stacks, which we call *regular for words*, is given as follows. A set of stacks is called regular for words, if the stacks read as well bracketed words form a set which is regular for words. This notion has been introduced in [BM04]. A different definition of regularity for sets of higher-order stacks was introduced independently in [Car05] and [Fra05a]. It bases on the properties of the symmetric operations and we call it *regular for (symmetric) operations*. In this case a set of higher-order stacks is regular for operations, if it can be obtained by applying a regular set of operation sequences to the empty stack. We need a definition of regularity for sets of higher-order stacks to get a finite representation of a possibly infinite set of higher-order stacks, e.g., when we need to represent the winning regions of a game played over the graph of a higher-order pushdown system.

Contribution 1: Strategies for Higher-Order Pushdown Parity Games. In the first part of the thesis we consider the higher-order pushdown graphs as game graphs. In [Cac03], Cachat has shown that the winner of a parity game defined by a level-k pushdown automaton starting from a given node, can be decided in k-ExpTime. In [CHM+08], it has been shown that when considering higher-order pushdown parity games defined using the unconditional deletion of a substack, the winning region is regular for words. The proof in [CHM+08] also provides a finite description of a winning strategy from a given vertex for one of the

players. The strategy is based on a higher-order pushdown automaton reading the moves of the play and outputting the next move. It is unknown if the notion of regularity for words can be used to describe positional winning strategies for higher-order pushdown parity games. A positional winning strategy for this kind of games means that we can define sets of stacks which are regular for words and these sets define a positional winning strategy for the according vertices. We focus in this work on the definition of higher-order pushdown systems which use the symmetric operations and define regularity for sets of higher-order stacks by using the regularity for operations. We show that we can construct global positional winning strategies for both players in their winning regions. Global strategies are strategies that are valid no matter in which vertex the game is started. For this, we define the positional strategies by sets of higher-order stacks which are regular for operations. We also show how to compute the winning region for both players, which can again be represented by a regular set of higher-order stacks.

Some of the results presented in this part of the thesis have been obtained in collaboration with Arnaud Carayol and have been published in [CS08].

Contribution 2: Parametrized Games. Another kind of games we consider in this thesis have at the first sight a different setting. They are based on a problem that has been stated by Church in [Chu63]. He considered the case that a requirement is given by some formula of a logic which has to be fulfilled by some circuit. The synthesis problem is to find such a circuit or to show that no such circuit exists. In our setting a circuit means a finite automaton with output. This framework can also be defined in a game setting where we have two players, here called Player Input (Player I) and Player Output (Player O). At each moment in time i first Player I chooses a bit $X(i)$ (0 or 1 in the simplest setting but it is also possible to take a letter from some finite alphabet Σ) and then Player O chooses a bit $Y(i)$. We assume here the case of only two bits 0 and 1. A play in this game is an infinite sequence of chosen bit-tuples $X(0)Y(0)X(1)Y(1)\ldots$ which defines via the concept of characteristic functions two sets $X, Y \subseteq \mathbb{N}$. The winning condition for these games is given as an ω-regular language. It can be given for example by some MSO-formula $\varphi(X, Y)$ over the structure $(\mathbb{N}, +1)$. These games are called regular infinite games as the winning condition

is given by some ω-regular language. When a play (X, Y) satisfies the formula $\varphi(X, Y)$ over $(\mathbb{N}, +1)$ Player O wins the play, otherwise Player I wins. If Player O has a winning strategy then in terms of Church's Problem this winning strategy corresponds to a circuit satisfying the specification. If Player I wins then there exists no circuit fulfilling the specification. These connection has been made by Büchi and Landweber in [BL69]. They have shown that given an MSO-formula $\varphi(X, Y)$ as winning condition, the game associated with φ is determined, one can decide who is the winner and construct from φ a finite state winning strategy.

We extend those games by adding a kind of third player which cannot choose his bits freely but follows some fixed behavior. More precisely, we have besides the two sets X and Y which are chosen by Player I and Player O a set $P \subseteq \mathbb{N}$ which is added as a "parameter". Now the winning condition is defined by some MSO-formula of the form $\varphi(P, X, Y)$ over the structure $(\mathbb{N}, +1, P)$. The parameter represents a game context that has some dynamic behavior over time but it is fixed and predictable. The addition of the parameter to the setting of Büchi and Landweber provides a natural step beyond the regular games, where new phenomena arise. We will call these extensions of regular games "parametrized regular games".

Rabinovich has considered parameters P such that $(\mathbb{N}, +1, P)$ has a decidable MSO-theory and showed that in this case an analogue of the Büchi-Landweber Theorem holds. In fact he showed in [Rab06, Rab07] that these parametrized regular games are determined, the winner can be computed and a recursive winning strategy can be constructed from the winning condition. We will first give a proof of this result that uses automata theoretic techniques rather than the composition method applied by Rabinovich.

In the further part of this work we give a more refined analysis. We want obtain sharper results on the format of the strategies, in dependence of the "complexity" of the parameter P. Thus, we are interested in strategies which can be generated by automata of various types, in our case, higher-order pushdown automata or collapsible pushdown automata. To guarantee the existence of such strategies, we demand that the parameter P can be generated by some related kind of deterministic machine. We introduce such "parameter generators", here with a memory structure consisting of higher-order stacks. The general approach to construct

corresponding strategies is based on a game product of a parameter generator and a parity automaton, that defines the winning condition of the parametrized regular game. Then, we apply the memoryless determinacy of parity games. After stating an abstract result on the solution of parametrized regular games we focus on parameters that can be constructed by different kinds of parameter generators and especially concentrate on those where $(\mathbb{N}, +1, P)$ belongs to the Caucal hierarchy, i.e., can be generated by some kind of higher-order pushdown automaton. This holds for example for the set of factorial numbers $n!$, the powers k^n and n^k for a fixed number k. In this case we can reduce the parametrized regular games onto higher-order pushdown parity games and solve them by using the previous results. Furthermore, we will consider the case that the parameters can be constructed by collapsible pushdown automata.

Some of the results presented in this part of the thesis have been obtained in collaboration with Paul Hänsch and Wolfgang Thomas; they have been published in [HST09a, HST09b].

Outlook: Results on Higher-Order Counter Systems. In the last chapter we consider higher-order counters, which are higher-order stacks that have only one stack symbol, i.e., the level-1 stacks reduce to a natural number. We focus on language theoretical aspects. We show that we can construct for each level-k higher-order pushdown automaton a level-$(k+1)$ higher-order counter automaton recognizing the same language. Furthermore, we show for some example languages how the automata recognizing them work.

Acknowledgment

I am heartily thankful to my supervisor, Wolfgang Thomas for his guidance and support through my time of studies. He has always been an inspiration for finding new perspectives and encouragement in my own skills.

Furthermore, I like to thank Arnaud Carayol for the fruitful cooperation and the joint work on parts of this thesis. He aroused my interest to the field of higher-order pushdown systems and his suggestions and comments have been immensely valuably.

I would also like to show my gratitude to Alexander Rabinovich who inspired the work on parametrized regular games and to Paul Hänsch who worked with me on this topic during the writing of his diploma thesis.

Special thanks to to my colleges Namit Chaturvedi, Ingo Felscher, Wladimir Fridman, Christof Löding, Jörg Olschewski, Wied Pakusa, Frank Radmacher, Roman Rabinovich and Stefan Schulz for their help in proofreading. I also like to thank all members of the chairs I7 and MGI for 4 years of collegiality and friendship.

I have to thank the DFG for funding my studies during my time in the research training group AlgoSyn, where I also met many inspiring people and made fruitful experiences.

Finally, I like to thank my family for supporting my idea to study computer science and to start my PhD. I am sorry that in the last months my time for them has decreased so much. I also would like to show my gratitude to my friends for supporting me and especially to Michal who told me several times that she believes in me and my skills and so let myself believe in it.

2 Preliminaries

In this chapter, we introduce the basic notions and facts used in this thesis. We start with the definition of different kinds of word automata and the monadic second-order logic. Afterwards, we define games of infinite duration played on finite game graphs and consider especially parity games. Furthermore, we give some fundamental information about infinite trees and tree automata. In the second part of the preliminaries, we introduce higher-order stacks and the operations used to manipulate them. Afterwards, we give a definition of regularity for sets of higher-order stacks and introduce different automata models that use higher-order stacks as storage mechanism.

2.1 Notations and Facts on Automata, Games and MSO

An *alphabet* is a finite, nonempty set of symbols respectively letters. For an alphabet Σ the set of all *finite words* over Σ is denoted by Σ^*, and the set of all *infinite words* over Σ by Σ^ω. We write ε for the *empty word*. For some finite word w we denote by $|w|$ its length, and for $a \in \Sigma$ by $|w|_a$ the number of a occurring in w. For some finite word $a = a_1 a_2 \dots a_n$ (respectively, some infinite word $a = a_1 a_2 a_3 \dots$) we denote by $a(i)$ the i-th letter a_i of a. For two words $a = a_1 a_2 \dots a_n$ and $b = b_1 b_2 \dots b_m$ we define a to be a prefix of b, written $a \sqsubseteq b$, if $a_i = b_i$ for $1 \leq i \leq n$ and $m \geq n$. A *word language* $\mathcal{L}$ is a subset of Σ^* or Σ^ω. In the second case we speak about an ω-language.

The set of *regular expressions* over an alphabet Σ is defined inductively as follows. The atomic regular expressions are $\emptyset$, ε, and a for all $a \in \Sigma$. They define the languages $L(\emptyset) := \emptyset$, $L(\varepsilon) := \{\varepsilon\}$, and $L(a) := \{a\}$. If r_1, r_2 are regular expressions then $r_1 + r_2$, $r_1 r_2$ and r_1^* are regular expressions, as well. They define the union $L(r_1 + r_2) = L(r_1) \cup L(r_2)$, the concatenation $L(r_1 r_2) = L(r_1)L(r_2)$, and the Kleene star $L(r_1^*) = L(r_1)^*$. We denote by $\mathrm{Reg}(\Sigma^*)$ the set of regular expressions over the alphabet

Σ. A language $L \subseteq \Sigma^*$ is called *regular* if and only if there exists a regular expression r with $L(r) = L$. For ω-languages the regular expressions are defined analogously. An *ω-regular expression* is of the form $r_1 s_1^\omega + r_2 s_2^\omega + \ldots + r_n s_n^\omega$ where r_i, s_i are the standard regular expression and $L(s_i^\omega) := L(s_i)^\omega$ for $i \in [1, n]$. An ω-language $L \subseteq \Sigma^\omega$ is called *ω-regular* if and only if there exists an ω-regular expression r with $L(r) = L$.

We denote the natural numbers including 0 by $\mathbb{N}$. The power set of a set S is written as $\mathcal{P}(S)$.

Next, we define a finite-state transducer in form of a *Mealy automaton*. As input it reads a finite word over some alphabet Σ_I and returns another word over the output alphabet Σ_O. Note, that that both words have the same length. By adding a set of accepting states we can restrict to those outputs for which the input is accepted by the automaton.

Definition 2.1. A *Mealy automaton* $\mathcal{A}$ is defined by a 7-tuple $(Q, \Sigma_I, \Sigma_O, \delta, \lambda, q_0, F)$ where Q is a finite set of states, Σ_I is an input alphabet, Σ_O is an output alphabet, $\delta\colon Q \times \Sigma_I \to Q$ is a transition function, $\lambda\colon Q \times \Sigma_I \to \Sigma_O$ is an output function, $q_0 \in Q$ is an initial state and $F \subseteq Q$ is a set of final states.

The automaton reads a word $i_1 i_2 \ldots i_n \in \Sigma_I$ as input and produces a word $o = o_1 o_2 \ldots o_n \in \Sigma_O$ as output. The run of $\mathcal{A}$ on $i_1 i_2 \ldots i_n$ starts in q_0 and for each $j \in [0, n-1]$ we have $\delta(q_j, i_{j+1}) = q_{j+1}$ and $\lambda(q_j, i_{j+1}) = o_{j+1}$. The run is called accepting if $q_n \in F$.

We proceed with two types of automata that recognize ω-languages, i.e., subsets of Σ^ω. We start with Büchi-automata and afterwards we introduce parity automata.

Definition 2.2. A *Büchi (word) automaton* $\mathcal{A}$ is defined by the tuple $(Q, \Sigma, q_0, \Delta, F)$ where Q is a finite set of states, q_0 is an initial state, Σ is a word alphabet, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation and $F \subseteq Q$ is a set of final states.

A run ρ of $\mathcal{A}$ on an infinite word $\alpha = \alpha(0)\alpha(1)\alpha(2)\ldots \in \Sigma^\omega$ is an infinite word $\rho = \rho(0)\rho(1)\rho(2)\ldots$ over Q with $\rho(0) = q_0$ and for all $i \in \mathbb{N}$ we have $(\rho(i), \alpha(i), \rho(i+1)) \in \Delta$.

A run is *accepting* if at least one state of F appears infinitely often in this run, i.e., $\forall i\, \exists j.\, j \geq i \wedge \rho(j) \in F$.

Next, we define parity automata which recognize infinite words, as the Büchi automata but with a different acceptance condition. First,

we consider the nondeterministic version and afterwards deterministic parity automata.

Definition 2.3. A *finite parity (word) automaton* $\mathcal{A}$ is defined by the tuple $(Q, \Sigma, q_0, \Delta, \Omega)$ where Q is a finite set of states, q_0 is an initial state, Σ is a word alphabet, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation and $\Omega\colon Q \to \{0, \ldots, n\}$ is a coloring of Q for some fixed number $n \in \mathbb{N}$.

A run ρ of $\mathcal{A}$ on an infinite word $\alpha = \alpha(0)\alpha(1)\alpha(2)\ldots \in \Sigma^\omega$ is an infinite sequence of states $\rho = \rho(0)\rho(1)\rho(2)\ldots \in Q^\omega$ with $\rho(0) = q_0$ and $(\rho(i), \alpha(i), \rho(i+1)) \in \Delta$ for all $i \in \mathbb{N}$.

A run is *accepting* if the smallest color which is seen infinitely often in the run is even, i.e., $\min\{\Omega(q)) \mid q \in \mathrm{Inf}(\rho)\}$ is even, where $\mathrm{Inf}(\rho) = \{q \in Q \mid q \text{ occurs infinitely often in } \rho\}$.

Definition 2.4. A finite parity word automaton is *deterministic* if we have, instead of a transition relation Δ, a transition function $\delta\colon Q \times \Sigma \to Q$, and in the run ρ we have $\delta(\rho(i), \alpha(i)) = \rho(i+1)$ for all $i \in \mathbb{N}$.

In the following we recall monadic second-order logic (MSO-logic); for details see [GTW01]. In this thesis we restrict to MSO-logic evaluated over the structures $(\mathbb{N}, +1)$ and $(\mathbb{N}, +1, P)$ where $P \subseteq \mathbb{N}$, hence to signatures $\{+1, P\}$.

A formula of the *monadic second-order logic* over the signature $\{+1, P\}$ is built up from a set of first order variables $x, y, z, \ldots$, second order variables $X, Y, Z, \ldots$, atomic formulas as $x = y$, $x + 1 = y$, $X(y)$, $P(x)$, the usual Boolean connectives $\neg$, $\vee$, $\wedge$ and the quantifiers $\exists$, $\forall$.

We state a known fact which will be needed later in this work. For details see [GTW01].

Proposition 2.5. *Each MSO-formula can be transformed into an equivalent (deterministic) finite parity automaton.*

2.1.1 Parity Games

An *(infinite) game* $\mathcal{G}$ is played between two players, which we call *Player 0* and *Player 1*. It is defined by a game graph and the formulation of a winning condition for Player 0. The *game graph* is given as a tuple (V_0, V_1, E), where V_i is the set of vertices of Player i for $i \in \{0, 1\}$ and $E \subseteq (V_0 \cup V_1) \times (V_0 \cup V_1)$ is the edge relation. We will focus in this section only on *parity games* and explain them, but the definitions can

easily be adapted to other winning conditions. The *winning condition* for a parity game is defined for some fixed number $n \in \mathbb{N}$ by a function $\Omega\colon (V_0 \cup V_1) \to \{1, \ldots, n\}$, also called coloring.

Player 0 and Player 1 *play* in $\mathcal{G}$ by moving a token from one vertex to another along the edges. A play π starts in some initial vertex v_0 and proceeds as follows: the player owning v_0 moves the token to a vertex v_1 such that $(v_0, v_1) \in E$. Then the player owning v_1 chooses a successor v_2, and so on. If at some point in the play one of the players cannot move, i.e., for the current vertex v there is no vertex v' such that $(v, v') \in E$, then the player owning v loses the play. Otherwise, the play goes on forever and yields an infinite word $\pi \in (V_0 \cup V_1)^\omega$. The play is *won* by Player 0 if the smallest color that appears infinitely often in π is even; otherwise Player 1 wins.

A *strategy* for Player i is a partial function $\varphi_i\colon (V_0 \cup V_1)^*(V_i) \to (V_0 \cup V_1)$ assigning to a play prefix ending in some vertex $v \in V_i$ a vertex v' such that $(v, v') \in E$. Player i plays according to a strategy φ_i in some play $\pi = v_0 v_1 v_2 \cdots$ if $v_{j+1} = \varphi_i(v_0 \cdots v_j)$, for all $j \geq 0$ where $v_j \in V_i$. A strategy φ_i for Player i is called *winning* from an initial position $v \in V_0 \cup V_1$ if Player i wins every play starting from v if he plays according to his strategy φ_i. A strategy φ_i for Player i is called *positional* if the value $\varphi_i(v_0 \ldots v_j)$ only depends on the last vertex v_j. Hence, a positional strategy is presentable as a partial function from V_i to $V_0 \cup V_1$. A vertex $v \in V_0 \cup V_1$ is *winning* for Player i if he has a winning strategy from v. We set of all vertices that are winning for one player is called *winning region*; so the winning region W_i of Player i consists of all winning vertices of Player i.

Parity games are *positional determined* [EJ91, Zie98]. This means that from every vertex either Player 0 or Player 1 has a positional winning strategy. This result can be extended to the assertion that we can find a *global positional winning strategy* φ_i for Player i such that φ_i is winning for Player i from all vertices in $\mathrm{Dom}(\varphi_i)$ where $\mathrm{Dom}(\varphi_i) = W_i \cap V_i$ (see [Tho97, Zie98]). From this results we can state the following known fact:

Proposition 2.6. *Parity games (even over infinite game arenas) are determined, and the winner has a (global) positional winning strategy.*

Example 1. In Figure 2.1 an example for a parity game is given. The vertices of Player 0 are depicted as circles and the vertices of Player 1 as squares, i.e., $V_0 = \{2, 4, 5, 6\}$ and $V_1 = \{1, 3, 7, 8\}$. The number inside a

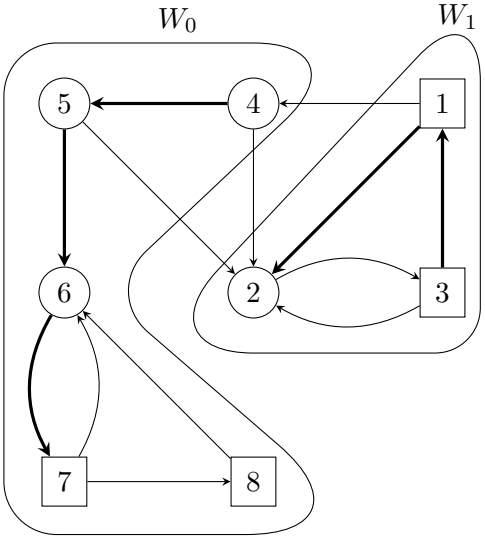

Figure 2.1: Parity Game with winning regions and positional strategies.

vertex represents its color. Usually, different vertices can have the same color, but to identify the vertices uniquely in this example, we chose each color only once. In the figure, the winning regions for Player 0 and Player 1 are marked. We have $W_0 = \{4, 5, 6, 7, 8\}$ and $W_1 = \{1, 2, 3\}$. The global positional winning strategy for Player 0 in W_0 is $\varphi(4) = 5$, $\varphi(5) = 6$ and $\varphi(6) = 7$. The global positional winning strategy for Player 1 in W_1 is $\varphi(1) = 2$ and $\varphi(3) = 1$.

2.1.2 Tree Automata Models

We introduce two automata models that run over infinite trees. First, we introduce the notion of *infinite trees*, called W-trees here, where W is a finite set of directions in the tree. Formally, a W-tree t is a non-empty prefix-closed subset of W^*, which means that if $w.d \in \mathrm{Dom}(t)$, where $w \in W^*$ and $d \in W$, then also $w \in \mathrm{Dom}(t)$. We call the elements in the tree t nodes; the empty word ε is the root of t. The direction of a node $w.d$ with $d \neq \varepsilon$ is d. For a direction $d \in W$, a node $wd \in \mathrm{Dom}(t)$ is a *d-son* of $w \in \mathrm{Dom}(t)$ and w is the *parent* of wd. The full infinite W-tree is W^*. In a tree t a finite path has the form $\beta = w_0 w_1 \dots w_n$ and an infinite path is given by $\beta' = w_0 w_1 w_2 \dots$, where for all $i \in \mathbb{N}$, $w_i \in \mathrm{Dom}(t)$ and there exists $d \in W$ such that $w_{i+1} = w_i.d$.

We introduce labelings of trees by an finite alphabet Σ. A *Σ-labeled W-tree* t can be considered as a partial function from W^* to Σ. Let Ξ be another finite alphabet, a Ξ-labeling of a Σ-labeled W-tree t is a $\Sigma \times \Xi$-labeled W-tree t' such that $\mathrm{Dom}(t) = \mathrm{Dom}(t')$ and for $w \in \mathrm{Dom}(t')$ we have $t'(w) = (t(w), \sigma)$ for some $\sigma \in \Xi$.

Before we introduce *two-way alternating parity tree automata* we give some intuition on how they work. Whenever a two-way alternating parity tree automaton is in a node of the input tree it can send several copies of itself to the sons of the node, to the parent node or to the node itself. In order to simulate navigation through the tree we introduce the set $\mathrm{ext}(W) := W \uplus \{\varepsilon, \uparrow\}$ of extended directions. By the symbol $\uparrow$ we indicate that the automaton should go to the parent node, by ε we want the automaton to stay at the current node, and for all symbols $d \in W$ the automaton should proceed to the respective d-son. We have for all $w \in W^*, d \in W$ that $w.\varepsilon = w$ and $wd \uparrow = w$. Note, that the node $\varepsilon \uparrow$ is not defined.

In the following Chapter 3, we will have to deal with incomplete W-trees where $\mathrm{Dom}(t) \neq W^*$. In these cases we assume that the labeling of a tree provides the directions to all sons of a node in the tree. So the automata run on $\mathcal{P}(W) \times \Sigma$-labeled W-trees t where for all $w \in \mathrm{Dom}(t)$, $t(w) = (\theta_w, \sigma_w)$ where $\theta_w = \{d \in W \mid wd \in \mathrm{Dom}(t)\}$.

Definition 2.7. A *two-way alternating parity tree automaton* (`2-PTA` for short) running over $\mathcal{P}(W) \times \Sigma$-labeled W-trees is defined by a tuple $\mathcal{A} = (Q, \Delta, I, \Omega)$ where Q is a finite set of states, $\Delta \subseteq Q \times (\mathcal{P}(W) \times \Sigma) \times \mathcal{P}(\mathrm{ext}(W) \times Q)$ is a transition relation, $I \subseteq Q$ is a set of initial states and $\Omega \colon Q \to \{1, \ldots, n\}$ is a coloring for some number $n \in \mathbb{N}$.

A transition $(q, (\theta, \sigma), \{(d_1, q_1), \ldots, (d_n, q_n)\}) \in \Delta$ will be written as $(q, (\theta, \sigma)) \to (d_1, q_1) \wedge \ldots \wedge (d_n, q_n)$. We assume that the set $\{d_1, \ldots, d_n\}$ is a subset of $\theta \cup \{\uparrow, \varepsilon\}$. The idea of such a transition is, that if the automaton is in state q in a node w with $t(w) = (\theta, \sigma)$ then it sends a copy into the direction d_i (i.e. into the node wd_i) with state q_i for all $i \in [1, n]$.

A run of a two-way alternating parity tree automaton $\mathcal{A}$ over a $\mathcal{P}(W) \times \Sigma$-labeled W-tree is another $W^* \times Q$-labeled tree r. Basically, this tree is an unfolding of the run where each node is a copy of the automaton in some node w of W^* and some state $q \in Q$. As the automaton can also send copies into the parent node and to the node itself, many nodes in r

can refer to the same node in W^*. A run r which is a $W^* \times Q$-labeled Υ-tree, where Υ is some almost arbitrary set of directions, has to fulfill the following conditions:

1. $\varepsilon \in \mathrm{Dom}(r)$ and $r(\varepsilon) = (\varepsilon, q_I)$ for $q_I \in I$.
2. If $y \in \mathrm{Dom}(r)$ with $r(y) = (w, q)$ then there is a transition $(q, t(w)) \to (d_1, q_1) \wedge \ldots \wedge (d_n, q_n)$ in Δ and for all $i \in [1, n]$ there is a $\mu_i \in \Upsilon$ such that $y.\mu_i \in \mathrm{Dom}(r)$ and $r(y.\mu_i) = (w.d_i, q_i)$.

A run is accepting if all its infinite paths satisfy the parity winning condition given by Ω, i.e., the smallest color which is seen infinitely often in the path is even.

We will use in the following a different characterization of the behavior of a 2-PTA which relies on a game based approach. The behavior of a 2-PTA $\mathcal{A} = (Q, \Delta, I, \Omega_{\mathcal{A}})$ over a $\mathcal{P}(W) \times \Sigma$-labeled W-tree t is now given by the parity game over the graph $\mathcal{G}_{\mathcal{A},t} = (V_0, V_1, E, \Omega')$ played between two players called Automaton and Pathfinder. The set V_0 of vertices of Automaton is $\mathrm{Dom}(t) \times Q$ and the set V_1 of vertices of Pathfinder is $\mathrm{Dom}(t) \times \Delta$. For all $w \in \mathrm{Dom}(t)$ and $q \in Q$, there is an edge $((w, q), (w, \delta)) \in E$ for each transition $\delta \in \Delta$ of the form $(q, t(w)) \to P$. Conversely for each transition $\delta = (q, t(w)) \to P \in \Delta$, there is an edge $((w, \delta), (wd_i, q_i))$ for each pair $(d_i, q_i) \in P$. We define $\Omega'((w, q)) = \Omega(q)$ for $(w, q) \in V_0$ and $\Omega'((w, \delta)) = \Omega(q)$ if $\delta = (q, t(w)) \to (d_1, q_1) \wedge \ldots \wedge (d_n, q_n)$ and $(w, \delta) \in V_1$.

Lemma 2.8. *The tree automaton $\mathcal{A}$ accepts the tree t iff in the parity game over $\mathcal{G}_{\mathcal{A},t}$* Automaton *wins from some vertex in $\{\varepsilon\} \times I$.*

The idea is that Automaton chooses repeatedly a transition to show that the tree t is accepted by $\mathcal{A}$ and Pathfinder chooses from the respective transition a pair consisting of a direction and state, to show that t is not accepted.

We will now introduce a special case of the 2-PTA where we only allow transitions where a copy of the automaton is sent only once to each son of the current node. Hence, the run of this automaton on an input tree has the same form as the input tree itself.

Definition 2.9. A one-way (non-deterministic) parity tree automaton (1-PTA) coincides with a 2-PTA where every transition is of the form

$$(q, (\theta, \sigma)) \to (d_1, q_1) \wedge \ldots \wedge (d_n, q_n),$$

where for all $i, j \in [1, n]$, $d_i, d_j \in W$ and $d_i = d_j$ implies $i = j$.

A 1-PTA is *deterministic* if for all $q \in Q$, $\theta \in \mathcal{P}(W)$ and $\sigma \in \Sigma$ there exists at most one transition $(q, (\theta, \sigma)) \to (d_1, q_1) \wedge \ldots \wedge (d_n, q_n)$ in Δ.

2.2 Higher-Order Pushdown Fundamentals

In this section we define the basics about higher-order stacks, the operations which are used to manipulate them, regularity for sets of higher-order stacks and different automata models which use these stacks as storage.

2.2.1 Higher-Order Stacks

We use an inductive definition to introduce higher-order stacks. A *level-1 stack* over some finite alphabet Γ can be represented as a word from Γ^*. The *empty level-1 stack* corresponds to the empty word ε and is denoted by $[\,]_1$. We write $\text{Stacks}_1(\Gamma) := \Gamma^*$ for the *set of all level-1 stacks* over Γ.

A *level-(k+1) stack* for $k \geq 1$ is defined as a nonempty sequence of level-k stacks. The *empty level-(k+1) stack* (written $[\,]_{k+1}$) is the stack that contains only the empty stack of level k, i.e., $[\,]_{k+1} = [[\ldots[\,]_1 \ldots]_k]_{k+1}$. The *set of all level-(k+1) stacks* is defined by $\text{Stacks}_{k+1}(\Gamma) := (\text{Stacks}_k(\Gamma))^+$. A level-($k$+1) stack s that consists of the sequence $s_1, \ldots, s_n$ of level-k stacks will be written as $[s_1, \ldots, s_n]_{k+1}$. We use the convention that s_n is the topmost level-k stack in s.

To address the topmost stack we define the function top_i for each level $i \geq 0$ by:

$$\begin{array}{lcll} top_0([\,]_1) & = & \text{non-defined} & \\ top_0([a_1, \ldots, a_n]_1) & = & a_n & \\ top_k([s_1, \ldots, s_n]_{k+1}) & = & s_n & \text{for } k \geq 1 \\ top_k([s_1, \ldots, s_n]_l & = & top_k(s_n) & \text{for } l > k+1 \end{array}$$

Example 2. An illustration of a level-3 stack can be found in Figure 2.2 to get an impression how higher-order stacks can be represented. Furthermore, we show some stacks and the application of the function *top* in the following examples:

$$\begin{array}{lcl} top_0([[abb][bba]]_2) & = & a \\ top_1([[abb][bba]]_2) & = & [bba]_1 \\ top_1([[abb][\,]]_2) & = & [\,]_1 \end{array}$$

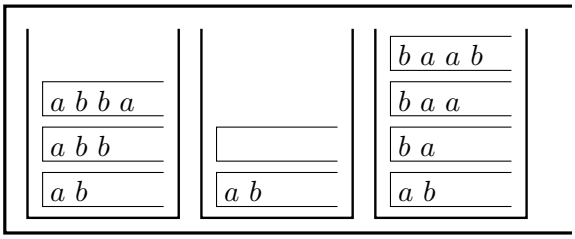

Figure 2.2: The level-3 stack $[[[ab][abb][abba]]\ [[ab][\ ]]\ [[ab][ba][baa][baab]]]_3$ is illustrated.

2.2.2 Operations

Next, we define some *operations* to process higher-order stacks. In this thesis we use the "symmetric" variant of higher-order operations. This means for every operation γ there is a *symmetric operation* $\bar{\gamma}$ which reverses the the effect of γ.

On level 1 we have the usual *push* and *pop* operations, where for *pop* the symbol which will be deleted has to be declared. For a level-1 stack $s = [a_1, \ldots, a_n]$ with $a_i \in \Gamma$ and all $x \in \Gamma$, we define:

$$\begin{aligned} push_x([a_1, \ldots, a_n]_1) &= [a_1, \ldots, a_n, x]_1, \\ pop_x([a_1, \ldots, a_n]_1) &= \begin{cases} [a_1, \ldots, a_{n-1}]_1 & \text{if } a_n = x \\ \text{non-defined} & \text{otherwise} \end{cases} \end{aligned}$$

On each level $(k{+}1) \geq 2$ we have the operation $copy_k$ which copies the topmost level-k stack and adds it to the level-$(k{+}1)$ stack. Additional we have the symmetric operation $\overline{copy_k}$ which deletes the topmost level-k stack but only if it is equal to the topmost but one level-k stack. For completeness we also define the non symmetric operation $destr_k$, which deletes the topmost level-k stack if there is at least one more level-k stack below it. Formally these operations are defined for a level-$(k{+}1)$ stack $s = [s_1, \ldots, s_n]_{k+1}$ by:

$$\begin{aligned} copy_k([s_1, \ldots, s_n]_{k+1}) &= [s_1, \ldots, s_n, s_n]_{k+1} \\ \overline{copy_k}([s_1, \ldots, s_{n-1}, s_n]_{k+1}) &= \begin{cases} [s_1, \ldots, s_{n-1}]_{k+1} & \text{if } s_n = s_{n-1} \\ \text{non-defined} & \text{otherwise} \end{cases} \\ destr_k([s_1, \ldots, s_{n-1}, s_n]_{k+1}) &= \begin{cases} [s_1, \ldots, s_{n-1}]_{k+1} & \text{if } n > 1 \\ \text{non-defined} & \text{otherwise} \end{cases} \end{aligned}$$

Besides these operations we have for each level k also a *test for emptiness* which is formally defined by:

$$T_{[\,]_k}(s) = \begin{cases} s & \text{if } s = [\,]_k \\ \text{non-defined} & \text{otherwise} \end{cases}$$

An operation γ of level k is extended to stacks of level $\ell > k$ using the definition $\gamma([s_0, \ldots, s_n]_\ell) = [s_0, \ldots, \gamma(s_n)]_\ell$. If we want to apply a sequence of operations $\mu = \gamma_1 \ldots \gamma_n$ onto some stack s we write $\mu(s)$ respectively $\gamma_1 \ldots \gamma_n(s)$. In this case first γ_1 is applied to s then γ_2 and so on.

Example 3. Consider $\Gamma = \{a, b\}$ and $s = [[[ab][\,]\,][[ab][ba][baa][baab]]]_3$. Now, we show the application of different stack operations onto s:

$$\begin{aligned}
&[[[ab][\,]\,][[ab][ba][baa][baab]]]_3 \xrightarrow{pop_b} [[[ab][\,]\,][[ab][ba][baa][baa]]]_3 \\
\xrightarrow{\overline{copy_1}}\ &[[[ab][\,]\,][[ab][ba][baa]]]_3 \xrightarrow{pop_a} [[[ab][\,]\,][[ab][ba][ba]]]_3 \\
\xrightarrow{\overline{copy_1}}\ &[[[ab][\,]\,][[ab][ba]]]_3 \xrightarrow{pop_a} [[[ab][\,]\,][[ab][b]]]_3 \\
\xrightarrow{pop_b}\ &[[[ab][\,]\,][[ab][\,]]]_3 \xrightarrow{T_{[\,]_1}} [[[ab][\,]\,][[ab][\,]]]_3 \\
\xrightarrow{\overline{copy_2}}\ &[[[ab][\,]]]_3 \xrightarrow{push_a} [[[ab][a]]]_3 \\
\xrightarrow{copy_1}\ &[[[ab][a][a]]]_3 \xrightarrow{copy_2} [[[ab][a][a]]\,[[ab][a][a]]]_3 \\
\xrightarrow{push_a}\ &[[[ab][a][a]]\,[[ab][a][aa]]]_3 \xrightarrow{destr_2} [[[ab][a][a]]]_3
\end{aligned}$$

Furthermore, we have $push_a push_b copy_1 pop_b([\,]_2) = [[ab][a]]_2$.

The *set of symmetric operations* $Ops_{k+1}(\Gamma)$ for some level $(k+1)$ is defined inductively by:

$$\begin{aligned}
Ops_1(\Gamma) &= \{push_x, pop_x \mid x \in \Gamma\} \cup \{T_{[\,]_1}\} \\
Ops_{k+1}(\Gamma) &= Ops_k(\Gamma) \cup \{copy_k, \overline{copy_k}, T_{[\,]_{k+1}}\}
\end{aligned}$$

Moreover, we denote by Ops_k^* the *monoid for the composition* of partial functions generated by $Ops_k(\Gamma)$. We will often write only Ops_k instead of $Ops_k(\Gamma)$ when Γ is clear from the context.

2.2.3 Instructions

Sometimes, we use a symbolic representation for the operations as some kind of abbreviation. We call those symbols *instructions*. We define

analogously to $Ops_k(\Gamma)$ the *set of instructions* by

$$\begin{aligned} \Gamma_1 &= \{a, \overline{a}, \bot_1 \mid a \in \Gamma\}, \\ \Gamma_{k+1} &= \Gamma_k \cup \{k, \overline{k}, \bot_{k+1}\} \end{aligned}$$

where a means $push_a$, $\overline{a}$ is the short form of pop_a, k represents $copy_k$, $\overline{k}$ is the abbreviation of $\overline{copy_k}$ and $\bot_k$ stands for $T_{[\,]_k}$. Furthermore, we define the set $\Gamma_k^T = \{\bot_\ell \mid \ell \in [1,k]\}$ which contains only the tests for emptiness and the set $\Gamma_k^O = \Gamma_k \setminus \Gamma_k^T$ which contains all other instructions which really change the stacks. We extend the bar notation to all symbols $\gamma \in \Gamma_k^O$ by taking $\overline{\overline{\gamma}} = \gamma$.

Example 4. Consider the stack alphabet $\Gamma = \{a, b, c\}$. The set of all instructions of level 3 is given by the two following sets where we differ between the operating instructions and the tests for emptiness; $\Gamma_3^O = \{a, b, c, \bar{a}, \bar{b}, \bar{c}, 1, \bar{1}, 2, \bar{2}\}$ and $\Gamma_3^T = \{\bot_1, \bot_2, \bot_3\}$. We can now write the instruction sequence $ab1\bar{b}$ instead of the operation sequence $push_a push_b copy_1 pop_b$.

We can use sequences of instructions to define stacks. If we start with the empty level-k stack of some level k and apply an instruction sequence, we get a unique level-k stack or the information that the stack is not defined for the case that one of the instructions is not defined along the way. For the reverse direction this is not that easy. If we want to give for some stack $s \in \mathrm{Stacks}_k(\Gamma)$ an instruction sequence that generates s starting from $[\,]_k$ it is not the case that we have a unique instruction sequence. Consider for example the stack $[[ab][aa]]_2$ it can be generated by $aa\bar{a}b1\bar{b}a1b\bar{b}\bar{1}$ as well as by $ab1\bar{b}a$. In the first case we see that there are some unnecessary instructions in the sequence which generate "loops" as by $a\bar{a}$ and $1b\bar{b}\bar{1}$ we come back to a stack that has been produced before. The second instruction sequence does not contain such loops. An instruction sequence that does not contain any loop will be called *reduced*. More formally a sequence $\mu \in \Gamma_k^*$ is *reduced* if for all stacks $s \in \mathrm{Stacks}_k(\Gamma)$ holds that there are no two sequences $\mu', \mu'' \in \Gamma_k$ such that $\mu' \sqsubseteq \mu$, $\mu'' \sqsubseteq \mu$ and $\mu'(s) = \mu''(s)$. Instead of this global property it is easier to check the following local property.

Definition 2.10. A sequence $\mu \in \Gamma_k^*$ is called *reduced* if μ contains neither a test in Γ_k^T nor the instruction $\overline{k-1}$ nor a subsequence of the form $\gamma\bar{\gamma}$ with $\gamma \in \Gamma_k^O$.

It is easy to transform a non reduced sequence stepwise into a corresponding reduced sequence. First, delete all tests. Afterwards, delete all subsequences of the form $\gamma\bar{\gamma}$. The deletion of $\gamma\bar{\gamma}$ has to be done repeatedly as these sequences can be nested. Note, that the transformation into a reduced sequence does in general not preserve the interpretation. Take for example the sequence $1b\bar{a}ab$ and its reduced sequence $1bb$. It holds for all stacks $s \in \mathrm{Stacks}_k(\Gamma)$ that $1bb(s)$ is defined, but there exists no stack $s \in \mathrm{Stacks}_k(\Gamma)$ such that $1b\bar{a}ab(s)$ is defined. Nevertheless the reduced sequences are very useful to describe stacks in a unique way by instruction sequences. For this we cite a result of Carayol:

Proposition 2.11 ([Car06])**.** *For all level $k \geq 1$ and all stacks $u, v \in \mathrm{Stacks}_k(\Gamma)$, there exists a unique reduced sequence $\rho_{u,v} \in \Gamma_k$ such that $v = \rho_{u,v}(u)$.*

A direct conclusion of Proposition 2.11 is that for each stack $s \in \mathrm{Stacks}_k(\Gamma)$ there exists a unique reduced sequence ρ_s which constructs s starting from the empty level-k stack.

Definition 2.12. For each stack $s \in \mathrm{Stacks}_k(\Gamma)$ let $\rho_s \in \Gamma_k^*$ be the *unique reduced sequence* such that $s = \rho_s([\,]_k)$.
Furthermore, we define a function $\mathrm{Last}\colon \mathrm{Stacks}_k(\Gamma) \to \Gamma_k^O \cup \{\diamond\}$ which returns for each stack s the last instruction of the reduced sequence ρ_s and if the stack $s = [\,]_k$, it returns $\diamond$.

$$\mathrm{Last}(s) = \begin{cases} \gamma_n & \text{if } s \neq [\,]_k \text{ and } \rho_s = \gamma_1 \dots \gamma_n \\ \diamond & \text{otherwise.} \end{cases}$$

The reduced sequence ρ_s of a stack s has the nice property that it is also the shortest unique sequence to produce the stack. Note, that if we consider instead of the symmetric operations the classical operations which contain $destr_k$ instead of $\overline{copy_k}$ for all $k \geq 1$, then this no longer holds. Take for example the level-3 stack $[[[a][abb]]\,[[a][a]]]_3$ which is generated by the reduced sequence $a1bb2\bar{b}\bar{b}$. It can be generated by the following 2 sequences of classical operations

$$\begin{aligned} [[[a][abb]]\,[[a][a]]]_3 &= push_a copy_1 push_b push_b copy_2 pop_b pop_b([\,]_3) \\ &= push_a copy_1 push_b push_b copy_2 destr_1 push_a([\,]_3) \end{aligned}$$

which have both the same length.

The following lemma shows what happens with level-$(k+1)$ stacks if we delete the instruction 1 from a instruction sequence that is applied

to it. More precisely, it shows that when we do not use k and $\overline{k}$ in an instruction sequence we only change the topmost level-k stack.

Lemma 2.13. *For every instruction sequence $\mu \in (\Gamma_{k+1} \setminus \{\overline{k}, \bot_{k+1}\})^*$ we define $\tilde{\mu} \in \Gamma_k^*$ to be the sequence that we get if we delete from μ all k. Then it holds for every stack $s \in \text{Stacks}_{k+1}(\Gamma)$ that $\mu(s)$ is defined if and only if $\tilde{\mu}(top_k(s))$ is defined. Furthermore we have $top_k(\mu(s)) = \tilde{\mu}(top_k(s))$ if both stacks are defined.*

Proof. We proof the lemma by an induction over the length of the instruction sequence μ. For the induction start let $|\mu| = 0$, i.e., μ is the empty instruction sequence. Then the claims follow directly. For the induction step assume $|\mu| = n + 1$, $\mu = \mu'\gamma$ and for μ' holds the induction hypothesis that for all stacks $s \in \text{Stacks}_{k+1}(\Gamma)$, $\mu'(s)$ is defined if and only if $\tilde{\mu'}(top_k(s))$ is defined and that $top_k(\mu'(s)) = \tilde{\mu'}(top_k(s))$ if both stacks are defined. Now assume $\gamma = k$. Note, that the application of a *copy* is defined for all stacks and that for a level-$(k+1)$ stack the operation $copy_k$ just copies the topmost level-k stack and does not change it. So, we have for all stacks $s \in \text{Stacks}_{k+1}(\Gamma)$ that $\mu' k(s)$ is defined if and only if $\tilde{\mu' k}(top_k(s)) = \tilde{\mu'}(top_k(s))$ is defined and that $top_k(\mu' k(s)) = \tilde{\mu' k}(top_k(s)) = \tilde{\mu'}(top_k(s))$. Note, for $\gamma \neq k$ that $\Gamma_{k+1} \setminus \{k, \overline{k}, \bot_{k+1}\} = \Gamma_k$ and so $\gamma \in \Gamma_k$ which means that γ can only operate on the topmost level-k stack of s. By this the claims follow directly. □

2.2.4 Regularity for Sets of Higher-Order Stacks

In order to speak about possibly infinite sets of higher-order stacks, it is essential to have a definition of regularity for these sets. By this we can provide a finite representation of a set which contains maybe infinitely many stacks. If we consider only sets of level-1 stacks we can use the definition of regularity for words as each level-1 stack can be considered as a word. Indeed Büchi has shown that the set of reachable stack contents of a pushdown system is regular [Büc64].

For all levels greater than 1 there are two main approaches to receive an adequate definition of *regularity for higher-order stacks*. The first definition considers a higher-order stack of level-k as some well-bracketed word of depth k. For example the stack $[[aa]_1[ab]_1]_2$ can be considered as the well-bracketed word $[[aa][ab]]$ of depth 2. So, a set of higher-order stacks is regular in this sense if the according set of well-bracketed words

is regular. This notion of regularity was introduced by [BM04]. We call a set of higher-order stacks fulfilling this condition *regular for words*. For example, the set of level-2 stacks $\{[a^n][b^m]]_2 \mid n, m \geq 1\}$ is regular for words.

The notion of regularity for sets of higher-order stacks that will be used in this work has been developed independently by Carayol, see [Car06, Car05], and Fratani [Fra05a, Fra05b]. In this case a set of higher-order stacks of some level k is regular if every stack can be constructed starting from the empty level-k stack by applying a regular sequence of level-k operations to it. We call a set of stacks fulfilling this condition *regular for operations*. Formally we define:

Definition 2.14. The set of all level-k stacks which are *regular for operations* is defined by

$$\mathrm{OReg}_k(\Gamma) = \mathrm{Reg}(Ops_k(\Gamma)^*)([\ ]_k) = \mathrm{Reg}(\Gamma_k^*)([\ ]_k),$$

or written in a different way $S \in \mathrm{OReg}_k(\Gamma)$ if there exists $R \in \mathrm{Reg}(\Gamma_k^*)$ and $S = \{\mu([\ ]_k) \mid \mu \in R\}$.

It is obvious that on level 1 the definitions for regularity for word and regularity for operations coincide. For all levels beyond 1 they differ. Take for example the following set of level-2 stacks, where all level-2 stacks are included that contain two level-1 stacks with the same number of a's:

$$S = \{[[a^n][a^n]]_2 \mid n \in \mathbb{N}\}$$

As the language $L = \{a^n b a^n \mid n \in \mathbb{N}\}$ over words is not regular, it is easy to see that S is also not regular for words. But S is regular for operations as it can be constructed by the application of the regular operation sequence $push_a^* copy_1$ applied to the empty level-2 stack.

For every level $k \geq 1$ the set $\mathrm{OReg}_k(\Gamma)$ is a boolean algebra. This result can be found in [Car05, Car06]. We come back to this later in Section 2.2.5 but first introduce some other ways to characterize the regular sets of higher-order stacks.

Note, that if we call a set of higher-order stacks regular without saying regular for word or regular for operations we always think of regular for operations.

2.2.5 Automata Models for Sets of Higher-Order Stacks

It is natural that besides a definition which relies on regularity for words and regular expressions, we can also give a description of $\mathrm{OReg}_k(\Gamma)$ by automata. For this we take automata which have as working alphabet the higher-order stack operations respectively instructions.

Definition 2.15. An *automaton* $\mathcal{A}$ *over* Γ_k is defined by (Q, I, F, Δ) where Q is a finite set of states, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states, and $\Delta \subseteq Q \times \Gamma_k \times Q$ is a transition relation.

Let $\mathcal{C}_\mathcal{A} = Q \times \mathrm{Stacks}_k(\Gamma)$ be the *set of all configurations* of $\mathcal{A}$. A single *configuration* is a tuple $(q, s) \in Q \times \mathrm{Stacks}_k(\Gamma)$. We write $(q, s) \xrightarrow{\gamma} (q', s')$ if $(q, \gamma, q') \in \Delta$ and $s' = \gamma(s)$.

A *run* of $\mathcal{A}$ is a sequence

$$(q_0, s_0)\gamma_1(q_1, s_1)\gamma_2 \dots \gamma_n(q_n, s_n) \in \mathcal{C}_\mathcal{A}(\Gamma_k \mathcal{C}_\mathcal{A})^*$$

such that for all $i \in [1, n]$ holds $(p_{i-1}, s_{i-1}) \xrightarrow{\gamma_i} (p_i, s_i)$. A stack $s \in \mathrm{Stacks}_k(\Gamma)$ is accepted by $\mathcal{A}$ if there exists a run with $q_0 \in I$, $s_0 = [\,]_k$, $q_n \in F$ and $s_n = s$. The set of stacks that is accepted by $\mathcal{A}$ is denoted by $\mathcal{S}(\mathcal{A})$. Sometimes, we call a regular set of stacks also a regular language.

Example 5. Consider the following set of level-2 stacks:

$$S_2 = \{[[a^n][a^{n-2}] \dots [aa][\,]]_2 \mid n \geq 0, n \text{ even}\}.$$

We define an automaton $\mathcal{A}_2 = (Q, I, F, \Delta)$ over Γ_2 that accepts S_2. For this let $Q = \{q_0, q_1, q_2, q_3, q_4, q_f\}$, $I = \{q_0\}$ and $F = \{q_f\}$. The transition relation Δ is defined by:

$$\begin{array}{llll} q_0 \xrightarrow{a} q_1 & q_1 \xrightarrow{a} q_0 & q_0 \xrightarrow{1} q_2 & \\ q_2 \xrightarrow{\bar{a}} q_3 & q_3 \xrightarrow{\bar{a}} q_4 & q_4 \xrightarrow{1} q_2 & q_2 \xrightarrow{\perp_1} q_f \end{array}$$

We can also represent the automaton graphically as shown in Figure 2.3. The run of S_2 that accepts the stack $[[a^4][a^2][\,]]_2$ looks as follows:

$$\begin{gathered} (q_0, [\,]_2) \xrightarrow{a} (q_1, [[a]]_2) \xrightarrow{a} (q_0, [[a^2]]_2) \xrightarrow{a} (q_1, [[a^3]]_2) \xrightarrow{a} \\ (q_0, [[a^4]]_2) \xrightarrow{1} (q_2, [[a^4][a^4]]_2) \xrightarrow{\bar{a}} (q_3, [[a^4][a^3]]_2) \xrightarrow{\bar{a}} \\ (q_4, [[a^4][a^2]]_2) \xrightarrow{1} (q_2, [[a^4][a^2][a^2]]_2) \xrightarrow{\bar{a}} (q_3, [[a^4][a^2][a]]_2) \\ \xrightarrow{\bar{a}} (q_4, [[a^4][a^2][\,]]_2) \xrightarrow{\perp_1} (q_f, [[a^4][a^2][\,]]_2). \end{gathered}$$

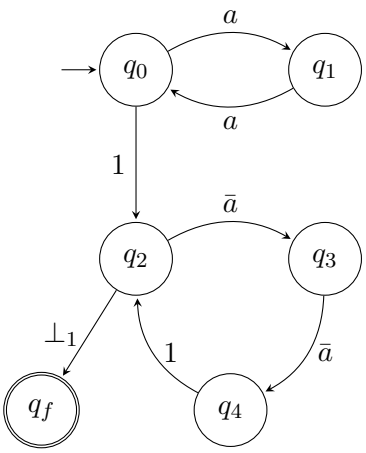

Figure 2.3: Illustration of the automaton $\mathcal{A}_2$ of Example 5.

The automata over Γ_k accept exactly the sets of stacks which are regular for operations, i.e., which belong to $\mathrm{OReg}_k(\Gamma)$. We add another characterization of the set $\mathrm{OReg}(\Gamma)$ by another automata model which has nicer closure properties. For this we use the definition of reduced instruction sequences and allow only runs that correspond to reduced sequences.

Definition 2.16. An automaton $\mathcal{A}$ over Γ_k is called *reduced* if for each run

$$(q_0, [\,]_k)\gamma_1(q_1, s_1)\gamma_2 \dots \gamma_n(q_n, s_n)$$

of $\mathcal{A}$ the sequence $\gamma_1 \dots \gamma_n \in \Gamma_k^*$ is reduced.
As the reduced sequence of a stack is unique, each stack $s \in \mathrm{Stacks}_k(\Gamma)$ with $\rho_s = \gamma_1 \dots \gamma_n$ that is accepted by $\mathcal{A}$ has to be accepted by a run of the form

$$(q_0, [\,]_k) \xrightarrow{\gamma_1} (q_1, s_1) \xrightarrow{\gamma_2} \dots \xrightarrow{\gamma_n} (q_n, s_n).$$

As the reduced sequences do not contain the tests for emptiness this holds also for the reduced automata over Γ_k. It is easy to see that the reduced automata over Γ_k are weaker than the automata over Γ_k and do not capture the set $\mathrm{OReg}_k(\Gamma)$. A simple example to illustrate this is the set of stacks $\{[[a^n][\,]]_2 \mid n \in \mathbb{N}\}$. It can be constructed by the regular instruction sequence $a^*1\bar{a}^*\perp_1$ applied to $[\,]_2$ but if the test for emptiness is not allowed the instruction sequence has to be of the form $a^n1\bar{a}^n$ for all $n \in \mathbb{N}$, which is no longer regular.

To enrich the reduced automata over Γ_k such that again all regular sets of stacks in $\mathrm{OReg}_k(\Gamma)$ can be recognized, we introduce the concept of tests. These tests are used to simulate the loops which are no longer allowed in the reduced sequences.

Definition 2.17. We define the new operation of *k-regular tests* for all $l \geq 1$ and $l \leq k$. For an associated regular language $L \subseteq \mathrm{Stacks}_l(\Gamma)$ we write $Test_L^k$ and define for all stacks $s \in \mathrm{Stacks}_k(\Gamma)$:

$$Test_L^k(s) \;=\; \begin{cases} s & \text{if } top_l(s) \in L \\ \text{non-defined} & \text{otherwise} \end{cases}$$

For the operation $Test_L^k$ we denote the corresponding instruction by T_L^k.

The automata with tests do not only use one test in their transitions but may use a finite number of tests. For this we denote by $\mathcal{L}$ a finite set of regular sets of stacks $L \in \mathrm{Stacks}_l(\Gamma)$, e.g., $\mathcal{L} = \{L_1, \ldots, L_m\}$ where $L_i \subseteq \mathrm{Stacks}_{l_i}(\Gamma)$ with $l_i \leq k$ for $i \in [1, m]$, $m \in \mathbb{N}$. The finite set of tests as instructions is denoted by $\mathcal{T}_{\mathcal{L}}^k = \{T_L^k \mid L \in \mathcal{L}\}$. As in one transition we allow to have several tests at the same time, we define for all subsets of tests $T \subseteq \mathcal{T}_{\mathcal{L}}^k$ the domain of T, $\mathrm{Dom}(T) \subseteq \mathrm{Stacks}_l(\Gamma)$ by:

$$\mathrm{Dom}(T) \;=\; \begin{cases} \bigcap_{i \in [1,n]} L_i & \text{if } T = \{T_{L_1}^k, \ldots, T_{L_n}^k\} \text{ with } n > 0 \\ \mathrm{Stacks}_l(\Gamma) & \text{if } T = \emptyset \end{cases}$$

Now, we define automata over Γ_k, which use these tests.

Definition 2.18. An *automaton* $\mathcal{A}$ *over* Γ_k *with tests* in a finite set $\mathcal{L}$ of regular subsets of $\mathrm{Stacks}_l(\Gamma)$ for $l \leq k$ is defined as a tuple (Q, I, F, Δ), where Q is a finite set of states, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states and $\Delta \subseteq Q \times \Gamma_k \times 2^{\mathcal{T}_{\mathcal{L}}^k} \times Q$ is a transition relation.

We write $(p, s) \xrightarrow{\gamma, T} (q, s')$ if $(p, \gamma, T, q) \in \Delta$, $s' = \gamma(s)$ is defined and $s' \in \mathrm{Dom}(T)$. A run of $\mathcal{A}$ is defined analogously to the automata over Γ_k, i.e., it starts in $(q_0, [\,]_k)$ with $q_0 \in I$, ends in (q_f, s) with $q_f \in F$ and follows the transition relation.

Example 6. Consider the following regular set of stacks:

$$S = \{[[w_0] \ldots [w_n]]_2 \mid n \geq 1, w_i \sqsubseteq w_{i+1}, |w_i|_a \text{ even}, \forall i \in [1, n]\}$$

It can be constructed by an automaton over Γ_2 with tests in $\mathrm{Stacks}_1(\Gamma)$. An automaton over Γ_2 with tests in $\mathrm{Stacks}_1(\Gamma)$ that accepts S is in shown

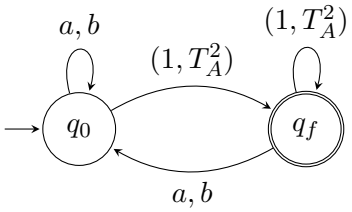

Figure 2.4: Automaton of Example 6.

in Figure 2.4. The used test T_A^2 is defined by the regular set of level-1 stacks A that contains all level-1 stacks with an even number of a, i.e., $A = \{[w]_1 \mid |w|_a \text{ even}\}$.

The definition of the automata over Γ_k with tests can be extended to *reduced automata over* Γ_k *with tests*. In this case the instruction sequence which is used in a run has to be reduced. We need reduced automata over Γ_k with tests later on to represent regular sets of stacks. We call them for short *reduced level-k automata*.

Definition 2.19. A *reduced level-k automaton* $\mathcal{A}$ is a finite automaton over Γ_k with tests in a finite set of regular subsets of $\text{Stacks}_{(k-1)}(\Gamma)$. It is given by (Q, I, F, Δ) together with a finite set of tests $\mathcal{R} \subset \text{OReg}_k(\Gamma)$ where $I \subseteq Q$ is s set of initial states, $F \subseteq Q$ is a set of final states and $\Delta : Q \times \Gamma_k \times 2^{\mathcal{R}} \times Q$ is a transition relation.

The transition relation and a run of the automaton is defined as in Definition 2.18 with the additional condition that the automaton follows only reduced instruction sequences as in Definition 2.16.

Theorem 2.20 ([Car05, Car06, Fra05a])**.** *For all $k \geq 1$, the sets of level-k stacks that are regular for operations are exactly those sets accepted by reduced automata over Γ_k with tests in* $\text{OReg}_{k-1}(\Gamma)$*. Moreover,* $\text{OReg}_k(\Gamma)$ *forms a Boolean algebra.*

2.2.6 Higher-Order Pushdown Automata

In this section we define higher-order pushdown systems. Later, we use them for example in the definition of games over higher-order pushdown graphs. Furthermore, we define higher-order pushdown automata that recognize languages over some alphabet Σ. For both automata models we use in the definition of the transition relation the instructions Γ_k. For

readability we sometimes use operations from $Ops_k(\Gamma)$ instead. We do not differentiate between these two notations, as both have an equivalent meaning.

Definition 2.21. A *level-k higher-order pushdown system* $\mathcal{A}$ (k-HOPDS for short) is defined by a tuple $(Q, \Sigma, \Gamma, \Delta)$ where Q is a finite set of states, Σ is an input alphabet, Γ is a stack symbol alphabet and $\Delta \subseteq Q \times \Sigma \times \Gamma_k \times Q$ is a transition relation.

A configuration is a pair $(p, s) \in Q \times \text{Stacks}_k(\Gamma)$. We write $(p, s) \xrightarrow{\alpha} (q, s')$ if there exists a transition $(p, \alpha, \gamma, q) \in \Delta$ such that $s' = \gamma(s)$ is defined.

A k-HOPDS is *deterministic* if for all $\alpha \in \Sigma$ and all configurations c, c' and c'', $c \xrightarrow{\alpha} c'$ and $c \xrightarrow{\alpha} c''$ imply that $c' = c''$.

Definition 2.22. A *level-k higher-order pushdown automaton* $\mathcal{A}$ (short: k-HOPDA) is defined as a tuple $(Q, \Sigma, \Gamma, \varepsilon, q_0, F, \Delta)$ where Q is a finite set of states, Σ is an input alphabet, Γ is a stack symbol alphabet, $\varepsilon \notin \Sigma$ is an extra symbol to Σ that has no influence on the accepted word, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states and $\Delta \subseteq Q \times \Sigma \cup \{\varepsilon\} \times \Gamma_k \times Q$ is a transition relation.

A configuration is a pair $(p, s) \in Q \times \text{Stacks}_k(\Gamma)$. We write $(p, s) \xrightarrow{\alpha} (q, s')$ if there exists a transition $(p, \alpha, \gamma, q) \in \Delta$ such that $s' = \gamma(s)$.

A run of a k-HOPDA $\mathcal{A}$ starts in the initial configuration $(q_0, [\,]_k)$ and proceeds by applying the transition relation. A word $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if there exists a run

$$(q_0, [\,]_k) \xrightarrow{\alpha_1} (q_1, s_1) \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} (q_n, s_n)$$

with $q_n \in F$ and we obtain w by deleting all ε from $\alpha_1 \dots \alpha_n$.

Example 7. As an example for a language recognized by a higher-order pushdown automaton we consider the language $L_{w\$w} = \{w\$w \mid w \in \{a, b\}^*\}$ of repeated words. It is known from automata theory that this language is not context free but context sensitive. As we will see, it can be recognized by a level-2 higher-order pushdown automaton. We define a level-2 HOPDA $\mathcal{A}$ that accepts the language $L_{w\$w}$ by $\mathcal{A}_{w\$w} = (Q, \Sigma, \Gamma, \varepsilon, q_0, \{q_f\}, \Delta)$ with $Q = \{q_0, q_1, q_2, q_3, q_4, q_f\}$, $\Sigma = \{a, b, \$\}$ and $\Gamma = \{a, b\}$.

The set of transitions Δ is given by:

$$\{(q_0, a, push_a, q_0), (q_0, b, push_b, q_0), (q_0, \$, copy_1, q_1), (q_1, \varepsilon, copy_1, q_2)$$
$$(q_2, \varepsilon, pop_a, q_1), (q_2, \varepsilon, pop_b, q_1), (q_1, \varepsilon, T_{[\,]_1}, q_3), (q_3, a, push_a, q_4)$$
$$(q_3, b, push_b, q_4), (q_4, \varepsilon, \overline{copy_1}, q_3), (q_3, \varepsilon, \overline{copy_1}, q_f)\}$$

The automaton is presented graphically in Figure 2.5. The idea for this automaton is the following. First the automaton remains in state q_0, reading the input word until \$ appears and pushes all letters up to \$ onto the stack. If \$ is reached, the automaton proceeds to state q_1 and copies the topmost level-1 stack. After that in every step the topmost level-1 stack is copied and the top symbol in the topmost level-1 stack is deleted until the topmost level-1 stack is empty. In these steps we use the silent input letter ε and no letter of the input word is read. Then in state q_3 we read again an input letter, put it onto the stack and then we can delete the topmost level-1 stack by $\overline{copy_1}$. In the last step we just delete the topmost level one stack by $\overline{copy_1}$ and reach the final state.

Now, we give an example run of $\mathcal{A}_{w\$w}$ for the input word $abb\$abb$:

$$(q_0, [\,]_2) \xrightarrow{a} (q_0, [[a]]_2) \xrightarrow{b} (q_0, [[ab]]_2) \xrightarrow{b} (q_0, [[abb]]_2) \xrightarrow{\$}$$
$$(q_1, [[abb][abb]]_2) \xrightarrow{\varepsilon} (q_2, [[abb][abb][abb]]_2) \xrightarrow{\varepsilon}$$
$$(q_1, [[abb][abb][ab]]_2) \xrightarrow{\varepsilon} (q_2, [[abb][abb][ab][ab]]_2) \xrightarrow{\varepsilon}$$
$$(q_1, [[abb][abb][ab][a]]_2) \xrightarrow{\varepsilon} (q_2, [[abb][abb][ab][a][a]]_2) \xrightarrow{\varepsilon}$$
$$(q_1, [[abb][abb][ab][a][\,]]_2) \xrightarrow{\varepsilon} (q_3, [[abb][abb][ab][a][\,]]_2) \xrightarrow{a}$$
$$(q_4, [[abb][abb][ab][a][a]]_2) \xrightarrow{\varepsilon} (q_3, [[abb][abb][ab][a]]_2) \xrightarrow{b}$$
$$(q_4, [[abb][abb][ab][ab]]_2) \xrightarrow{\varepsilon} (q_3, [[abb][abb][ab]]_2) \xrightarrow{b}$$
$$(q_4, [[abb][abb][abb]]_2) \xrightarrow{\varepsilon} (q_3, [[abb][abb]]_2) \xrightarrow{\varepsilon} (q_f, [[abb]]_2)$$

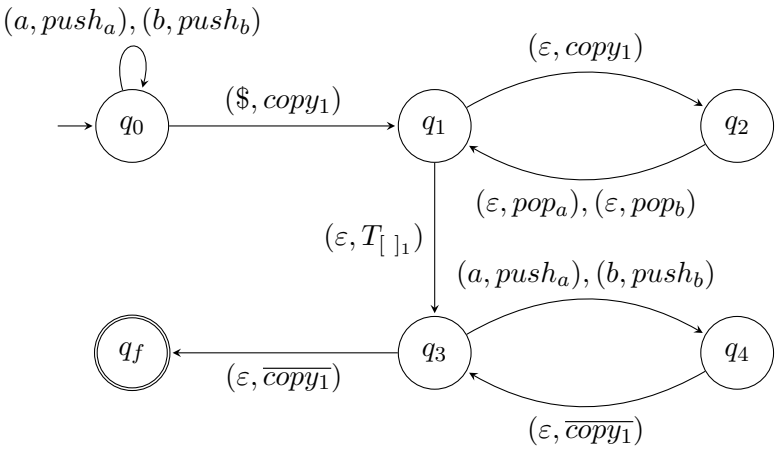

Figure 2.5: Higher-order pushdown automaton to accept the language $L_{w\$w}$ of Example 7.

3 Higher-Order Pushdown Parity Games

In this chapter we consider parity games played on the configuration graph of a higher-order pushdown system. We compute the winning regions and global positional winning strategies for both players. The main focus is to obtain global positional strategies in a "regular" representation, i.e., given by sets of stacks that are regular for operations.

We start by introducing higher-order pushdown parity games in Section 3.1. After this we state the main theorem and give an outline of the proof in Section 3.2. The proof of the main theorem is divided into two parts given in Section 3.3 and Section 3.4. In Section 3.5 the results of the previous sections are combined to give the formal proof of the main theorem of this chapter. In Section 3.6 we consider higher-order pushdown games with two different winning conditions. First, we consider in Section 3.6.1 higher-order pushdown Büchi games and afterwards in Section 3.6.2 higher-order pushdown request-response games and show how to solve them.

3.1 Definition of Higher-Order Pushdown Parity Games

We begin by defining *higher-order pushdown parity games*. These are parity games played on a game graph, which is built by a higher-order pushdown system. Higher-oder pushdown systems provide by their transition relation the possibility to construct an infinite graph with a finite representation. We use the states of the higher-order pushdown system to define the partitioning of the game graph into vertices of Player 0 and Player 1 as well as to assign the color to a vertex. It is also possible to define the partitioning and the coloring by regular sets of higher-order pushdown stacks; but the definition by the states also suffices [Cac01].

Definition 3.1. A *level-k higher-order pushdown parity game* $\mathcal{G}$ (short: k-HOPDPG) is given by a deterministic k-HOPDS $P = (Q, \Sigma, \Gamma, \delta)$ together with a partition of the states $Q_0 \uplus Q_1$ and a coloring $\Omega_P \colon Q \to \{0, \ldots, n\}$ for some fixed number $n \in \mathbb{N}$. The game $\mathcal{G}$ is then defined by the game graph (V_0, V_1, E, Ω) where

- $V_0 = Q_0 \times \mathrm{Stacks}_k(\Gamma)$,
- $V_1 = Q_1 \times \mathrm{Stacks}_k(\Gamma)$,
- $E \subseteq (Q \times \mathrm{Stacks}_k(\Gamma)) \times \Sigma \times (Q \times \mathrm{Stacks}_k(\Gamma))$ with $(p, s) \xrightarrow{\alpha} (q, s') \in E$ if $(p, \alpha, \gamma, q) \in \delta$ and $s' = \gamma(s)$,
- Ω is defined for $(p, s) \in Q \times \mathrm{Stacks}_k(\Gamma)$ by $\Omega(p, s) := \Omega_P(p)$.

We use a labeled edge relation in the games to get a nice way to represent the positional winning strategies. So the labels in Σ on the edges do not play any role for the game itself but only for the representation of the strategies. For this we use that the k-HOPDS P is assumed to be deterministic. This means that for all configurations c, c', c'' and all $\alpha \in \Sigma$ we have that if $c \xrightarrow{\alpha} c'$ and $c \xrightarrow{\alpha} c''$ then $c' = c''$. So we can conclude from a configuration $c \in Q \times \mathrm{Stacks}_k(\Gamma)$ and a label $\alpha \in \Sigma$ the succeeding configuration and which transition of P has been used. Note, that in this context each higher-order pushdown system can be made deterministic, e.g., by using for each transition a different symbol in Σ.

We describe a *positional strategy* φ_i for Player i by a partial function from V_i to Σ. We can equivalently describe a positional winning strategy by a family $(F^i_\alpha)_{\alpha \in \Sigma}$ of subsets of V_i, i.e., $F^i_\alpha := \{(p, s) \in V_i \mid \varphi_i((p, s)) = \alpha\}$. This means if $(p, s) \in Q_i \times \mathrm{Stacks}_k(\Gamma)$ is in the set F_α of Player 1 then Player 1 has to choose the edge labeled by α if he is in vertex (p, s).

We will represent a configuration (p, s) by the stack $push_p(s)$. In this case these sets F_α are sets of stacks instead of sets of configurations. We say that a positional strategy defined by a family $(F_\alpha)_{\alpha \in \Sigma}$ is *regular* if all these sets are regular for operations.

Example 8. Let the level-2 higher-order pushdown parity game $\mathcal{G}$ be defined by the game graph (V_0, V_1, E, Ω) which is given by the deterministic level-2 higher-order pushdown system

$$P = (\{q_0, q_1, q_2\}, \{\alpha, \alpha', \beta, \beta', \gamma, \gamma'\}, \{a\}, q_0, \Delta\}$$

with

$$\begin{aligned}\Delta \;=\; & \{(q_0,\alpha,1,q_0),(q_0,\alpha',a,q_1),(q_1,\beta,a,q_2),\\ & (q_1,\beta',\overline{a},q_0),(q_2,\gamma,a,q_0),(q_2,\gamma',\overline{a},q_1)\},\end{aligned}$$

the state partition $Q_0 = \{q_1, q_2\}$, $Q_1 = \{q_0\}$ and the coloring Ω_P with $\Omega_P(q_0) = 1$ and $\Omega_P(q_1) = \Omega_P(q_2) = 0$. A small part of the game graph is shown in Figure 3.1.

The winning region of Player 0 consists of all the vertices which belong to him without the configurations where the state is q_2 and the topmost level-1 stack is empty, i.e., $W_0 = V_0 \backslash \{(q_2, s) \mid s \in \Gamma_2^* \bot_1([\,]_2)\}$; the winning region of Player 1 is accordingly equal to $V_1 \cup \{(q_2, s) \mid s \in \Gamma_2^* \bot_1([\,]_2)\}$. The winning strategy for Player 0 can be given by:

- $\varphi_0((q_1, s)) = \beta$,
- $\varphi_0((q_2, s)) = \gamma'$,

for all $s \in \mathrm{Stacks}_2(\Gamma) \setminus \{(q_2, s) \mid s \in \Gamma_2^* \bot_1([\,]_2)\}$. Alternatively we can define the winning strategy by the following three regular sets of higher-order stacks:

- $F^0_\beta = \{push_{q_1}(s) \mid s \in \mathrm{Stacks}_2(\Gamma)\}$
- $F^0_{\gamma'} = \{push_{q_2}(s) \mid s \in \mathrm{Stacks}_2(\Gamma) \setminus \{\Gamma_2^* \bot_1([\,]_2)\}\}$
- $F^0_\alpha = F^0_{\alpha'} = F^0_{\beta'} = F^0_\gamma = \emptyset$

The winning strategy of Player 1 is $\varphi_1((q_0, s)) = \alpha$ for all $s \in \mathrm{Stacks}_2(\Gamma)$. Alternatively we can define the winning strategy by the following two regular sets of higher-order stacks:

- $F^1_\alpha = \{push_{q_0}(s) \mid s \in \mathrm{Stacks}_2(\Gamma)\}$
- $F^1_{\alpha'} = F^1_\beta = F^1_{\beta'} = F^1_\gamma = F^1_{\gamma'} = \emptyset$

3.2 Main Theorem and Outline of the Proof

We state here the main theorem of this chapter and provide an outline of its proof. The detailed proof is given in the remaining sections of this chapter.

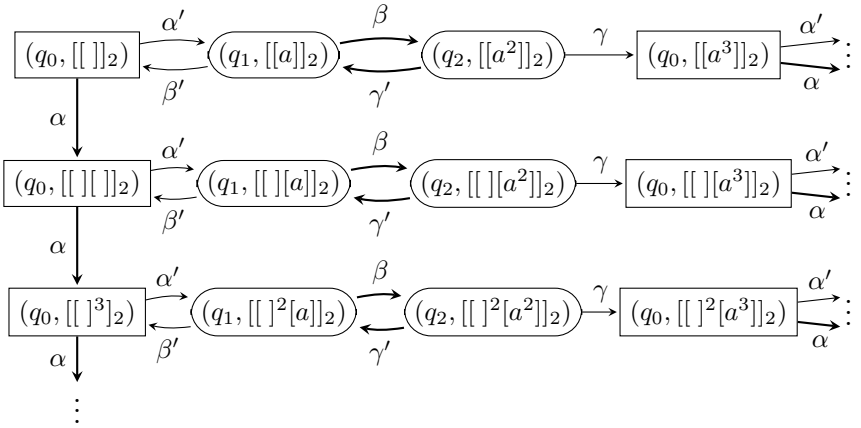

Figure 3.1: Outline of a part of the game graph of the higher-order pushdown parity game described in Example 8. The winning strategies of both players are marked by thick arrows.

Theorem 3.2 (Main Theorem)**.** *Given a level-k higher-order pushdown parity game, we can construct in k-ExpTime reduced level-k automata describing the winning region and a global positional winning strategy for each player.*

To prove the theorem we use a property of the higher-order pushdown stacks which allows us to arrange them as a tree. So, we define in Section 3.3 for every level k a tree $\mathbf{t}_k$ (see Fig. 3.2) which includes and arranges all level-k stacks uniquely. The clue is that the branches of $\mathbf{t}_k$ correspond to the reduced sequences of the level-k stacks.

The goal of this chapter is to compute a finite representation of the winning regions and global positional winning strategies for a parity game $\mathcal{G}$ which is played on the configuration graph of a level-k higher-order pushdown system P. We start in Section 3.3 with the construction of a two-way alternating parity tree automaton (2-PTA) $\mathcal{A}_P$ running on $\mathbf{t}_k$. The 2-PTA captures the game $\mathcal{G}$ and its size is polynomial in the size of P. By this we reduce the problem of computing regular winning regions and regular global positional strategies for both players in the higher-order pushdown parity game to the problem of computing regular global positional strategies for the 2-PTA $\mathcal{A}_P$ on $\mathbf{t}_k$. Intuitively such a strategy for $\mathcal{A}_P$ consists for every node u of the tree $\mathbf{t}_k$ and every state q of the automaton in either providing a transition of the automaton

starting with state q which can be applied at node u or a set of directions and states which refute any transitions of the automaton that can be applied at node u in state q.

In Section 3.4, we show how to compute regular global positional strategies for a 2-PTA $\mathcal{A}_k$ running on the trees $\mathbf{t}_k$. The proof proceeds by induction on the level k of the tree $\mathbf{t}_k$ and is divided into two steps for each level. The first step is based on a construction from [Var98]. Starting from a 2-PTA $\mathcal{A}_k$ a non-deterministic one-way parity tree automaton (1-PTA) $\mathcal{B}_k$ is constructed, where $\mathcal{B}_k$ accepts the labellings of $\mathbf{t}_k$ which correspond to regular global positional winning strategies of $\mathcal{A}_k$. This is done in Proposition 3.9. The second step reduces the level of the underlying tree $\mathbf{t}_k$. Therefore, the 1-PTA $\mathcal{B}_k$ running on $\mathbf{t}_k$ is transformed into a 2-PTA $\mathcal{C}_{k-1}$ running on $\mathbf{t}_{k-1}$, such that from a global positional winning strategy of $\mathcal{C}_{k-1}$ over $\mathbf{t}_{k-1}$ defined by regular sets of level-$(k-1)$ stacks, we can construct a global positional winning strategy (for $\mathcal{A}_k$) accepted by $\mathcal{B}_k$ defined by regular sets of level-k stacks. This is shown in Proposition 3.10. These two results are combined in Theorem 3.16. Afterwards, the proof of the main theorem stated above is given in Section 3.5.

3.3 From Games to Trees

In this section we show that for each level k we can arrange the set of all stacks of this level in form of an infinite tree which we call $\mathbf{t}_k$. The idea for this arrangement relies on the fact that each stack has a unique reduced sequence. On these trees we can run a two-way alternating parity tree automata (2-PTA) or an one-way non-deterministic parity tree automata (1-PTA) which have been introduced in Section 2.1.2. We show that the problem of computing global positional winning strategies for a level-k pushdown parity game can be reduced in polynomial time to the problem of computing global positional strategies for a 2-PTA running on the tree $\mathbf{t}_k$. By global positional strategies for a 2-PTA we mean the strategies for the players Automaton and Pathfinder in the parity game $\mathcal{G}_{\mathcal{A},t}$ which is induced by the 2-PTA (see Section 2.1.2).

As we have seen in Section 2.2.3, each level-k stack s is uniquely characterized by its reduced sequence ρ_s. We use this to arrange all level-k stacks in form of a tree. We start at the root with the empty level-k stack. For each node in the tree corresponding to a stack s we

apply all instructions $\gamma \in \Gamma_k^O$ to s which are defined and where $\rho_s\gamma$ is a reduced sequence. The resulting stack $\gamma(s)$ is added as a son of s. Consider for example the level-2 stack $[[ab][ab]]_2$ with $\Gamma = \{a, b\}$; it has the sons $[[ab][aba]]_2$, $[[ab][abb]]_2$, $[[ab][a]]_2$ and $[[ab][ab][ab]]_2$. The parent is $[[ab]]_2$.

As the trees $\mathbf{t}_k$ for $k \geq 2$ are no longer regular we add some informations by using a *labeling* of the nodes. By this we increase the expressivity of the automata running on these trees. We label each node by a finite set of informations about the stacks that surround the node. We define the *surrounding* of a stack $s \in \text{Stacks}_k(\Gamma)$ by a triple $\ell(s) = (d, D, e)$ where

- $d \in \Gamma_k^O \cup \{\diamond\}$ is the last symbol of ρ_s if $\rho_s \neq \varepsilon$ and is equal to $\diamond$ otherwise,
- D is the set $\{\gamma \in \Gamma_k^O \mid \exists s' \in \text{Stacks}_k(\Gamma), \rho_{s'} = \rho_s\gamma\}$,
- $e \in [0, k]$ is the maximum of $\{n \in [1, k] \mid \bot_n(s) = s\} \cup \{0\}$.

Later, we see that in the surrounding we have all the information to simulate a higher-order pushdown parity game by a 2-PTA which runs on $\mathbf{t}_k$. By d we know the instruction to go up in the tree. In D we have the instructions that are defined on the current stack while still having a reduced sequence and so for all $d' \in D$ there is a d'-son in the tree $\mathbf{t}_k$. In e we have the information that the topmost level-e stack is empty which is needed when in the game an emptiness test is used, if $e = 0$ we have that the stack is non-empty for any level.

Formally, the tree $\mathbf{t}_k$ is defined for all stacks $s \in \text{Stacks}_k(\Gamma)$ by $\mathbf{t}_k(\rho_s) = \ell(s)$. When referring to the nodes of $\mathbf{t}_k$, we do not distinguish between the stack s and its reduced sequence ρ_s. Furthermore we consider a Ξ-labeling t of $\mathbf{t}_k$ to be regular if for all $x \in \Xi$, the set of level-k stacks $\mathcal{S}_x = \{s \in \text{Stacks}_k(\Gamma) \mid t(\rho_s) = (\mathbf{t}_k(\rho_s), x)\}$ is regular for operations. We can represent t by a family of reduced level-k automata $(\mathcal{A}_x)_{x\in\Xi}$.

For $\Gamma = \{a, b\}$, the tree $\mathbf{t}_1$ is essentially the full binary tree. The tree $\mathbf{t}_2$ for $\Gamma = \{a, b\}$ which is partially depicted in Figure 3.2 is neither complete nor regular.

Now, we reduce the decision problem for level-k higher-order pushdown parity games, i.e., the computation of the winning regions and of the global positional winning strategies for the two players, to the acceptance problem of two-way alternating parity tree automata which run on $\mathbf{t}_k$.

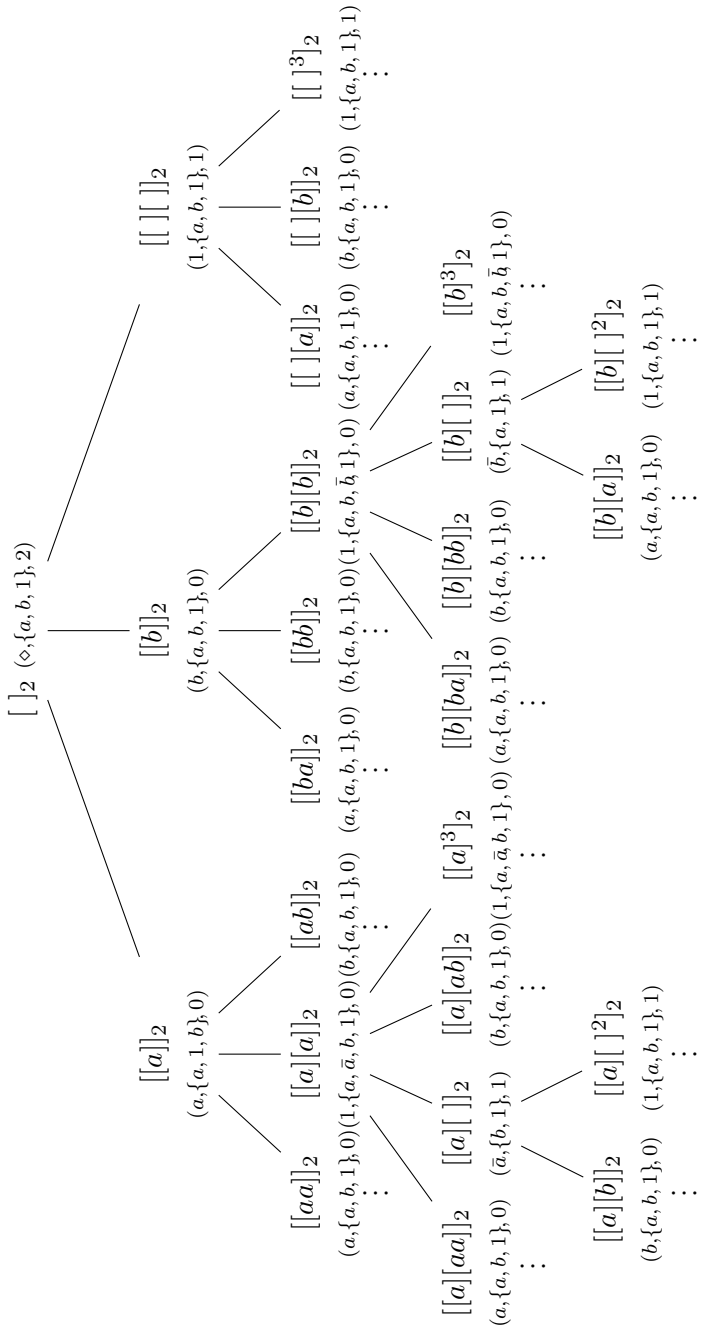

Figure 3.2: The tree $\mathbf{t}_2$ for $\Gamma = \{a, b\}$ where the labels of the surrounding appear in parenthesis below the corresponding node.

We generalize a construction that has been done for (level-1) pushdown parity games by [Var98, KV00].

Intuitively, the non-determinism of the 2-PTA is used to reflect the choices of Player 0 and the alternation is used to reflect the choices of Player 1. To receive global strategies we have to add some initial phase. In the initial phase the 2-PTA proceeds by sending copies into all nodes of the tree $\mathbf{t}_k$, afterwards in every node and with every state of the HOPDA (which defines the level-k parity game) it starts the simulation of the parity game. By this we make sure that we have a strategy from every possible vertex in the higher-order pushdown parity game for one of the players.

Proposition 3.3. *Given a higher-order pushdown parity game $\mathcal{G}$ of level k, we can construct a 2-PTA $\mathcal{A}$ running on $\mathbf{t}_k$ such that Player 0 wins from (q, s) in $\mathcal{G}$ if and only if $\mathcal{A}$ accepts $\mathbf{t}_k$ starting from ρ_s and in state q. Furthermore, the size of $\mathcal{A}$ is polynomial in the size of the level-k pushdown automaton defining $\mathcal{G}$.*

Proof. First, we give the construction of the 2-PTA, then we show the correctness of the construction, and afterwards we consider the complexity.

Construction. Let the higher-order parity game $\mathcal{G}$ be defined by the deterministic k-HOPDS $P = (Q_P, \Sigma, \Gamma, \Delta_P)$, the state partition Q_0, Q_1 and the coloring Ω_P.

We first define the 2-PTA $\mathcal{A} = (Q_A, \Delta_A, I, \Omega_A)$. The states of $\mathcal{A}$ include all the states of P and a fresh state q_0 which we need for the initial phase, where we send copies of the automaton into all directions, i.e., into all stacks. So we have $Q_A = Q_P \cup \{q_0\}$ and the set of initial states of $\mathcal{A}$ is $I = \{q_0\}$. The coloring Ω_A of $\mathcal{A}$ is defined as Ω_P for states in Q_P and assigns to q_0 an even color which is greater than the maximal color appearing in Ω_P.

Before we define Δ_A, we introduce for a surrounding $\tau = (d, D, e)$ and an instruction $\gamma \in \Gamma_k$ the corresponding direction $[\![\gamma]\!]_\tau$ on $\mathbf{t}_k$ by:

$$
\begin{array}{ll}
[\![\gamma]\!]_\tau = \gamma & \text{if } \gamma \in D \\
[\![\gamma]\!]_\tau = \uparrow & \text{if } \gamma = \overline{d} \\
[\![\gamma]\!]_\tau = \varepsilon & \text{if } \gamma = \bot_j \text{ and } e \geq j \\
[\![\gamma]\!]_\tau \text{ undefined} & \text{otherwise.}
\end{array}
$$

To define the transitions of the 2-PTA, we distinguish between two phases. In the initial phase the automaton sends copies into all possible directions which stay in the initial phase. At the same time the automaton sends copies for each $q \in Q_P$ which stay at the current node but change the state into one of the states of Q_P and proceed to the simulation phase. We need this initial phase since we consider games over the whole configuration graph of a HOPDS and do not restrict to the configurations which are reachable from some initial configuration. So, we have to compute global winning strategies no matter in which configuration the game starts.

In the simulation phase the automaton starts to simulate the higher-order pushdown parity game, where we have two cases depending whether the transition starts in a state of Player 0 or Player 1. If the transition starts in a Player 0 state the nondeterminism of the automaton is used, so in $\mathcal{A}$ Automaton can choose how to proceed. If the transition starts in a Player 1 state the alternation is used to reflect the choice of Player 1 which means that in $\mathcal{A}$ Pathfinder can choose how to go on.

Initial phase. For all labeling $\tau = (d, D, e)$, we have

$$(q_0, \tau) \to \bigwedge_{d' \in D} (d', q_0) \wedge \bigwedge_{q \in Q_P} (\varepsilon, q) \in \Delta_A.$$

In the simulation phase we have to distinguish for the states $p \in Q_P$ to which player they belong.

Case $\mathbf{p \in Q_0}$. For each transition $\delta = (p, \alpha, \gamma, q) \in \Delta_P$ of the k-HOPDS, we introduce a transition of the 2-PTA simulating the transition. More precisely, for each labeling $\tau = (d, D, e)$ we add the following transition to Δ_A

$$(p, \tau) \to (\llbracket \gamma \rrbracket_\tau, q),$$

if $\llbracket \gamma \rrbracket_\tau$ is defined.

Case $\mathbf{p \in Q_1}$. Let $\delta_1, \ldots, \delta_n$ be the set of all transitions in Δ_P starting in state p, i.e., having the form $\delta_i = (p, \alpha_i, \gamma_i, q_i)$ for all $i \in [1, n]$. We add for each labeling $\tau = (d, D, e)$ the following transition to Δ_A:

$$(p, \tau) \to \bigwedge_{i \in [1,n] \wedge \llbracket \gamma_i \rrbracket_\tau \text{def.}} (\llbracket \gamma_i \rrbracket_\tau, q_i).$$

Proof of Correctness. Now we have to show for each $\hat{s} \in \text{Stacks}_k$ and each $\hat{q} \in Q_P$ that Player 0 wins the parity game $\mathcal{G}$ starting from $(\hat{q}, \hat{s})$ if and only if $\mathcal{A}$ accepts $\mathbf{t}_k$ starting from $\rho_{\hat{s}}$ in state $\hat{q}$.

As a preparation we first have to make some remark about the connection between $\gamma(s)$ and $\rho_s[\![\gamma]\!]$. We have to show that both generate the same stack, respectively, that for the second case we end in $\mathbf{t}_k$ in the node representing the stack $\gamma(s)$. Let the surrounding of ρ_s be (d, D, e). We have do make a case disjunction depending on γ. If $\gamma \in D$ then $[\![\gamma]\!] = \gamma$ and by definition of the surrounding $\rho_s\gamma$ is still a reduced sequence and represents the stack $\gamma(s)$. If $\gamma = \bar{d}$ then $[\![\gamma]\!] =\uparrow$ and by definition d is the last operation of ρ_s so $\rho_s\gamma$ is no longer reduced and the last two operations substitute each other. The deletion of the last operation of ρ_s means that in $\mathbf{t}_k$ we go one step upwards in the tree. So if $\rho_s = \gamma_1 \dots \gamma_{n-1} d$ we have $\rho_s \uparrow = \gamma_1 \dots \gamma_{n-1}$ which represents the stack $\gamma(s) = \gamma_1 \dots \gamma_{n-1} d\gamma([\ ]_k) = \gamma_1 \dots \gamma_{n-1}([\ ]_k)$. If $\gamma = \bot_j$ and $e \geq j$ then $[\![\gamma]\!] = \varepsilon$ so ρ_s does not change. The same holds if $\bot_j$ is applied to s as $e \geq j$ indicates that the topmost level e stack is empty. If none of the cases applies and $[\![\gamma]\!]$ is not defined $\gamma(s)$ is also not defined.

Direction from left to right: To show the direction from left to right, we first assume that Player 0 wins the higher-order pushdown parity game $\mathcal{G}$ starting from $(\hat{q}, \hat{s})$ for some stack $\hat{s} \in \text{Stacks}_k$ and some state $\hat{q} \in Q_P$. In particular this means that Player 0 has a winning strategy φ_0 for every play starting in $(\hat{q}, \hat{s})$ in the game $\mathcal{G}$, where whatever Player 1 does, the play fulfills the winning condition. A winning strategy φ_0 for Player 0 assigns to every $(q, s) \in V_0$ which is reached in a play a symbol α, i.e., a transition $(q, \alpha, \gamma, q') \in \Delta_P$ such that the successor configuration in the play is (q', s') with $s' = \gamma(s)$.

Now we have to show that $\mathcal{A}$ accepts $\mathbf{t}_k$ starting from $\rho_{\hat{s}}$ in state $\hat{q}$ which means that the Player Automaton has a winning strategy in the parity game $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ defining the behavior of $\mathcal{A}$ starting from $(\rho_{\hat{s}}, \hat{q})$.

Remember that the vertices of Automaton are in $\text{Dom}(\mathbf{t}_k) \times Q_A$ and the vertices of Pathfinder are in $\text{Dom}(\mathbf{t}_k) \times \Delta_A$. Furthermore for all $w \in \text{Dom}(\mathbf{t}_k)$ and $q \in Q_A$ there is an edge $((w, q), (w, \delta)) \in E$ for all transitions $\delta \in \Delta_A$ of the form $(q, \mathbf{t}_k(w)) \to P$. Conversely for every transition $\delta = (q, \mathbf{t}_k(w)) \to P \in \Delta_A$, there is an edge $((w, \delta), (wd_i, q_i))$ for all $(d_i, q_i) \in P$.

Let the parity game $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ start in vertex $(\rho_{\hat{s}}, \hat{q})$ which belongs to Automaton. Considering the vertex $(\hat{q}, \hat{s})$ in $\mathcal{G}$ we have to differentiate between two cases depending on to which player the vertex belongs.

Case $(\hat{\mathbf{q}}, \hat{\mathbf{s}})$ belongs to Player 0. We have assumed that Player 0

wins the game with the winning strategy φ_0. Let now $\varphi_0((\hat{s},\hat{q})) = \alpha$ with $\delta = (\hat{q}, \alpha, \gamma, q') \in \Delta_P$ such that the successor configuration in the play is (q', s') with $s' = \gamma(\hat{s})$. By construction there has to be for each labeling $\tau = (d, D, e)$ a transition of the form $(\hat{q}, \tau) \to ([\![\gamma]\!]_\tau, q')$ in Δ_A. So Automaton can choose the transition $(\hat{q}, \tau_{\hat{s}}) \to ([\![\gamma]\!]_{\tau_{\hat{s}}}, q')$ in the game $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ in vertex $(\rho_{\hat{s}}, \hat{q})$ where $\rho_{\hat{s}}$ is labeled by $\tau_{\hat{s}}$. We end in the vertex $(\rho_{\hat{s}}, (\hat{q}, \tau_{\hat{s}}) \to ([\![\gamma]\!]_{\tau_{\hat{s}}}, q'))$ of Pathfinder. Now Pathfinder can only choose the pair $([\![\gamma]\!]_{\tau_{\hat{s}}}, q')$, i.e., go to vertex $(\rho_{\hat{s}}[\![\gamma]\!]_{\tau_{\hat{s}}}, q')$ of Automaton which corresponds to the vertex $(q', \gamma(\hat{s})) = (q', s')$ in $\mathcal{G}$ as shown above. There we can apply the same case disjunction as before again and continue.

Case $(\hat{\mathbf{q}}, \hat{\mathbf{s}})$ belongs to Player 1. In this case we have to take all possible choices of Player 1 into account. So let $\delta_1, \ldots, \delta_n$ be the set of all transitions in Δ_P starting in state $\hat{p}$, i.e., having the form $\delta_i = (\hat{p}, \alpha_i, \gamma_i, q_i)$ for all $i \in [1, n]$. So Player 1 can choose to go to one of the successor vertices $(q_i, \gamma_i(\hat{s}))$ for $i \in [1, n]$ and Player 0 can from there still win the game. By construction we have for each labeling $\tau = (d, D, e)$ the transition $(\hat{q}, \tau) \to \bigwedge_{i\in[1,n]\wedge[\![\gamma_i]\!]_\tau \text{def.}}([\![\gamma_i]\!]_\tau, q_i)$ in Δ_A. So Automaton can only choose the transition $\Big((\hat{q}, \tau_{\hat{s}}) \to \bigwedge_{i\in[1,n]\wedge[\![\gamma_i]\!]_{\tau_{\hat{s}}} \text{def.}}([\![\gamma_i]\!]_{\tau_{\hat{s}}}, q_i)\Big) =: \delta_\wedge$ in the game $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ in vertex $(\rho_{\hat{s}}, \hat{q})$ where $\rho_{\hat{s}}$ is labeled by $\tau_{\hat{s}}$. The game proceeds in vertex $(\rho_{\hat{s}}, \delta_\wedge)$ where Pathfinder has to choose one pair $([\![\gamma_i]\!]_{\tau_{\hat{s}}}, q_i)$. Assume w.l.o.g. that Pathfinder chooses $([\![\gamma_1]\!]_{\tau_{\hat{s}}}, q_1)$ and we go on in vertex $(\rho_{\hat{s}}[\![\gamma_1]\!]_{\tau_{\hat{s}}}, q_1)$ which again belongs to Automaton. This choice of Pathfinder is somehow equivalent to the choice of Player 1 in $\mathcal{G}$, i.e., we choose here one arbitrary pair as Automaton has to win whatever Pathfinder chooses. The vertex $(\rho_{\hat{s}}[\![\gamma_1]\!]_{\tau_{\hat{s}}}, q_1)$ corresponds to the vertex $(q_i, \gamma_1(\hat{s}))$ in $\mathcal{G}$. Now we can apply the same case disjunction as before and continue.

As the states $q \in Q_P$ are colored the same way in both games the game $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ starting in vertex $(\rho_{\hat{s}}, \hat{q})$ is won by Automaton if the `HOPDPG` $\mathcal{G}$ is won by Player 0 starting from $(\hat{p}, \hat{s})$. So it follows that $\mathcal{A}$ accepts $\mathbf{t}_k$ starting from $\rho_{\hat{s}}$ in state $\hat{q}$ if Player 0 wins the `HOPDPG` $\mathcal{G}$ starting from $(\hat{q}, \hat{s})$.

Direction from right to left: Now we have to show the other direction, i.e., that if $\mathcal{A}$ accepts $\mathbf{t}_k$ starting from $\rho_{\hat{s}}$ in state $\hat{q}$ then Player 0 wins the parity game $\mathcal{G}$ from vertex $(\hat{q}, \hat{s})$.

The 2-`PTA` $\mathcal{A}$ accepts $\mathbf{t}_k$ starting from $\rho_{\hat{s}}$ in state $\hat{q}$ if Automaton has

a winning strategy in $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ from $(\rho_{\hat{s}}, \hat{q})$. Assume we are in $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ in some vertex (ρ_s, q)[1] where $q \in Q_A \setminus \{q_0\}$. We have again to distinguish whether $q \in Q_0$ or $q \in Q_1$.

Case $\mathbf{q \in Q_0}$. By construction there is a set of transitions of the form $(q, \tau) \to (\llbracket\gamma\rrbracket_\tau, q')$ with $(q, \alpha, \gamma, q') \in \Delta_P$ which can be applied, where τ is the labeling of ρ_s. As Automaton has a winning strategy, we pick one and assume that the strategy chooses the transition $(p, \tau) \to (\llbracket\gamma\rrbracket_\tau, q')$ and goes to vertex $(\rho_s, (p, \tau) \to (\llbracket\gamma\rrbracket_\tau, q'))$. Pathfinder can only choose the pair $(\llbracket\gamma\rrbracket_\tau, q')$ and end in the vertex $(\rho_s\llbracket\gamma\rrbracket, q')$ of Automaton. So in the parity game $\mathcal{G}$ Player 0 has to choose the edge α (the transition (q, α, γ, q')) as his winning strategy in (q, s) and come to vertex $(q', \gamma(s))$.

Case $\mathbf{q \in Q_1}$. By construction there is only one possible transition for Automaton to choose for the current labeling τ of ρ_s in $\mathbf{t}_k$. Let this transition be $(q, \tau) \to (\llbracket\gamma_1\rrbracket_\tau, q_1) \wedge \ldots \wedge (\llbracket\gamma_n\rrbracket_\tau, q_n)$. This means in the parity game $\mathcal{G}$ that Player 1 has to choose one of the transitions $(q, \alpha_1, \gamma_1, q_1), \ldots, (q, \alpha_n, \gamma_n, q_n)$. In the game $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ Pathfinder has to choose one pair $(\llbracket\gamma_i\rrbracket_\tau, q_i)$. So afterwards the vertex $(\rho_s\llbracket\gamma_i\rrbracket, q_i)$ of Automaton is reached. From this we can conclude that Player 1 chooses the transition $(q, \alpha_i, \gamma_i, q_i)$ and ends in $(q_i, \gamma_i(s))$.

Recall that the colors in the game $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ are defined in accordance with the colors of the parity game $\mathcal{G}$. As for each play starting in $(\rho_{\hat{s}}, \hat{q})$ Automaton has a winning strategy the smallest color appearing infinitely often has to be even. This has to hold also for each play starting from $(\hat{q}, \hat{s})$ when Player 0 chooses the edges according to the choices of Automaton. So it follows that Player 0 wins the parity game $\mathcal{G}$ starting from $(\hat{q}, \hat{s})$ if $\mathcal{A}$ accepts $\mathbf{t}_k$ starting from $\rho_{\hat{s}}$ in state $\hat{q}$.

Complexity. The number of states of $\mathcal{A}$ is linear in the number of states in P. The number of transitions of $\mathcal{A}$ is polynomial in the number of transitions in P but gets an exponential blow-up in the size of the instruction set Γ_k^O. More precisely we have

$$\begin{aligned} |Q_A| &= |Q_P| + 1, \\ |\Delta_A| &= |\Delta_P| \cdot (|\Gamma_k^O| \cdot 2^{|\Gamma_k^O|} \cdot k) + (|\Gamma_k^O| \cdot 2^{|\Gamma_k^O|} \cdot k). \end{aligned}$$

□

The relation between the HOPDPG $\mathcal{G}$ and the 2-PTA $\mathcal{A}$ constructed in

[1] For the start we are in $(\rho_{\hat{s}}, \hat{q})$.

Proposition 3.3 can be lifted to strategies. Before stating this correspondence, we show how to represent a pair of *global positional winning strategies* φ_{aut} and φ_{path} in $\mathcal{G}_{\mathcal{A},\mathbf{t}_k}$ for Automaton and Pathfinder as a labeling of $\mathbf{t}_k$ by a finite amount of information. The labeling set is $\mathcal{F}_0 \times \mathcal{F}_1$ where $\mathcal{F}_0$ is the set of all partial functions from Q to Δ and $\mathcal{F}_1$ is the set of all partial functions from Q to $\mathcal{P}(\text{ext}(W) \times Q)$.

The strategy φ_{aut} of Automaton at a node $w \in \text{Dom}(t)$ is given by a partial function ν_0^w from Q to Δ. If the strategy is defined on a state q, it gives the transition which should be chosen in the configuration (w, q). Formally, for all $q \in Q$ and $w \in \text{Dom}(t)$, we have $\nu_0^w(q) = \delta$ iff $\varphi_{\text{aut}}(w, q) = \delta$.

The strategy φ_{path} of Pathfinder can be given by a partial function ν_1^w from Q to $\mathcal{P}(\text{ext}(W) \times Q)$. For all $q \in Q$, we have two cases depending on who wins the game from (w, q). If Automaton wins from (w, q) there exists at least one transition $\delta = (q, t(w)) \to P$, such that Automaton wins no matter which pair $(d_i, q_i) \in P$ Pathfinder would choose, so $\varphi_{\text{path}}(w, \delta)$ is undefined. There might be transitions $\delta_i = (q_i, t(w)) \to P_i$ which Automaton could choose but then there exists a pair $(d_{i,j}, q_{i,j}) \in P_i$ such that Pathfinder wins from $(wd_{i,j}, q_{i,j})$. In this case we have $\varphi_{\text{path}}(w, \delta_i) = (d_{i,j}, q_{i,j})$. But no matter which of this two cases appears $\nu_1^w(q)$ is undefined. If Pathfinder wins from (w, q) then for all transitions $\delta_1, \ldots, \delta_n$ starting with $(q, t(w))$ (i.e., $\delta_j = (q, t(w)) \to P_j$), we have $\varphi_{\text{path}}(w, \delta_j) = (d_{j_i}, q_{j_i})$ for some $(d_{j_i}, p_{j_i}) \in P_j$ for all $j \in [1, n]$. In this case $\nu_1^w(q)$ is equal to $\{(d_{j_i}, q_{j_i}) \mid j \in [1, n]\}$. Intuitively $\nu_1^w(q)$ is defined only if Pathfinder wins from (w, q), i.e., he can respond to every choice of a transition δ of Automaton. In this case $\nu_1^w(q)$ corresponds to a set of directions and states that can refute any transition of Automaton that can be applied in (w, q). Note that for any node $w \in \text{Dom}(t)$, $\text{Dom}(\nu_0^w)$ and $\text{Dom}(\nu_1^w)$ form a partition of Q.

Conversely we say that a $\mathcal{F}_0 \times \mathcal{F}_1$-labeling of $\mathbf{t}_k$ is a global positional strategy Φ for $\mathcal{A}$ on $\mathbf{t}_k$ if it induces a pair of global winning strategies for each player.

Note that for sets of vertices of $\mathbf{t}_k$ the term "regular" means in fact regular for operations.

Proposition 3.4. *Let $\mathcal{G}$ be a level-k higher-order pushdown parity game and let $\mathcal{A}$ be the 2-PTA given by Proposition 3.3. Given a regular global positional strategy Φ for $\mathcal{A}$ on $\mathbf{t}_k$, we can compute, for each player, a*

regular global positional winning strategy and a regular representation of the winning region.

Proof. Let the regular global positional strategy for $\mathcal{A}$ be given by Φ where for all $(\nu_0, \nu_1) \in \mathcal{F}_0 \times \mathcal{F}_1$ the set

$$S_{(\nu_0,\nu_1)} = \{s \in \text{Stacks}_k(\Gamma) \mid \Phi(\rho_s) = (\nu_0^{\rho_s}, \nu_1^{\rho_s}) \wedge (\nu_0^{\rho_s} = \nu_0) \wedge (\nu_1^{\rho_s} = \nu_1)\}$$

is regular for operations. Remember that for all $\rho_s \in \text{Dom}(\mathbf{t}_k)$, the sets $\text{Dom}(\nu_0^{\rho_s})$ and $\text{Dom}(\nu_1^{\rho_s})$ form a partition of Q_A.

We want to compute the regular positional global winning strategies for Player i in $\mathcal{G}$ as a family of sets $(S_\alpha^i)_{\alpha\in\Sigma}$ with $S_\alpha^i \subseteq V_i$ where $(q, s) \in S_\alpha^i$ means that at node $(q, s) \in V_i$ Player i chooses the α-edge.

By the construction of $\mathcal{A}$ we know that if Player 0 chooses at a vertex (p, s) the edge labeled by α, i.e., the transition $\delta = (p, \alpha, \gamma, q) \in \Delta_P$, this is reflected in $\mathcal{A}$ by the choice of a transition of the form $(p, \mathbf{t}_k(\rho_s)) \to (\llbracket\gamma\rrbracket_{\mathbf{t}_k(\rho_s)}, q) \in \Delta_A$ at the node ρ_s in $\mathbf{t}_k$.

So we define for Player 0:

$$\begin{aligned} S_\alpha^0 = \bigcup_{S_{(\nu_0,\nu_1)}} \{(p, s) \in V_0 \mid\ & s \in S_{(\nu_0,\nu_1)} \ \wedge\ p \in \text{Dom}(\nu_0) \wedge \\ & \nu_0(p) = \big((p, \mathbf{t}_k(\rho_s)) \to (\llbracket\gamma\rrbracket_{\mathbf{t}_k(\rho_s)}, q)\big) \\ & \wedge\ (p, \alpha, \gamma, q) \in \Delta_P\} \end{aligned}$$

By construction of $\mathcal{A}$ a choice of Player 1 at a vertex (p, s) in the game is reflected in $\mathcal{A}$ by a transition of the form $(p, \mathbf{t}_k(\rho_s)) \to (\llbracket\gamma_1\rrbracket_{\mathbf{t}_k(\rho_s)}, q_1) \wedge \ldots \wedge (\llbracket\gamma_n\rrbracket_{\mathbf{t}_k(\rho_s)}, q_n)$ where $\delta_1, \ldots, \delta_n$ are all transitions in P starting with p and $\delta_i = (p, \alpha_i, \gamma_i, q_i) \in \Delta_P$ for $i \in [1, n]$. Player 1 can then choose one of the δ_i respectively α_i.

So we define for Player 1:

$$\begin{aligned} S_{\alpha_i}^1 = \bigcup_{S_{(\nu_0,\nu_1)}} \{(p, s) \in V_1 \mid\ & s \in S_{(\nu_0,\nu_1)} \ \wedge\ p \in \text{Dom}(\nu_1) \wedge \\ & (\llbracket\gamma_i\rrbracket_{\mathbf{t}_k(\rho_s)}, q_i) \in \nu_1(p) \\ & \wedge\ \delta_i = (p, \alpha_i, \gamma_i, q_i)\} \end{aligned}$$

By the construction given before and the properties of Φ the sets $(S_\alpha^0)_{\alpha\in\Sigma}$ and $(S_\alpha^1)_{\alpha\in\Sigma}$ are indeed regular positional global winning strategies for Player 0 and Player 1.

The regular winning regions for both players are computed as follows:

$$W_0 = \bigcup_{\alpha\in\Sigma} S_\alpha^0 \cup (V_1 \setminus \bigcup_{\alpha\in\Sigma} S_\alpha^1)$$

$$W_1 = \bigcup_{\alpha \in \Sigma} S^1_\alpha \cup (V_0 \setminus \bigcup_{\alpha \in \Sigma} S^0_\alpha)$$

□

3.4 Computing Strategies over $\mathbf{t}_k$

In this section, we show how to compute a regular global positional strategy for a 2-PTA $\mathcal{A}$ running on $\mathbf{t}_k$. For this we proceed by induction on the level k of the tree $\mathbf{t}_k$. Remember we have shown in the last section, that if we simulate a higher-order pushdown parity game by a 2-PTA, we can compute from the strategies for the 2-PTA, the strategies for the according higher-order pushdown parity game.

In every step of the induction from level $(k+1)$ to level k we have to make two steps. The first is to transform the 2-PTA on $\mathbf{t}_{k+1}$ into a 1-PTA on $\mathbf{t}_{k+1}$ which accepts a labeling that represents a global positional winning strategy for the 2-PTA. The second step transforms the 1-PTA on $\mathbf{t}_{k+1}$ back into a 2-PTA on $\mathbf{t}_k$ accepting the same labeling.

3.4.1 From Two-Way to One-Way

We start with the first step which is based on [Var98]. It consists in showing that for any 2-PTA $\mathcal{A}$ running on $\mathbf{t}_k$, one can construct a 1-PTA $\mathcal{B}$ accepting a $\mathcal{F}_0 \times \mathcal{F}_1$-labeling of $\mathbf{t}_k$ representing a global positional winning strategy for $\mathcal{A}$ on $\mathbf{t}_k$.

In [Var98], a 1-PTA $\mathcal{B}$ is constructed that accepts the trees representing a strategy for Automaton winning from a given state of $\mathcal{A}$ (see [Cac01] for a detailed presentation of the construction). The following proposition simply adapts the construction to make it symmetric between Automaton and Pathfinder.

To prove that we can construct for each 2-PTA on $\mathbf{t}_k$ a 1-PTA on $\mathbf{t}_k$ which accepts a labeling that represents global positional winning strategies for Automaton and Pathfinder for the 2-PTA we proceed in several steps. We first recall the definition of a global positional strategy for $\mathcal{A}$ as labeling of W-trees. We give a characterization of well-formed strategy trees which represent global positional strategies and a 1-PTA that checks them. In the next step we consider the detours that can occur if the same node in the tree is visited several times and define valid annotations which substitute the loops. This can again be verified by

a 1-PTA. In the third step it is tested that there are no losing paths for both players. In the last step these three tests are combined and done by one 1-PTA which accepts the labelings representing global positional strategies for the 2-PTA.

Well Formed Strategy Trees.

Let Σ and W be two finite alphabets and $\mathcal{A} = (Q, I, \Delta, \Omega)$ be a 2-PTA running on a Σ-labeled W-tree. Remember that we have defined $\mathcal{F}_0$ to be the set of all partial functions from Q to Δ and $\mathcal{F}_1$ to be the set of all partial functions from Q to $\mathcal{P}(\text{ext}(W) \times Q)$. To refer to the global positional winning strategies we deal with $\mathcal{F}_0 \times \mathcal{F}_1$-labelings of Σ-labeled W-trees called *strategy trees*.

We introduce in the following some notations and short hands. Let t be a Σ-labeled W-tree and st be a $\mathcal{F}_0 \times \mathcal{F}_1$-labeling of t. For $w \in \text{Dom}(t)$, we denote by ν_0^w and ν_1^w the elements of $\mathcal{F}_0$ and $\mathcal{F}_1$ such that $st(w) = (t(w), \nu_0^w, \nu_1^w)$. Moreover, for $p \in Q$ and $(d, q) \in \text{ext}(W) \times Q$, we write $(d, q) \in \nu_0(p)$ if $\nu_0(p)$ is a transition of the form $(p, \tau) \to P$ with $(d, q) \in P$. Using this convention we have $\nu_0(p) = \{(d, q) \mid (d, q) \in P\}$ for $\nu_0(p) = (p, \tau) \to P$. We use both notations of ν_0 in parallel. By this we do not always have to distinguish between ν_0 and ν_1 in the following.

For a Σ-labeled W-tree t we say that a $\mathcal{F}_0 \times \mathcal{F}_1$-labeling st of t is *well-formed* if for all $w \in \text{Dom}(t)$ holds:

1. $\text{Dom}(\nu_0^w)$ and $\text{Dom}(\nu_1^w)$ form a partition of Q,
2. for all $q \in \nu_0^w$, $\nu_0^w(q)$ is a transition starting with $(q, t(w))$,
3. for all $q \in \nu_1^w$, if $\delta_1 \dots \delta_n$ is the set of transitions in Δ starting with $(q, t(w))$ then for all $i \in [1, n]$, $\nu_1^w(q) \cap P_i \neq \emptyset$ where $\delta_i = (q, t(w)) \to P_i$,
4. for $i \in \{0, 1\}$, $p \in \text{Dom}(\nu_i)$, if $(d, q) \in \nu_i^w(p)$ then wd is defined and $q \in \text{Dom}(\nu_i^{wd})$.

These conditions guaranty that st induces two positional strategies φ_{aut} for Automaton and φ_{path} for Pathfinder. The strategy φ_{aut} is defined by $\varphi_{\text{aut}}(w, p) = \nu_0(p)$ for $w \in \text{Dom}(t)$ and $p \in Q$. We know by Condition 2 that $\nu_0(p)$ is a transition starting with $(p, t(w))$. The definition of φ_{path} is more involved. This is due to the fact that $\nu_1^w(p)$ has to prepare for

each transition that Automaton can choose an answer by Pathfinder, i.e., a pair (d, q). So, by $\varphi_{\text{path}}(w, \delta)$ the correct pair for δ has to be found.

For $w \in \text{Dom}(t)$ and $\delta = (p, t(w)) \to P$ with $P = \{(d_1, q_1), \ldots, (d_n, q_n)\}$, we define φ_{path} by distinguishing several cases:

- If $\nu_1^w(p)$ is defined, $\nu_1^w(p) \cap P$ is non-empty by Condition 3. Let (d, q) be the smallest element of $\nu_1^w(p) \cap P$ for some fixed order, we take $\varphi_{\text{path}}(w, \delta) = (d, q)$.
- If $\nu_1^w(p)$ is undefined and there is an $i \in [1, n]$ such that $q_i \in \text{Dom}(\nu_1^{wd_i})$. Let i_0 be the smallest such i, we take $\varphi_{\text{path}}(w, \delta) = (d_{i_0}, q_{i_0})$.
- Otherwise $\varphi_{\text{path}}(w, \delta)$ is undefined.

The first case considers a configuration (w, p) where Pathfinder has a winning strategy so $\nu_1^w(p)$ is defined and we can choose one arbitrary pair in $\nu_1^w(p) \cap P$. In the second case $\nu_1^w(p)$ is not defined but we can choose a pair (d_i, q_i) to go on such that $\nu_1^{wd_i}(q_i)$ is defined. In the third case Pathfinder will lose whatever he does, so a winning strategy is not defined.

Due to the local nature of the definition of well-formed strategy trees, we can check that a strategy tree is well-formed using only a deterministic 1-PTA.

Lemma 3.5. *For a 2-PTA $\mathcal{A}$ running on Σ-labeled W-trees, there exists a deterministic 1-PTA $\mathcal{A}^I$ accepting the well-formed $\mathcal{F}_0 \times \mathcal{F}_1$-labelings of Σ-labeled W-trees. The number of states of $\mathcal{A}^I$ is $2^{4|Q|}$.*

Proof. The states of $\mathcal{A}^I$ are of the form $(\mathcal{P}(Q) \times \mathcal{P}(Q))^2$. The meaning of the state $((Q_0^\uparrow, Q_0), (Q_1^\uparrow, Q_1))$ is that $Q_0^\uparrow$ corresponds to the set of states for which at the parent node the strategy is defined for Automaton and Q_0 is the set of states for which the parent node requires the strategy to be defined for Player Automaton at the current node and similarly for Pathfinder. In the initial state all four sets are empty, i.e., the initial state is $((\emptyset, \emptyset), (\emptyset, \emptyset))$. For $D \subseteq W$, $\sigma \in \Sigma$, $\nu_0 \in \mathcal{F}_0$ and $\nu_1 \in \mathcal{F}_1$, we have the transition

$$\Big(((Q_0^\uparrow, Q_0), (Q_1^\uparrow, Q_1)), (D, \sigma, \nu_0, \nu_1)\Big) \to \bigwedge_{d \in D} (d, ((\text{Dom}(\nu_0), Q_0^d), (\text{Dom}(\nu_1), Q_1^d)))$$

where

- ν_0 and ν_1 satisfy Conditions 1-3,
- $Q_0 \subseteq \mathrm{Dom}(\nu_0)$ and $Q_1 \subseteq \mathrm{Dom}(\nu_1)$,
- for $i \in \{0,1\}$, if $(q,\varepsilon) \in \nu_i(p)$ for some $p \in Q$, then $q \in \mathrm{Dom}(\nu_i)$,
- for all $d \in D$ and $i \in \{0,1\}$, Q_i^d is the set $\{q \in Q \mid (d,q) \in \nu_i(p) \text{ for some } p \in Q\}$,
- for $i \in \{0,1\}$, if $(q,\uparrow) \in \nu_i(p)$ for some $p \in Q$, then $q \in Q_i^{\uparrow}$.

The acceptance condition is simply that the run is infinite. For this we set the parity for all states to be 0. □

Removing Detours Using Annotations.

Our next task is to check that given a well-formed $\mathcal{F}_0 \times \mathcal{F}_1$-labeling the induced positional strategies φ_{aut} and φ_{path} are winning for the corresponding player from every node in their domain, where they are defined. In this case, we say that the $\mathcal{F}_0 \times \mathcal{F}_1$-labeling is winning. Remark that it follows from Condition 1 that these two strategies are global strategies.

Let st be a strategy tree for $\mathcal{A}$ on a tree t.

For $i \in 0,1$, an i-path is an infinite sequence $(w_0,q_0),(w_1,q_1),\ldots \in (\mathrm{Dom}(t) \times Q)^\omega$ such that for all $n \in \mathbb{N}$, $q_n \in \mathrm{Dom}(\nu_i^{w_n})$ and there exists $(d_n, q_{n+1}) \in \nu_i^{w_n}$ such that $w_{n+1} = w_n d_n$. A 0-path is winning if the associated sequence of states satisfies the parity condition. A 1-path is winning if the associated sequence of states does not satisfy the parity condition. Similar, we define the notion of finite i-paths.

Fact: *A well-formed strategy tree st is winning if and only if all 0-paths and all 1-paths are winning.*

In order to check this condition using an one-way tree automaton, we need to remove the detours in the paths. A *detour* on $w \in \mathrm{Dom}(t)$ is a finite path $\pi = (w_0,q_0),\ldots,(w_n,q_n)$ with $n \geq 1$ such that $w_0 = w_n = w$ and for all $i \in [0,n]$, w is a prefix of w_i. Intuitively, a detour on w is a finite path that never goes above w. From the point of view of the automaton $\mathcal{A}$, a detour on w can be subsumed by the triple (q_0, f, q_n) where f is the smallest parity of the states $q_0,\ldots,q_n$.

The idea is that if we consider the strategy tree annotated with the information of the possible detours, we can check that the strategy tree is winning considering only downward paths.

Let Det be the finite set $\mathcal{P}(Q \times [0,m] \times Q)$ where m is the maximal color used in the 2-PTA $\mathcal{A}$. So, we have sets with entries of the form (p,f,q) which mean that the detour starts in state p goes down in the tree, comes back in state q and the minimal color which is seen in the detour is f. We now consider Det $\times$ Det-labelings of the well-formed strategy trees of $\mathcal{A}$. Such labeling will be referred to as *annotation* of the strategy tree. Consider such an annotation at of a strategy tree st. For all $w \in \mathrm{Dom}(at)$, we denote by L_0^w and L_1^w the elements of Det, labeling w in at.

The annotation at is said to be *valid* if for all $w \in \mathrm{Dom}(at)$ and $i \in \{0,1\}$ holds that if there exists a detour on w from p to q with parity f then $(p,f,q) \in L_i^w$. Intuitively, the labeling contains all the detours on st but may contain more.

The set of valid labelings can be characterized by the following local conditions for $i \in \{0,1\}$ and $w \in \mathrm{Dom}(st)$:

1. for $p \in \mathrm{Dom}(\nu_i^w)$, $(\varepsilon, q) \in \nu_i^w(p) \Rightarrow (p, \min(\Omega(p), \Omega(q)), q) \in L_i^w$,
2. $(p,f,p') \in L_i^w$ and $(p',f',p'') \in L_i^w \Rightarrow (p, \min(f,f'), p'') \in L_i^w$,
3. for $p \in \mathrm{Dom}(\nu_i^w)$ and $d \in W$, $(d,p') \in \nu_i^w(p)$ and $(\uparrow, p'') \in \nu_i^{wd}(p')$ imply that $(p, \min(\Omega(p), \Omega(p'), \Omega(p'')), p'') \in L_i^w$,
4. for $p \in \mathrm{Dom}(\nu_i^w)$ and $d \in W$, $(d,p') \in \nu_i^w(p) \wedge (p',f,p'') \in L_i^{wd} \wedge (\uparrow, p''') \in \nu_i^{wd}(p'')$ imply $(p, \min(\Omega(p), f, \Omega(p''')), p''') \in L_i^w$.

An illustration of the four conditions can be found in Figure 3.3. Note that Condition 3 is not a particular case of Condition 4 as $(p', \Omega(p'), p')$ does not necessarily belong to $L_i^w d$.

Using these local conditions, we define a deterministic 1-PTA accepting the valid Det $\times$ Det-labelings of well-formed strategy trees. So, altogether the deterministic 1-PTA runs on a $(\mathcal{F}_0 \times \mathcal{F}_1) \times (\mathrm{Det} \times \mathrm{Det})$-labeling of a Σ-labeled W-tree.

Lemma 3.6. *Given a 2-PTA $\mathcal{A}$ running on Σ-labeled W-trees, there exists a deterministic 1-PTA $\mathcal{A}^{II}$ accepting the annotations of strategy trees of $\mathcal{A}$ which are valid. Moreover, the number of states of $\mathcal{A}^{II}$ is $2^{2|Q|^2(m+1)}$ where m is the maximal color in $\mathcal{A}$.*

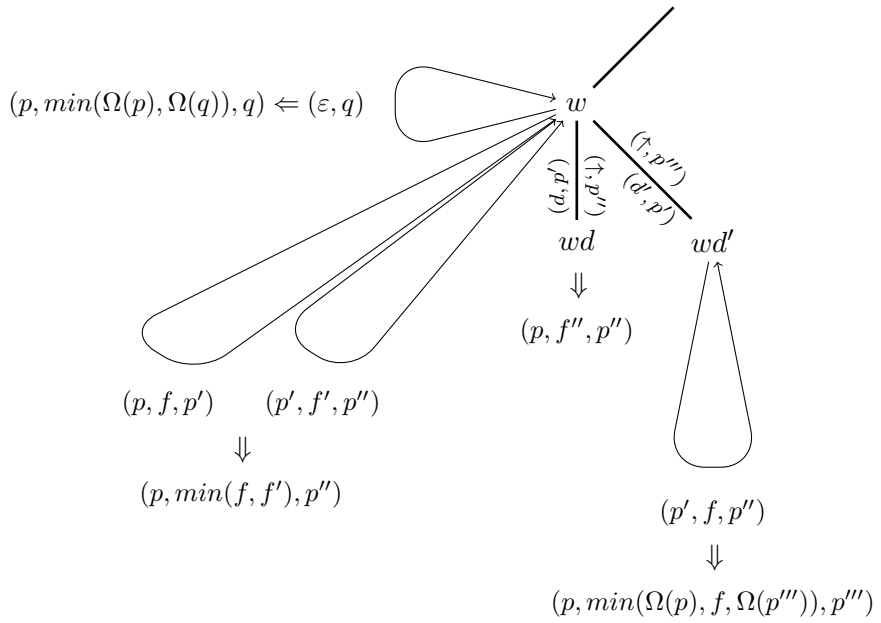

Figure 3.3: Visualization of annotation. $f'' = min(\Omega(p), \Omega(p'), \Omega(p''))$.

Proof. The Conditions 1 and 2 can be checked without memory. To check Conditions 3 and 4, we need to memorize for each player the set of detours of the parent node together with part of the strategy of the parent node concerning the current node.

The states of $\mathcal{A}^{II}$ have the form $(\text{Det} \times \mathcal{P}(Q \times Q))^2$. The intuitive meaning of a state $((L_0^{\uparrow}, R_0), (L_1^{\uparrow}, R_1))$ at a node w is that $L_0^{\uparrow}$ is the set of detours at the parent node for Automaton and $(p, q) \in R_0$ means that at the parent node of w the strategy for Automaton sends for state p the state q to w. For Pathfinder $L_1^{\uparrow}$ and R_1 are defined analogously. The initial state is $((\emptyset, \emptyset), (\emptyset, \emptyset))$.

For $D \subseteq W$, $\sigma \in \Sigma$, $\nu_0 \in \mathcal{F}_0$, $\nu_1 \in \mathcal{F}_1$ and $L_0, L_1 \in \text{Det}$, we have the transition

$$\Big(((L_0^{\uparrow}, R_0), (L_1^{\uparrow}, R_1)), (D, \sigma, \nu_0, \nu_1, L_0, L_1)\Big) \to \bigwedge_{d \in D} (d, ((L_0, R_0^d), (L_1, R_1^d)))$$

where

- L_0 and L_1 satisfy Condition 1 and 2,
- for $i \in \{0, 1\}$, $R_i^d = \{(p, q) \mid (q, d) \in \nu_i(p)\}$,

- if $(p, p') \in R_i$ and $(p'', \uparrow) \in \nu_i(p')$ then $(p, \min(\Omega(p), \Omega(p'), \Omega(p'')), p'') \in L_i$ (Condition 3),
- if $(p, p') \in R_i$, $(p', f, p'') \in L_i$ and $(p''', \uparrow) \in \nu_i(p'')$ then $(p, \min(\Omega(p), f, \Omega(p''')), p''') \in L_i$ (Condition 4).

The acceptance condition is simply that the run is infinite, i.e., all states have the same even parity. □

Elimination of Losing Paths.

In an annotation at of a strategy tree st for $i \in \{0, 1\}$, a downward i-path is a sequence $(w_0, q_0), l_0, (w_1, q_1), l_1 \ldots \in ((\mathrm{Dom}(t) \times Q)(\mathrm{Det} \cup \varepsilon))^\omega$ such that for all $n \in \mathbb{N}$ either $l_n = (q_n, f_n, q'_n) \in L_i^{w_n}$ and there exists $(d_n, q_{n+1}) \in \nu_i^{w_n}(q'_n)$ for some $d_n \in W$ with $w_{n+1} = w_n d_n$ or $l_n = \varepsilon$ and there exists $(d_n, q_{n+1}) \in \nu_i^{w_n}(q_n)$ for some $d_n \in W$ with $w_{n+1} = w_n d_n$. The parity of this path is the smallest parity appearing infinitely often as parity of a state or as one of the f_n.

A *losing path* for Automaton on at is either an infinite downward 0-path that does not satisfy the parity condition or a node $w \in \mathrm{Dom}(t)$ such that for some $q \in Q$ and odd parity f, $(q, f, q) \in L_0^w$. A losing path for Pathfinder on at is either an infinite downward 1-path that satisfy the parity condition or a node $w \in \mathrm{Dom}(t)$ such that for some $q \in Q$ and even parity f, $(q, f, q) \in L_0^w$.

Lemma 3.7. *Let $\mathcal{A}$ be a 2-PTA running on Σ-labeled W-trees. A well-formed strategy tree st for $\mathcal{A}$ is winning iff there exists a valid annotation at of st that does not contain any losing path for* Automaton *and* Pathfinder.

Proof. For the direct implication assume that st is winning. It is enough to show that for the annotation at, corresponding to the actual detours, st does not contain any losing path for Automaton or Pathfinder. It is easy to see that from any losing path on at for Automaton (resp. for Pathfinder), we can construct a 0-path (respectively 1-path) on st that does not satisfy the parity condition (respectively that satisfies the parity condition). This contradicts the assumption that st is winning.

For the converse implication assume that there exists a valid annotation at of the strategy tree st which does not contain any losing paths for Automaton or Pathfinder. Suppose by contradiction that st is not

winning. We can assume w.l.o.g. that there exists a path π in st for Automaton that does not satisfy the parity condition. We show that there exists in at a losing 0-path. We distinguish two cases (see also Figure 3.4).

For the first case we assume that there exists a node in $\mathrm{Dom}(st)$ appearing infinitely often in π. Let w_0 be a node appearing infinitely often in π which is minimal for the prefix order. Let q_0 be a state of $\mathcal{A}$ such that the configuration (w_0, q_0) appears infinitely often in π. By minimality of w_0, there exists a suffix of π that can be written as an infinite concatenation of detours on w_0 starting from q_0 to q_0. As π does not satisfy the parity condition, there exists a detour on w_0 from q_0 to q_0 with an odd parity f. Finally as at is valid it contains all detours on st. Hence (q_0, f, q_0) belongs to $L_0^{w_0}$ and at contains a losing path for Automaton.

For the second case there is no node which appears infinitely often in π. In this case we can construct an infinite 0-path $(w_0, q_0)l_0(w_1, q_1)\dots$ on at with the same parity as π. Let w_0 be the smallest node appearing in π. The path can be uniquely written as $\pi'(w_0, q_0)\pi_0$ where π' does not contain w_0. By minimality of w_0, all nodes in π_0 have w_0 as a prefix. We define $l_i \in (\mathrm{Det} \cup \varepsilon), w_i \in \mathrm{Dom}(st), q_i \in Q$ and π_i a suffix of π by induction on i. The induction property is that π_i start with the configuration (q_i, w_i) and that all nodes appearing in π_i have w_i as a prefix. Assume that for some $i \geq 0$, we have defined $l_i \in (\mathrm{Det} \cup \varepsilon), w_i \in \mathrm{Dom}(st), q_i \in Q$ and π_i a suffix of π. If w_i does not appear in π_i, we take $l_i = \varepsilon$, (w_{i+1}, q_{i+1}) to be the first element of π_i and π_{i+1} to be π_i deprived of its first element. If w_i appears in π_i, then as w_i does not appear infinitely often, π_i can be written as $\pi'(w_{i+1}, q_{i+1})\pi_{i+1}$ where π' is a detour on w_i form q_i to q_i' and w_i does not appear in $(w_{i+1}, q_{i+1})\pi_{i+1}$. We take l_i to be the value of the detour π'. The infinite 0-path we constructed has the same parity as π which yields the contradiction. □

We show next that it is possible to accept winning and valid annotations of a well-formed strategy tree using a deterministic 1-PTA.

Lemma 3.8. *Let $\mathcal{A}$ be a 2-PTA running on Σ-labeled W-trees. There exists a deterministic 1-PTA $\mathcal{A}^{III}$ accepting the valid annotations of well-formed strategy trees for $\mathcal{A}$ that are winning. Moreover the number of states of $\mathcal{A}^{III}$ is exponential in the number of states of $\mathcal{A}$ and the number of colors of $\mathcal{A}^{III}$ is linear in the number of colors of $\mathcal{A}$.*

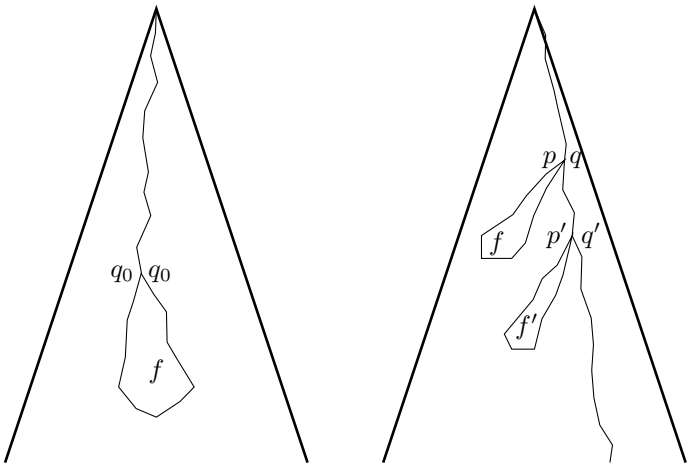

Figure 3.4: Illustration of Lemma 3.7.

Proof. An infinite branch in a valid annotation at of a well formed strategy tree st is a word in $V \times W$, where V is the set of labelings of at. We can construct a non-deterministic parity word automaton $\mathcal{D}$ which accepts the set of branches $\mathcal{L}$ containing a suffix corresponding to a losing path for Automaton or Pathfinder. The automaton $\mathcal{D}$ uses a number of states polynomial in the number of states of $\mathcal{A}$. So the parity word automaton $\mathcal{D}$ guesses a losing path in the tree. Using the standard construction, we can compute a deterministic parity word automaton $\mathcal{E}$ accepting the complement of $\mathcal{L}$. This construction leads to an exponential blowup in the number of states of $\mathcal{E}$. From $\mathcal{E}$, we obtain a deterministic 1-PTA $\mathcal{A}^{III}$ with a same number of states which accepts a valid annotation at of a well formed strategy if and only if all its branches belong to the complement of $\mathcal{L}$ or in other words, if at does not contain any losing paths for Automaton or Pathfinder. □

Combining the Automata.

Now we have all requirements we need to show the following proposition which combines the results we have shown before.

Proposition 3.9. *Given a 2-PTA $\mathcal{A}$ running on $\mathbf{t}_k$, we can construct a 1-PTA $\mathcal{B}$ accepting the trees representing a global positional winning*

strategy of $\mathcal{A}$. Furthermore, the size of $\mathcal{B}$ is exponential in the size of $\mathcal{A}$ but the number of colors of $\mathcal{B}$ is linear in the number of colors of $\mathcal{A}$.

Proof. Let $\mathcal{A} = (Q, I, \Delta, \Omega)$ be a 2-PTA running Σ-labeled W-tree. If we build the product automaton of the deterministic 1-PTAs $\mathcal{A}^I$, $\mathcal{A}^{II}$ and $\mathcal{A}^{III}$ constructed in Lemma 3.5, 3.6 and 3.8, we obtain a deterministic 1-PTA $\mathcal{A}^{IV}$ running on annotations of strategy trees of $\mathcal{A}$ such that $\mathcal{A}^{IV}$ accepts an annotation at of a strategy tree st if and only if st is well-formed and at is a valid and winning annotation. The number of states of $\mathcal{A}^{IV} = (Q_{IV}, q_0^{IV}, \Delta_{IV}, \Omega_{IV})$ is exponential in the number of states of $\mathcal{A}$ and the number of colors of $\mathcal{A}^{IV}$ is linear in the number of colors of $\mathcal{A}$.

We now consider the non-deterministic 1-PTA $\mathcal{B}$ obtained from $\mathcal{A}^{IV}$ by guessing the annotations instead of reading them. For this the guessed annotations become part of the states. Formally the set of states $Q_{\mathcal{B}}$ of $\mathcal{B}$ is $Q_{IV} \times \mathrm{Det} \times \mathrm{Det}$. The set of initial states is $\{q_0^{IV}\} \times \mathrm{Det} \times \mathrm{Det}$. The parity function $\Omega^{\mathcal{B}}$ is defined by $\Omega_{\mathcal{B}}(q, L_0, L_1) = \Omega_{IV}(q)$. Finally, the transition relation $\Delta_{\mathcal{B}}$ is defined as follows. For

$$(p, (D, \sigma, \nu_0, \nu_1, L_0, L_1)) \to \bigwedge_{d \in D} (d, q_d) \in \Delta_{IV}$$

we have:

$$((p, L_0, L_1), (D, \sigma, \nu_0, \nu_1)) \to \bigwedge_{d \in D} (d, (q_d, L_0^d, L_1^d)) \in \Delta_{\mathcal{B}}$$

where for all $d \in D$, L_0^d and L_1^d belong to Det.

The 1-PTA $\mathcal{B}$ accepts a strategy tree st if and only if there exists a winning and valid annotation at of st. By Lemma 3.7, $\mathcal{B}$ accepts a strategy tree st of $\mathcal{A}$ if and only if st represents a global positional strategy of $\mathcal{A}$.

The number of states of $\mathcal{B}$ is exponential in the number of states of $\mathcal{A}$ and the number of colors of $\mathcal{B}$ is linear in the number of colors of $\mathcal{A}$. □

3.4.2 Reducing the Level

Using Proposition 3.9, we have reduced our initial problem to compute a global positional strategy for a 2-PTA $\mathcal{A}$ running on $\mathbf{t}_{k+1}$ to the problem of computing a regular Ξ-labeling of $\mathbf{t}_{k+1}$ accepted by a 1-PTA $\mathcal{B}$. In the next step, we reduce this problem to the problem of computing a global positional strategy for a 2-PTA $\mathcal{C}$ running on $\mathbf{t}_k$.

The construction is based on the fact that $\mathcal{B}$ does not use the $\uparrow$ direction and hence when running on $\mathbf{t}_{k+1}$ only take directions in $\Gamma_k^O \cup \{k\}$. From the point of view of the actions over the stacks, $\mathcal{B}$ does not perform the $\overline{copy}_k$ operation and hence can only access the top-most level-k stack. Intuitively, we can simulate the same behavior at one level below by replacing the direction k by ε. For this see also Lemma 2.13. Now, we need to use alternation instead of performing the $copy_k$ operation. By deleting the $copy_k$, respectively, substituting it by ε and working on level-k instead of level-$(k+1)$, we have to use in the 2-PTA again the $\uparrow$-direction. In the 1-PTA on $\mathbf{t}_{k+1}$ we have followed by construction of $\mathbf{t}_{k+1}$ in each path only reduced sequences but if we delete all appearances of $copy_k$ on $\mathbf{t}_k$ this no longer holds. Consider for example the reduced sequence $ab1\bar{b}1\bar{a}b$, applied to $[\ ]_2$ we get $[[ab][a][b]]_2$. If we delete all 1's we get $ab\bar{b}\bar{a}b$ which is no longer reduced.

The construction of the 2-PTA is a bit more technical as we need to relate the surroundings (d, D, e) in $\mathbf{t}_{k+1}$ which belong to a node corresponding to a level-$(k+1)$ stack s to the surroundings (d', D', e') in $\mathbf{t}_k$ of the node corresponding to the level-k stack $top_k(s)$.

First, we state the proposition and give the construction of the 2-PTA. Next, we state some lemmas we need to prove the correctness of the construction. Afterwards, the correctness of the construction is shown. In the last step, we demonstrate how to reconstruct the regular Ξ-labeling.

Proposition 3.10. *Given a 1-PTA $\mathcal{B}$ accepting at least one Ξ-labeling of $\mathbf{t}_{k+1}$, we can construct a 2-PTA $\mathcal{C}$ running on $\mathbf{t}_k$ such that given a regular global positional strategy for $\mathcal{C}$ on $\mathbf{t}_k$, we can construct a regular Ξ-labeling of $\mathbf{t}_{k+1}$ accepted by $\mathcal{B}$. Furthermore, the size of $\mathcal{C}$ is polynomial in the size of $\mathcal{B}$.*

Construction. Let $\mathcal{B} = (Q_\mathcal{B}, \Delta_\mathcal{B}, I_\mathcal{B}, \Omega_\mathcal{B})$ be a 1-PTA accepting a Ξ-labeling of $\mathbf{t}_{k+1}$. We can assume w.l.o.g. that the automaton $\mathcal{B}$ runs on $\mathbf{t}_{k+1}$ and guesses the Ξ-labeling (i.e., the states of $Q_\mathcal{B}$ are of the form $Q'_\mathcal{B} \times \Xi$.

As we are only interested in the behavior of $\mathcal{B}$ on $\mathbf{t}_{k+1}$, we can assume w.l.o.g. that for all transitions $(q, (d, D, e)) \to (q_1, \gamma_1) \wedge \ldots (q_n, \gamma_n) \in \Delta_\mathcal{B}$, the set D contains k and does not contain $\overline{k}$ and $\overline{d}$. Indeed transitions which do not satisfy these conditions are not applicable in a run of $\mathcal{B}$ on $\mathbf{t}_{k+1}$.

We define the 2-PTA $\mathcal{C} = (Q_\mathcal{C}, \Delta_\mathcal{C}, I_\mathcal{C}, \Omega_\mathcal{C})$ with $Q_\mathcal{C} = Q_\mathcal{B} \times (\Gamma_k^O \cup \{k, \diamond\})$, $I_\mathcal{C} = I_\mathcal{B} \times \{\diamond\}$ and $\Omega_\mathcal{C}(p, d) = \Omega_\mathcal{B}(p)$ for all $d \in \Gamma_k^O \cup \{k, \diamond\}$.

For each transition

$$\delta := (q, (d, D, e)) \to (\gamma_1, q_1) \wedge \ldots \wedge (\gamma_n, q_n) \in \Delta_\mathcal{B},$$

and for all $d' \in \Gamma_k^O \cup \{\diamond\}$, we add the following transition to $\Delta_\mathcal{C}$

$$\delta_{\downarrow_{d'}} := ((q, d), (d', D', e')) \to (\gamma_1', (q_1, \gamma_1)) \wedge \ldots \wedge (\gamma_n', (q_n, \gamma_n))$$

where

1. $d = \diamond \Rightarrow d' = \diamond$ and $(d' \neq \diamond \wedge d' \neq d) \Rightarrow \overline{d'} \in D$,
2. $D' = (D \cup \{\overline{d}\}) \setminus \{\overline{d'}, k, \overline{k}\}$,
3. $e' = e$ if $e < k + 1$ and $e' = k$ if $e = k + 1$, i.e., $e' = \min\{e, k\}$,
4. for all $i \in [1, n]$, $\gamma_i' = \begin{cases} \varepsilon & \gamma_i = k \\ \uparrow & \gamma_i = \overline{d'} \\ \gamma_i & \text{otherwise.} \end{cases}$

Intuitively, the automaton $\mathcal{C}$ simulates the actions of $\mathcal{B}$ on the top-most level-k stack. This is enough to capture the whole behavior of $\mathcal{B}$ as $\mathcal{B}$ never performs the $\overline{copy_k}$ operation and so cannot reach some level-k stack below the topmost one. The Conditions 1 to 3 relate the surrounding (d, D, e) of a node ρ_s that corresponds to a level-$(k{+}1)$ stack s in $\mathbf{t}_{k+1}$ to the surrounding (d', D', e') of the node $\rho_{top_k(s)}$ corresponding to the level-k stack $top_k(s)$ in $\mathbf{t}_k$. Condition 4 reflects the fact that the $copy_k$ operation does not modify the top-most level-k stack (i.e., the direction k is replaced by ε). Furthermore, Condition 4 takes into account that if the operation $copy_k$ is deleted from a reduced instruction sequence then the resulting sequence does no longer have to be reduced. So it has to be checked if $\gamma_i = \overline{d'}$ and if so the direction in the transition has to be $\uparrow$.

The number of states of $\mathcal{C}$ is polynomial in the number of states of $\mathcal{B}$ and the size of instruction set Γ_k^O. More precisely we have $|Q_\mathcal{C}| = |Q_\mathcal{B}| \cdot (|\Gamma_k^O| + 2)$. The number of transitions of $\mathcal{C}$ is polynomial in the number of transitions of $\mathcal{B}$ and the size of instruction set Γ_k^O. More precisely we have $|\Delta_\mathcal{C}| = |\Delta_\mathcal{B}| \cdot (|\Gamma_k^O| + 1)$.

An important property for the transitions of the 2-PTA $\mathcal{C}$ is that for every $\delta \in \Delta_{\mathcal{C}}$, there exists a unique transition $\delta^{\uparrow} \in \Delta_{\mathcal{B}}$ and a unique $d' \in \Gamma_k^O \cup \{\diamond\}$ such that $\delta = (\delta^{\uparrow})_{\downarrow_{d'}}$. Moreover, if the labels of δ and $\delta^{\uparrow}$ are respectively (d', D', e') and (d, D, e), we have $D = (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}$. We prove these properties of $\Delta_{\mathcal{C}}$ in the following Lemma 3.11. These properties allow us to lift a regular positional strategy for $\mathcal{C}$ on $\mathbf{t}_k$ to a regular positional strategy of $\mathcal{B}$ on $\mathbf{t}_{k+1}$. Consequently, we can compute a regular Ξ-labeling accepted by $\mathcal{B}$ following the regular global positional strategy for $\mathcal{B}$.

Lemma 3.11. *For every $\delta \in \Delta_{\mathcal{C}}$, there exists a unique transition $\delta^{\uparrow} \in \Delta_{\mathcal{B}}$ and a unique $d' \in \Gamma_k^O \cup \{\diamond\}$ such that $\delta = (\delta^{\uparrow})_{\downarrow_{d'}}$. Moreover, if the labels of δ and $\delta^{\uparrow}$ are respectively (d', D', e') and (d, D, e), we have $D = (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}$.*

Proof. Consider the transition $\delta' \in \Delta_{\mathcal{C}}$ which has the labeling (d', D', e'). We first establish that for all $\delta \in \Delta_{\mathcal{B}}$ with labeling (d, D, e) where $(\delta)_{\downarrow_{d'}} = \delta'$, we have:

$$D = (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}$$

By definition of $\delta_{\downarrow_{d'}}$, we have:

$$D' = (D \cup \{\overline{d}\}) \setminus \{\overline{d'}, k, \overline{k}\}$$

We distinguish four cases depending on whether $d = \diamond$, $d \neq \diamond$ and $d = d'$, $d = k$ or $d \notin \{d', k, \diamond\}$. We use several time the convention that $\overline{\diamond}$ is not defined. This follows from the fact that $\overline{\diamond}$ would mean to go up when being at the root of the tree and this is not possible.
If $d = \diamond$ then $d \neq k$ and $d \neq \bar{k}$. By convention holds that $k \in D$ and $\bar{k} \notin D$ and by definition of $\delta_{\downarrow_{d'}}$ we know that $d' = \diamond$.

It follows that:

$$\begin{array}{lcll} D' & = & (D \cup \{\overline{\diamond}\}) \setminus \{\overline{\diamond}, k, \overline{k}\} & \\ D' & = & D \setminus \{k, \overline{k}\} & \textbf{as} \quad \overline{\diamond} \text{ is not defined} \\ D' & = & (D \setminus \{k\}) & \textbf{as} \quad \overline{k} \notin D \\ D' \cup \{k\} & = & D & \textbf{as} \quad k \in D \end{array}$$

So using the convention that $\overline{\diamond}$ is not defined, we can conclude $D = (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}$.
If $d \neq \diamond$ **and** $d = d'$ we know $d \neq k$ and $d \neq \bar{k}$ and by convention $k \in D$ and $\bar{k} \notin D$.

It follows that:

$$\begin{array}{lcll} D' & = & (D \cup \{\overline{d}\}) \setminus \{\overline{d'}, k, \overline{k}\} & \\ D' & = & (D \setminus \{k, \overline{k}\}) & \textbf{as } \overline{d} = \overline{d'}, \overline{d} \notin \{k, \overline{k}\} \textbf{ and } \overline{d} \notin D \\ D' & = & (D \setminus \{k\}) & \textbf{as } \overline{k} \notin D \\ D' \cup \{k\} & = & D & \textbf{as } k \in D \end{array}$$

By definition of $\delta_{\downarrow_{d'}}$, we have $\overline{d'} \notin D'$ and by the precondition we know $\overline{d} = \overline{d'}$. So we have $D = (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}$.

If $d = k$ then $d \neq d'$ and $d \neq \bar{k}$ and by convention $k \in D$ and $\bar{k} \notin D$.

If $d' = \diamond$ then it follows that:

$$\begin{array}{lcll} D' & = & (D \cup \{\overline{k}\}) \setminus \{\overline{\diamond}, k, \overline{k}\} & \\ D' & = & (D \cup \{\overline{k}\}) \setminus \{k, \overline{k}\} & \textbf{as } \overline{\diamond} \text{ not defined} \\ D' & = & D \setminus \{k\} & \textbf{as } \overline{k} \notin D \\ D' \cup \{k\} & = & D & \textbf{as } k \in D \end{array}$$

As $\bar{k} \notin D'$ and by using the convention that $\overline{\diamond}$ is not defined, we can conclude that $D = (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}$.

If $d' \neq \diamond$, then by definition of $\delta_{\downarrow_{d'}}$ we have $\overline{d'} \in D$ and it follows that:

$$\begin{array}{lcll} D' & = & (D \cup \{\overline{k}\}) \setminus \{\overline{d'}, k, \overline{k}\} & \\ D' & = & D \setminus \{\overline{d'}, k\} & \textbf{as } \overline{k} \notin D \\ D' \cup \{\overline{d'}, k\} & = & D & \textbf{as } \{k, \overline{d'}\} \subseteq D \end{array}$$

As $\bar{k} \notin D'$, $D = (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}$.

If $d \notin \{k, d', \diamond\}$ then we know by convention that $d \neq \bar{k}$, $k \in D$ and $\bar{k} \notin D$.

If $d' = \diamond$ then it follows that:

$$\begin{array}{lcll} D' & = & (D \cup \{\overline{d}\}) \setminus \{\overline{\diamond}, k, \overline{k}\} & \\ D' & = & (D \cup \{\overline{d}\}) \setminus \{k, \overline{k}\} & \textbf{as } \overline{\diamond} \text{ not defined} \\ D' & = & (D \setminus \{k, \overline{k}\}) \cup \{\overline{d}\} & \textbf{as } \overline{d} \notin \{k, \overline{k}\} \\ D' \setminus \{\overline{d}\} & = & D \setminus \{k, \overline{k}\} & \textbf{as } \overline{d} \notin D \\ (D' \setminus \{\overline{d}\}) \cup \{k\} & = & D & \textbf{as } k \in D \textbf{ and } \overline{k} \notin D \\ (D' \cup \{k\}) \setminus \{\overline{d}\}) & = & D & \textbf{as } \overline{d} \neq k \end{array}$$

Using the convention that $\overline{\diamond}$ is not defined we have $D = (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}$.

If $d' \neq \diamond$, then by definition of $\delta_{\downarrow_{d'}}$, $\overline{d'} \in D$ and it follows that:

$$\begin{array}{lll} D' & = (D \cup \{\overline{d}\}) \setminus \{\overline{d'}, k, \overline{k}\} & \\ D' & = (D \setminus \{\overline{d'}, k, \overline{k}\}) \cup \{\overline{d}\} & \textbf{as } \overline{d} \notin \{\overline{d'}, k, \overline{k}\} \\ D' \setminus \{\overline{d}\} & = D \setminus \{\overline{d'}, k, \overline{k}\} & \textbf{as } \overline{d} \notin D \\ (D' \setminus \{\overline{d}\}) \cup \{\overline{d'}, k\} & = D & \textbf{as } k \in D, \overline{d'} \in D \textbf{ and } \overline{k} \notin D \\ (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}) & = D & \textbf{as } k \neq \overline{d} \textbf{ and } d \neq d' \end{array}$$

In all cases, it follows that $D = (D' \cup \{\overline{d'}, k\}) \setminus \{\overline{d}\}$. So we have shown that from δ' we can uniquely determine D. The e can be uniquely determined by the fact that e' is always equal to e except for the case where we are at the root $[\ ]_{k+1}$ but this can be easily seen because in this case we have $d = \diamond$. The rest of the transition $\delta^{\uparrow}$ is known because it is stored in the states of $\mathcal{C}$, i.e., we store the last direction which is taken in the states. □

The following lemma establishes some basics about reduced sequences which are introduced in Section 2.2.3.

Lemma 3.12. *For every stack $s \in \mathrm{Stacks}_{k+1}(\Gamma)$, the reduced sequence $\rho_{top_k(s)}$ of $top_k(s)$ is obtained by reducing the sequence obtained by erasing all occurrences of k in ρ_s. In particular, considering two stacks s and s' such that $\rho_s = \rho_{s'}\gamma$ for some $\gamma \in \Gamma_k^O \cup \{k\}$, the respective reduced sequences ρ and ρ' of $top_k(s)$ and $top_k(s')$ are linked by*

$$\rho = \begin{cases} \rho' & \text{if } \gamma = k \\ \rho'[1, |\rho'| - 1] & \text{if } \gamma = \overline{\mathrm{Last}(\rho')} \\ \rho'\gamma & \text{otherwise} \end{cases}$$

Proof. The first statement is a direct consequence of Lemma 2.13 and can be proved by induction on the length of ρ_s. The second can be shown by a case distinction on γ. □

The next remark quotes some basic facts of the surroundings of the nodes in the trees $\mathbf{t}_k$ and $\mathbf{t}_{k+1}$.

Remark 3.13. *For all stacks $s \in \mathrm{Stacks}_{k+1}(\Gamma)$ with the surroundings $\mathbf{t}_k(\rho_{top_k(s)}) = (d, D, e)$ and $\mathbf{t}_{k+1}(\rho(s)) = (d', D', e')$, we have:*

$$\begin{aligned} \ell_{k+1}(s) &= (\mathrm{Last}(s), (D \cup \{\overline{d}, k\}) \setminus \overline{\mathrm{Last}(s)}, e'') \\ &\quad \text{where } e'' = k+1 \text{ if } \mathrm{Last}(s) = \diamond \text{ and } e'' = e \text{ otherwise} \\ \ell_k(top_k(s)) &= (\mathrm{Last}(top_k(s)), (D' \cup \{\overline{d'}\}) \setminus \{\overline{\mathrm{Last}(top_k(s))}, k, \overline{k}\}, e''') \\ &\quad \text{where } e''' = \min\{e', k\}. \end{aligned}$$

Note that we still imply that $\overline{\diamond}$ is not defined as it would mean to go up at the root of the tree.

Example 9. We summarize in the following some properties of a level-3 stack s concerning the surrounding. Let s be the following level-3 stack

$$s = [[[ab][abb]]\ [[ab][aa][aa]]\ [[ab][aa][aa]]]_3.$$

- $\rho_s = ab1b2\bar{b}\bar{b}a12$
- $top_2(s) = [[ab][aa][aa]]_2$ with $\rho_{top_2(s)} = ab1\bar{b}a1$
- $\mathbf{t}_3(\rho_s) = (2, \{a, \bar{a}, b, 1, \bar{1}, 2\}, 0)$
- $\mathbf{t}_2(\rho_{top_2(s)}) = (1, \{a, \bar{a}, b, 1\}, 0)$

So it follows that:

- $\ell_3(s) = (2, \{a, \bar{a}, b, 1\} \cup \{\bar{1}, 2\} \setminus \{\bar{2}\} = \{a, \bar{a}, b, 1, \bar{1}, 2\}, 0)$ and
- $\ell_2(top_2(s)) = (1, \{a, \bar{a}, b, 1, \bar{1}, 2\} \cup \{\bar{2}\} \setminus \{\bar{1}, 2, \bar{2}\} = \{a, \bar{a}, b, 1\}, 0)$.

Now we can prove the correctness of the construction for Proposition 3.10, i.e.,we show that $\mathcal{B}$ accepts $\mathbf{t}_{k+1}$ if and only if $\mathcal{C}$ accepts $\mathbf{t}_k$.

Direct Implication. Assume that the 1-PTA $\mathcal{B}$ accepts $\mathbf{t}_{k+1}$ with a run $r_{\mathcal{B}} \colon \mathrm{Dom}(\mathbf{t}_{k+1}) \to Q_{\mathcal{B}}$. We construct an accepting run $r_{\mathcal{C}}$ of $\mathcal{C}$ on $\mathbf{t}_k$. We take $\mathrm{Dom}(r_{\mathcal{C}}) = T$ equal to $\mathrm{Dom}(r_{\mathcal{B}})$ and the labeling $r_{\mathcal{C}} \colon T \to \mathrm{Dom}(\mathbf{t}_k) \times Q_{\mathcal{C}}$ is defined for a stack $s \in \mathrm{Stacks}_{k+1}(\Gamma)$ with $\rho_s \in T$ by

$$r_{\mathcal{C}}(\rho_s) = (\rho_{top_k(s)}, (r_{\mathcal{B}}(\rho_s), \mathrm{Last}(\rho_s))).$$

To show that $r_{\mathcal{C}}$ is an accepting run for $\mathcal{C}$ on $\mathbf{t}_k$, we only need to show that $r_{\mathcal{C}}$ is well-formed. Indeed the fact that $r_{\mathcal{C}}$ satisfies the acceptance condition follows from the fact that $r_{\mathcal{B}}$ does.

Let $s \in \mathrm{Stacks}_{k+1}(\Gamma)$ be such that ρ_s belongs to $T = \mathrm{Dom}(r_{\mathcal{B}})$. We write $d = \mathrm{Last}(\rho_s)$ and $d' = \mathrm{Last}(top_k(\rho_s))$. As ρ_s belongs to $\mathrm{Dom}(r_{\mathcal{B}})$, there exists a transition

$$\delta := (r_{\mathcal{B}}(\rho_s), (d, D, e)) \to (\gamma_1, q_1) \wedge \ldots \wedge (\gamma_n, q_n) \in \Delta_{\mathcal{B}}$$

such that for all $i \in [1, n]$, $r_{\mathcal{B}}(\rho_s\gamma_i) = q_i$. By definition of $\Delta_{\mathcal{C}}$, the following transition belongs to $\Delta_{\mathcal{C}}$

$$\begin{aligned} \delta' \quad := \quad & \big((r_{\mathcal{B}}(\rho_s), d), (d', (D \cup \{\overline{d}\}) \setminus \{\overline{d'}, k, \overline{k}\}, e')\big) \\ & \to (\gamma'_1, (q_1, \gamma_1)) \wedge \ldots \wedge (\gamma'_n, (q_n, \gamma_n)) \end{aligned}$$

where e' and γ'_i are defined as above.

We claim that the transition δ' can be applied at the node ρ_s of $r_{\mathcal{C}}$. By Lemma 3.13, the labeling of $\rho_{top_k(s)}$ in $\mathbf{t}_k$ is equal to $(d', (D \cup \{\overline{\mathrm{Last}(\rho_s)}\} \setminus \{\overline{d'}, k, \overline{k}\}, e')$.

It remains to show for all $i \in [1,n]$ that $r_{\mathcal{C}}(\rho_s\gamma_i) = (\rho_{top_k(s)}\gamma'_i, (q_i,\gamma_i))$. We denote by s_i the level-(k+1) stack with the reduced sequence $\rho_s\gamma_i$. By definition of the run $r_{\mathcal{C}}$ we have $r_{\mathcal{C}}(\rho_s\gamma_i) = (\rho_{top_k(s_i)}, (r_{\mathcal{B}}(\rho_s\gamma_i), \mathrm{Last}(\rho_s\gamma_i))$. By δ we know $r_{\mathcal{B}}(\rho_s\gamma_i) = q_i$ and it is obvious that $\mathrm{Last}(\rho_s\gamma_i) = \gamma_i$. Using Lemma 3.12 we can conclude that $\rho_{top_k(s_i)} = \rho_{top_k(s)}\gamma'_i$ which establishes the desired equality.

Converse Implication. Assume that $\mathcal{C}$ accepts $\mathbf{t}_k$ with a run $r_{\mathcal{C}}$. We start by making some basic remark on the structure of $\mathrm{Dom}(r_{\mathcal{C}})$. For this assume that $\mathrm{Dom}(r_{\mathcal{C}})$ is a W-tree. We write for $v \in \mathrm{Dom}(r_{\mathcal{C}})$, $r_{\mathcal{C}}(v) = (\pi_v, (q_v, \gamma_v))$. To every node $v \in \mathrm{Dom}(r_{\mathcal{C}})$, we associate the sequence ρ_v in $(\Gamma^O_{k+1})^*$ defined by $\gamma_{v_1} \dots \gamma_{v_{|v|}}$ where for all $i \in [1,|v|]$, v_i is the prefix of v of length i with $r_{\mathcal{C}}(v_i) = (\pi_{v_i}, (q_{v_i}, \gamma_{v_i}))$.

Let us establish some basic facts on ρ_v.

Lemma 3.14. *For all $v \in \mathrm{Dom}(r_{\mathcal{C}})$, ρ_v is the reduce sequence of a level-(k+1) stack s_v. Moreover, the level-k stack $top_k(s_v)$ has π_v as reduced sequence.*

Proof. As $\mathcal{B}$ is a 1-PTA on $\mathbf{t}_{k+1}$ where each path in $\mathbf{t}_{k+1}$ corresponds to a reduced instruction sequence, we know by construction of $\mathcal{C}$ that the sequence ρ_v is also reduced. Furthermore, it follows that $\rho_v([\,]_{k+1})$ is defined and equal to some level-(k+1) stack s_v. A straightforward induction on the length of ρ_v using Lemma 3.12 shows that the reduced sequence of $top_k(s_v)$ is equal to π_v. □

Now, we define a run $r_{\mathcal{B}}$ of $\mathcal{B}$ on $\mathbf{t}_{k+1}$. We take $r_{\mathcal{B}}(\rho_v) = q_v$ for all $v \in \mathrm{Dom}(r_{\mathcal{B}})$. As $\mathcal{B}$ is a 1-PTA on $\mathbf{t}_{k+1}$, it follows by definition that for all $v \neq v' \in \mathrm{Dom}(r_{\mathcal{B}})$, we have $\rho_v \neq \rho_{v'}$.

To show that $r_{\mathcal{B}}$ is an accepting run for $\mathcal{B}$ on $\mathbf{t}_{k+1}$, we only need to show that $r_{\mathcal{B}}$ is well-formed. Indeed the fact that $r_{\mathcal{B}}$ satisfies the acceptance condition follows from the fact that $r_{\mathcal{C}}$ does as the states are colored in the same way, i.e., $\Omega_C(p,d) = \Omega_B(p)$ for all $d \in \Gamma^O_k \cup \{k, \diamond\}$, $p \in Q_{\mathcal{B}}$.

Let $v \in \mathrm{Dom}(r_{\mathcal{C}})$ and $r_{\mathcal{C}}(v) = (\pi_v, (q_v, \gamma_v))$. We write respectively (d, D, e) and (d', D', e') for the surroundings of s_v and $top_k(s_v)$.

We know there exists a transition

$$\delta' := ((q_v,\gamma_v),(d',D',e')) \to (\gamma'_1,(q_1,\gamma_1)) \wedge \dots \wedge (\gamma'_n,(q_n,\gamma_n)) \in \Delta_{\mathcal{C}}$$

such that for all $i \in [1,n]$, there is $d_i \in W$ with $v.d_i \in \mathrm{Dom}(r_{\mathcal{C}})$ and $r_{\mathcal{C}}(v.d_i) = (\pi_v\gamma_i', (q_i, \gamma_i))$. Remark that by definition of s_v, $\gamma_v = \mathrm{Last}(\rho_v) = d$.

By Lemma 3.11, there exists a unique transition

$$\delta^{\uparrow} := (q_v, (\gamma_v, D, e)) \to (\gamma_1, q_1) \wedge \ldots \wedge (\gamma_n, q_n) \in \Delta_{\mathcal{B}}$$

such that $\delta' = (\delta^{\uparrow})_{\downarrow_{d'}}$.

It remains to prove that for all $i \in [1,n]$ holds that $r_{\mathcal{B}}(\rho_v\gamma_i) = q_i$. Recall that for all i, there exists $d_i \in W$ such that $v.d_i \in \mathrm{Dom}(r_{\mathcal{C}})$ and $r_{\mathcal{C}}(v.d_i) = (\pi_v\gamma_i', (q_i, \gamma_i))$. By definition $\rho_{v.d_i}$ is equal to $\rho_v\gamma_i$. Hence, by definition of $r_{\mathcal{B}}(\rho_v\gamma_i) = q_{v.d_i} = q_i$.

Reconstruction of Regular Ξ-Labeling. Now it remains to show that we can reconstruct the regular Ξ-labeling of $\mathbf{t}_{k+1}$ which is accepted by $\mathcal{B}$. Remember that the Ξ-labeling has been introduced in Proposition 3.9 to represent the global positional winning strategies of the 2-PTA $\mathcal{A}$. So we need to show that given a global positional strategy for $\mathcal{C}$ on $\mathbf{t}_k$ we can construct a regular Ξ-labeling of $\mathbf{t}_{k+1}$ which is accepted by $\mathcal{B}$.

Consider a regular global positional strategy for $\mathcal{C}$ on $\mathbf{t}_k$, i.e., the strategy for Automaton to choose the correct transitions. The strategy is given by a family $(R_{\delta_C})_{\delta_C \in \Delta_C}$ of regular sets of level-k stacks[2].

We construct a regular global positional strategy for $\mathcal{B}$ accepting $\mathbf{t}_{k+1}$, from this strategy we reconstruct the Ξ-labeling.

Remark 3.15. *Consider a positional strategy $\varphi_C \colon Q_C \times \mathrm{Dom}(\mathbf{t}_k) \to \Delta_C$ for $\mathcal{C}$ on $\mathbf{t}_k$ which is winning starting in the initial state of $\mathcal{C}$ from the root of $\mathbf{t}_k$. The positional strategy $\varphi_{\mathcal{B}} \colon Q \times \mathrm{Dom}(\mathbf{t}_{k+1}) \to \Delta_{\mathcal{B}}$ for $\mathcal{B}$ on $\mathbf{t}_{k+1}$ which is defined by*

$$\varphi_{\mathcal{B}}(q, \rho_s) = (\varphi_C((q, \mathrm{Last}(\rho_s)), \rho_{top_k(s)}))^{\uparrow}$$

is winning from the root of $\mathbf{t}_{k+1}$ starting in the initial state of $\mathcal{B}$.

Proof. The statement of the remark follows directly by Lemma 3.11 and the converse implication of the proof of Proposition 3.10. □

[2]Remember that (as explained before) in a stack $s \in R_{\delta_C}$ the state of $\mathcal{C}$ is stored as topmost level-1 symbol, i.e., $top_1(s) = q$ for some $q \in Q_C$ and the stack s represents the node ρ_s in $\mathbf{t}_k$.

For each transition $\delta_{\mathcal{B}} \in \Delta_{\mathcal{B}}$, we define the set of level-$(k+1)$ stacks

$$S_{\delta_{\mathcal{B}}} = \bigcup_{\gamma \in \Gamma_k^O \cup \{k,\diamond\} \wedge\, \delta_C \in *} \{s \mid \mathrm{Last}(s) = \gamma\}^3 \cap \{s \mid top_k(s) \in R_{\delta_C}\}$$

where $\delta_C \in *$ means that we take all transitions $\delta_C \in \Delta_C$ starting in a state of the form (q, γ) for some $q \in Q_{\mathcal{B}}$ and it holds that $(\delta_C)^{\uparrow} = \delta_{\mathcal{B}}$. As both sets $\{s \mid \mathrm{Last}(s) = \gamma\}$ and $\{s \mid top_k(s) \in R_{\delta_C}\}$ are regular sets of stacks, it follows that $S_{\delta_{\mathcal{B}}}$ is a regular set for all $\delta_{\mathcal{B}} \in \Delta_{\mathcal{B}}$. It follows from Remark 3.15 that $(S_{\delta_{\mathcal{B}}})_{\delta_{\mathcal{B}} \in \Delta_{\mathcal{B}}}$ describes a regular positional strategy for $\mathcal{B}$ on $\mathbf{t}_{k+1}$.

Remember now that we have assumed that $\mathcal{B}$ guesses the Ξ-labeling in its states, such that we have $Q_{\mathcal{B}} = Q'_{\mathcal{B}} \times \Xi$. From this we can reconstruct the regular Ξ-labeling. We define it as the union of the sets $S_{\delta_{\mathcal{B}}}$ where $\delta_{\mathcal{B}}$ starts in a state with the respectively Ξ element, i.e., for each $\xi \in \Xi$:

$$U_\xi = \bigcup_{\delta_{\mathcal{B}} \in \Delta_{\mathcal{B}}} \{s \in \mathrm{Stacks}_{k+1}(\Gamma) \mid \delta_{\mathcal{B}} = ((q_{\mathcal{B}}, d, \xi), \tau) \to R, s \in S_{\delta_{\mathcal{B}}}\}.$$

It is clear by construction that ξ-labeling is regular. The fact that the Ξ-labeling of $\mathbf{t}_{k+1}$ is accepted by $\mathcal{B}$ follows as $(S_{\delta_{\mathcal{B}}})_{\delta_{\mathcal{B}} \in \Delta_{\mathcal{B}}}$ is an accepting strategy.

3.5 Combining the Results

Now we can combine the shown results and prove that we can compute a regular global positional strategy for a 2-PTA running on $\mathbf{t}_k$. Afterwards we come back to the main theorem stated in Section 3.2 and give the formal proof.

Theorem 3.16. *Given a 2-PTA $\mathcal{A}$ running on $\mathbf{t}_k$, we can compute in k-ExpTime a regular global positional strategy for $\mathcal{A}$ on $\mathbf{t}_k$.*

Proof. We show the theorem by an induction over the level of the underlying tree $\mathbf{t}_k$ and combine therefore the results of the previous sections, i.e., of the Propositions 3.9 and 3.10.

Given a 2-PTA $\mathcal{A}_{k+1}$ running on $\mathbf{t}_{k+1}$ for $k \geq 1$ we first apply the construction introduced in Proposition 3.9 and get a 1-PTA $\mathcal{B}_{k+1}$ accepting Ξ-labeling of $\mathbf{t}_{k+1}$ which represent regular global positional strategies of

[3] As we store the state at the top of the stack we have actually $\{s \mid \mathrm{Last}(pop_q(s)) = \gamma \text{ if } top_1(s) = q\}$.

$\mathcal{A}_{k+1}$. Next, we use Proposition 3.10 to construct a 2-PTA $\mathcal{A}_k$ running on $\mathbf{t}_k$ such that from a regular global positional strategy for $\mathcal{A}_k$ we can reconstruct a regular Ξ-labeling of $\mathbf{t}_{k+1}$ accepted by $\mathcal{B}_{k+1}$. Combining the two steps yield an exponential blow-up in the number of states of $\mathcal{A}_{k+1}$ where the number of colors remains linear.

The induction base considers the case of a 2-PTA $\mathcal{A}_1$ running over the tree $\mathbf{t}_1$, which is for example when we consider a stack alphabet Γ with $|\Gamma| = 2$ the binary tree. First, we apply again the construction of Proposition 3.9 and generate a 1-PTA $\mathcal{B}_1$ over $\mathbf{t}_1$ such that $\mathcal{B}_1$ accepts the trees representing global positional winning strategies of $\mathcal{A}_1$. This step provides an exponential blow-up in the number of states but the number of colors remains linear. To decide if there exists such a tree accepted by $\mathcal{B}_1$ and construct a regular tree if it is the case can be done e.g. accordingly to [Zie98]. The computation of a winning strategy for a parity game over a finite graph is exponential in the number of colors but stays polynomial in the number of states [Jur00].

Hence, we obtain altogether a k-ExpTime procedure for computing a regular global positional strategy for a 2-PTA on $\mathbf{t}_k$. □

Now, we prove formally the main theorem of this chapter.

Proof of Theorem 3.2. We have to show that given a level-k higher-order pushdown parity game $\mathcal{G}$ we can construct in k-ExpTime reduced level-k automata describing the winning regions and global positional winning strategies for both players.

By Proposition 3.3 we can construct a 2-PTA $\mathcal{A}$ running on $\mathbf{t}_k$ such that Player 0 wins from a vertex (q, s) in the game graph of the k-HOPDPG $\mathcal{G}$ if and only if $\mathcal{A}$ accepts $\mathbf{t}_k$ starting from ρ_s in state q. Furthermore, the size of $\mathcal{A}$ is polynomial in the size of the level-k higher-order pushdown automaton defining the game graph of $\mathcal{G}$. As this holds for all possible vertices (q, s) in the game graph of $\mathcal{G}$ we can compute from a regular global positional strategy for $\mathcal{A}$ on $\mathbf{t}_k$ for each player regular global positional winning strategies and a regular representation of the winning region. This has been shown in Proposition 3.4. The regular global positional strategy for $\mathcal{A}$ on $\mathbf{t}_k$ can be computed in k-ExpTime using Theorem 3.16. □

3.6 Further Results

Until yet we have discussed higher-order pushdown games with parity winning condition. In this section we consider higher-order pushdown games with restricted winning conditions and present their reduction to parity games. For this we first treat the case of Büchi-games over higher-oder pushdown graphs (which are parity games, where only two colors are present) . After this we introduce request-response games over higher-order pushdown graphs, where the reduction to parity games is more involved.

3.6.1 Higher-Order Pushdown Büchi Games

First, we give a general definition of *Büchi games* to formalize the winning condition for Player 0.

Definition 3.17. The winning condition for *Büchi games* over some game graph $G = (V, V_0, V_1, E)$ is defined by a set $F \subseteq V$. An infinite play $\rho = \rho_1\rho_2\rho_3 \ldots \in V^\omega$ is won by Player 0 if the set F is visited infinitely often in the play. Formally we define

$$\forall i \in \mathbb{N}\ \exists j \geq i \text{ such that } \rho_j \in F.$$

Now, we define *higher-order pushdown Büchi games*. Here, the idea is the same as that for higher-order pushdown parity games, i.e., we use a deterministic higher-order pushdown system to construct a game graph and its states to define a state partition and a winning condition.

Definition 3.18. A *level-k higher-order pushdown Büchi game* $\mathcal{G}$ (short: k-`HOPDBG`) is given by a deterministic level-k higher-order pushdown system $\mathcal{A} = (Q, \Sigma, \Gamma, \delta)$, a state partition Q_0, Q_1 of the states in Q and a Büchi winning condition over Q, i.e. $F^{\mathcal{A}} \subseteq Q$.

The game graph of $\mathcal{G}$ is defined by the configuration graph of the higher-order pushdown system $\mathcal{A}$ such that:

- $V_0 = Q_0 \times \text{Stacks}_k(\Gamma)$,
- $V_1 = Q_1 \times \text{Stacks}_k(\Gamma)$,
- $E \subseteq (Q \times \text{Stacks}_k(\Gamma)) \times \Sigma \times (Q \times \text{Stacks}_k(\Gamma))$ with $(p, s) \xrightarrow{\alpha} (q, s') \in E$ if $(p, \alpha, \gamma, q) \in \delta$ and $s' = \gamma(s)$,

- $F = F^{\mathcal{A}} \times \text{Stacks}_k(\Gamma)$.

As Büchi games are a special case of parity games we can simply use the methods developed for parity games to solve them.

Remark 3.19. *Every level-k higher-order pushdown Büchi game $\mathcal{G}_B$ defined a deterministic k-HOPDS $\mathcal{A}$, Q_0, Q_1, F is also a level-k higher-order pushdown parity game $\mathcal{G}_P$ which has only two colors. More precise $\mathcal{G}_P$ is defined by the same deterministic k-HOPDS $\mathcal{A}$, the same state partition Q_0, Q_1 and*

$$\Omega((p,s)) = \begin{cases} 0 & \text{if } (p,s) \in F, \\ 1 & \text{if } (p,s) \notin F. \end{cases}$$

By this remark we can reuse the results of the higher-order pushdown parity games of Section 3.2 to solve the higher-order pushdown Büchi games, i.e., we can compute the winning region and a global positional winning strategy for both players.

An open problem is to develop an approach to solve higher-order pushdown Büchi games directly, not using the results on parity games. One idea goes into the direction of recurring reachability. An approach for solving reachability games by a saturation method has been introduced over level-1 pushdown graphs in [BEM97] and over higher-order pushdown graphs in [HO08]. Nevertheless, it seems not possible to gain a huge effort in the complexity as it relies on the exponential blow-ups in the level reductions.

3.6.2 Higher-Order Pushdown Request-Response Games

In the following we consider *request-response games*. The idea behind these games lies more in the practical application and the modeling of processes, e.g., constructing a controller for a lift. In this case there can be different types of requests, which all have to be fulfilled finally by an according response. Formally the winning condition is formalized by pairs of sets (P_i, Q_i) and says that if some request in P_i occurs, then it has to be followed by some response in Q_i at the same moment or later on. Request-response games have been introduced in [WHT03] for the case of a finite game arena. First, we state the general wining condition and then we introduce the *higher-order pushdown request-response games*.

Definition 3.20. The winning condition for *request-response games* over some game graph $G = (V, V_0, V_1, E)$ is defined by a set $\Omega = \{(P_1, Q_1), \ldots, (P_r, Q_r)\}$ with $P_\ell, Q_\ell \subseteq V$ for $1 \leq \ell \leq r$. An infinite play $\rho = \rho_1 \rho_2 \ldots \in V^\omega$ is won by Player 0 if for $1 \leq \ell \leq r$ and for all $i \in \mathbb{N}$ holds, if $\rho_i \in P_\ell$ then there exists $j \geq i$ where $\rho_j \in Q_\ell$. Formally we define

$$\forall\ 1 \leq \ell \leq r \text{ and } \forall i \in \mathbb{N} : \rho_i \in P_\ell \Rightarrow \exists j \geq i : \rho_j \in Q_\ell.$$

Definition 3.21. A *level-k higher-order pushdown request-response game* $\mathcal{G}$ (short: k-HOPDRRG) is given by a deterministic level-k higher-order pushdown system $\mathcal{A} = (Q, \Sigma, \Gamma, \delta)$, a state partition Q_0, Q_1 of the states in Q and a request-response winning condition over Q, i.e. $\Omega^{\mathcal{A}} = \{(P_1^{\mathcal{A}}, Q_1^{\mathcal{A}}), \ldots, (P_r^{\mathcal{A}}, Q_r^{\mathcal{A}})\}$ with $P_\ell^{\mathcal{A}}, Q_\ell^{\mathcal{A}} \subseteq Q$ for $1 \leq \ell \leq r$.

The game graph of $\mathcal{G}$ is defined by the configuration graph of the higher-order pushdown system $\mathcal{A}$ such that

- $V_0 = Q_0 \times \text{Stacks}_k(\Gamma)$,
- $V_1 = Q_1 \times \text{Stacks}_k(\Gamma)$,
- $E \subseteq (Q \times \text{Stacks}_k(\Gamma)) \times \Sigma \times (Q \times \text{Stacks}_k(\Gamma)$ with $(p, s) \xrightarrow{\alpha} (q, s') \in E$ if $(p, \alpha, \gamma, q) \in \delta$ and $s' = \gamma(s)$.

The request-response winning condition $\Omega = \{(P_1, Q_1), \ldots, (P_r, Q_r)\}$ over the higher-order pushdown game is defined by using $\Omega^{\mathcal{A}}$ and only the control state, i.e., for $1 \leq \ell \leq r$ we define

- if $p \in P_\ell^{\mathcal{A}}$ then $(p, s) \in P_\ell$ for all $s \in \text{Stacks}_k(\Gamma)$,
- if $p \in Q_\ell^{\mathcal{A}}$ then $(p, s) \in Q_\ell$ for all $s \in \text{Stacks}_k(\Gamma)$.

Reduction to Higher-Order Pushdown Büchi Game

We continue by showing a game reduction where we reduce higher-order pushdown request-response games to higher-order pushdown Büchi games. The idea is similar to the one for the case of games on a finite game arena introduced in [WHT03]. We store in the states of the Büchi automaton all currently activated requests, one particular active request and a bit indicating if vertex is in F or not. If the marked request gets a response we visit once the set F and then we mark the next active request.

Theorem 3.22. *Given a higher-order pushdown request-response game $\mathcal{G}_{RR}$ we can construct a higher-order pushdown Büchi game $\mathcal{G}_B$ such that Player 0 wins a play ρ on $\mathcal{G}_{RR}$ iff Player 0 wins the induced play ρ' on $\mathcal{G}_B$*

Proof. Let the level-k higher-order pushdown request-response game be given by the deterministic k-HOPDS $\mathcal{A}^{RR} = (Q^{RR}, \Sigma, \Gamma, \delta^{RR})$, the state partition Q_0^{RR}, Q_1^{RR} and the request-response condition $\Omega^{RR} = \{(P_1, Q_1), \ldots, (P_r, Q_r)\}$ over the states of $\mathcal{A}^{RR}$.

To define the level-k higher-order pushdown Büchi game we first define a deterministic k-HOPDS $\mathcal{A}^B = (Q^B, \Sigma, \Gamma, \delta^B)$ together with a state partition Q_0^B, Q_1^B and the set F^B. From this we construct a higher-order pushdown Büchi game which fulfills the required condition. Furthermore, we need a set I^B that represents the starting states of the game simulation. A state of Q^B has the form $(p, \{n_1, \ldots, n_i\}, m, b)$. We have that p is current state of the request-response game. The numbers $\{n_1, \ldots, n_i\}$ respond to the currently open requests, i.e., Player 0 needs to visit vertices from the sets $Q_{n_1}, \ldots, Q_{n_i}$. Furthermore, we wait at the moment for the specific respond of a request in P_m. The bit b is equal to 1 if the last marked request has been fulfilled.

We define:

- $Q^B = Q^{RR} \times 2^{\{1,\ldots,r\}} \times \{0, \ldots, r\} \times \{0, 1\}$.
- $$\begin{aligned} I^B \;=\; & \{(q, M, \min(M), 0) \mid M \neq \emptyset,\, q \in Q^{RR}\} \\ & \cup \{(q, \emptyset, 0, 1) \mid M = \emptyset, q \in Q^{RR}\}, \end{aligned}$$ where $M = \{1 \leq \ell \leq r \mid q \in P_\ell \wedge q \notin Q_\ell\}$.
- $((p, M, m, f), \alpha, \gamma, (q, M', m', f'))) \in \delta^B \Leftrightarrow$
  - $(p, \alpha, \gamma, q) \in \delta^{RR}$,
  - $M' = M \cup \{1 \leq \ell \leq r \mid q \in P_\ell\} \setminus \{1 \leq \ell \leq r \mid q \in Q_\ell\}$
  - $m' = \begin{cases} 0 & \text{, if } M' = \emptyset \\ m & \text{, if } m \in M' \\ \min(m \leq \ell \leq r \mid \ell \in M') & \text{, if it exists} \\ \min(1 \leq \ell \leq r \mid \ell \in M') & \text{, otherwise} \end{cases}$
  - $f' = \begin{cases} 0 & \text{, if } m = m' \neq 0 \\ 1 & \text{, otherwise.} \end{cases}$
- $Q_0^B = Q_0^{RR} \times 2^{\{1,\ldots,r\}} \times \{0, \ldots, r\} \times \{0, 1\}$.

- $Q_1^B = Q_1^{RR} \times 2^{\{1,\ldots,r\}} \times \{0,\ldots,r\} \times \{0,1\}$.
- $F = Q^{RR} \times 2^{\{1,\ldots,r\}} \times \{0,\ldots,r\} \times \{1\}$.

Assume some arbitrary play $\rho = (p_0, s_0), (p_1, s_1), \ldots$ in $\mathcal{G}_{RR}$ with start vertex (p_0, s_0). The induced play $\rho' = (q_0, s_0), (q_1, s_1), \ldots$ in $\mathcal{G}_B$ is constructed according to the above definitions. So, we have $q_i = (p_i, M_i, m_i, f_i)$ for all $i \in \mathbb{N}$ where all M_i, m_i, f_i are defined by the local testable conditions given in the construction.

We have to show that Player 0 wins the play ρ in $\mathcal{G}_{RR}$ iff Player 0 wins the induced play ρ' in $\mathcal{G}_B$.

We assume that Player 0 does not win ρ' and show that Player 0 also does not win ρ. Player 0 does not win the play ρ' iff from one point onwards there is never seen a vertex in F, i.e.,

$$\exists i \in \mathbb{N}\ \forall j > i :\ (q_j, s_j) \notin F_B.$$

So, we have by definition of F that:

$$\exists i \in \mathbb{N} \forall j > i \colon q_j \notin \{(p_j, M_j, m_j, 1) \,|\, p_j \in Q^P, M_j \subseteq \{1, ..., r\}, m_j \in [1, r]\}$$

From this we know that from one point onwards all f_j are 0. This is only the case if the numbers m_{j-1} and m_j stay the same and are not 0, i.e.

$$\begin{aligned} \exists i \in \mathbb{N}\ \forall j > i, \quad & q_{j-1} = (p_{j-1}, M_{j-1}, m_{j-1}, 0) \wedge q_j = (p_j, M_j, m_j, 0) \\ & \wedge\ m_{j-1} = m_j \neq 0. \end{aligned}$$

So, it follows that

$$\exists i \in \mathbb{N}, q_i = (p_i, M_i, m_i, f_i) \text{ such that } \forall j > i, q_j = (p_j, M_j, m_i, 0),$$

where $m_i \in M_j$. This means that there is some request which is never fulfilled. So, we have:

$$\begin{aligned} & \exists i \in \mathbb{N} : q_i = (p_i, M_i, m_i, f_i) \wedge \\ & \exists i' < i : q_{i'} = (p_{i'}, M_{i'}, m_{i'}, f_{i'}) \text{ such that} \\ & p_{i'} \in P_{m_i} \wedge \forall j \geq i' : q_j = (p_j, M_j, m_j, f_j) \text{ holds:} \\ & p_j \notin Q_{m_i} \end{aligned}$$

From this follows that Player 0 also does not win ρ.

This argumentation can be followed in the opposite direction to show that from the assumption that Player 0 does not win ρ it follows that Player 0 does not win ρ'. □

Computation of the winning regions and winning strategies

We proceed by considering the computation of the winning regions and winning strategies for both players.

Lemma 3.23. *For a level-k higher-order pushdown request-response game $\mathcal{G}_{RR}$, we can compute the winning regions of Player 0 and Player 1 from the winning regions of the induced level-k higher-order pushdown Büchi game $\mathcal{G}_B$.*

Proof. Let the k-HOPDRRG $\mathcal{G}_{RR}$ be given by $\mathcal{A}^{RR}, Q_0^{RR}, Q_1^{RR}, \Omega^{RR}$ and let the k-HOPDBG $\mathcal{G}_B$ be constructed by $\mathcal{A}^B$, Q_0^B, Q_1^B, F^B, I^B like in Theorem 3.22.

The k-HOPDBG $\mathcal{G}_B$ can be solved by first using Remark 3.19 to get a level-k higher-order pushdown parity game, and then solving the latter as in Theorem 3.2. Let W_0^B and W_1^B be the computed winning regions for Player 0 and Player 1 in the k-HOPDPG which can be adopted for $\mathcal{G}_B$. From this we get the winning regions of $\mathcal{G}_{RR}$ as follows:

- $W_0 = \{(p,s) \in Q^{RR} \times \text{Stacks}_k(\Gamma) \mid ((p,M,m,f),s) \in W_0^B \wedge (p,M,m,f) \in I^B\}$
- $W_1 = \{(p,s) \in Q^{RR} \times \text{Stacks}_k(\Gamma) \mid ((p,M,m,f),s) \in W_1^B \wedge (p,M,m,f) \in I^B\}$

□

Lemma 3.24. *For a level-k higher-order pushdown request-response game $\mathcal{G}_{RR}$ we can compute a winning strategy for Player 0 by a level-k higher-order pushdown strategy automaton with tests iff Player 0 wins the induced level-k higher-order pushdown Büchi game $\mathcal{G}_B$.*

Proof. Let the k-HOPDRRG $\mathcal{G}_{RR}$ be given by $\mathcal{A}^{RR} = (Q^{RR}, \Sigma, \Gamma, \delta^{RR})$, Q_0^{RR}, Q_1^{RR}, Ω^{RR} and let $\mathcal{G}_B$ be the k-HOPDBG constructed by the k-HOPDS $\mathcal{A}^B = (Q^B, \Sigma, \Gamma, \delta^B)$, Q_0^B, Q_1^B, F^B, I^B like in Theorem 3.22. The k-HOPDBG $\mathcal{G}_B$ can be solved by first using Remark 3.19 to get a k-HOPDPG, and then the latter can be solved by Theorem 3.2. Let the positional winning strategy for Player 0 in $\mathcal{G}_B$ be given by $S_\alpha \in \text{OReg}_k(\Gamma)$ with $\alpha \in \Sigma$, as defined in Section 3.1. Note that S_α contains all stacks of the form $push_p(s)$ where in the Büchi game in vertex (p,s) the edge labeled by α should be taken, i.e., there has to be a transition in $\mathcal{A}^B$ of the form (p,α,γ,q) which can be applied at (p,s). As $\mathcal{A}^B$ is deterministic there is

an unique transition fulfilling this condition. For the strategy automaton we extract the states and define for each state $p \in Q^B$ and $\alpha \in \Sigma$

$$s \in S_\alpha^p \text{ iff } push_p(s) \in S_\alpha.$$

We compute a level-k higher-order pushdown strategy automaton $\mathcal{A}^S = (Q, \delta^{RR}, q_0, \sigma, \tau)$ with tests in $\mathcal{T} = \{S_\alpha^p \mid \alpha \in \Sigma, p \in Q^B\}$ for Player 0 for the higher-order pushdown request-response game starting from some vertex $(p_0, s_0) \in Q^{RR} \times \text{Stacks}_k(\Gamma)$. The construction for Player 1 is analogous.

- $Q = Q^{RR} \times 2^{\{1,\dots,r\}} \times \{0, \dots, r\} \times \{0, 1\}$
- $q_0 = \begin{cases} (p_0, M, min(M), 0) & \text{if } M \neq \emptyset \\ (\emptyset, 0, 1) & \text{otherwise} \end{cases}$
  for $M = \{1 \leq \ell \leq r \mid p_0 \in P_\ell \wedge p_0 \notin Q_\ell\}$,
- $\sigma : Q \times \Sigma \times \Gamma_k \times Q$
  $((p, M, m, f), \alpha, \gamma, (p', M', m', f') \in \sigma$ iff
  - $(p, \alpha, \gamma, p') \in \delta^{RR}$,
  - $M' = M \cup \{1 \leq \ell \leq r \mid q \in P_\ell\} \setminus \{1 \leq \ell \leq r \mid q \in Q_\ell\}$
  - $m' = \begin{cases} 0 & \text{, if } M' = \emptyset \\ m & \text{, if } m \in M' \\ \min(m \leq \ell \leq r \mid \ell \in M') & \text{, if it exists} \\ \min(1 \leq \ell \leq r \mid \ell \in M') & \text{, otherwise} \end{cases}$
  - $f' = \begin{cases} 0 & \text{, if } m = m' \neq 0 \\ 1 & \text{, otherwise} \end{cases}$
- $\tau : Q_0^B \times 2^{\mathcal{T}} \mapsto \delta^{RR}$

$$\tau((p, M, m, f), S_\alpha^{(p,M,m,f)}) = \alpha \quad \text{for all } \alpha \in \Sigma$$

□

Note that there is an exponential blow-up in the construction in the number of pairs in the winning condition.

4 Parametrized Regular Games

In this chapter we introduce *parametrized regular games*. They extend the idea of Gale-Stewart games, where two players choose bits (0 or 1) in alternation by adding a kind of third player who also provides a bit in every round which is fixed in advance, in terms of some set $P \subseteq \mathbb{N}$. This set is called a *parameter* and these games are called *parametrized regular games* since the winning condition is given by some regular ω-language. Another way to motivate those games is to consider Church's Synthesis Problem for a given MSO-formula $\varphi(X, Y)$ over the structure $(\mathbb{N}, +1)$. It asks whether there exists an operator F such that $\mathbb{N} \models \forall X \varphi(X, F(X))$ and if so, to construct this operator [Chu63]. This problem has been solved positively by Büchi and Landweber in [BL69]. This setting can be extended by a parameter P, where we have as problem instance an MSO-formula of the form $\varphi(X, Y, P)$ over the structure $(\mathbb{N}, +1, P)$.

First, we introduce parametrized regular games in Section 4.1 and give a new proof of a result of Rabinovich stating that for parameters P, where $(\mathbb{N}, +1, P)$ has a decidable MSO-theory, parametrized regular games are determined. Moreover, the winner, as well as a recursive winning strategy for him, can be computed.

In the second part of this chapter we give a more refined analysis. We want obtain sharper results on the format of the strategies, in dependence of the "complexity" of the parameter P. Thus, we are interested in strategies which can be generated by automata of various types, in our case, higher-order pushdown automata or collapsible pushdown automata. To guarantee the existence of such strategies, we demand that the parameter P can be generated by some related kind of deterministic machine. We introduce such a "parameter generator", here with a memory structure consisting of higher-order stacks. The general approach to construct corresponding strategies is based on a game product of a parameter generator and a parity automaton, that defines the winning condition of the parametrized regular game. Then, we apply the memoryless determinacy of parity games. This is the content of the abstract result in Section 4.2.

Following this, we give a concrete and effective construction for the case that the structure $(\mathbb{N}, +1, P)$ lies in the Caucal hierarchy. We show that if the structure $(\mathbb{N}, +1, P)$ lies in the Caucal hierarchy, equivalently the parameter can be constructed by some new kind of higher-order pushdown automaton (see Section 4.3.1). In particular, we show in Section 4.3 that in this case we can reduce the parametrized regular games to higher-order pushdown parity games and solve them by using the results of Chapter 3. Second, we apply our approach in Section 4.4 for the case that the parameters can be constructed by collapsible pushdown automata.

4.1 Parametrized Games

In this chapter we consider infinite two-player games which are defined slightly differently than the parity games we have considered in Chapter 3. In case of parity games we have some finite game arena (or as in the last chapter an infinite game arena defined by a higher-order pushdown system), the player owning a vertex has to choose the successor and the winning condition is defined by a coloring of the vertices. Now, we consider *regular infinite two-player games* in the framework of Gale-Stewart games [GS53], where we have the following more general situation. The underlying game graph is in this case the infinite binary tree, where we consider the root to be labeled by ε and where each node $w \in \{0,1\}^*$ has the left son $w0$ and the right son $w1$. We call the players here Player I and Player O for Input and Output. Player I starts the play at the root and afterwards both players choose a direction from $\{0,1\}$ in alternation. The game is round based; in every round first Player I chooses a bit (0 or 1) and then Player O chooses a bit. A play is an infinite $0,1$-sequence

$$X(0), Y(0), X(1), Y(1), \ldots$$

where $X(i)$ is supplied by Player I and $Y(i)$ by Player O for all $i \in \mathbb{N}$. We consider here the case that the winning condition is given by an ω-regular language over the alphabet $\{0,1\}^2$. To express the winning condition formally we may use formulas $\varphi(X, Y)$ of monadic second-order logic (MSO-logic) over $(\mathbb{N}, +1)$. As the MSO-logic allows to define precisely the ω-regular languages we speak of *regular games*. In the following we also define regular winning conditions by deterministic finite parity automata. Deterministic finite parity automata are known

to have the same expressive power as the MSO-logic over $(\mathbb{N}, +1)$ (see Proposition 2.5). Player O wins a play (X, Y) if (X, Y) satisfies the formula $\varphi(X, Y)$ over the structure $(\mathbb{N}, +1)$; otherwise Player I wins. If the winning condition is given by a deterministic finite parity automaton, Player O wins if the word $(X, Y) = \binom{X(0)}{Y(0)}\binom{X(1)}{Y(1)}\binom{X(2)}{Y(2)}\ldots$ is accepted by the automaton; otherwise Player I wins.

We extend regular games by adding a kind of third player whose behavior is in a sense predictable. More precisely we add a *parameter* P which is a subset of the natural numbers. Before we get into the details let us first consider the relation between a (possibly infinite) subset of the natural numbers P and an infinite $0, 1$ sequence χ_P. We define the *characteristic sequence* χ_P of a set $P \subseteq \mathbb{N}$ by $\chi_P(i) = 1$ if $i \in P$ and $\chi_P(i) = 0$ otherwise for all $i \in \mathbb{N}$, where $\chi_P(i)$ is the i-th letter of χ_P. For example the set $\{1, 3, 5, 6\}$ has the characteristic sequence 01010110^ω. We use here freely the correspondence between a set P of natural numbers and its characteristic bit sequence χ_P.

When we extend regular games by adding a parameter $P \subseteq \mathbb{N}$ respectively its characteristic sequence χ_P, we define the winning condition by a formula $\varphi(P, X, Y)$ over the structure $(\mathbb{N}, +1, P)$. In this case we speak of a *parametrized regular game*. A play of this game can be viewed as an ω-word over the alphabet $\{0, 1\}^3$. For every point in time i, first the bit $\chi_P(i)$ of the parameter P is provided, afterwards Player I chooses a bit $X(i)$ and then Player O chooses a bit $Y(i)$. So the start of an example play may look as follows:

$$\begin{array}{lccccccc} \text{Parameter} & 0 & 1 & 1 & 0 & 1 & \ldots & = P \\ \text{Player I} & 0 & 1 & 0 & 1 & 1 & \ldots & = X \\ \text{Player O} & 1 & 0 & 1 & 0 & 1 & \ldots & = Y. \end{array}$$

A play (P, X, Y) is won by Player O if it satisfies the formula $\varphi(P, X, Y)$ over the structure $(\mathbb{N}, +1, P)$; otherwise it is won by Player I.

Alternatively, we can use a deterministic finite parity automaton $\mathcal{C}$ over $\{0, 1\}^3$ to define the winning condition of the parametrized regular game, where Player O wins if the play (P, X, Y) considered as the word

$$\begin{pmatrix} \chi_P(1) \\ X(1) \\ Y(1) \end{pmatrix} \begin{pmatrix} \chi_P(2) \\ X(2) \\ Y(2) \end{pmatrix} \begin{pmatrix} \chi_P(3) \\ X(3) \\ Y(3) \end{pmatrix} \ldots$$

is in $\mathcal{L}(\mathcal{C})$.

We write $\mathcal{G}(P, \mathcal{C})$ for the parametrized regular game with the parameter P and the winning condition given by a deterministic finite parity automaton $\mathcal{C}$.

The goal for the parametrized regular games is to decide which player wins the game and to find a winning strategy for him. Whether this is possible depends on the kind of parameter which is chosen.

Rabinovich has investigated in this topic of solving parametrized regular games, e.g., in [Rab06, Rab07, Rab09]. He motivated his research by Church's synthesis problem, which asks when giving an MSO-formula $\varphi(X, Y)$ if there exists a C-operator[1] F such that $\mathbb{N} \models \forall X \varphi(X, F(X))$ and, if so, construct it.

We now state some results of Rabinovich which are presented in [Rab06, Rab07, Rab09] and thereafter we give a new proof of one of the results.

In [Rab07] Rabinovich extended the problem of solving Church's Problem by adding parameters $P \subseteq \mathbb{N}$ in the following ways. As input a MSO-formula $\varphi(X, Y, P)$ is assumed:

Problem 1: Check whether there is a C-operator $Y = F(X, P)$ such that $\mathbb{N} \models \forall X \varphi(X, F(X, P), P)$ and if there is such a recursive operator construct it.

Problem 2: Check whether there is a recursive C-operator $Y = F(X, P)$ such that $\mathbb{N} \models \forall X \varphi(X, F(X, P), P)$ and if there is such a recursive operator construct it.

Problem 3: Check whether there is a C-operator $Y = F(X)$ such that $\mathbb{N} \models \forall X \varphi(X, F(X), P)$ and if there is such a recursive operator construct it.

Problem 4: Check whether there is a finite state C-operator $Y = F(X, P)$ such that $\mathbb{N} \models \forall X \varphi(X, F(X, P), P)$ and if there is such a recursive operator construct it.

Problem 5: Check whether there is a finite state C-operator $Y = F(X)$ such that $\mathbb{N} \models \forall X \varphi(X, F(X), P)$ and if there is such a recursive operator construct it.

He showed that the following conditions are equivalent and imply the solvability of the problems 4 and 5:

- The parameter P is ultimately periodic.

[1] In our terms a C-operator is a winning strategy for Player O.

- For every MSO-formula $\varphi(X, Y, P)$ either there is a finite state C-operator F such that $\mathbb{N} \models \forall X \varphi(X, F(X, P), P)$ or there is a finite state SC-operator[2] G such that $\mathbb{N} \models \forall Y \neg\varphi(G(Y, P), Y, P)$. Moreover it is decidable which of these cases holds and the corresponding operator is computable from φ.

- For every MSO-formula $\varphi(X, Y, P)$ either there is a finite state C-operator F such that $\mathbb{N} \models \forall X \varphi(X, F(X), P)$ or there is a finite state SC-operator G such that $\mathbb{N} \models \forall Y \neg\varphi(G(Y), Y, P)$. Moreover it is decidable which of these cases holds and the corresponding operator is computable from φ.

This result is an extension of the Büchi-Landweber theorem to ultimately periodic parameters.

He also showed that that the following conditions are equivalent:

- The Problems 1-3 are solvable for P.

- The monadic theory of $(\mathbb{N}, +1, P)$ is decidable.

- For every MSO-formula $\varphi(X, Y, P)$ either there is a recursive C-operator F such that $\mathbb{N} \models \forall X \varphi(X, F(X), P)$ or there is a recursive SC-operator G such that $\mathbb{N} \models \forall Y \neg\varphi(G(Y), Y, P)$. Moreover it is decidable which of these cases holds and the corresponding operator is computable from φ.

In [Rab09] Rabinovich introduces a class of increasing recursive ω-sequences of integers which are "effectively sparse" and "effectively ultimately reducible". This class is called ER and contains many interesting predicates as for example the set of factorial numbers, the sets $\{k^n \mid n \in \mathbb{N}\}$ and $\{n^k \mid n \in \mathbb{N}\}$. It is shown that for $P \in ER$ there exists an algorithm that decides for every MSO-formula $\varphi(P, X, Y)$ over $(\mathbb{N}, +1, P)$ whether Player I has a finite-memory winning strategy in the induced game and if so constructs it.

In the following we focus on the main result of Rabinovich presented in [Rab06, Rab07] and give a new proof that relies more on automata theoretic concepts rather than on the logical techniques which have been used by Rabinovich.

[2] In our terms a strategy for Player I.

Theorem 4.1 ([Rab06, Rab07])**.** *Parameterized regular games are determined, and if the MSO-theory of the structure* $(\mathbb{N}, +1, P)$ *is decidable then the winner can be computed and a recursive winning strategy can be constructed from the winning condition.*

To show the Theorem 4.1 we proceed here as follows. We first state a basic fact known from automata theory. Then we present the proof sketch for Theorem 4.1 which is structured into 6 steps. In the first step from the parity automaton, which defines the winning condition, a parity game is constructed. The second step introduces an infinite game graph taking the concrete parameter P into account. This infinite game graph is structured into "slices" such that it can be coded as an infinite word over these slices. In Step 3 it is shown that some memoryless strategy suffices which can also be coded as an infinite word. It can be tested that the strategy is valid by a deterministic parity automaton which works on an alphabet consisting of the slices and the strategy. This automaton is transformed in Step 4 such that it runs no longer over the slices and the strategy but on the characteristic sequence of P and the strategy. Here it is used that the slices can be reconstructed from the characteristic sequence. This new automaton tests also if the strategy is valid. In Step 5 the automaton is changed to guess the strategy and otherwise works as the automaton of Step 4. Furthermore, it is transformed from a parity automaton to an equivalent Büchi automaton. Step 6 uses the MSO-theory of P as an oracle to construct an accepting run of the Büchi automaton and thus obtains a winning strategy for the game.

We start with a fundamental result (for details and definitions see [GTW01]):

Proposition 4.2. (Known Fact) *The MSO-theory of* $(\mathbb{N}, +1, P)$ *is decidable iff the following decision problem* Aut_P *is decidable.*
Aut_P: Given a parity (or Büchi) automaton $\mathcal{A}$, does $\mathcal{A}$ accept χ_P?

Proof. (of Theorem 4.1.)

Let the parametrized regular game with a fixed parameter P be defined with a winning condition given by $\varphi(P, X, Y)$ and let the MSO-theory of the structure $(\mathbb{N}, +1, P)$ be decidable. By Proposition 2.5 we know there exists a parity automaton $\mathcal{A}_\varphi = (Q, \{0,1\}^3, q_0, \delta, \Omega_\varphi)$ which is equivalent to φ. More precisely a word over $\{0,1\}^3$ is accepted by $\mathcal{A}_\varphi$ iff the word fulfills the formula $\varphi(P, X, Y)$. Assume that $|Q| = n$.

$\delta(q_0, (0,0,0)) = q_0$
$\delta(q_0, (0,0,1)) = q_1$
$\delta(q_0, (0,1,0)) = q_1$
$\delta(q_0, (0,1,1)) = q_2$
$\delta(q_0, (1,0,0)) = q_3$
$\delta(q_0, (1,0,1)) = q_2$
$\delta(q_0, (1,1,0)) = q_3$
$\delta(q_0, (1,1,1)) = q_3$

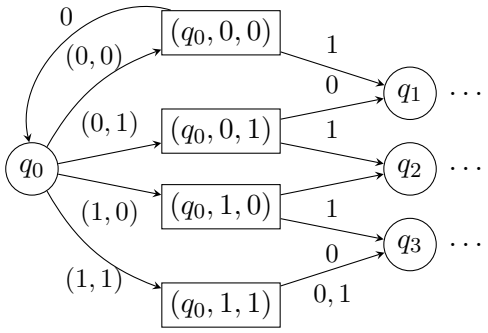

Figure 4.1: Example for Step 1 in the proof of Theorem 4.1. For the given subset of transitions of δ the corresponding game graph $\mathcal{G}_\varphi$ is depicted.

Step 1. In this step we construct a parity game starting from the parity automaton $\mathcal{A}_\varphi$ where Player 1 supplies the bit of the parameter and the bit of Player I in one step. Player 0 only supplies the bit of Player O. More precisely we transform $\mathcal{A}_\varphi$ into a game arena $\mathcal{G}_\varphi = (V, V_0, V_1, E, \Omega)$ as follows. The vertices of the game graph are given by $V = Q \cup (Q \times \{0,1\} \times \{0,1\})$ where the vertices of Player 1 are $V_1 = Q$ and for Player 0 we have $V_0 = Q \times \{0,1\} \times \{0,1\}$. For a transition $\delta(p, (b_0, b_1, b_2)) = q$ we introduce two edges in the game graph:

- The first edge is $p \xrightarrow{(b_0,b_1)} (p, b_0, b_1)$. It takes the "context bit" b_0 of P into account and the choice of Player I.
- The second edge is $(p, b_0, b_1) \xrightarrow{b_2} q$. Here Player 0 provides the bit b_2 of Player O and reaches the state q.

We obtain a finite game arena $\mathcal{G}_\varphi$. The parity acceptance condition of the automaton is turned into a winning condition over $\mathcal{G}_\varphi$; the coloring of vertices is inherited from the coloring of the states of the automaton, i.e., we define the coloring function Ω for the parity game by $\Omega(q) = \Omega_\varphi(q)$ and $\Omega((q, b_0, b_1)) = \Omega_\varphi(q)$ for all $q \in Q$ and $b_0, b_1 \in \{0,1\}$.

A small example can be found in Figure 4.1.

Step 2. Now we transform the finite game graph $\mathcal{G}_\varphi$ into an infinite game graph $\mathcal{G}_\varphi^P$. The graph $\mathcal{G}_\varphi^P$ takes into account the fixed choices of bits of the parameter P and unfolds the game graph $\mathcal{G}_\varphi$. For this we extend the

vertices of $\mathcal{G}_\varphi$ by natural numbers, i.e., instead of p and (p, b_0, b_1) we have $(p)_i$ and $(p, b_0, b_1)_i$ for all numbers $i \in \mathbb{N}$. We proceed in the following way. The initial vertex is $(q_0)_0$. If we have an edge $p \xrightarrow{(b_0,b_1)} (p, b_0, b_1)$ in $\mathcal{G}_\varphi$ we add for each $i \in \mathbb{N}$ the edge $(p)_i \xrightarrow{(b_0,b_1)} (p, b_0, b_1)_i$ to $\mathcal{G}_\varphi^P$ if $b_0 = \chi_P(i)$, i.e., if $i \in P$ then $b_0 = 1$ and otherwise $b_0 = 0$. Furthermore, we add $(p, b_0, b_1)_i \xrightarrow{b_2} (q)_{i+1}$ to $\mathcal{G}_\varphi^P$ if and only if $(p, b_0, b_1) \xrightarrow{b_2} q$ is an edge in $\mathcal{G}\varphi$.

The coloring of the vertices stays the same, i.e., we define the coloring of $\mathcal{G}_\varphi^P$ by $\Omega'((p)_i) = \Omega(p)$ and $\Omega'((p, b_0, b_1)_i) = \Omega(p, b_0, b_1))$ for all $i \in \mathbb{N}$, $p \in Q$ and $b_0, b_1 \in \{0, 1\}$. For an example see Figure 4.2.

Lemma 4.3. *Player O wins the parametrized regular game defined by φ iff Player 0 wins the parity game over $\mathcal{G}_\varphi^P$ from its initial vertex $(q_0)_0$.*

Lemma 4.3 follows directly by the construction of $\mathcal{G}_\varphi^P$.

The constructed game graph $\mathcal{G}_\varphi^P$ is acyclic as for each round (each player chooses once) in the game the number i is increased. We structure the game graph into slices $S_0, S_1, S_2, \ldots$. The k-th slice for $k = 2i$, $i \in \mathbb{N}$ contains only vertices of the form $(q)_i$ and the k-th slice for $k = 2i + 1$, $i \in \mathbb{N}$ contains only vertices of the form $(p, b_0, b_1)_i$. So, the slices with an even number can contain at most $n = |Q|$ vertices and the slices with an odd number at most $2n$ as the entry for the bit of the parameter b_0 is fixed for a given number $i \in \mathbb{N}$ and only the bit of Player I has to be remembered.

In order to have the same time scale in the characteristic sequence χ_P and the sequence of slices, we group the slices into a sequence of slice pairs $(S_0, S_1), (S_2, S_3), \ldots$. These slice pairs code the game graph $\mathcal{G}_\varphi^P$ in form of an ω-word over an appropriate alphabet Σ. We denote by α_φ^P the ω-word coding $\mathcal{G}_\varphi^P$.

Remark that the transformation of this step can be implemented by a finite automaton, uniformly in P:

Lemma 4.4. *Given a finite game arena $\mathcal{G}_\varphi$, there is a finite-state transducer (in the format of a Mealy automaton) which transforms each characteristic sequence of a set P into the corresponding sequence α_φ^P.*

Step 3. We know as stated in Proposition 2.6 that parity games are memoryless determined. This also holds for parity game over $\mathcal{G}_\varphi^P$ and so one of the two players (Player 0 and Player 1) has starting from the

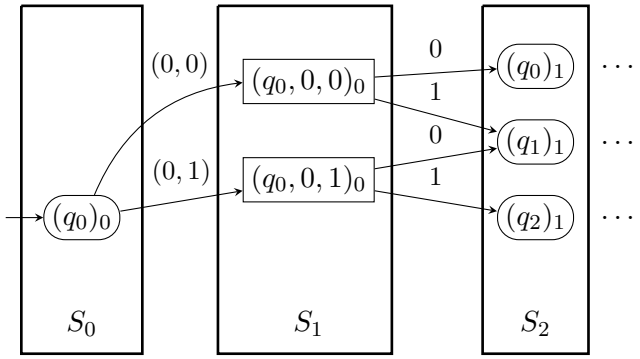

Figure 4.2: Example for Step 2 in the proof of Theorem 4.1 which extends the example given in Figure 4.1. We assume $0 \notin P$ and show the unfolding of the game graph $\mathcal{G}_\varphi$ (of Figure 4.1) over time which results in the depicted game graph $\mathcal{G}_\varphi^P$. Further more we have marked the slices S_0, S_1, S_2.

initial vertex $(q_0)_0$ a memoryless winning strategy. From this it follows that also the parametrized regular game with parameter P is determined which is one point of the Theorem 4.1 we wanted to show. To show that a recursive winning strategy can be computed we focus only on Player 0 which can be done due to the symmetry of the games. So, we assume that Player 0 wins the game starting from $(q_0)_0$.

A memoryless strategy for Player 0 is a function that maps each vertex of the form $(p, b_0, b_1)_i$ to some vertex $(q)_{i+1}$, i.e., for each i we apply a mapping from a set with at most $2n$ elements into a set with at most n elements. Let Υ be the finite set of these maps. A memoryless strategy of Player 0 is thus coded by an ω-word $\upsilon = \upsilon(0)\upsilon(1)\ldots$ over Υ where $\upsilon(i)$ is the mapping applied at moment i by Player 0.

One can construct a deterministic parity automaton $\mathcal{T}_\varphi^P$ that runs on words over the alphabet $\Sigma \times \Upsilon$ and tests for an ω-word $\alpha_\varphi^P \circ \upsilon := (\alpha_\varphi^P(0), \upsilon(0)),\ (\alpha_\varphi^P(1), \upsilon(1)), \ldots$ whether υ represents a winning strategy in the parity game coded by α_φ^P.

This is done as follows. The parity automaton $\mathcal{T}_\varphi^P$ has to check that each path within a given ω-word $\alpha_\varphi^P \circ \upsilon$ gives a play won by Player 0.

We work with an intermediate parity automaton $\mathcal{A}_I$ over the alphabet $\Sigma \times \Upsilon \times Q$ that checks whether in an ω-word

$$(\alpha_\varphi^P(0), \upsilon(0), q^{(0)}), (\alpha_\varphi^P(1), \upsilon(1), q^{(1)}), \ldots \qquad (*)$$

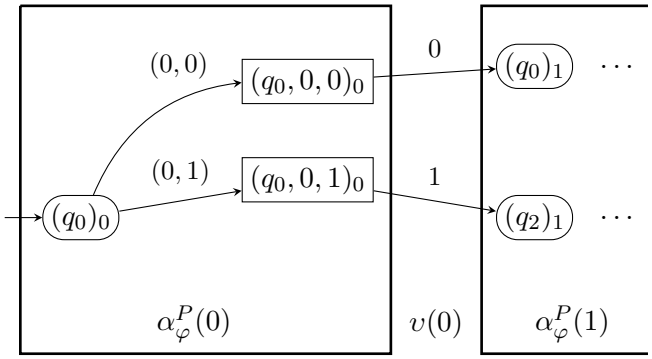

Figure 4.3: Example for Step 3 in the proof of Theorem 4.1 which extends the example given in Figure 4.2. We assume that the winning strategy for Player 0 is to choose in $(q_0, 0, 0)_0$ the 0-edge and in $(q_0, 0, 1)_0$ the 1-edge.

the following holds: if the sequence $q^{(0)}q^{(1)}q^{(2)} \ldots \in Q^\omega$ is chosen over α_φ^P according to υ then it satisfies the parity condition. The tester $\mathcal{T}_\varphi^P$ now accepts $\alpha_\varphi^P \circ \upsilon$ iff for all sequences $q^{(0)}q^{(1)}q^{(2)} \ldots \in Q^\omega$, the ω-word $(*)$ is accepted by $\mathcal{A}_I$. It is obtained from $\mathcal{A}_I$ by a complementation, a projection onto $\Sigma \times \Upsilon$ and another complementation.

Step 4. By using the transducer given by Lemma 4.4 of Step 2, we can transform the parity automaton $\mathcal{T}_\varphi^P$ into a parity automaton that runs over the input alphabet $\{0,1\} \times \Upsilon$ rather than over $\Sigma \times \Upsilon$. The automaton $\mathcal{T}_\varphi^{P'}$ that runs on the input $\chi_P \circ \upsilon$ computes, using the transducer, from χ_P the sequence α_φ^P. Furthermore, it simulates simultaneously on $\alpha_\varphi^P \circ \upsilon$ the work of $\mathcal{T}_\varphi^P$. We call $\mathcal{T}_\varphi^{P'}$ a "winning strategy *t*ester" for φ.

Corollary 4.5. *The parity automaton $\mathcal{T}_\varphi^{P'}$ accepts $\chi_P \circ \upsilon$ iff υ represents a winning strategy of Player 0 in the parametrized regular game with parameter P and winning condition φ.*

Step 5. In the fifth step we transform the deterministic winning strategy tester $\mathcal{T}_\varphi^{P'}$ of Step 4 into a nondeterministic "winning strategy *guesser*" $\mathcal{T}_\varphi^{P''}$ that runs over the input alphabet $\{0,1\}$ only. If $\mathcal{T}_\varphi^{P''}$ gets as input the characteristic sequence χ_p of P, it guesses the winning strategy $\upsilon \in \Upsilon^\omega$ and then it works on $\chi_P \circ \upsilon$ like $\mathcal{T}_\varphi^{P'}$. The automaton $\mathcal{T}_\varphi^{P''}$ is a nondeterministic parity automaton but for the next step we assume it

to be transformed into an equivalent nondeterministic Büchi automaton $\mathcal{B}^P_\varphi$.

Proposition 4.6. *The Büchi automaton $\mathcal{B}^P_\varphi$ accepts the characteristic sequence χ_P of a set P iff Player 0 has a winning strategy in the parametrized regular game with parameter P and winning condition φ.*

With this proposition we show the first effectiveness claim of the Theorem 4.1, i.e., if the MSO-theory of $(\mathbb{N}, +1, P)$ is decidable, one can decide whether Player O wins the parametrized regular game with winning condition φ. It suffices to apply Proposition 4.2, which says that if the MSO-theory of $(\mathbb{N}, +1, P)$ is decidable we can decide if $\mathcal{B}^P_\varphi$ accepts χ_P.

Step 6. In the last step it remains to show how to construct a recursive strategy for the winner of a parametrized regular game. We still assume that Player O wins the parametrized regular game with parameter P and winning condition given by φ. By Step 5 we know that it suffices to construct effectively an accepting run for $\mathcal{B}^P_\varphi$ on χ_P (which has to exists) to get a winning strategy. In terms of MSO-logic, this amounts to the proof of a Selection Lemma:

Lemma 4.7. *Assume that the MSO-theory of $(\mathbb{N}, +1, P)$ is decidable. If $(\mathbb{N}, +1) \models \exists Z \psi(P, Z)$ then a satisfying recursive set Z can be constructed (i.e., a procedure that decides for each i whether $i \in Z$).*

We give a proof, following an argument of Siefkes [Sie75], in automata theoretic terminology. It involves the well-known merging relation that was already used by McNaughton [McN66] in his proof of determinization of Büchi automata.

Definition 4.8. Let $\mathcal{B}$ be a Büchi automaton with state set $Q_\mathcal{B}$.

We call two words $\mathcal{B}$-*equivalent* (short $u \sim_\mathcal{B} v$) if for each pair s, s' of states, $\mathcal{B}$ can reach s' from s via u iff $\mathcal{B}$ can reach s' from s via v.

Denote by $P[i, j]$ the segment $\chi_P(i) \dots \chi_P(j)$ of the characteristic sequence of P.

Call two positions i, j *mergable* if there is a position $k > i, j$ such that $P[i, k] \sim_\mathcal{B} P[j, k]$. This is an equivalence relation over $\mathbb{N}$ of finite index.

We use the MSO-theory of $(\mathbb{N}, +1, P)$ repeatedly as "oracle". First, to compute a representative for each merge-equivalence class for the

parameter P and the Büchi-automaton B_φ^P we have constructed in Step 5. Afterwards, we use it in order to determine that enough representatives, say $n_1, \ldots, n_m$, have been computed. This can be done by formulating the according sentences by an MSO-formula. Just observe that i and j merge iff $(\mathbb{N}, +1, P)$ satisfies the sentence expressing $\exists z P[i, z] \sim_{\mathcal{B}} P[j, z]$. We know that all representatives occur up to position k by checking the truth of the sentence expressing "$\forall\, x > k\ \exists\, y \leq k \colon x, y$ merge". Again using the MSO-theory of $(\mathbb{N}, +1, P)$ as oracle, we pick one representative n of a merge-equivalence class from $n_1, \ldots, n_m$ with the following property:

> *There is a $\mathcal{B}$-run ρ_{acc} on χ_P that visits a certain fixed final state q_f at infinitely many times k that merge with n.*

It is clear how to express this property of n. Note, that such q_f and n can be found by a finite search process, due to the finite index of the merging relation and the assumption that an accepting run exists.

Using q_f and n, we construct effectively a run ρ of $\mathcal{B}$ on χ_P visiting q_f infinitely often, thus accepting χ_P. We proceed as follows.

We start out by looking for a first position $p_0 \in \mathbb{N}$ which fulfills the following conditions:

- p_0 merges with n and
- q_f is reachable from the initial state q_0 of $\mathcal{B}$ via $P[0, p_0 - 1]$ using a finite run ρ_0.

Such p_0 exists by assumption on n. The run ρ_0 is an initial segment of the desired accepting run ρ.

For some $k_0 > n, p_0$ we know $P[n, k_0] \sim_{\mathcal{B}} P[p_0, k_0]$, since p_0 and n merge. Hence, ρ_0 can be extended such that at position k_0 the same state as that of ρ_{acc} is reached.

We can now pick $p_1 > k_0$ such that p_1 merges again with n *and* such that q_f is reachable from q_0 via $P[0, p_1 - 1]$, by a finite run which is an extension of ρ_0. Call this finite run ρ_1.

Continuing in this way by successive finite extensions each of which is computable and ends by a final state, we construct the infinite accepting run ρ of $\mathcal{B}_\varphi^P$ on χ_P as desired.

By the choice of $n_1, \ldots, n_m$, the number $n \in \{n_1, \ldots, n_m\}$, and the state q_f using the MSO-theory of $(\mathbb{N}, +1, P)$ as oracle, we can check for the merge equivalence between n and candidate numbers c by an

effective procedure. Note that for sufficiently high k we always find that $P[c,k] \sim_{\mathcal{B}} P[n_i,k]$ for some i. So, the sequence of numbers $p_0, p_1, \ldots$ is computable if P is recursive. For arbitrary P the sequence is recursive in P. □

We have shown how to construct a recursive winning strategy for a parametrized regular game where for the parameter P the MSO-theory of $(\mathbb{N}, +1, P)$ is decidable. Nevertheless, the construction of this strategy involves an unbounded number of queries to the MSO-theory of $(\mathbb{N}, +1, P)$. Remember that these queries where needed to find the representatives $n_1, \ldots, n_m$ of the merge equivalence class, a special class of them n and some final state q_f of $\mathcal{B}^P_\varphi$. For the original specification φ let φ_P be the corresponding tuple $(n_1, \ldots, n_m, n, q_f)$ of parameters. Let the function $F\colon \varphi \mapsto \varphi_P$ capture the complexity of the synthesis problem for the set P. This function (or rather its graph considered as a set S) is Turing-reducible to the MSO-theory of $(\mathbb{N}, +1, P)$. We do not know whether this reducibility relation can be sharpened to tt-reducibility (truth-table reducibility; see [Rog87]). It is known that in general the MSO-theory of $(\mathbb{N}, +1, P)$ is tt-reducible (but not btt-reducible) to the second jump P'' of P ([Tho78]). So, for the set S coding the construction of winning strategies we have

$$S \leq_{\mathrm{T}} \text{MSO-theory of } (\mathbb{N}, +1, P) \leq_{\mathrm{tt}} P''.$$

4.2 Parametrized Regular Games based on Parameter Generators

In this section we introduce a different approach to compute the winner of a parametrized regular game and a strategy for the winner of the game. For this we define the general concept of *parameter generator*, which is a deterministic machine that defines a parameter $P \subseteq \mathbb{N}$, or more precise, generates its characteristic sequence χ_P. In this section we give a general definition of parameter generators and analyze in the following two sections two special cases of generators, which use higher-order stacks.

We assume in the following that the winning condition for the parametrized regular game is always given by a deterministic finite parity automaton over the alphabet $\{0,1\}^3$, rather than by of an MSO-formula.

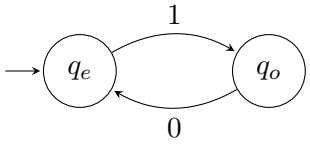

Figure 4.4: Parameter Generator $\mathcal{A}_{even}$ of Example 10.

We start with the definition of the parameter generator and give a small example.

Definition 4.9. A *parameter generator* is a deterministic machine $\mathcal{A} = (\text{Conf}, \Sigma, \text{conf}_0, \delta)$ over the alphabet $\Sigma = \{0, 1, \varepsilon\}$ with an at most countable set of configurations[3] Conf, an initial configuration $\text{conf}_0 \in \text{Conf}$ and a transition function[4] $\delta \colon \text{Conf} \to \Sigma \times \text{Conf}$.

A parameter generator $\mathcal{A}_P$ *constructs* in the natural way exactly one word $w \in \{0,1\}^\omega$ as follows. When $\mathcal{A}_P$ is started from conf_0 and the transition function δ is applied repeatedly, an infinite run

$$c_0 \xrightarrow{\alpha_0} c_1 \xrightarrow{\alpha_1} c_2 \xrightarrow{\alpha_2} \dots$$

is obtained with $c_0 = \text{conf}_0$, $\alpha_i \in \Sigma$ and $\delta(c_i) = (\alpha_i, c_{i+1})$ for all $i \in \mathbb{N}$, and the word w is produced by deleting all ε from $\alpha_0\alpha_1\alpha_2\dots$.

To get a first impression of such a parameter generator, we give a simple example based on a deterministic finite automaton.

Example 10. The parameter generator $\mathcal{A}_{even}$ constructs the characteristic sequence of the parameter $P_{even} = \{2x \mid x \in \mathbb{N}\}$, as a deterministic two-state finite automaton which switches states in each step (see Figure 4.4 where the output letters appear as labels on the transitions).

In the following we combine a parameter generator $\mathcal{A}_P$ and a deterministic finite parity automaton $\mathcal{C}$ into a parity game graph using a product construction called here *game product* $\mathcal{A}_P \otimes \mathcal{C}$. Remember that the parity automaton has $\{0,1\}^3$ as input alphabet because in each

[3] For automata with an additional storage, like for example pushdown automata, we have a state space and a set of stacks defining a set of configurations. For a simple finite automaton we have only a finite set of states.

[4] Later, we allow also a transition relation. In this case we demand, that there is exactly one infinite word over $0, 1$ generated and all other words are finite, i.e., the run aborts or produces only ε after some finite time.

round of a parametrized regular game three bits are chosen: the bit of the parameter, the bit of Player I and the bit of Player O. The parity automaton reads these bits and computes if the resulting infinite word over $\{0,1\}^3$ is accepted and if so the parametrized regular game is won by Player O.

To construct $\mathcal{A}_P \otimes \mathcal{C}$ we build the product of four components: the parameter generator $\mathcal{A}_P$, the parity automaton $\mathcal{C}$, the indicators for the turns between $\mathcal{A}_P$, Player I, Player O, and the $\mathcal{C}$-update step, and the memory of the last three chosen bits. For the indicators, we use the auxiliary alphabet $\Phi := \{\Phi_P, \Phi_I, \Phi_O, \Phi_C\}$. The symbol Φ_P indicates that the next bit of χ_P is computed by $\mathcal{A}_P$, the symbol Φ_I that Player I chooses a bit, Φ_O that Player O chooses a bit, and the symbol $\Phi_\mathcal{C}$ that the next state of $\mathcal{C}$ is computed by evaluating the three chosen bits. The coloring of the vertices with the indicator $\Phi_\mathcal{C}$ is adopted from the color of the current state of $\mathcal{C}$ in the vertices. The color of all other vertices is an even number higher than all numbers in the coloring of $\mathcal{C}$. All vertices with the indicators Φ_P, Φ_I and $\Phi_\mathcal{C}$ belong to Player 1 and all vertices with the indicator Φ_O to Player 0.

Definition 4.10. A *game product* $\mathcal{G}_P = \mathcal{A}_P \otimes \mathcal{C}$ of a parameter generator $\mathcal{A}_P = (\mathrm{Conf}^\mathcal{A}, \Sigma, \mathrm{conf}_0^\mathcal{A}, \delta^\mathcal{A})$ and a deterministic finite parity automaton $\mathcal{C} = (Q^\mathcal{C}, \{0,1\}^3, q_0^\mathcal{C}, \delta^\mathcal{C}, \Omega^\mathcal{C})$ working over $\{0,1\}^3$, is a parity game graph defined as follows.

The transition graph of $\mathcal{G}_P$ is given by a set of vertices $V \subseteq \mathrm{Conf}^\mathcal{A} \times Q^\mathcal{C} \times \{\Phi_P, \Phi_O, \Phi_I, \Phi_\mathcal{C}\} \times \{0,1\}^3$ and an edge relation E.

The edge relation E of $\mathcal{G}_P$ is defined by:

- For a transition $\delta^\mathcal{A}(c_1^\mathcal{A}) = (\varepsilon, c_2^\mathcal{A})$ in $\mathcal{A}_P$ and all vertices of the form $(c_1^\mathcal{A}, q^\mathcal{C}, \Phi_P, (b_0, b_1, b_2)) \in V$ we add the following edge to E:
$$(c_1^\mathcal{A}, q^\mathcal{C}, \Phi_P, (b_0, b_1, b_2)) \to (c_2^\mathcal{A}, q^\mathcal{C}, \Phi_P, (b_0, b_1, b_2))$$
- For a transition $\delta^\mathcal{A}(c_1^\mathcal{A}) = (0, c_2^\mathcal{A})$ in $\mathcal{A}_P$ and all vertices of the form $(c_1^\mathcal{A}, q^\mathcal{C}, \Phi_P, (b_0, b_1, b_2)) \in V$ we add the following edge to E:
$$(c_1^\mathcal{A}, q^\mathcal{C}, \Phi_P, (b_0, b_1, b_2)) \to (c_2^\mathcal{A}, q^\mathcal{C}, \Phi_I, (0, b_1, b_2))$$
- For a transition $\delta^\mathcal{A}(c_1^\mathcal{A}) = (1, c_2^\mathcal{A})$ in $\mathcal{A}_P$ and all vertices of the form $(c_1^\mathcal{A}, q^\mathcal{C}, \Phi_P, (b_0, b_1, b_2)) \in V$ we add the following edge to E:
$$(c_1^\mathcal{A}, q^\mathcal{C}, \Phi_P, (b_0, b_1, b_2)) \to (c_2^\mathcal{A}, q^\mathcal{C}, \Phi_I, (1, b_1, b_2))$$

- For each vertex of the form $(c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_I, (b_0, b_1, b_2)) \in V$ we have the following two edges in E:

$$(c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_I, (b_0, b_1, b_2)) \to (c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0, 0, b_2))$$
$$(c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_I, (b_0, b_1, b_2)) \to (c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0, 1, b_2))$$

- For each vertex of the form $(c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0, b_1, b_2)) \in V$ we have the following two edges in E:

$$(c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0, b_1, b_2)) \to (c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0, b_1, 0))$$
$$(c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0, b_1, b_2)) \to (c^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0, b_1, 1))$$

- For a transition $\delta^{\mathcal{C}}(q_1^{\mathcal{C}}, (b_1, b_2, b_3)) = q_2^{\mathcal{C}}$ in $\mathcal{C}$ and all vertices of the form $(c^{\mathcal{A}}, q_1^{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0, b_1, b_2)) \in V$ we add the following edge to E:

$$(c^{\mathcal{A}}, q_1^{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0, b_1, b_2)) \to (c^{\mathcal{A}}, q_2^{\mathcal{C}}, \Phi_P, (b_0, b_1, b_2))$$

The coloring of the game product $\mathcal{G}_P$ is defined for all vertices in V by

$$\Omega((c^{\mathcal{A}}, q^{\mathcal{C}}, x, (b_0, b_1, b_2))) = \begin{cases} \Omega^{\mathcal{C}}(q^{\mathcal{C}}) & \text{if } x = \Phi_{\mathcal{C}}, \\ 2 \cdot n & \text{if } x \in \{\Phi_P, \Phi_I, \Phi_O\} \end{cases}$$

where n is the maximal color in $\Omega^{\mathcal{C}}$.

The partitioning of the vertices of $\mathcal{G}_P$ is defined as follows

- $V_0 \subseteq \mathrm{Conf}^{\mathcal{A}} \times Q^{\mathcal{C}} \times \{\Phi_O\} \times \{0,1\}^3$ and
- $V_1 \subseteq \mathrm{Conf}^{\mathcal{A}} \times Q^{\mathcal{C}} \times \{\Phi_P, \Phi_{\mathcal{C}}, \Phi_I\} \times \{0,1\}^3$.

The following result is an easy observation which develops the context of the succeeding sections. We apply the theorem on memoryless determinacy of parity games which works even over countable game graphs (see Proposition 2.6) to products of the form $\mathcal{A}_P \otimes \mathcal{C}$.

Proposition 4.11. *Given a parameter generator $\mathcal{A}_P$ and a deterministic finite parity automaton $\mathcal{C}$, the parity game over the game graph $\mathcal{A}_P \otimes \mathcal{C}$ is memoryless determined.*

This proposition yields information about the winning strategies needed in the parametrized regular game $\mathcal{G}(P, \mathcal{C})$ with parameter P, generated by parameter generator $\mathcal{A}_P$, and the regular winning condition defined by $\mathcal{C}$. Obviously, the winning player can use the product $\mathcal{A}_P \otimes \mathcal{C}$ and

the memoryless strategy coded therein as a memoryless structure to win $\mathcal{G}(P, \mathcal{C})$: During a play in the parametrized regular game, an according path in $\mathcal{A}_P \otimes \mathcal{C}$ is generated and the respective next chosen bit for the winning player is given by the next available transition coded by the strategy in $\mathcal{A}_P \otimes \mathcal{C}$. This shows that up to a finite factor $\mathcal{C}$, the format of the winning strategy is tightly connected to the parameter generator $\mathcal{A}_P$ and in this sense has the "same structural complexity".

The subsequent considerations sharpen this result by more specific algorithmic claims in the case where $\mathcal{A}_P$ is a parameter generator which uses higher-order stacks, in two different versions. For the case of a higher-order pushdown parameter generator it follows already from Proposition 4.11 that the memoryless winning strategies can be interpreted as higher-order pushdown strategies. We show now additionally that one can decide who wins, and present a concrete higher-order pushdown winning strategy for the respective winner. Also complexity statements on the construction are proved.

4.3 Parametrized Regular Games with Higher-Order Pushdown Parameters

In this section we consider a special class of parameters where the structure $(\mathbb{N}, +1, P)$ belongs to the Caucal hierarchy. Remember that the Caucal hierarchy has been introduced by Caucal in [Cau02]. It is a large class of infinite graphs which can be generated starting from finite trees and graphs by applying MSO-interpretations and unfoldings in alternation. The resulting hierarchy is a rich collection of models, each of them having a decidable MSO-theory. In this hierarchy for example the following parameters can be constructed: the set of factorial numbers and for all $c \in \mathbb{N}$ the sets $\{n^c \mid n \in \mathbb{N}\}$ and $\{c^n \mid n \in \mathbb{N}\}$. In [CW03] Carayol and Wöhrle showed that the graphs of the Caucal hierarchy coincide with the transition graphs of higher-order pushdown automata.

For the case that the structure $(\mathbb{N}, +1, P)$ belongs to the Caucal hierarchy we construct a parameter generator that bases on higher-order pushdown automata. As shown in Chapter 3 we can solve parity games over graphs defined by higher-order pushdown systems and compute positional winning strategies. So we can apply the definition of a game product to solve parametrized regular games with those parameters and

the results of Chapter 3.

First, we define in Section 4.3.1 the higher-order pushdown parameter generators and give in Section 4.3.2 an example. Afterwards, we show in Section 4.3.3 how to solve parametrized regular games and give a winning strategy for the player winning the game.

4.3.1 Higher-Order Pushdown Parameter Generators

We have seen in Section 4.2 how to define parameter generators in a very general way. We get here more concrete and consider the case that the automata have as additional benefit a storage in form of a higher-order pushdown stack and use the symmetric operations $Ops_k(\Gamma)$ or equivalently the instructions Γ_k which we have defined in Section 2.2.2 and 2.2.3. The *higher-order pushdown parameter generators*, which we define in the following, construct an infinite 0-1-sequence as output. More precisely the output alphabet is $\Sigma = \{0, 1, \varepsilon\}$, where the ε serves as output token for intermediate transitions which neither produce 0 nor 1. Furthermore, we introduce the *identity function id* for higher-order stacks that just returns the stack to which it is applied to. We add *id* to Ops_k and to Γ_k. It is defined by $id(s) = s$ for all stacks $s \in \text{Stacks}_k(\Gamma)$.

Definition 4.12. A *level-k higher-order pushdown parameter generator* (short: k-`HOPG`) is a level-k higher-order pushdown automaton $\mathcal{A}$ that constructs exactly one infinite word over $\{0, 1\}$. It is given by the tuple $(Q, \Sigma, \Gamma, q_0, \Delta)$ where Q is a finite set of states, $\Sigma = \{0, 1, \varepsilon\}$ is the output alphabet, Γ is a stack alphabet, $q_0 \in Q$ is an initial state, and $\Delta \subseteq Q \times \Sigma \times \Gamma_k \times Q$ is an transition relation.

In the terminus of Section 4.2, we refer to the countable set of configurations Conf as the set $Q \times \text{Stacks}_k(\Gamma)$ and to the initial configuration $\text{conf}_0 = (q_0, [\ ]_k)$. We write $(p, s) \xrightarrow{\alpha} (q, s')$ if there exists a transition $(p, \alpha, \gamma, q) \in \Delta$ such that $s' = \gamma(s)$ is defined.

As before, an ω-word $\alpha \in \{0, 1\}^\omega$ is *constructed* by the automaton $\mathcal{A}$ if there exists an infinite run

$$(q_0, [\ ]_k) \xrightarrow{\alpha_0} (q_1, s_1) \xrightarrow{\alpha_1} (q_2, s_2) \xrightarrow{\alpha_2} (q_3, s_3) \xrightarrow{\alpha_3} \dots$$

such that α is obtained from $\alpha_0\alpha_1\alpha_2\alpha_3\dots$ by deleting all occurrences of ε. We assume that there is exactly one run which generates an infinite word over $\{0, 1\}$. There may be other runs but they either abort or produce eventually only ε.

Now, we introduce the term "level-k-constructible", which means that a parameter P can be constructed by a k-HOPG. Afterwards, we show that a structure $(\mathbb{N}, +1, P)$ is in level k of the Caucal hierarchy if and only if P is level-k-constructible.

Definition 4.13. A set $P \subseteq \mathbb{N}$ is *level-k-constructible* if there is a level-k higher-order pushdown parameter generator $\mathcal{A}$ that constructs the characteristic sequence χ_P of P.

Theorem 4.14. *A structure $(\mathbb{N}, +1, P)$ is in level k of the Caucal hierarchy iff P is level-k-constructible.*

Proof. The theorem follows almost directly from [CW03]. There it is shown that, for every level k, a graph G is the ε-closure of a configuration graph of a level-k higher-order pushdown system if and only if G is in the k-th level of the Caucal hierarchy.

So if the structure $(\mathbb{N}, +1, P)$ is in the k-th level of the Caucal hierarchy it means that there is a graph G which is in principle a path, where each vertex refers to a number in $\mathbb{N}$. Each vertex, except the first, has two neighbors connected by an edge (a predecessor and a successor). The first vertex has only one successor. The vertices which correspond to numbers in P are marked; we label their outgoing edges as follows. The edges starting in a vertex which is marked (in P) are labeled by 1 and the others by 0.

By [CW03] follows that G is the ε-closure of the configuration graph of a level-k higher-order pushdown system. As G is only a path labeled by $\{0, 1\}$ in the configuration graph of the higher-order pushdown system, there is also only one infinite path labeled by $\{0, 1\}$. Due to the ε transitions there may be ε between the 0 and 1. There might also be paths that branch off the 0-1-path but they can only be labeled by ε. Such the conditions of a higher-order pushdown parameter generator are fulfilled.

For the converse direction let P be level-k-constructible by a level-k higher-order pushdown parameter generator $\mathcal{A}$. Due to the definition of the HOPG the ε-closure of its configuration graph is an infinite $\{0, 1\}$-labeled path, which we denote by G. It follows from [CW03] that G is in the k-th level of the Caucal hierarchy. If we again consider the vertices of the path G as the natural numbers and add the "numbers" of the vertices with an outgoing 1 edge to the set P we get that $(\mathbb{N}, +1, P)$ is in the k-th level of the Caucal hierarchy. □

4.3.2 Example

In the following we present an example for a parametrized regular game where the parameter is constructed by a higher-order pushdown parameter generator. The parameter we consider here is the set of numbers which are a power of 2. For this we first give a 2-HOPG to generate the set of numbers which are powers of 2 respectively construct the according characteristic sequence. Afterwards we formulate a winning condition by a parity automaton.

As an example for the application of higher-order pushdown parameter generators, let us describe the idea for a 2-HOPG defining the set $P_2 = \{2^i \mid i \in \mathbb{N}\}$ of the powers of 2. Note that after the output of a 1 at position 2^i, the next output of 1 occurs 2^i steps later at position 2^{i+1}.

The idea for the generator is the following. The generator performs a binary counting where it uses that by the $\overline{copy_1}$ it can differ if the two topmost stacks are equal or not, which counts as 0 or 1. For the main procedure we need in principle only a counter, which means a pushdown automaton with only one stack symbol (here x). However, to mark the deepest level-1 stack we need a second stack symbol (here y). To explain the algorithm we use the convention that the level-1 stack $[x^i]_1$ is represented by the number i.

At the beginning the first three bits (011) of the characteristic sequence of P_2 are generated; then the real computation starts. The deepest level-1 stack is marked by adding at its top a y onto the current number of x we are processing. To explain the idea we now forget about this deepest stack and consider it to consist of x^i. We have to produce $2^i - 1$ times 0 as output in this case. We differ in the computation between two main phases; the "up" phase and the "down" phase. The states in the definition of the 2-HOPG are named accordingly.

In the algorithm first the stack $[i, i-1, i-1, i-2, i-2, \ldots, 1, 1, 0, 0]_2$ is built in the "up" phase. In the "down" phase the stack is reformed. Here we have three cases to distinguish. If the two topmost level-1 stacks are equal a $\overline{copy_1}$ is performed and the "up"-phase restarted, if the two topmost stacks are different they are made equal and also a $\overline{copy_1}$ is performed and we stay in the "down"-phase. If the topmost but one stack is the deepest level-1 stack with the y at the top, we are done with the output of 0^{2^i-1}. The next 1 is produced and a further x is added to start again with x^{i+1}. A 0 is produced whenever in the "up"-phase the

$(q_0^{\text{ini}}, 0, push_y, q_1^{\text{ini}})$ $(q_1^{\text{ini}}, 1, id, q_2^{\text{ini}})$ $(q_2^{\text{ini}}, 1, id, q_1^{\text{prep}})$
$(q_1^{\text{prep}}, \varepsilon, copy_1, q_2^{\text{prep}})$ $(q_2^{\text{prep}}, \varepsilon, pop_y, q_1^{\text{up}})$
$(q_1^{\text{up}}, 0, copy_1, q_2^{\text{up}})$ $(q_2^{\text{up}}, \varepsilon, pop_x, q_3^{\text{up}})$ $(q_3^{\text{up}}, \varepsilon, copy_1, q_1^{\text{up}})$
$(q_2^{\text{up}}, \varepsilon, T_{[\,]_1}, q^{\text{stop}})$ $(q^{\text{stop}}, \varepsilon, \overline{copy_1}, q_1^{\text{down}})$
$(q_1^{\text{down}}, \varepsilon, \overline{copy_1}, q_1^{\text{up}})$
$(q_1^{\text{down}}, \varepsilon, push_x, q_2^{\text{down}})$ $(q_2^{\text{down}}, \varepsilon, \overline{copy_1}, q_1^{\text{down}})$
$(q_1^{\text{down}}, \varepsilon, push_y, q_1^{\text{next}})$ $(q_1^{\text{next}}, \varepsilon, \overline{copy_1}, q_2^{\text{next}})$
$(q_2^{\text{next}}, \varepsilon, pop_y, q_3^{\text{next}})$ $(q_3^{\text{next}}, 1, push_x, q_4^{\text{next}})$ $(q_4^{\text{next}}, \varepsilon, push_y, q_1^{\text{prep}})$.

Table 4.1: Definition of the transition relation Δ of the 2-HOPG $\mathcal{A}_{P_2}$.

first time the stack is copied.

The 2-HOPG to generate the parameter $P_2 = \{2^i \mid i \in \mathbb{N}\}$ is defined by $\mathcal{A}_{P_2} = (Q, \{0, 1, \varepsilon\}, \{x, y\}, q_0^{\text{ini}}, \Delta)$, with $Q = \{q_i^{\text{ini}}, q_1^{\text{prep}}, q_2^{\text{prep}}, q_j^{\text{up}}, q^{\text{stop}}, q_1^{\text{down}}, q_2^{\text{down}}, q_k^{\text{next}} \mid i \in [0, 2], j \in [1, 3], k \in [1, 4]\}$. The transition relation Δ is given in Table 4.1.

The higher-order pushdown parameter generator $\mathcal{A}_{P_2}$ is also depicted in Figure 4.5. To illustrate the mode of operation of the parameter generator $\mathcal{A}_{P_2}$ a bit more we give the start of its production of χ_{P_2} in Table 4.2.

Let us continue the example and discuss a parametrized regular game where we have the parameter $P_2 = \{2^i \mid i \in \mathbb{N}\}$. We consider as winning condition the property that Player O copies the bits played by Player I except for the moments i where $i+1 \in P_2$; in these moments the converse bit is required. The start of an example play that is won by Player O (if it continues in the correct way) could be:

P_2	0	1	1	0	1	0	0	0	1	0	...
Player I	0	1	0	0	1	0	0	1	1	1	...
Player O	1	0	0	1	1	0	0	0	1	1	...

The deterministic parity word automaton over $\{0, 1\}^3$ that defines this winning condition is given in Figure 4.6. For this example a safety automaton (A-automaton) would suffice; but as we deal in our proof with parity automata defining the winning condition, we use here also a parity automaton.

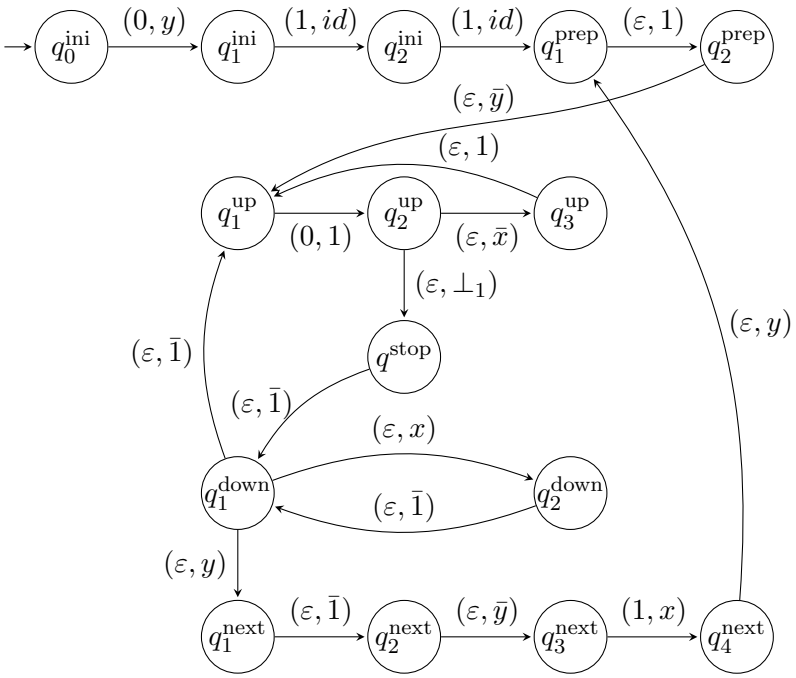

Figure 4.5: 2-HOPG to generate P_2. The transitions are labeled by tuples (i, j) where $i \in \{0, 1, \varepsilon\}$ is the output of the automaton and $j \in \Gamma_2$ is the instruction applied in this transition.

$$(q_0^{\text{ini}}, [\,]_2) \xrightarrow{0} (q_1^{\text{ini}}, [[y]]_2) \xrightarrow{1} (q_2^{\text{ini}}, [[y]]_2) \xrightarrow{1} (q_1^{\text{prep}}, [[y]]_2) \xrightarrow{\varepsilon}$$
$$(q_2^{\text{prep}}, [[y][y]]_2) \xrightarrow{\varepsilon} (q_1^{\text{up}}, [[y][\,]]_2) \xrightarrow{0} (q_2^{\text{up}}, [[y][\,][\,]]_2) \xrightarrow{\varepsilon}$$
$$(q^{\text{stop}}, [[y][\,][\,]]_2) \xrightarrow{\varepsilon} (q_1^{\text{down}}, [[y][\,]]_2) \xrightarrow{\varepsilon} (q_1^{\text{next}}, [[y][y]]_2) \xrightarrow{\varepsilon}$$
$$(q_2^{\text{next}}, [[y]]_2) \xrightarrow{\varepsilon} (q_3^{\text{next}}, [\,]_2) \xrightarrow{1} (q_4^{\text{next}}, [[x]]_2) \xrightarrow{\varepsilon} (q_1^{\text{prep}}, [[xy]]_2)$$
$$\xrightarrow{\varepsilon} (q_2^{\text{prep}}, [[xy][xy]]_2) \xrightarrow{\varepsilon} (q_1^{\text{up}}, [[xy][x]]_2) \xrightarrow{0} (q_2^{\text{up}}, [[xy][x][x]]_2) \xrightarrow{\varepsilon}$$
$$(q_3^{\text{up}}, [[xy][x][\,]]_2) \xrightarrow{\varepsilon} (q_1^{\text{up}}, [[xy][x][\,][\,]]_2) \xrightarrow{0} (q_2^{\text{up}}, [xy][x][\,][\,][\,]]_2) \xrightarrow{\varepsilon}$$
$$(q^{\text{stop}}, [[xy][x][\,][\,][\,]]_2) \xrightarrow{\varepsilon} (q_1^{\text{down}}, [[xy][x][\,][\,]]_2) \xrightarrow{\varepsilon} (q_1^{\text{up}}, [[xy][x][\,]]_2) \xrightarrow{0}$$
$$(q_2^{\text{up}}, [[xy][x][\,][\,]]_2) \xrightarrow{\varepsilon} (q^{\text{stop}}, [[xy][x][\,][\,]]_2) \xrightarrow{\varepsilon} (q_1^{\text{down}}, [[xy][x][\,]]_2) \xrightarrow{\varepsilon}$$
$$(q_2^{\text{down}}, [[xy][x][x]]_2) \xrightarrow{\varepsilon} (q_1^{\text{down}}, [[xy][x]]_2) \xrightarrow{\varepsilon} (q_1^{\text{next}}, [[xy][xy]]_2) \xrightarrow{\varepsilon}$$
$$(q_2^{\text{next}}, [[xy]]_2) \xrightarrow{\varepsilon} (q_3^{\text{next}}, [[x]]_2) \xrightarrow{1} (q_4^{\text{next}}, [[xx]]_2) \xrightarrow{\varepsilon} (q_1^{\text{prep}}, [[xxy]]_2) \cdots$$

Table 4.2: Initial part of the run of HOPG $\mathcal{A}_{P_2}$ to generate χ_{P_2}.

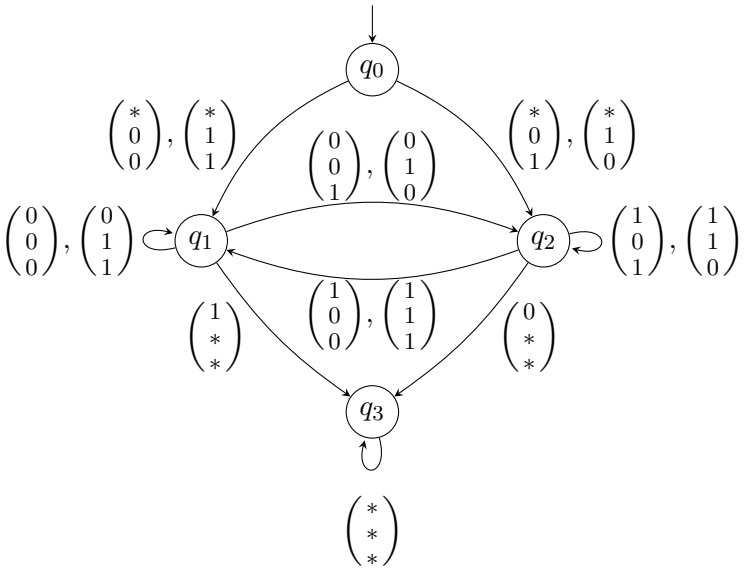

Figure 4.6: Deterministic parity automaton with coloring $q_0 = q_1 = q_2 = 0$, $q_3 = 1$ for the definition of the winning condition of the example in Section 4.3.2. The $*$ in the labeling of the transitions means either 0 or 1 is possible.

It is easy to see that a finite-state winning strategy does not suffice for Player O to win this game as no finite memory suffices to determine the moments i with $i+1 \in P_2$. On the other hand, if Player O can use an additional storage in form of a higher-order pushdown parameter generator that defines P_2, he can detect the critical moments.

Another way to detect these critical moments in this example, without using a higher-order pushdown parameter generator, could be for example to use a lookahead on the parameter P_2. A lookahead on the parameter means that Player O knows at the moment i in the play not only the parameter up to $\chi_P(i)$ but he also knows the value of $\chi_P(i+1)$.

4.3.3 Solution

In the following we show that a parametrized regular game, where the parameter P can be constructed by a level-k higher-order pushdown parameter generator, can be solved in k-ExpTime. This means we can compute which player wins, and for the winner we return a winning strategy which is describable by a level-k pushdown automaton.

We present a detailed proof here where we use the idea of building a game product in the special case of higher-order pushdown parity games. We define a higher-order pushdown strategy automaton for the winner and analyze the complexity of the algorithm.

Theorem 4.15. *Let the parameter $P \subseteq \mathbb{N}$ be defined by a level-k higher-order pushdown parameter generator $\mathcal{A}_P$. The parametrized regular game $\mathcal{G}(P, \mathcal{C})$ where the winning condition is given by a deterministic parity word automaton $\mathcal{C}$ over $\{0,1\}^3$ is (determined and) solvable: it can be decided who wins the game and for the winner one can construct a level-k HOPDA that defines a winning strategy.*

To prove this theorem we use the result of Theorem 3.2 over higher-order pushdown parity games stated in Chapter 3. For this we simulate the parametrized regular game and especially the computation of the parameter by using the higher-order stacks in the higher-order pushdown parity game. In the parametrized regular game Player O is the one who wants to fulfill the winning condition. He is simulated in the HOPDPG by Player 0; Player I is simulated by Player 1.

We show in the proof first how to compute the winner of the game, then give the higher-order pushdown automaton which defines a winning strategy, and afterwards show the complexity of the computation.

Proof. The idea is the following. We construct a higher-order pushdown parity game $\mathcal{G}_P$. In every round we first compute with the help of the higher-order pushdown parameter generator $\mathcal{A}_P$, i.e., the level-k stack, the next bit of the characteristic sequence of P. Then we first let Player I (in $\mathcal{G}_P$ Player 1) choose a bit and afterwards Player O (in $\mathcal{G}_P$ Player 0) chooses a bit. These three bits are stored in the state of the current vertex of $\mathcal{G}_P$. Finally these three bits are evaluated by the parity automaton $\mathcal{C}$ defining the winning condition and so the color of the vertices is computed. For this we give $\mathcal{C}$ these three bits as input. The parity game $\mathcal{G}_P$ is won by Player 0 if and only if the given parametrized regular game is won by Player O. Using this allows us to invoke Theorem 3.2 to solve the higher-order pushdown parity game $\mathcal{G}_P$ and to compute a winning strategy.

Construction. Let $\mathcal{A}_P = (Q^{\mathcal{A}}, \Sigma^{\mathcal{A}}, \Gamma^{\mathcal{A}}, q_0^{\mathcal{A}}, \Delta^{\mathcal{A}})$ be a k-HOPG constructing P, and let $\mathcal{C} = (Q^{\mathcal{C}}, \Sigma^{\mathcal{C}}, q_0^{\mathcal{C}}, \delta^{\mathcal{C}}, \Omega^{\mathcal{C}})$ be a deterministic parity word automaton over the alphabet $\Sigma^{\mathcal{C}} = \{0,1\}^3$ defining the winning condition of the parametrized regular game.

In order to simulate the parametrized regular game between Player I and Player O with the parameter P we construct a level-k higher-order pushdown parity game (k-HOPDPG) $\mathcal{G}_P$ which is defined by a level-k higher-order pushdown system (k-HOPDS) $\mathcal{A}_{\mathcal{G}} = (Q, \Sigma^{\mathcal{A}}, \Gamma^{\mathcal{A}}, q_0, \Delta)$, a state partition Q_0, Q_1 and a coloring Ω.

The k-HOPDS $\mathcal{A}_{\mathcal{G}}$ that defines the game graph works repeatedly in four phases, indicated by the different symbols of the alphabet $\Phi := \{\Phi_P, \Phi_I, \Phi_O, \Phi_{\mathcal{C}}\}$. The symbol Φ_P indicates that the next bit of χ_P is computed by $\mathcal{A}_P$, the symbol Φ_I that Player I chooses a bit, Φ_O means Player O chooses a bit, and the symbol $\Phi_{\mathcal{C}}$ that the next state of $\mathcal{C}$ is computed by evaluating the chosen bits.

The k-HOPDS $\mathcal{A}_{\mathcal{G}}$ has the state set $Q = Q^{\mathcal{A}} \times Q^{\mathcal{C}} \times \Phi \times \{0,1\}^3$ where for a state $(q^{\mathcal{A}}, q^{\mathcal{C}}, x, (b_0, b_1, b_2)) \in Q$ we have that $q^{\mathcal{A}}$ is the current state in $\mathcal{A}_P$ and $q^{\mathcal{C}}$ is the current state in $\mathcal{C}$. Furthermore, by the third component we know in which phase of a round we are. By (b_0, b_1, b_2) we memorize the current bit of χ_P and the last bit chosen by Player I and Player O. The initial state is $q_0 = (q_0^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_P, (0,0,0))$. The transition relation Δ is defined in the following. Note that the bits b_0', b_1', b_2' are the current choices for χ_P, Player I, and Player O. The transitions are defined for all $q^{\mathcal{A}} \in Q^{\mathcal{A}}$, $q^{\mathcal{C}} \in Q^{\mathcal{C}}$, and $b_1, b_2, b_3 \in \{0,1\}$.

- For $(q_1^{\mathcal{A}}, \varepsilon, \gamma, q_2^{\mathcal{A}}) \in \Delta^{\mathcal{A}}$ we have $\big((q_1^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_P, (b_0, b_1, b_2)), \varepsilon, \gamma, (q_2^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_P, (b_0, b_1, b_2))\big) \in \Delta$.
- For $b_0' \in \{0,1\}$ and $(q_1^{\mathcal{A}}, b_0', \gamma, q_2^{\mathcal{A}}) \in \Delta^{\mathcal{A}}$ we have $\big((q_1^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_P, (b_0, b_1, b_2)), b_0', \gamma, (q_2^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_I, (b_0', b_1, b_2))\big) \in \Delta$.
- For $b_1' \in \{0,1\}$ we have $\big((q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_I, (b_0, b_1, b_2)), b_1', id, (q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0, b_1', b_2))\big) \in \Delta$.
- For $b_2' \in \{0,1\}$ we have $\big((q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0, b_1, b_2)), b_2', id, (q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0, b_1, b_2'))\big) \in \Delta$.
- For $\delta^{\mathcal{C}}(q_1^{\mathcal{C}}, (b_0, b_1, b_2)) = q_2^{\mathcal{C}}$ we have $\big((q^{\mathcal{A}}, q_1^{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0, b_1, b_2)), \varepsilon, id, (q^{\mathcal{A}}, q_2^{\mathcal{C}}, \Phi_P, (b_0, b_1, b_2))\big) \in \Delta$.

The state partitioning is defined by

$$Q_1 = Q^{\mathcal{A}} \times Q^{\mathcal{C}} \times \{\Phi_I, \Phi_P, \Phi_{\mathcal{C}}\} \times \{0,1\}^3$$
$$Q_0 = Q^{\mathcal{A}} \times Q^{\mathcal{C}} \times \{\Phi_O\} \times \{0,1\}^3.$$

Player 0 chooses only the bits of Player O. We let Player 1 do the computation of the characteristic sequence of P, the choice of the bits of Player I and the computation of the run of $\mathcal{C}$. As $\mathcal{C}$ is deterministic there is no real choice in this step as there is only one possibility to go on for Player 1. By the computation of χ_P we know by the definition of the higher-order pushdown parameter generators that there are two possibilities. The first is that Player 1 chooses always the correct transitions and produces χ_P. In the second case Player 1 moves in an unintended way. Then either the run aborts or there are only ε produced forever. Player 1 is punished for choosing wrong by losing the game, so we assume later that he chooses appropriately.

The coloring is defined by

$$\Omega((q^{\mathcal{A}}, q^{\mathcal{C}}, x, (b_0, b_1, b_2))) = \begin{cases} \Omega^{\mathcal{C}}(q^{\mathcal{C}}) & \text{if } x \in \{\Phi_{\mathcal{C}}, \Phi_I, \Phi_O\} \\ (2{\cdot}n) & \text{if } x = \Phi_P \end{cases}$$

where n is the maximal color in $\Omega^{\mathcal{C}}$. All vertices which are not in the Φ_P-phase are colored according to the $\mathcal{C}$-state. The vertices in the Φ_P-phase are colored by a maximal even color. So if Player 1 chooses the edges in an incorrect way and produces in the Φ_P-phase only ε's he stays there forever and lose.

Correctness of Construction. Now we prove the correctness of the construction. For this it remains to show the following:

> *Player O wins the parametrized regular game with the winning condition defined by $\mathcal{C}$ if and only if the higher-order pushdown parity game $\mathcal{G}_P$ is won by Player 0 from the initial configuration.*

Direction left to right: We assume that Player O wins the parametrized regular game. For the parameter P with $\chi_P = p_0p_1p_2\ldots$ let $\rho^{\mathcal{A}}$ be the run of $\mathcal{A}_P$ producing χ_P. More precisely, we assume $\rho^{\mathcal{A}}(0) = (q_0^{\mathcal{A}}, [\,]_k)$ and $\rho^{\mathcal{A}}(i) \xrightarrow[\mathcal{A}_P]{a_i^j} \rho^{\mathcal{A}}(i{+}1)$ for all $i \geq 0$ where $j \in \mathbb{N} \cup \{\bullet\}$ and $j = \ell$ if $a_i^j = p_\ell$ for all $\ell \in \mathbb{N}$ and $j = \bullet$ if $a_i^j \notin \{0,1\}$. Let for all $i \in \mathbb{N}$, $\rho^{\mathcal{A}}(i) = (q_i, s_i)$.

If Player O wins the parametrized regular game Player O can choose for every possible input of Player I a correct output such that the winning condition, given by $\mathcal{C}$, is fulfilled. In particular for all $X = x_0x_1\ldots \in \{0,1\}^\omega$ chosen by Player I, Player O can construct $Y = y_0y_1\ldots \in \{0,1\}^\omega$ such that $(P, X, Y) \in (\{0,1\}^3)^\omega$ is accepted by $\mathcal{C}$. Fix now some X and Y chosen by Player I and Player O. Let $\rho^{\mathcal{C}} \in (Q^{\mathcal{C}})^\omega$ be the accepting run of $\mathcal{C}$ on (P, X, Y) with $\rho^{\mathcal{C}}(0) = q_0^{\mathcal{C}}$ and $\delta^{\mathcal{C}}(\rho^{\mathcal{C}}(i), (p_i, x_i, y_i)) = \rho^{\mathcal{C}}(i+1)$ for all $i \geq 0$.

We have to show that in the `HOPDPG` $\mathcal{G}_P$, which is constructed as described above and where Player 1 and Player 0 choose the bits according to the choices of Player I and Player O, in the parametrized regular game, Player 0 wins this play starting from the initial configuration $((q_0^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_P, (0,0,0)), [\,]_k)$.

Due to the definition of the `HOPDPG` $\mathcal{G}_P$, starting from the initial vertex $((q_0^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_P, (0,0,0)), [\,]_k)$, Player 1 has to compute the first bit of the parameter P. Let in $\rho^{\mathcal{A}}$ for some $i \in \mathbb{N}$ a_i^0 be the first time when a bit is produced in the run $\rho^{\mathcal{A}}$, i.e., for

$$\rho^{\mathcal{A}}(0) \xrightarrow[\mathcal{A}_P]{a_0^{j_0}} \rho^{\mathcal{A}}(1) \xrightarrow[\mathcal{A}_P]{a_1^{j_1}} \ldots \xrightarrow[\mathcal{A}_P]{a_i^{j_i}} \rho^{\mathcal{A}}(i+1)$$

we have for all $\ell \in [1, i-1]$ that $a_\ell^{j_\ell} \notin \{0,1\}$ (so we have $j_\ell = \bullet$) and $a_i^{j_i} \in \{0,1\}$ (from which follows that $j_i = 0$). So we reach the vertex $((q_{i+1}^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_I, (p_0,0,0)), s_{i+1})$.

There are only two possibilities for Player 1 to go on, i.e., either to choose 0 or 1 as first bit. By assumption Player 1 chooses the

bit x_0 and we get to the configuration $((q^{\mathcal{A}}_{i+1}, q^{\mathcal{C}}_0, \Phi_O, (p_0, x_0, 0)), s_{i+1})$. From there Player 0 chooses the bit y_0 according to the choice of Player O (in the parametrized regular game) and we arrive in configuration $((q^{\mathcal{A}}_{i+1}, q^{\mathcal{C}}_0, \Phi_{\mathcal{C}}, (p_0, x_0, y_0)), s_{i+1})$.

In this configuration again Player 1 decides how to go on. However, due to the definition of the higher-order pushdown parity game and because $\mathcal{C}$ is a deterministic parity automaton Player 1 has only one choice, namely to go to $((q^{\mathcal{A}}_{i+1}, \rho^{\mathcal{C}}(1), \Phi_P, (p_0, x_0, y_0)), s_{i+1})$, as $\delta^{\mathcal{C}}(q^{\mathcal{C}}_0, (p_0, x_0, y_0)) = \rho^{\mathcal{C}}(1)$.

We assume that further in the play Player 1 chooses in the Φ_I-vertices the bits according to X and Player 0 chooses in the Φ_O-vertices the bits according to Y. We get the following play with the start as described above and

$$
\begin{array}{ll}
((q^{\mathcal{A}}_j, \rho^{\mathcal{C}}(i), \Phi_P, (p_{i-1}, x_{i-1}, y_{i-1})), s_j) & \xrightarrow{a^{\bullet}_{j+1}} \cdots \xrightarrow{a^{\bullet}_{j+(k-2)}} \xrightarrow{a^{i}_{j+(k-1)}} \\
((q^{\mathcal{A}}_{j+k}, \rho^{\mathcal{C}}(i), \Phi_I, (p_i, x_{i-1}, y_{i-1})), s_{j+k}) & \xrightarrow{x_i} \\
((q^{\mathcal{A}}_{j+k}, \rho^{\mathcal{C}}(i), \Phi_O, (p_i, x_i, y_{i-1})), s_{j+k}) & \xrightarrow{y_i} \\
((q^{\mathcal{A}}_{j+k}, \rho^{\mathcal{C}}(i), \Phi_{\mathcal{C}}, (p_i, x_i, y_i)), s_{j+k}) & \xrightarrow{\varepsilon} \\
((q^{\mathcal{A}}_{j+k}, \rho^{\mathcal{C}}(i+1), \Phi_P, (p_i, x_i, y_i)), s_{j+k}) &
\end{array}
$$

where for all $i > 0$, $j, k, l \geq 0$ holds $a^i_{j+(k-1)} = p_i$ and $a^{\bullet}_l = \varepsilon$.

It remains to show that Player 0 wins this play. This follows as all vertices except the Φ_P-vertices are colored according to the states of $\mathcal{C}$. As $\rho^{\mathcal{C}}$ is accepting and the Φ_P-vertices have a color that is larger than every color in $\Omega^{\mathcal{C}}$, Player 0 wins.

Direction right to left: Assume that Player 0 wins the game $\mathcal{G}_P$ from the initial configuration $((q^{\mathcal{A}}_0, q^{\mathcal{C}}_0, \Phi_P, (0, 0, 0)), [\,]_k)$. In this case Player 0 has a winning strategy φ so that every play compliant to φ fulfills the winning condition. We have to differ between two cases:

1. Player 1 loses because he made a mistake in the computation of P and chooses an edge which refers to a wrong transition in the run of $\mathcal{A}_P$. In this case either the play aborts in a vertex of Player 1 or the play stays forever in the Φ_P-component. So the only color which is seen infinitely often is even. This case we want to exclude because here Player 0 does not necessarily have a winning strategy. We can immediately determine if this case has appeared because it is the only way the play can abort or the color $2 \cdot n$ appears as smallest color which is seen infinitely often.

2. Player 1 loses because the smallest color which appears infinitely often in the play is even and not equal to $2 \cdot n$. This happens only if Player 1 chooses the correct way to construct χ_P.

We concentrate on the second case.

As Player 0 wins the game from the initial configuration for every possible choice of Player 1 we can assume some particular play starting from the initial configuration $((q_0^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_P, (0,0,0)), [\,]_k)$. Due to the definition of the game $\mathcal{G}_P$ and the assumption that Player 1 chooses the correct edges in the $\mathcal{A}_P$ component, the play must have the following form where $l_i \in \mathbb{N}$ for all $i \in \mathbb{N}$:

$$
\begin{array}{lll}
i = 0: & ((q_0^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_P, (0,0,0)), [\,]_k) & \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \xrightarrow{p_0} \\
& ((q_{\ell_1}^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_I, (p_0,0,0)), s_{\ell_1}) & \xrightarrow{x_0} \\
& ((q_{\ell_1}^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_O, (p_0,x_0,0)), s_{\ell_1}) & \xrightarrow{y_0} \\
& ((q_{\ell_1}^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_{\mathcal{C}}, (p_0,x_0,y_0)), s_{\ell_1}) & \xrightarrow{\varepsilon} \\
& ((q_{\ell_1}^{\mathcal{A}}, q_1^{\mathcal{C}}, \Phi_P, (p_0,x_0,y_0)), s_{\ell_1}) & \\
i > 0: & ((q_{\ell_i}^{\mathcal{A}}, q_i^{\mathcal{C}}, \Phi_P, (p_{i-1},x_{i-1},y_{i-1})), s_{\ell_i}) & \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \xrightarrow{p_i} \\
& ((q_{\ell_{i+1}}^{\mathcal{A}}, q_i^{\mathcal{C}}, \Phi_I, (p_i,x_{i-1},y_{i-1})), s_{\ell_{i+1}}) & \xrightarrow{x_i} \\
& ((q_{\ell_{i+1}}^{\mathcal{A}}, q_i^{\mathcal{C}}, \Phi_O, (p_i,x_i,y_{i-1})), s_{\ell_{i+1}}) & \xrightarrow{y_i} \\
& ((q_{\ell_{i+1}}^{\mathcal{A}}, q_i^{\mathcal{C}}, \Phi_{\mathcal{C}}, (p_i,x_i,y_i)), s_{\ell_{i+1}}) & \xrightarrow{\varepsilon} \\
& ((q_{\ell_{i+1}}^{\mathcal{A}}, q_{i+1}^{\mathcal{C}}, \Phi_P, (p_i,x_i,y_i)), s_{\ell_{i+1}}). &
\end{array}
$$

We have that $\chi_P = p_0 p_1 p_2 \ldots$ by construction. Assume that in the parametrized regular game Player I chooses $X = x_0 x_1 x_2 \ldots$ and Player O chooses $Y = y_0 y_1 y_2 \ldots$ as specified by the choices of Player 0 and Player 1 above. By construction the infinite word (P, X, Y) is accepted by $\mathcal{C}$ as the run of $\mathcal{C}$ on (P, X, Y) is simulated in the $\Phi_{\mathcal{C}}$-phase of the higher-order pushdown parity game. So the parametrized regular game (P, X, Y) is won by Player O.

Winning Strategy. In the next part of the proof we want to construct a strategy automaton for the player who wins the parametrized regular game. This automaton is a k-HOPDA with tests in $\mathrm{OReg}_k(\Gamma)$. Remember, that the set $\mathrm{OReg}_k(\Gamma)$ is the set of all stacks of level-k which are regular for operations. The idea for the construction of the strategy automaton for the player winning the game is similar as above and uses the strategy which is delivered by the construction given in Theorem 3.2.

Now, we show in detail:

From a positional strategy for Player 0 (Player 1) winning the game $\mathcal{G}_P$ starting from the initial configuration we can construct a winning strategy for Player O (Player I) in the parametrized regular game.

We assume here that Player 0 wins the game $\mathcal{G}_P$; accordingly Player O wins the parametrized regular game. If Player 1 wins the game the strategy also includes the choices where the parameter P and the next state of the automaton $\mathcal{C}$ is computed. For the strategy in the parametrized regular game those choices are irrelevant so we omit this and show only the construction if Player 0 wins.

Now, we show how to compute a winning strategy for Player O in form of a strategy automaton which uses higher-order pushdown stacks with tests. Let φ be the positional winning strategy for Player 0 in the game $\mathcal{G}_P$ starting from the initial configuration which is computed by the algorithm of Theorem 3.2. It is represented by reduced level-k automata. In particular, the strategy for Player 0 is defined here by two regular sets of stacks[5] since Player 0 can always choose between exactly two alternatives (0 and 1). So we have the sets $S_0 \subseteq V_2$ and $S_1 \subseteq V_2$; for all vertices in S_0 Player 0 has to take the 0 edge, and for all vertices in S_1 Player 0 has to take the 1 edge. Player O just has to to follow the game $\mathcal{G}_P$ and has to mimic the choices of Player 0 in $\mathcal{G}_P$.

The idea to compute the strategy for Player O is to use a `HOPDA` similar to $\mathcal{A}_\mathcal{G}$. In this `HOPDA` we need to have tests, so we can decide for a configuration of Player O whether it belongs to S_0 or S_1. As in S_0 and S_1 also the state of the configuration is stored we have to split the sets such that we have for each state $q \in Q$ sets S_0^q and S_1^q with

$$s \in S_i^q \text{ iff } \mathit{push}_q(s) \in S_i \qquad \text{for } i \in \{0,1\}.$$

Before we construct the higher-order pushdown strategy automaton for Player O, we give a short definition of higher-order pushdown strategy automata. A higher-order pushdown strategy automaton is a k-`HOPDA` with a finite set of tests $\mathcal{R} \subset \mathrm{OReg}_k(\Gamma)$, where we have instead of one transition relation Δ the transition function $\sigma : Q \times \Sigma \times \Gamma_k \times 2^\mathcal{R} \to Q$ for the memory update and the output function $\tau : Q_0 \times 2^\mathcal{R} \to \{0,1\}$.

For the construction of a higher-order pushdown strategy automaton $\mathcal{A}_S$ for Player O in the parametrized regular game we proceed

[5] The state of the configuration (p, s) is stored in the stack by pushing it onto the topmost stack, i.e., we have $\mathit{push}_p(s)$.

as follows. We define the higher-order pushdown strategy automaton $\mathcal{A}_S = (Q, \Sigma^{\mathcal{A}}, \Gamma^{\mathcal{A}}, q_0, \sigma, \tau)$ with test in the set $\{S_i^q \mid i \in \{0,1\}, q \in Q\}$, where Q and q_0 are defined as in the HOPDS $\mathcal{A}_{\mathcal{G}}$ above, i.e., we have $Q = Q^{\mathcal{A}} \times Q^{\mathcal{C}} \times \{\Phi_P, \Phi_1, \Phi_2, \Phi_{\mathcal{C}}\} \times \{0,1\}^3$ and the start state is $q_0 = (q_0^{\mathcal{A}}, q_0^{\mathcal{C}}, \Phi_P, (0,0,0))$.

The transition function σ for the memory update is defined as follows, for all $q^{\mathcal{A}} \in Q^{\mathcal{A}}$, $q^{\mathcal{C}} \in Q^{\mathcal{C}}$, and $b_0, b_1, b_2 \in \{0,1\}$:

$$\begin{aligned}
\sigma((q_1^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_P, (b_0,b_1,b_2)), \varepsilon, \gamma, \emptyset) &= (q_2^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_P, (b_0,b_1,b_2)) \\
&\text{if } (q_1^{\mathcal{A}}, \varepsilon, \gamma, q_2^{\mathcal{A}}) \in \Delta^{\mathcal{A}} \\
\sigma((q_1^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_P, (b_0,b_1,b_2)), 0, \gamma, \emptyset) &= (q_2^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_I, (0,b_1,b_2)) \\
&\text{if } (q_1^{\mathcal{A}}, 0, \gamma, q_2^{\mathcal{A}}) \in \Delta^{\mathcal{A}} \\
\sigma((q_1^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_P, (b_0,b_1,b_2)), 1, \gamma, \emptyset) &= (q_2^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_I, (1,b_1,b_2)) \\
&\text{if } (q_1^{\mathcal{A}}, 1, \gamma, q_2^{\mathcal{A}}) \in \Delta^{\mathcal{A}} \\
\sigma((q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_1, (b_0,b_1,b_2)), b_1', id, \emptyset) &= (q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0,b_1',b_2)) \\
&\text{for } b_1' \in \{0,1\} \\
\sigma((q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0,b_1,b_2)), 0, id, \{S_0^{(q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0,b_1,b_2))}\}) & \\
&= (q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0,b_1,0)) \\
\sigma((q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0,b_1,b_2)), 1, id, \{S_1^{(q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0,b_1,b_2))}\}) & \\
&= (q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0,b_1,1)) \\
\sigma((q^{\mathcal{A}}, q_1^{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0,b_1,b_2)), \varepsilon, id, \emptyset) &= (q^{\mathcal{A}}, q_2^{\mathcal{C}}, \Phi_P, (b_0,b_1,b_2)) \\
&\text{if } \delta^{\mathcal{C}}(q_1^{\mathcal{C}}, (b_0,b_1,b_2)) = q_2^{\mathcal{C}}.
\end{aligned}$$

The output function τ is defined by:

$$\begin{aligned}
\tau((q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0,b_1,b_2)), \{S_0^{(q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0,b_1,b_2))}\}) &= 0 \\
\tau((q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_2, (b_0,b_1,b_2)), \{S_1^{(q^{\mathcal{A}}, q^{\mathcal{C}}, \Phi_O, (b_0,b_1,b_2))}\}) &= 1.
\end{aligned}$$

According to the proof for the correctness of the construction to compute the winner of the parametrized regular game, we can conclude that the defined higher-order pushdown strategy automaton computes a winning strategy for Player O in the parametrized regular game iff Player 0 wins the higher-order pushdown parity game $\mathcal{G}_P$ with the higher-order pushdown strategy given by S_0 and S_1. □

Proposition 4.16. *The computation of the winner and the winning strategy in Theorem 4.15 is possible in k-exponential time.*

Proof. By Theorem 3.2 we have a k-ExpTime procedure to compute the winner of the game $\mathcal{G}_P$ and the positional winning strategy for the player winning $\mathcal{G}_P$. As the construction of $\mathcal{G}_P$ is polynomial in the size of the automata $\mathcal{A}_P$ and $\mathcal{C}$, we have altogether again an algorithm running in k-exponential time to compute the winner of the parametrized regular game as well as the desired higher-order pushdown strategy automaton for the winner. □

4.4 Parametrized Regular Games with Collapsible Pushdown Parameters

In this section we consider parametrized regular games where the parameter is constructed by a collapsible pushdown automaton. *Collapsible pushdown automata* use like higher-order pushdown automata higher-order stacks as additional storage. They extend the idea of higher-order pushdown automata by adding to each symbol in the stack a link to a stack somewhere below it. Furthermore, collapsible pushdown automata have an additional operation called *collapse*, which returns the stack where the link of the topmost stack symbol points to. As the (deterministic) collapsible pushdown automata fulfill the requirements of Theorem 4.11 we can solve parameterized games with parameters constructed by them.

It is unknown if the usage of the collapse increases the power of the automata concerning the recognition of languages. It is known that deterministic order-2 collapsible pushdown automata are stronger than deterministic level-2 higher-order pushdown automata. Parys has shown in [Par11] that a variant of the Urzyczyn language [AdMO05] is recognizable by a deterministic order-2 collapsible pushdown automaton but not by a deterministic level-2 higher-order pushdown automaton. It is even unknown if the Urzyczyn language is recognizable by any deterministic higher-order pushdown automata of a higher level than 2. Nevertheless, a nondeterministic level-2 higher-order pushdown automaton can be constructed to recognize this language.

As even for finite languages it is not known if the deterministic collapsible pushdown automata are more expressive than the deterministic higher-order pushdown automata, it is also not clear if there are parameters $P \subseteq \mathbb{N}$ which can be constructed by deterministic collapsible

pushdown automata and not by deterministic higher-order pushdown automata.

In Section 4.4.1 we introduce the stacks used by collapsible pushdown automata and their operations. Afterwards in Section 4.4.2 we define collapsible pushdown parameter generators which use these stacks and operations. Finally we show in Section 4.4 how to solve parametrized regular games where the parameter is constructed by a collapsible pushdown parameter generator. Here we use the general approach given in Section 4.2 and a result from [CHM$^+$08] about collapsible pushdown parity games.

4.4.1 Background on Collapsible Pushdown Automata

Now we introduce the stacks and operations that are used by the collapsible pushdown automata formally. The stack alphabet is assumed to be Γ which contains a distinguished bottom-of-stack symbol $\bot$.

The stacks on which the collapsible pushdown automata work are quite similar to the higher-order pushdown stacks we used before. The difference is that the stacks of the collapsible pushdown automata add to each symbol in the stack a *link*. The link points to the stack which would remain as topmost stack if the operation *collapse* is applied. The links that are added to the stack symbols can either be indicated by an arrow or as we use them here by two numbers j, k written $a^{(j,k)}$ for $a \in \Gamma$, $j \in [1, n]$ and $k \geq 1$ where n is the order of the stack. These two numbers indicate also in which stack the application of the collapse operation would result in. They will be explained later in detail when the stack operations are introduced.

An *order-0 stack* is just a stack symbol. An *order-*$(n+1)$ *stack* $s = [s_1, \ldots, s_\ell]$ is a nonempty sequence of order-n stacks, such that every Γ symbol a except $\bot$ that appears in s has a link of some order $k \leq n$ to a stack situated below it in s. This link is called a $(k+1)$*-link*. By $ord(s)$ the *order* (respectively level) of a stack s is given. As usual the *bottom-of-stack symbol* $\bot$ can neither be popped from a stack nor pushed onto it. The *empty* k*-stack* is written $\bot_k$ and we define inductively $\bot_0 = \bot$ and $\bot_{k+1} = [\bot_k]$.

Example 11. In the Figure 4.7 an order-3 stack is shown. First the links are depicted by arrows and second the links are annotated beside the stack symbols. For the 1-links we omit the arrows which would point

$$[\,[\,[\,a\,]\,]\,[\,[\;\;]\,[\,a\,b\,c\,]\,[\,a\,b\,c\,]\,]\,] \;=\; [\,[\,[\,a^{(1,1)}\,]\,]\,[\,[\;\;]\,[\,a^{(1,1)}\,b^{(2,1)}\,c^{(3,1)}\,]\,[\,a^{(1,1)}\,b^{(2,2)}\,c^{(3,1)}\,]\,]\,]$$

Figure 4.7: Illustration of an order-3 stack.

directly in front of the symbol. Furthermore, we omit the bottom-of-stack symbol $\perp$ in the stacks, i.e., we write $[[a][\,]]_2$ instead of $[[\perp\!\!\curvearrowleft\! a][\perp]]_2$.

Next, we define the set of *order-n operations* denoted by Op_n. For this consider first the auxiliary operation top_i which takes a stack s and returns the top $(i{-}1)$-stack of s, i.e., for $s = [s_1, \ldots, s_\ell]$ and $i \in [1, ord(s)]$ define:

$$top_i([s_1, \ldots, s_\ell]) = \begin{cases} s_\ell & \text{if } ord(s) = i \\ top_i(s_\ell) & \text{if } ord(s) > i \end{cases}$$

We introduce the operation pop_i for $i \in [1, ord(s)]$ next, since there the links play no role. In difference to the *pop* and $\overline{copy_i}$ we used for the higher-order stacks before, the *pop* here is not defined in the symmetric way. So the deletion of the topmost order-$(i{-}1)$ stack is done without any further condition. We define for $s = [s_1, \ldots, s_\ell, s_{\ell+1}]$ and $i \in [1, ord(s)]$:

$$pop_i([s_1, \ldots, s_\ell, s_{\ell+1}]) = \begin{cases} [s_1, \ldots, s_\ell] & \text{if } ord(s) = i \wedge l \geq 1 \\ [s_1, \ldots, s_\ell, pop_i(s_{\ell+1})] & ord(s) > i \end{cases}$$

Now, we consider the *push-operations* and the *links*. When a stack symbol a is pushed onto the topmost order-1 stack of a stack s, an j-link can be added for some order $j \in [1, n]$ which means that the link points to the $(j-1)$-stack that is immediately below the top $(j-1)$-stack of s. These links are represented by a tuple (j, k) which is attached to the stack symbol, i.e., $a^{(j,k)}$ for $a \in \Gamma$, $1 \leq j \leq n$ and $k \geq 1$. In this case j represents the order of the link and k is needed, when a *collapse* is performed, to indicate how often a pop_j has to be performed to reach the linked stack. For $j = 1$ we have always $k = 1$ and even though there is no link from $\perp$ we assume if $a = \perp$ that $j = k = 1$. We define for $s = [s_1, \ldots, s_\ell]$, $a \in \Gamma$, $j \in [1, ord(s)]$ and $i \in [2, ord(s)]$

$$push_1^{a,j}([s_1, \ldots, s_\ell]) = \begin{cases} \left[s_1, \ldots, s_\ell, a^{(j,1)}\right] & \text{if } ord(s) = 1 \\ \left[s_1, \ldots, push_1^{a,j}(s_\ell)\right] & \text{if } ord(s) > 1. \end{cases}$$

and

$$push_i([s_1,\dots,s_\ell]) = \begin{cases} \left[s_1,\dots,s_\ell,s_\ell^{\langle i\rangle}\right] & \text{if } ord(s)=i \\ [s_1,\dots,push_i(s_\ell)] & \text{if } ord(s)>i \end{cases}$$

where $s_\ell^{\langle i\rangle}$ means that in s_ℓ each stack symbol with superscript (i,k_i) (for some k_i) is replaced by (i,k_i+1), except for $i=1$ there $k_i=1$ holds always.

The last operation we want to define formally is the *collapse*. The collapse follows the link of the top_1 stack symbol and deletes everything until the source of the link is reached, i.e., if the top_1 symbol of the stack s is superscripted by (j,k), it means that k-times a pop_j has to be performed (short: pop_j^k). So we define

$$collapse(s) = pop_j^k(s) \qquad \text{where } top_1(s) = a^{(j,k)}.$$

Altogether, the set of *order-n operations* is given by:

$$Op_n = \{pop_j, push_1^{a,j}, push_i, collapse \mid a \in \Gamma \setminus \{\bot\}, j \in [1,n], i \in [2,n]\}.$$

Furthermore, let CStacks_n be the set of all order-n stacks.

Example 12. We give some examples to illustrate the collapsible stacks and their operations. Take as starting point the order-3 stack $s_1 = [[[a^{(1,1)}]]\,[[\bot][a^{(1,1)}]]]$. We simplify the notation and write a instead of $a^{(1,1)}$.

$$\begin{aligned}
push_1^{b,2}(s_1) &= [[[a]]\,[[\bot][ab^{(2,1)}]]] = s_2\\
push_1^{c,3}(s_2) &= [[[a]]\,[[\bot][ab^{(2,1)}c^{(3,1)}]]] = s_3\\
push_2(s_3) &= [[[a]]\,[[\bot][ab^{(2,1)}c^{(3,1)}][ab^{(2,2)}c^{(3,1)}]]] = s_4\\
push_3(s_3) &= [[[a]]\,[[\bot][ab^{(2,1)}c^{(3,1)}]]\,[[\bot][ab^{(2,1)}c^{(3,2)}]]] = s_5\\
collapse(s_4) &= [[[a]]] = s_6\\
collapse(s_5) &= [[[a]]] = s_6\\
pop_1(s_4) &= [[[a]]\,[[\bot][ab^{(2,1)}c^{(3,1)}][a,b^{(2,2)}]]] = s_7\\
collapse(s_7) &= [[[a]]\,[[\bot]]] = s_8\\
pop_1(s_5) &= [[[a]]\,[[\bot][ab^{(2,1)}c^{(3,1)}]]\,[[\bot][ab^{(2,1)}]]] = s_9\\
collapse(s_9) &= [[[a]]\,[[\bot][ab^{(2,1)}c^{(3,1)}]]\,[[\bot]]]
\end{aligned}$$

We shortly come back to the Urzyczyn language [AdMO05]. It is defined over the alphabet $\Sigma = \{(,), *\}$. A word from the language has the following form. It starts with a prefix of a bracket expression, i.e., in each prefix of the word the number of closing brackets is not greater than the number of opening brackets. Afterwards, a finite sequence of "$*$" is added. For a prefix of a bracketed expression w let v be the longest suffix which is a full bracketed expression. Then the number of "$*$" which follows w is the number of opening brackets in w without v. For example the words $((()())*$, $()$ and $()((()***$ belong to the language and the words $())()*$, $()*$ and $(()(()**$ do not.

A deterministic collapsible pushdown automaton of order-2 that recognizes the Urzyczyn language works as follows. When the automaton reads a "(" it performs $push_2; push_1^{a,2}$. When the automaton reads a ")" it performs a pop_1. When the automaton reads the first "$*$" it performs first a collapse and then for each further "$*$" a pop_2. If at the end of the word the topmost order-1 stack is empty the automaton accepts.

4.4.2 Collapsible Pushdown Parameter Generators

We go on by introducing a collapsible pushdown automaton to construct the parameters, i.e., a collapsible pushdown parameter generator.

Definition 4.17. A *collapsible pushdown parameter generator* (short: n-CPDPG) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta)$ where $\Sigma = \{0, 1, \varepsilon\}$ is the word alphabet, Γ is a stack alphabet, Q is a finite set of states, $q_0 \in Q$ is a initial state and $\Delta \subseteq Q \times \Gamma \times \Sigma \times Q \times Op_n$ is a transition relation.

In terminus of Section 4.2 we refer to the countable set of configurations Conf as the set $Q \times \mathrm{CStacks}_n$, i.e., a configuration of a n-CPDPG is a pair (q, s) where $q \in Q$ and $s \in \mathrm{CStacks}_n$. The initial configuration conf_0 is $(q_0, \bot_n)$. The transition relation Δ induces a labeled transition relation over configurations by $(q, s) \xrightarrow{a} (q', s')$ if $(q, top_1(s), a, q', \gamma) \in \Delta$ and $s' = \gamma(s)$.

As before an ω-word $\alpha \in \{0, 1\}^\omega$ is *constructed* by the n-CPDPG $\mathcal{A}$ if there exists an infinite run

$$(q_0, \bot_n) \xrightarrow{\alpha_0} (q_1, s_1) \xrightarrow{\alpha_1} (q_2, s_2) \xrightarrow{\alpha_2} (q_3, s_3) \xrightarrow{\alpha_3} \dots$$

such that α is obtained from $\alpha_0\alpha_1\alpha_2\alpha_3\dots$ by deleting all occurrences of ε. We assume that there is exactly one run which generates an infinite

word over $\{0,1\}$. There may be other runs but they either abort or produce eventually only ε.

By this definition the collapsible pushdown parameter generators fulfill the conditions of the general parameter generators given in Definition 4.9.

4.4.3 Solution

To solve parametrized regular games, where the parameter is constructed by a collapsible pushdown parameter generator, we can use Definition 4.10 and Proposition 4.11 as well as a result on collapsible pushdown games. First, we define games played on the configuration graph of a collapsible pushdown system. Afterwards, we cite the result that these games can be solved [HMOS08]. Then, we show how to solve the parametrized regular games where the parameter is constructed by a collapsible pushdown parameter generator.

To define games over collapsible pushdown graphs we first introduce collapsible pushdown systems and then the game graphs they produce.

Definition 4.18. An *order-n collapsible pushdown system* (short n-CPDS) is defined by a 4-tuple $\mathcal{A} = (Q, \Gamma, \Delta, q_0)$ where Γ is a stack alphabet, Q is a finite set of states with an initial state q_0 and $\Delta \subseteq Q \times \Gamma \times Q \times Op_n$ is a transition relation.

A configuration is a tuple $(q, s) \in Q \times \mathrm{CStacks}_n$. The transition relation is defined by $(q, s) \xrightarrow{\gamma} (q', s')$ if and only if $(q, top_1(s), q', \gamma) \in \Delta$ and $s' = \gamma(s)$. We have as initial configuration $(q_0, \bot_n)$. The configuration graph of $\mathcal{A}$ is given by taking all from $(q_0, \bot_n)$ with $\xrightarrow{\gamma}$ reachable configurations as vertices, and the edge relation is defined by $\xrightarrow{\gamma}$ restricted to the reachable configurations.

To define a parity game played over the configuration graph of an n-CPDS we need besides an n-CPDS $\mathcal{A}$ a partitioning $Q_0 \cup Q_1$ of Q (the state set of $\mathcal{A}$) and a coloring function $\Omega\colon Q \to \{1, \ldots, n\}$ for some $n \in \mathbb{N}$. If $G = (V, E)$ is the configuration graph of $\mathcal{A}$ then we define V_0 and V_1 to be those vertices of V where the control state is in Q_0 respectively in Q_1. The color of a vertex $(q, s) \in V$ is defined by $\Omega(q)$. So altogether, we define a *n-CPDS parity game $\mathcal{G}$ over a n-CPDS game graph* given by (V, E, V_0, V_1, Ω).

Theorem 4.19 ([HMOS08])**.** *The problem of solving an n-CPDS parity game is n-ExpTime complete. Further one can build an n-CPDA with*

output that realizes a winning strategy for the winning player.

Theorem 4.20. *Let $P \subseteq \mathbb{N}$ be defined by a order-n CPDPG $\mathcal{A}_P$. The parametrized regular game where the winning condition is given by a deterministic finite parity word automaton $\mathcal{C}$ over $\{0,1\}^3$ is (determined and) solvable: It can be decided who wins the game and for the winner one can construct a n-CPDA with output that realizes a winning strategy.*

Proof. Using Definition 4.10 we construct from the order-n CPDPG $\mathcal{A}_P$ and the deterministic finite parity automaton automaton $\mathcal{C}$ defining the winning condition of the parametrized regular game $\mathcal{G}(P, \mathcal{C})$ a game product $\mathcal{A}_P \otimes \mathcal{C}$. Note that the game product $\mathcal{A}_P \otimes \mathcal{C}$ defines a order-n collapsible pushdown game. This game can be solved using Theorem 4.19. From the winner of $\mathcal{A}_P \otimes \mathcal{C}$ we know the winner of $\mathcal{G}(P, \mathcal{C})$. Furthermore, the winning strategy for the winning player of the order-n collapsible pushdown game which is computed by Theorem 4.19 can be adapted as strategy for the winner of the parametrized regular game $\mathcal{G}(P, \mathcal{C})$. So, the computation of the winner of $\mathcal{G}(P, \mathcal{C})$ and his winning strategy has the same time complexity as the computation of the order-n collapsible pushdown game defined by $\mathcal{A}_P \otimes \mathcal{C}$, i.e., it is n-ExpTime complete. □

5 On Higher-Order Counter Automata

In this chapter we consider higher-order stacks that use only one stack symbol and the corresponding automata as language recognizers. In this case a level-1 stack can be regarded as a natural number. These kinds of stacks with only one stack symbol are called counters. When we extend the idea to higher-order counters we have on level 2 a stack of numbers, on level 3 a stack of stacks of numbers and so on.

In this chapter we restrict the focus on the language theoretic aspects and thus consider languages which can be recognized when using higher-order counters instead of higher-order stacks.

First, we introduce higher-order counters and the corresponding automata in Section 5.1. In Section 5.2 we show that for a language that can be recognized by a level-k higher-order pushdown automaton we can construct a level-$(k+1)$ higher-order counter automaton which recognizes the same language. Afterwards, we give example languages and higher-order counter automata that recognize them to illustrate cases where the pushdown automata and the counter automata are of the same level. In Section 5.3 we present a conjecture which says that there are languages which can be recognized by a level-k higher-order pushdown automaton but not by a level-k higher-order counter automaton. For $k = 1$ we give an automata theoretic proof concerning a language that is recognizable by a deterministic pushdown automaton but not by a deterministic counter automaton.

5.1 Preliminaries

We introduce the *higher-order counters* which are a special case of the higher-order stacks. In this case the stack alphabet Γ contains only one symbol. For simplification we assume that this stack symbol is x, i.e., $\Gamma = \{x\}$. We write for the level-1 stack $[x^n]_1$ simply n for

each number $n \in \mathbb{N}$. For level 2 we denote for example the counter $[[x^4][x^3][\,][x]]_2$ by $[4,3,0,1]_2$. Furthermore, we denote the *set of all higher-order counters of level k* by Counter_k. We define $\text{Counter}_1 = \mathbb{N}$ and $\text{Counter}_{n+1} = (\text{Counter}_n)^+$. The *operations over higher-order counters* are defined accordingly to the operations over higher-order pushdown stacks but we write *inc* instead of $push_x$ and instead of pop_x we write *dec*. So we have:

$$\begin{aligned} Ops_1^C &= \{inc, dec, T_{[\,]_1}, id\} \\ Ops_{k+1}^C &= Ops_k^C \cup \{copy_k, \overline{copy}_k, T_{[\,]_{k+1}}\} \end{aligned}$$

In the set of *instructions over higher-order counters* we replace x by $+$ and $\overline{x}$ by $-$. We write:

$$\begin{aligned} \Gamma_1^C &= \{+, -, \bot_1, id\} \\ \Gamma_{k+1}^C &= \Gamma_k^C \cup \{k, \overline{k}, \bot_{k+1}\} \end{aligned}$$

Higher-order counter automata are defined similar to higher-order pushdown automata but work on higher-order counters instead of higher-order stacks.

Definition 5.1. A *level-k higher-order counter automaton* $\mathcal{C}$ (k-HOCA for short) is defined as a tuple $(Q, \Sigma, \Gamma, \varepsilon, q_0, F, \Delta)$ where Q is a finite set of states, Σ is an input alphabet, $\Gamma = \{x\}$ is the stack symbol alphabet consisting only of one symbol, $\varepsilon \notin \Sigma$ is an extra symbol to Σ that does not count for the accepted word, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states and $\Delta \subseteq Q \times \Sigma \times \Gamma_k^C \times Q$ is a transition relation[1].

A configuration of a k-HOCA $\mathcal{C}$ is a pair $(p, c) \in Q \times \text{Counter}_k$. We write $(p, c) \xrightarrow{\alpha} (q, c')$ if there exists a transition $(p, \alpha, \gamma, q) \in \Delta$ such that $c' = \gamma(c)$.

A run of a k-HOCA $\mathcal{C}$ starts in the initial configuration $(q_0, [\,]_k)$ and proceeds by applying the transition relation. A word $w \in \Sigma^*$ is recognized by $\mathcal{C}$ if there exists a run

$$(q_0, [\,]_k) \xrightarrow{\alpha_1} (q_1, c_1) \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} (q_n, c_n)$$

with $q_n \in F$ and we obtain w by deleting all ε from $\alpha_1 \ldots \alpha_n$.

[1] We could also use the operations and write $\Delta \subseteq Q \times \Sigma \times Ops_k^C \times Q$.

5.2 Higher-Order Counters vs. Higher-Order Stacks

In this section we show that we can simulate each level-k higher-order pushdown automaton by a level-$(k+1)$ higher-order counter automaton. The idea is to assign to each stack symbol in the stack alphabet of the `HOPDA` a number and then use a whole level-1 counter for the representation of a single symbol in a level-1 stack. Using this idea the "higher level operations" of the stacks, i.e., $copy_l$, $\overline{copy_l}$ and $T_{[\,]_l}$ for $l \in [1,k]$, are shifted by one level in the counters. Only for the level-1 operations, *push* and *pop*, we have to make some extra steps to simulate them.

We first introduce two functions that define a correspondence between a level-k higher-order stack and a level-$(k+1)$ higher-order counter. For this a level-1 counter is associated with some symbol in the stack alphabet of the higher-order stack. If the level-1 counter corresponds to some number which is higher than the total number of symbols in the stack alphabet we return a failure as in this case the translation into a stack fails. The same holds if there is an empty level-1 counter besides other level-1 counters in a level-2 counter. We will need the two following functions for the proof of Theorem 5.3.

Definition 5.2. We define two functions: The function $\eta\colon \text{Stacks}_k \to \text{Counter}_{k+1}$ which assigns a level-$(k+1)$ counter to a level-k stack and the function $\bar{\eta}\colon \text{Counter}_{k+1} \to \text{Stacks}_k$ which assigns a level-k stack to a level-$(k+1)$ counter. For this we assume that the stack alphabet is $\Gamma = \{a_1, \ldots, a_n\}$ and the counter alphabet $\Gamma' = \{x\}$.
We give an inductive definition for η by:

$$\begin{array}{lcll}
\eta(a_i) & = & [x^i]_1 & \text{for all } i \in [1,n] \\
\eta([\,]_1) & = & [[\,]_1]_2 & \\
\eta([a_{i_1}, \ldots, a_{i_m}]_1) & = & [\eta(a_{i_1}), \ldots, \eta(a_{i_m})]_2 & \text{for } m \geq 1, a_{i_j} \in \Gamma \\
\eta([s_1, \ldots, s_m]_\ell) & = & [\eta(s_1), \ldots, \eta(s_m)]_{\ell+1} & \text{for } m \geq 1, \ell \in [2,k]
\end{array}$$

We define $\bar{\eta}$ inductive by:

$$\begin{array}{lcll}
\bar{\eta}([x^i]_1) & = & \begin{cases} e & \text{if } i = 0 \\ a_i & \text{if } 1 \leq i \leq n \\ f & \text{otherwise} \end{cases} & \\
\bar{\eta}([c_1, \ldots, c_m]_2) & = & \begin{cases} [\,]_1 & \text{if } m = 1, c_1 = [\,]_1 \\ [\bar{\eta}(c_1), \ldots, \bar{\eta}(c_m)]_1 & \text{otherwise} \end{cases} & \\
\bar{\eta}([c_1, \ldots, c_m]_{\ell+1}) & = & [\bar{\eta}(c_1), \ldots, \bar{\eta}(c_m)]_\ell & \text{for } m \geq 1, \ell \in [2,k]
\end{array}$$

Later, we assume that no level-1 counter contains more than n times x or an empty level-1 stack beside other level-1 stacks, as otherwise the correspondence fails. For completeness, we add here two new letters f, e to Γ. Into f all counters greater than n are translated and e is used for the case that an empty level-1 counter is somewhere between some other level-1 counters.

Now, we show that for each language which is recognized by some k-HOPDA we can construct a $(k+1)$-HOCA that recognizes the same language. Note that there are also languages where this additional level is not necessary, but we come back to this later. The idea to use a level-1 counter as a stack symbol has been given before.

To simulate in this case the stack operations $push_a$ or pop_a we proceed as follows. Take for example the stack symbol b and assume that it represented by the number 2. If $push_b$ is applied by a k-HOPDA onto a stack $s \in \text{Stacks}_k(\Gamma)$ we simulate this in the $(k+1)$-HOCA by first doing a $copy_1$, then decrementing the top level-1 counter until it is empty, and afterwards we increment two times. For the case that the $push_b$ is applied to an empty level-1 stack we do not have to perform the $copy_1$ and just increment the counter two times. If pop_b is applied by a k-HOPDA to a stack $s \in \text{Stacks}_k(\Gamma)$ we first decrement the top level-1 counter two times and check that it is empty. Afterwards, we increment as long as it is needed to perform a $\overline{copy_1}$, i.e., this depends on the stack symbol below the b and if b has been the only symbol in the stack, we end in the counter with the emptiness test. The remaining stack operations $copy_\ell$, $\overline{copy_\ell}$ and $T_{[\,]_\ell}$ are simulated in the $(k+1)$-HOCA by the according operations on level higher, i.e., $copy_{\ell+1}$, $\overline{copy_{\ell+1}}$ and $T_{[\,]_{\ell+1}}$.

Theorem 5.3. *For each level k, $k \geq 1$ holds that if a language $\mathcal{L}$ is recognized by a k-HOPDA $\mathcal{A}$ then there exists a $(k+1)$-HOCA $\mathcal{C}$ which recognizes $\mathcal{L}$.*

Proof. Let $\mathcal{A} = (Q_\mathcal{A}, \Sigma, \Gamma_\mathcal{A}, \varepsilon, q_0^\mathcal{A}, F_\mathcal{A}, \Delta_\mathcal{A})$ be a k-HOPDA recognizing $\mathcal{L}$ and assume $\Gamma_\mathcal{A} = \{a_1, \ldots, a_n\}$. We use here the operations instead of instructions in the transition relation to ease the readability. We define a $(k+1)$-HOCA $\mathcal{C}$ by the tuple $(Q_\mathcal{C}, \Sigma, \{x\}, \varepsilon, q_0^\mathcal{C}, F_\mathcal{C}, \Delta_\mathcal{C})$ with:

- $Q_\mathcal{C} = Q_\mathcal{A} \cup \{q_-^j, q_+^j, q_-^i, q_+^{d,j} \mid \forall q \in Q_\mathcal{A}, j \in [1,n]\}$[2],

[2]The index i in the name of the states stands for increment and the d for decrement.

- $q_0^{\mathcal{C}} = q_0^{\mathcal{A}}$,
- $F_{\mathcal{C}} = F_{\mathcal{A}}$,
- $\Delta_{\mathcal{C}}$:
  - for $(p, \alpha, push_{a_j}, q) \in \Delta_{\mathcal{A}}$, $j \in [1, n]$ add:

$$\begin{gathered}(p, \alpha, T_{[\,]_2}, q_+^j),\\ (p, \alpha, copy_1, q_+^{d,j}),\\ (q_+^{d,j}, \varepsilon, dec, q_+^{d,j}),\\ (q_+^{d,j}, \varepsilon, T_{[\,]_1}, q_+^j),\\ (q_+^j, \varepsilon, inc, q_+^{j-1}), \ldots, (q_+^1, \varepsilon, inc, q)\end{gathered}$$

  - for $(p, \alpha, pop_{a_j}, q) \in \Delta_{\mathcal{A}}$, $j \in [1, n]$ add:

$$\begin{gathered}(p, \alpha, dec, q_-^j),\\ (q_-^j, \varepsilon, dec, q_-^{j-1}), \ldots, (q_-^2, \varepsilon, dec, q_-^1),\\ (q_-^1, \varepsilon, T_{[\,]_2}, q),\\ (q_-^1, \varepsilon, T_{[\,]_1}, q_-^i)\\ (q_-^i, \varepsilon, inc, q_-^i)\\ (q_-^i, \varepsilon, \overline{copy_1}, q)\end{gathered}$$

  - for $(p, \alpha, copy_\ell, q) \in \Delta_{\mathcal{A}}$, $\ell \in [1, k-1]$ add $(p, \alpha, copy_{\ell+1}, q)$,
  - for $(p, \alpha, \overline{copy_\ell}, q) \in \Delta_{\mathcal{A}}$, $\ell \in [1, k-1]$ add $(p, \alpha, \overline{copy_{\ell+1}}, q)$,
  - for $(p, \alpha, T_{[\,]_\ell}, q) \in \Delta_{\mathcal{A}}$, $\ell \in [1, k]$ add $(p, \alpha, T_{[\,]_{\ell+1}}, q)$.

Now, it remains to show the correctness of the construction, i.e., that both automata recognize the same language. For this we show for all $w \in \Sigma^*$ that $w \in \mathcal{L}(\mathcal{A})$ if and only if $w \in \mathcal{L}(\mathcal{C})$.

Direction from left to right: Take some word $w = w_1 \ldots w_\ell \in \Sigma^*$ with $w \in \mathcal{L}(\mathcal{A})$. Then there exists a run $\rho_{\mathcal{A}}$ of $\mathcal{A}$ of the form

$$(q_0^{\mathcal{A}}, [\,]_k) \xrightarrow{\alpha_1} (q_1, s_1) \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} (q_m, s_m)$$

such that there exists $i_1, \ldots, i_\ell \in [1, m]$, $i_1 < i_2 < \ldots < i_\ell$ with $w_j = \alpha_{i_j}$ for $j \in [1, \ell]$ and $\alpha_j = \varepsilon$ for all $j \notin \{i_1, \ldots, i_\ell\}$. Furthermore, we have $\gamma_1, \ldots, \gamma_m \in \Gamma_k$ with $\gamma_i(s_{i-1}) = s_i$ and $(q_{i-1}, \alpha_i, \gamma_i, q_i) \in \Delta_{\mathcal{A}}$ for all $i \in [1, m]$ and $q_m \in F_{\mathcal{A}}$.

Now, we show how to construct an accepting run $\rho_{\mathcal{C}}$ of $\mathcal{C}$ on w by induction on the length of the run $\rho_{\mathcal{A}}$. The run $\rho_{\mathcal{C}}$ is in the majority of cases longer as $\rho_{\mathcal{A}}$ because it takes several steps to simulate a *push* or *pop*. So, we show by induction on the length of $\rho_{\mathcal{A}}$ for all $i \in [0, m]$ and (q_i, s_i) in $\rho_{\mathcal{A}}$, that there exists $j \geq i$ with (q_j, c_j) in $\rho_{\mathcal{C}}$ where $q_i = q_j$ and $\eta(s_i) = c_j$.

The run $\rho_{\mathcal{A}}$ of length 0 is just the initial configuration $(q_0^{\mathcal{A}}, [\,]_k)$ and we get the initial configuration $(q_0^{\mathcal{C}}, \eta([\,]_k)) = (q_0^{\mathcal{C}}, [\,]_{k+1})$ with $q_0^{\mathcal{A}} = q_0^{\mathcal{C}}$ from the definition of $\mathcal{C}$.

Now, assume for the run $\rho_{\mathcal{A}}$ of length $v-1$, $v \geq 1$ we have constructed a corresponding run $\rho_{\mathcal{C}}$ of $\mathcal{C}$ ending in a configuration $(q_{v-1}, \eta(s_{v-1})) = (q_{v-1}, c_{v-1})$ and the next transition which is taken in $\rho_{\mathcal{A}}$ to get from (q_{v-1}, s_{v-1})[3] to (q_v, s_v) via α_v is $(q_{v-1}, \alpha_v, \gamma_v, q_v) \in \Delta_{\mathcal{A}}$. We have to consider five different cases depending on γ_v:

1. If $\gamma_v = push_{a_i}$ for some $a_i \in \Sigma$ we have to add several steps to the run $\rho_{\mathcal{C}}$. We have to distinguish between two cases. In the first case the topmost level-1 stack of s_{v-1} is empty. In the second case we assume $top_0(s_{v-1}) = a_j$ for some $j \in [1, n]$.

   For the first case $T_{[\,]_1}(s_{v-1}) = s_{v-1}$ and $T_{[\,]_2}(c_{v-1}) = c_{v-1}$. The run $\rho_{\mathcal{C}}$ continues as follows:

   $$(q_{v-1}, c_{v-1}) \xrightarrow{\alpha_v} (q^i_{v,+}, T_{[\,]_2}(c_{v-1})) \xrightarrow{\varepsilon} (q^{i-1}_{v,+}, T_{[\,]_2} inc(c_{v-1})) \xrightarrow{\varepsilon}_{i-2} (q^1_{v,+}, T_{[\,]_2} inc^{i-1}(c_{v-1})) \xrightarrow{\varepsilon} (q_v, T_{[\,]_2} inc^i(c_{v-1}))$$

   The transitions used in the run have to be in $\Delta_{\mathcal{C}}$ by construction. Note that $T_{[\,]_2} inc^i(c_{v-1}) = \eta(s_v)$.

   For the second case assume that $top_1(c_{v-1}) = [x^j]_1$, respectively, that $top_0(s_{v-1}) = a_j$ for some $j \in [1, n]$. The run $\rho_{\mathcal{C}}$ continues as follows:

   $$(q_{v-1}, c_{v-1}) \xrightarrow{\alpha_v} (q^{d,i}_{v,+}, copy_1(c_{v-1})) \xrightarrow{\varepsilon}_j (q^{d,i}_{v,+}, copy_1 dec^j(c_{v-1})) \xrightarrow{\varepsilon} (q^i_{v,+}, copy_1 dec^j T_{[\,]_1}(c_{v-1})) \xrightarrow{\varepsilon} (q^{i-1}_{v,+}, copy_1 dec^j T_{[\,]_1} inc(c_{v-1})) \xrightarrow{\varepsilon}_{i-2} (q^1_{v,+}, copy_1 dec^j T_{[\,]_1} inc^{i-1}(c_{v-1})) \xrightarrow{\varepsilon} (q_v, copy_1 dec^j T_{[\,]_1} inc^i(c_{v-1}))$$

   The transitions used in the run have to be in $\Delta_{\mathcal{C}}$ by construction. Note that $copy_1 dec^j T_{[\,]_1} inc^i(c_{v-1}) = \eta(s_v)$.

[3] Note that the state q_{v-1} is the same in the runs $\rho_{\mathcal{A}}$ and $\rho_{\mathcal{C}}$. The stacks s_{v-1} and c_{v-1} are different which is clear as s_{v-1} is a stack of level k and c_{v-1} is a counter of level $k+1$ but it holds that $\eta(s_{v-1}) = c_{v-1}$ and $\bar{\eta}(c_{v-1}) = s_{v-1}$ by construction.

2. If $\gamma_v = pop_{a_i}$ for some $a_i \in \Sigma$, we have to add several steps to the run $\rho_{\mathcal{C}}$. We have to consider two cases, where we differ between the case that there is at least one more symbol a_j in the level-1 stack after we have poped a_i and the case that the level-1 stack is empty after the pop of a_i. For the first case assume that $top_0(pop_{a_i}(s_{v-1})) = a_j$ for some $j \in [1, n]$. The run $\rho_{\mathcal{C}}$ continues as follows:

$$(q_{v-1}, c_{v-1}) \xrightarrow{\alpha_v} (q^i_{v,-}, dec(c_{v-1})) \xrightarrow{\varepsilon} (q^{i-1}_{v,-}, dec^2(c_{v-1})) \xrightarrow{\varepsilon}_{i-2}$$
$$(q^1_{v,-}, dec^i(c_{v-1})) \xrightarrow{\varepsilon} (q_{v,-}, dec^i T_{[\,]_1}(c_{v-1})) \xrightarrow{\varepsilon}_{j}$$
$$(q_{v,-}, dec^i T_{[\,]_1} inc^j(c_{v-1})) \xrightarrow{\varepsilon} (q_v, dec^i T_{[\,]_1} inc^j \overline{copy_1}(c_{v-1}))$$

   The transitions that are used in the run have to be in $\Delta_{\mathcal{C}}$ by construction. Note that $dec^i T_{[\,]_1} inc^j \overline{copy_1}(c_{v-1}) = \eta(s_v)$.

   For the second case the run $\rho_{\mathcal{C}}$ continues as follows:

$$(q_{v-1}, c_{v-1}) \xrightarrow{\alpha_v} (q^i_{v,-}, dec(c_{v-1})) \xrightarrow{\varepsilon} (q^{i-1}_{v,-}, dec^2(c_{v-1})) \xrightarrow{\varepsilon}_{i-2}$$
$$(q^1_{v,-}, dec^i(c_{v-1})) \xrightarrow{\varepsilon} (q_v, dec^i T_{[\,]_2}(c_{v-1}))$$

   The transitions that have been used in the run have to be in $\Delta_{\mathcal{C}}$ by construction. Note that $dec^i T_{[\,]_2}(c_{v-1}) = \eta(s_v)$.

3. If $\gamma_v = copy_\ell$ for some $\ell \in [1, k-1]$, we add the letter α_v and the configuration $(q_v, copy_{\ell+1}(c_{v-1}))$ to the run $\rho_{\mathcal{C}}$. This can be done as the corresponding transition $(q_{v-1}, \alpha_v, copy_{\ell+1}, q_v)$ has to be in $\Delta_{\mathcal{C}}$ by definition of the construction. Note that $copy_{\ell+1}(c_{v-1}) = \eta(copy_\ell(s_{v-1})) = \eta(s_v)$.

4. If $\gamma_v = \overline{copy_\ell}$ for some $\ell \in [1, k-1]$, we add the letter α_v and the configuration $(q_v, \overline{copy_{\ell+1}}(c_{v-1}))$ to the run ρ_C. This can be done as the corresponding transition $(q_{v-1}, \alpha_v, \overline{copy_{\ell+1}}, q_v)$ has to be in $\Delta_{\mathcal{C}}$ by definition of the construction. As $c_{v-1} = \eta(s_{v-1})$ and $\overline{copy_\ell}(s_{v-1}) = s_v$, $\overline{copy_{\ell+1}}(c_{v-1})$ has to be defined and is equal to $\eta(s_v)$.

5. If $\gamma_v = T_{[\,]_\ell}$ for some $\ell \in [1, k]$, we add the letter α_v and the configuration $(q_v, T_{[\,]_{\ell+1}}(c_{v-1}))$ to the run $\rho_{\mathcal{C}}$. This can be done as the transition $(q_{v-1}, \alpha_v, T_{[\,]_{\ell+1}}, q_v)$ has to be in $\Delta_{\mathcal{C}}$ by definition of the construction. As $c_{v-1} = \eta(s_{v-1})$ and $T_{[\,]_\ell}(s_{v-1}) = s_v$, i.e., the test is successful, $T_{[\,]_{\ell+1}}(c_{v-1})$ has also to be defined and is equal to $\eta(s_v)$.

As the run $\rho_{\mathcal{C}}$ ends in the same state as $\rho_{\mathcal{A}}$, it follows that $\rho_{\mathcal{C}}$ is accepting only if $\rho_{\mathcal{A}}$ is accepting.

Direction from right to left: Now, it remains to show the converse direction, i.e., that if there exists an accepting run of $\mathcal{C}$ for some word $w \in \Sigma^*$ then there is also an accepting run of $\mathcal{A}$. For this assume $w = w_1 \dots w_\ell \in \mathcal{L}(\mathcal{C})$. Then there exists a run $\rho_{\mathcal{C}}$ on w of the following form:

$$(q_0^{\mathcal{C}}, [\ ]_{k+1}) \xrightarrow{\alpha_1} (q_1^{\mathcal{C}}, s_1) \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} (q_m^{\mathcal{C}}, s_m)$$

such that there exists $i_1, \dots, i_\ell \in [1, m]$, $i_1 < i_2 < \dots < i_\ell$ with $w_j = \alpha_{i_j}$ for $j \in [1, \ell]$ and $\alpha_j = \varepsilon$ for all $j \notin \{i_1, \dots, i_\ell\}$. Furthermore, we have $\gamma_1, \dots, \gamma_m \in \Gamma_{k+1}^{C}$ with $\gamma_i(s_{i-1}) = s_i$ and $(q_{i-1}^{\mathcal{C}}, \alpha_i, \gamma_i, q_i^{\mathcal{C}}) \in \Delta_{\mathcal{C}}$ for all $i \in [1, m]$ and it holds that $q_m^{\mathcal{C}} \in F_{\mathcal{C}}$.

We show how to construct an accepting run $\rho_{\mathcal{A}}$ of $\mathcal{A}$ on w by induction on the length of the run $\rho_{\mathcal{C}}$ where, due to the construction, sometimes, we have to merge several steps of the run of $\mathcal{C}$ into one step of the run of $\mathcal{A}$, as the run $\rho_{\mathcal{C}}$ is in the majority of cases longer as $\rho_{\mathcal{A}}$. So, the induction is somehow more on the length of $\rho_{\mathcal{A}}$ again. We have to find the points in the run $\rho_{\mathcal{C}}$ where we can construct a corresponding point in $\rho_{\mathcal{A}}$, i.e., the states are the same and also the stacks when we use the η- or respectively the $\bar{\eta}$-function.

So, we have to show for all $i \in [1, m]$ where $q_i^{\mathcal{C}} \in Q_{\mathcal{A}}$ for $(q_i^{\mathcal{C}}, c_i) \in \rho_{\mathcal{C}}$ that there exists $(q_j^{\mathcal{A}}, s_j) \in \rho_{\mathcal{A}}$ with $q_i^{\mathcal{C}} = q_j^{\mathcal{A}}$ and $\bar{\eta}(c_i) = s_j$.

The run $\rho_{\mathcal{C}}$ of length 0 is just the initial configuration $(q_0^{\mathcal{C}}, [\ ]_{k+1})$ and in $\rho_{\mathcal{A}}$ we have $(q_0^{\mathcal{A}}, \bar{\eta}([\ ]_{k+1})) = (q_0^{\mathcal{A}}, [\ ]_k)$, where $q_0^{\mathcal{C}} = q_0^{\mathcal{A}}$ by definition.

Assume we have constructed for the run $\rho_{\mathcal{C}}$ of length $v-1$, for $v \geq 1$, and $q_{v-1} \in Q_{\mathcal{A}}$, a corresponding run $\rho_{\mathcal{A}}$ of $\mathcal{A}$ ending in a configuration $(q_{v-1}, \bar{\eta}(c_{v-1})) = (q_{v-1}, s_{v-1})$[4].

Let the next transition which is taken in $\rho_{\mathcal{C}}$ to get from (q_{v-1}, c_{v-1}) to (q_v, c_v) via α_v be $(q_{v-1}, \alpha_v, \gamma_v, q_v) \in \Delta_{\mathcal{C}}$. We have to consider different cases depending on γ_v:

1. If $\gamma_v = copy_1$ then, by construction, the run $\rho_{\mathcal{C}}$ has to go on in the

[4]Note that the state q_{v-1} is the same in the runs $\rho_{\mathcal{A}}$ and $\rho_{\mathcal{C}}$. The stacks c_{v-1} and s_{v-1} are different which is clear as c_{v-1} is a counter of level $(k+1)$ and s_{v-1} is a stack of level k but it holds that $\bar{\eta}(c_{v-1}) = s_{v-1}$ and $\eta(s_{v-1}) = c_{v-1}$ by construction.

following way until a state in $Q_{\mathcal{A}}$ is reached again

$$(q_{v-1}, c_{v-1}) \xrightarrow[copy_1]{\alpha_v} (q_v, c_v) \xrightarrow[dec]{\varepsilon}{}^{i} (q_{v+i}, c_{v+i}) \xrightarrow[T_{[\,]_1}]{\varepsilon} (q_{v+i+1}, c_{v+i+1}) \xrightarrow[inc]{\varepsilon}{}^{j} (q_{v+i+1+j}, c_{v+i+1+j})$$

for $i \in [1, n]$ where $top_1(\bar{\eta}(c_{v-1})) = a_i$ and some $j \in [1, n]$. Assume p is the first state we reach that is again in $Q_{\mathcal{A}}$ then we can replace the "dummy" states as follows:

$$\begin{aligned}(q_{v-1}, c_{v-1}) &\xrightarrow{\alpha_v} (p_+^{d,j}, copy_1(c_{v-1})) \xrightarrow{\varepsilon}{}^{i} (p_+^{d,j}, copy_1 dec^i(c_{v-1})) \xrightarrow{\varepsilon} \\ (p_+^j, copy_1 dec^i T_{[\,]_1}(c_{v-1})) &\xrightarrow{\varepsilon}{}^{j-1} (p_+^1, copy_1 dec^i T_{[\,]_1} inc^{j-1}(c_{v-1})) \\ &\xrightarrow{\varepsilon} (p, copy_1 dec^i T_{[\,]_1} inc^j(c_{v-1}))\end{aligned}$$

By construction there has to be a transition $(q_{v-1}, \alpha_v, push_{a_j}, p)$ in $\Delta_{\mathcal{A}}$. We add

$$(q_{v-1}, s_{v-1}) \xrightarrow{\alpha_v} (p, push_{a_j}(s_{v-1})) = (p, \bar{\eta}(c_{v+i+1+j}))$$

to the run $\rho_{\mathcal{A}}$.

2. If $\gamma_v = dec$ then, by construction, the run $\rho_{\mathcal{C}}$ has to go on in one of the two following ways until a state in $Q_{\mathcal{A}}$ is reached again. The first way considers the case that there is at least one other level-1 counter below the current, and the second considers the case that there is only the current level-1 counter in the level-2 counter. The first case proceeds as follows:

$$(q_{v-1}, c_{v-1}) \xrightarrow[dec]{\alpha_v} (q_v, c_v) \xrightarrow[dec]{\varepsilon}{}^{i-1} (q_{v+i-1}, c_{v+i-1}) \xrightarrow[T_{[\,]_1}]{\varepsilon} (q_{v+i}, c_{v+i}) \xrightarrow[inc]{\varepsilon}{}^{j} (q_{v+i+j}, c_{v+i+j}) \xrightarrow[\overline{copy_1}]{\varepsilon} (q_{v+i+j+1}, c_{v+i+j+1})$$

for $i \in [1, n]$ and $j \in [1, n]$ where $top_0(pop_{a_i}(\bar{\eta}(c_{v-1}))) = a_j$. Assume p to be the first state which is reached that is in $Q_{\mathcal{A}}$, then we can replace the "dummy" states as follows:

$$\begin{aligned}(q_{v-1}, c_{v-1}) &\xrightarrow{\alpha_v} (p_-^j, dec(c_{v-1})) \xrightarrow{\varepsilon}{}^{i-1} (p_-^1, dec^i(c_{v-1})) \xrightarrow{\varepsilon} \\ (p_-, dec^i T_{[\,]_1}(c_{v-1})) &\xrightarrow{\varepsilon}{}^{j} (p_-, dec^i T_{[\,]_1} inc^j(c_{v-1})) \xrightarrow{\varepsilon} \\ &(p, dec^i T_{[\,]_1} inc^j \overline{copy_1}(c_{v-1}))\end{aligned}$$

By construction there has to be a transition $(q_{v-1}, \alpha_v, pop_{a_i}, p)$ in $\Delta_{\mathcal{A}}$. We add

$$(q_{v-1}, s_{v-1}) \xrightarrow{\alpha_v} (p, pop_{a_i}(s_{v-1})) = (p, \bar{\eta}(c_{v+i+j+1}))$$

to the run $\rho_{\mathcal{A}}$.

In the second case we have the following situation:

$$(q_{v-1}, c_{v-1}) \xrightarrow[dec]{\alpha_v} (q_v, c_v) \xrightarrow[dec]{\varepsilon}{}^{i-1} (q_{v+i-1}, c_{v+i-1}) \xrightarrow[T_{[\,]_2}]{\varepsilon} (q_{v+i}, c_{v+i})$$

for $i \in [1, n]$ where $top_0(pop_{a_i}(\bar{\eta}(c_{v-1})))$ is not defined because the level-1 stack is empty. Assume p is the first state that is reached which is in $Q_{\mathcal{A}}$. Then we can replace the "dummy" states as follows:

$$(q_{v-1}, c_{v-1}) \xrightarrow{\alpha_v} (p^j_-, dec(c_{v-1})) \xrightarrow{\varepsilon}{}^{i-1} (p^1_-, dec^i(c_{v-1})) \xrightarrow{\varepsilon} (p, dec^i T_{[\,]_2}(c_{v-1}))$$

By construction there has to be a transition $(q_{v-1}, \alpha_v, pop_{a_i}, p)$ in $\Delta_{\mathcal{A}}$. We add to $\rho_{\mathcal{A}}$: $(q_{v-1}, s_{v-1}) \xrightarrow{\alpha_v} (p, pop_{a_i}(s_{v-1})) = (p, \bar{\eta}(c_{v+i}))$.

3. If $\gamma_v = copy_{\ell+1}$, for some $\ell \in [1, k]$ then there has to be a transition $(q_{v-1}, \alpha_v, copy_\ell, q_v)$ in $\Delta_{\mathcal{A}}$ by construction and we add $(q_{v-1}, s_{v-1}) \xrightarrow{\alpha_v} (q_v, copy_\ell(s_{v-1})) = (q_v, \bar{\eta}(c_v))$ to the run $\rho_{\mathcal{A}}$.

4. If $\gamma_v = \overline{copy_{\ell+1}}$, for some $\ell \in [1, k]$ then there has to be a transition $(q_{v-1}, \alpha_v, \overline{copy_\ell}, q_v)$ in $\Delta_{\mathcal{A}}$ by construction and we add $(q_{v-1}, s_{v-1}) \xrightarrow{\alpha_v} (q_v, \overline{copy_\ell}(s_{v-1})) = (q_v, \bar{\eta}(c_v))$ to the run $\rho_{\mathcal{A}}$. It is clear that $\overline{copy_\ell}(s_{v-1})$ is defined as $\overline{copy_{\ell+1}}(c_{v-1})$ is defined and $\bar{\eta}(c_{v-1}) = s_{v-1}$).

5. If $\gamma_v = T_{[\,]_2}$ we have to distinguish between two cases. To know which one is present we need to take the next transition in $\rho_{\mathcal{C}}$ into account. If it contains as next operation an *inc* then the run has to continue as follows until again a state in $Q_{\mathcal{A}}$ is reached

$$(q_{v-1}, c_{v-1}) \xrightarrow[T_{[\,]_2}]{\alpha_v} (q_v, c_v) \xrightarrow[inc]{\varepsilon}{}^{j} (q_{v+j}, c_{v+j})$$

for some $j \in [1, n]$. Assume p is the first state we reach that is again in $Q_{\mathcal{A}}$ then we can replace the "dummy" states as follows:

$$(q_{v-1}, c_{v-1}) \xrightarrow{\alpha_v} (p^j_+, T_{[\,]_2}(c_{v-1})) \xrightarrow{\varepsilon}{}^{j-1} (p^1_+, inc^{j-1}(c_{v-1})) \xrightarrow{\varepsilon} (p, inc^j(c_{v-1}))$$

By construction there has to be a transition $(q_{v-1}, \alpha_v, push_{a_j}, p)$ in $\Delta_{\mathcal{A}}$. We add

$$(q_{v-1}, s_{v-1}) \xrightarrow{\alpha_v} (p, push_{a_j}(s_{v-1})) = (p, \bar{\eta}(c_{v+i+1+j}))$$

to the run $\rho_{\mathcal{A}}$.

In the other cases where the operation in the next transition is not *inc* we simply refer to the next Point 6.

6. If $\gamma_v = T_{[\,]_{\ell+1}}$, $\ell \in [1,k]$ then there has to be a transition of the form $(q_{v-1}, \alpha_v, T_{[\,]_\ell}, q_v)$ in $\Delta_{\mathcal{A}}$ by construction and we add $(q_{v-1}, s_{v-1}) \xrightarrow{\alpha_v} (q_v, T_{[\,]_\ell}(s_{v-1})) = (q_v, \bar{\eta}(c_v))$ to $\rho_{\mathcal{A}}$. It is clear that $T_{[\,]_\ell}(s_{v-1})$ is defined as $T_{[\,]_{\ell+1}}(c_{v-1})$ is defined and $\bar{\eta}(c_{v-1}) = s_{v-1}$.

7. By the assumption that $q_{v-1} \in Q_{\mathcal{A}}$ it follows by the given construction that $\gamma_v \notin \{inc, \overline{copy_1}, T_{[\,]_1}\}$.

As the run $\rho_{\mathcal{A}}$ ends in the same state as $\rho_{\mathcal{C}}$ it follows that $\rho_{\mathcal{A}}$ is accepting only if $\rho_{\mathcal{C}}$ is accepting. □

The next two examples show how the higher-order counter automata can be used to compute the tower of two, i.e., for some number n to compute the number 2^n and 2^{2^n}. The idea for this counter automata can be easily extended to further levels, i.e., to compute for some $n \in \mathbb{N}$ the number $2^{\cdot^{\cdot^{\cdot^{2^n}}}\}\text{k}}$ for some fixed k. These kind of languages show that it is not always necessary to have stacks instead of counters as it has been shown in [CW03, Cau02] that for each $k \in \mathbb{N}$ the languages $\{a^n b^{2^{\cdot^{\cdot^{\cdot^{2^n}}}\}\text{k}}} \mid n \in \mathbb{N}\}$ can be recognized by a level-$(k+1)$ higher-order pushdown automaton, i.e., both higher-order pushdown automata and higher-order counter automata need the same level.

Corollary 5.4. *For each level $k \geq 1$ there are languages such that the level k of the higher-order pushdown automaton and the higher-order counter automaton recognizing this languages are the same.*

Example 13. In this example we define a level-2 higher-order counter automaton $\mathcal{C}$ that recognizes the language $\{a^n b^{2^n} \mid n \geq 0\}$. We define $\mathcal{C}$ by $(Q, \{a,b\}, \{x\}, \varepsilon, q_0, F, \Delta)$ with $Q = \{q_i, q_f \mid i \in [0,8]\}$, $F = \{q_f\}$ and Δ is given in Table 5.1. The whole higher-order counter automaton is also depicted in Figure 5.1.

The idea for the counter automaton is the following. First, we store the number of a's -1 in the counter, i.e., assume that we read i times an a. Then the counter $[i-1, i-2, i-2, i-3, i-3, \ldots, 1, 1, 0, 0]_2$ is built by cycling through the sequence $q_4, copy_1, q_5, dec, q_6, copy_1, q_4$. If the two topmost counters are 0 and we are in state q_8 then a new subpart

(q_0, a, id, q_1)	(q_0, b, id, q_f)	
(q_1, a, inc, q_1)		
$(q_1, b, T_{[\,]_2}, q_2)$	(q_2, b, id, q_f)	
(q_1, b, dec, q_3)	$(q_3, \varepsilon, inc, q_4)$	
$(q_4, b, copy_1, q_5)$	$(q_5, \varepsilon, dec, q_6)$	$(q_6, \varepsilon, copy_1, q_4)$
	$(q_5, \varepsilon, T_{[\,]_1}, q_7)$	$(q_7, \varepsilon, \overline{copy_1}, q_8)$
$(q_8, \varepsilon, \overline{copy_1}, q_4)$		
$(q_8, \varepsilon, inc, q_9)$	$(q_9, \varepsilon, \overline{copy_1}, q_8)$	
$(q_8, \varepsilon, dec, q_{10})$	$(q_{10}, \varepsilon, dec, q_{10})$	$(q_{10}, \varepsilon, T_{[\,]_2}, q_f)$

Table 5.1: Transition relation Δ of Example 13.

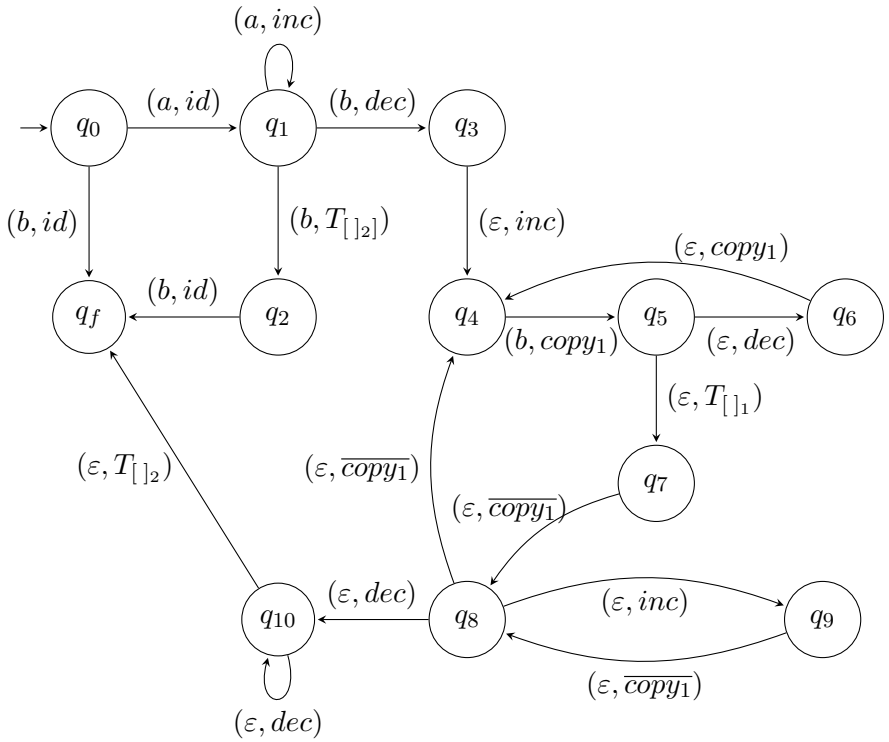

Figure 5.1: Higher-order counter automaton of Example 13.

$$
\begin{aligned}
&(q_0, [0]_2) \xrightarrow{a} (q_1, [0]_2) \xrightarrow{a} (q_1, [1]_2) \xrightarrow{a} (q_1, [2]_2) \xrightarrow{b} (q_3, [1]_2) \\
&\to (q_4, [2]_2) \xrightarrow{b} (q_5, [2, 2]_2) \to (q_6, [2, 1]_2) \to (q_4, [2, 1, 1]_2) \\
&\xrightarrow{b} (q_5, [2, 1, 1, 1]_2) \to (q_6, [2, 1, 1, 0]_2) \to (q_4, [2, 1, 1, 0, 0]_2) \\
&\xrightarrow{b} (q_5, [2, 1, 1, 0, 0, 0]_2) \to (q_7, [2, 1, 1, 0, 0, 0]_2) \to \\
&(q_8, [2, 1, 1, 0, 0]_2) \to (q_4, [2, 1, 1, 0]_2) \xrightarrow{b} (q_5, [2, 1, 1, 0, 0]_2) \\
&\to (q_7, [2, 1, 1, 0, 0]_2) \to (q_8, [2, 1, 1, 0]_2) \to (q_9, [2, 1, 1, 1]_2) \\
&\to (q_8, [2, 1, 1]_2) \to (q_4, [2, 1]_2) \xrightarrow{b} (q_5, [2, 1, 1]_2) \\
&\to (q_6, [2, 1, 0]_2) \to (q_4, [2, 1, 0, 0]_2) \xrightarrow{b} (q_5, [2, 1, 0, 0, 0]_2) \\
&\to (q_7, [2, 1, 0, 0, 0]_2) \to (q_8, [2, 1, 0, 0]_2) \to (q_4, [2, 1, 0]_2) \\
&\xrightarrow{b} (q_5, [2, 1, 0, 0]_2) \to (q_7, [2, 1, 0, 0]_2) \to (q_8, [2, 1, 0]_2) \to \\
&(q_9, [2, 1, 1]_2) \to (q_8, [2, 1]_2) \to (q_9, [2, 2]_2) \to (q_8, [2]_2) \\
&\to (q_{10}, [1]_2) \to (q_{10}, [0]_2) \to (q_f, [0]_2)
\end{aligned}
$$

Table 5.2: Run for the word a^3b^8 of the counter $\mathcal{C}$ of Example 13.

starts. If a $\overline{copy_1}$ can be performed directly, it is done and the cycling is started again. If first an increment of the counter is needed to perform the $\overline{copy_1}$ we go again into state q_8. For the case that the current counter contains only one level 1 counter we are done. The idea is similar to binary counting, where e.g., two times the same number in two succeeding counters means 1 and if a number is contained only once in the counters it means 0. An example run for the word a^3b^8 is shown in Table 5.2. The whole automaton is also quite similar to the higher-order pushdown parameter generator in the example of Section 4.3.2 which generates the characteristic sequence of the set $\{2^n \mid n \in \mathbb{N}\}$.

Example 14. In this example we show a level-3 higher-order counter automaton $\mathcal{C}$ that recognizes the language $\{a^n b^{2^{2^n}} \mid n \geq 0\}$. We define $\mathcal{C}$ by $(Q, \{a, b\}, \{x\}, \varepsilon, q_0, F, \Delta)$ with $Q = \{q_i, q', q'', q_f \mid i \in [0, 18]\}$, $F = \{q_f\}$ and Δ is given in Table 5.3.

In this level-3 higher-order counter automaton the idea of the level-2 higher-order counter automaton of Example 13 is used and transfered to one level above. Now, besides the idea to use the level-1 counter to count binary, we also use the level 2. There we can, by the use of the $copy_2$ and $\overline{copy_2}$, have either two times the same level-2 counter or only one of them.

(q_0, a, id, q_1)	(q_0, b, id, q_f)		
(q_1, a, inc, q_1)			
$(q_1, b, T_{[\,]_3}, q_2)$	(q_2, b, id, q')	(q', b, id, q'')	(q'', b, id, q_f)
(q_1, b, dec, q_3)	$(q_3, \varepsilon, inc, q_4)$		
$(q_4, b, copy_2, q_5)$			
$(q_5, \varepsilon, copy_1, q_6)$	$(q_6, \varepsilon, dec, q_7)$	$(q_7, \varepsilon, copy_1, q_8)$	
	$(q_6, \varepsilon, T_{[\,]_1}, q_9)$	$(q_9, \varepsilon, \overline{copy_1}, q_{10})$	
$(q_8, \varepsilon, copy_2, q_4)$			
$(q_{10}, \varepsilon, \overline{copy_1}, q_{11})$	$(q_{11}, \varepsilon, copy_2, q_4)$		
$(q_{10}, \varepsilon, inc, q_{12})$	$(q_{12}, \varepsilon, \overline{copy_1}, q_{10})$		
$(q_{10}, \varepsilon, dec, q_{13})$	$(q_{13}, \varepsilon, dec, q_{13})$		
	$(q_{13}, \varepsilon, T_{[\,]_2}, q_{14})$	$(q_{14}, b, copy_2, q_{15})$	
$(q_{10}, \varepsilon, T_{[\,]_2}, q_{18})$	$(q_{18}, \varepsilon, \overline{copy_2}, q_{15})$		
$(q_{15}, \varepsilon, \overline{copy_2}, q_4)$			
$(q_{15}, \varepsilon, inc, q_{16})$	$(q_{16}, \varepsilon, inc, q_{16})$	$(q_{16}, \varepsilon, copy_1, q_{16})$	
	$(q_{16}, \varepsilon, dec, q_{16})$	$(q_{16}, \varepsilon, \overline{copy_2}, q_{15})$	
$(q_{15}, \varepsilon, dec, q_{17})$	$(q_{17}, \varepsilon, dec, q_{17})$	$(q_{17}, \varepsilon, T_{[\,]_3}, q_f)$	

Table 5.3: Transition relation Δ of Example 14.

5.3 A Conjecture

We have seen in the last section that for each language which can be recognized by a k-HOPDA we can define a $(k+1)$-HOCA to recognize the same language. Furthermore, we have seen two examples for languages where the HOPDA needs the same level as the HOCA to recognize the language. This leads to the question whether the HOPDA are needed at all or, respectively, whether there are languages where a stack alphabet with more than one letter is really needed. For level-1 counter automata one finds a proof in [Ber79] that there are languages which can be recognized by a level-1 pushdown automaton but not by a level-1 counter automaton. Examples for this kind of languages are: the language containing only mirror words, i.e. $\{w\$w^R \mid w \in \{a,b\}^+\}$, and the language containing only correctly bracketed words with two kinds of brackets.

While trying to show that the result of Berstel can also be lifted to all further levels, we give here a new automata theoretic proof for level-1. We restrict here to the case of deterministic automata, where no unbounded number of ε-steps are allowed between the reading two letters of the input word. The language we use in this case contains only words of the form $a^n b^m \$ b^m a^n$ for $n, m \geq 1$.

Theorem 5.5. *The language $L_1' = \{a^n b^m \$ b^m a^n \mid n, m \geq 1\}$ is recognizable by a deterministic 1-HOPDA but not by a deterministic 1-HOCA.*

Proof. We first define a deterministic 1-HOPDA $\mathcal{A}$ that recognizes the language L_1'. So let $\mathcal{A} = (Q, \{a, b, \$\}, \{a, b\}, q_0, \{q_f\}, \Delta)$ be defined with $Q = \{q_0, q_1, q_2, q_3, q_f\}$ and:

$$\begin{aligned}\Delta \;=\; & \{(q_0, a, push_a, q_0), (q_0, b, push_b, q_1), (q_1, b, push_b, q_1),\\ & (q_1, \$, id, q_2), (q_2, b, pop_b, q_2), (q_2, a, pop_a, q_3),\\ & (q_3, a, pop_a, q_3), (q_3, \varepsilon, T_{[\,]_1}, q_f)\}.\end{aligned}$$

The automaton is also given in Figure 5.2.

To show that the language L_1' is not recognizable by a deterministic level-1 counter automaton, we assume that it is recognizable and then this leads to a contradiction. The idea for the proof is to use a pumping argument. We take n and m big enough to find several repetitions of a part of the run such that we start and end in the same state, we read at least one letter a or b in between and some number x_a respectively x_b is added to the counter. In the part of the run where the a's are read

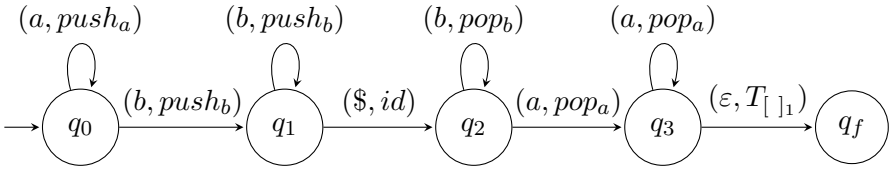

Figure 5.2: Deterministic 1-HOPDA for the language $L_1' = \{a^n b^m \$ b^m a^n \mid n, m \geq 1\}$.

we delete this part x_b times and in the part where the b's are read we add the part x_a time. So, the numbers of a's and b's change but the run reaches the \$ in the same configuration as before. Now, we show this in detail.

Let $\mathcal{C}$ be a deterministic 1-HOCA recognizing L_1' with k states. Then there has to be an accepting run on each word of the form $a^n b^m \$ b^m a^n$ for $n, m \geq 1$. While reading the part of the word before the \$ the height of the counter has in general to be growing (besides short periods of decreasing). Otherwise if the counter height would repeatedly be set to 0 or would stay below some fixed number, then for big enough numbers n, m a repetition of a configuration would appear and could be used for pumping the word which leads to a contradiction. So, the counter has to be growing while reading $a^n b^m$.

Now, we consider very large numbers n and m and regard the accepting run π of $\mathcal{C}$ on $a^n b^m \$ b^m a^n$. The run π is a sequence in $(Q \times \mathbb{N})(\{a, b, \$, \varepsilon\}(Q \times \mathbb{N}))^*$. Let us split the run into three parts, the part π_a where a^n is read, the part π_b where b^m is read and the rest π_r where $\$ b^m a^n$ is read.

Let us first consider the run of $\mathcal{C}$ on a^n, i.e., π_a. For very large n there has to be at least one state which has to be repeated several times in the configurations but with different counter heights. Lets fix this state to be p. So let $t_1, t_2, \ldots, t_i$ be the points with $\pi_a(t_j) = (p, c_{t_j})$ where c_{t_j} is the height of the counter at point t_j for $j \in [1, i]$. Then we consider the differences between the counter heights of two succeeding configurations of this set, i.e., let $\lambda_j = c_{t_{j+1}} - c_{t_j}$ for $j \in [1, i-1]$. Again for a large enough n at least one difference λ_j has to be seen repeatedly as the distance between two points t_j and t_{j+1} cannot increase more and more. Furthermore, we can assume that we find two configurations where in between at least one a is read. But this is only the case if we consider

deterministic automata as otherwise there could be ε-transitions which increase the stack more and more staying in the same state.

Subsuming, we have a segment of π_a starting in (p, c_a) ending in (p, c'_a) where at least one a occurs in between, with $c'_a - c_a = \Lambda$ and this segment occurs several times in π_a for different c_a, c'_a if n is large enough.

We go on considering the part of the run where the b^m is read. Here, we can make the same assumptions and find a repeated segment in π_b starting in a configuration (p', c_b) ending in (p', c'_b) where at least one b occurs in between, with $c'_b - c_b = \Lambda'$ and this segment occurs several times in π_b for different c_b, c'_b if m is large enough.

Let now n and m be such that in the run on a^n at least Λ' times a segment appears with the difference Λ between two configurations with the state p. Furthermore, we define $|\Lambda|_a$ to be the number of times that an a is read between the configurations $(p, c_1), (p, c_2)$ with $\Lambda = c_2 - c_1$ and $|\Lambda'|_b$ to be the number of times that a b is read, accordingly.

Now, we construct an accepting run on a word which is not in L'_1. Take the run we considered before and in the part π_a where the a^n is read we cut out Λ' times a segment between two configurations (p, c_1) and (p, c_2) with $c_2 - c_1 = \Lambda$. So now only $n - (\Lambda' \cdot |\Lambda|_a)$ times an a is read and the stack height is $\Lambda' \cdot \Lambda$ times lower. Note that after reading $a^{n-(\Lambda' \cdot |\Lambda|_a)}$ we are in the same state as in the original run after a^n. In the next part of the run π_b where b^m is read, we repeat Λ times a segment of the run between (p', c'_1) and (p', c'_2) with $c'_2 - c'_1 = \Lambda'$. So we have $m + (\Lambda \cdot |\Lambda'|_b)$ times read b and the stack height is increased by $\Lambda \cdot \Lambda'$. Furthermore, we end after $a^{n-(\Lambda' \cdot |\Lambda|_a)} b^{m+(\Lambda \cdot |\Lambda'|_b)}$ in the same state as after reading $a^n b^m$. As we have first subtracted $\Lambda' \cdot \Lambda$ from the stack height and then added back the same number we end actually in the same configuration. From this it follows that $a^{n-(\Lambda' \cdot |\Lambda|_a)} b^{m+(\Lambda \cdot |\Lambda'|_b)} \$ b^m a^n$ is recognized by $\mathcal{C}$ but the word is not in L'_1 which contradicts our assumption that $\mathcal{C}$ recognizes exactly L'_1. An illustration of the proof idea can be found in Figure 5.3. □

In order to show that we can find for each level k, where $k \geq 1$, languages which can be recognized by a level-k higher-order pushdown automaton but cannot be recognized by any level-k higher-order counter automaton and so to lift this result to any further level, we propose the following set of languages.

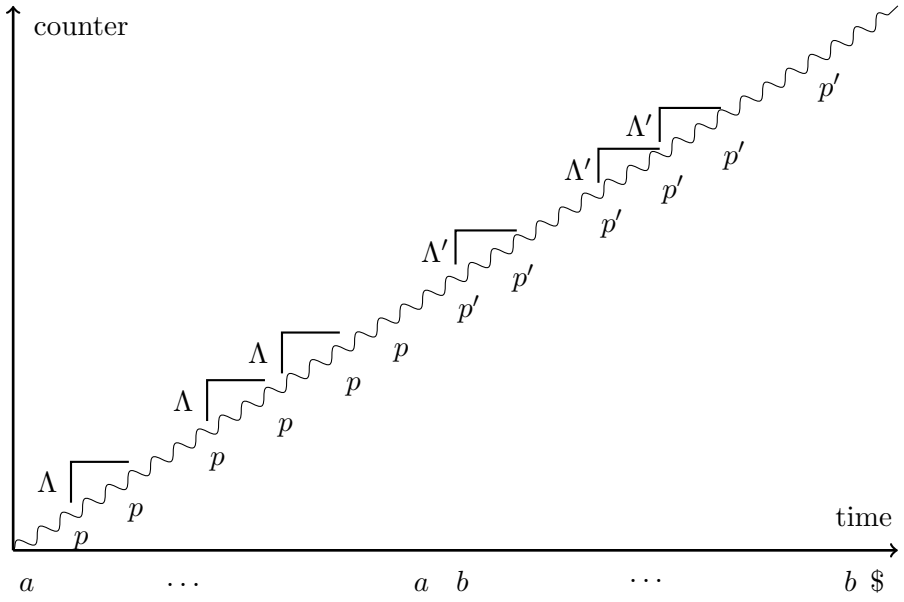

Figure 5.3: Illustration of the proof idea of Theorem 5.5.

We define a set of languages L_k, $k \geq 1$ as follows:

$$
\begin{aligned}
k = 1: &\quad L_1 = \{w\$w^R \mid w \in \{a,b\}^+\} \\
k = 2: &\quad L_2 = \{w\$w \mid w \in \{a,b\}^+\} \\
k > 2: &\quad L_k = \{w\$(w^R)^{2^{\cdot^{\cdot^{\cdot^{2^{|w|+1}}}}}\}\text{k-2}} \mid w \in \{a,b\}^+\}
\end{aligned}
$$

Conjecture 5.6. *For each level k, $k \geq 1$ the language L_k is recognized by a level k-HOPDA but there exists no level k-HOCA recognizing L_k.*

One can construct the k-HOPDA for the languages L_k: The one for L_2 is given in Example 7, and for the higher levels we can combine the idea from L_2 with the one used to construct the tower of powers of two shown in Examples 13 and 14. A problem in a proof for non-recognizability for counter automata is the handling of blocks of ε-transitions.

6 Conclusion

We have considered in this work the use of higher-order pushdown stacks in different settings. First, we used the higher-order pushdown stacks as a store in higher-order pushdown systems, where we took the configuration graph of the higher-order pushdown system to define parity games. For these games, we computed global positional winning strategies. Furthermore, we used the higher-order stacks in parametrized regular games, where we took higher-order pushdown parameter generators as one way to construct the parameters and solve the games. Moreover, we considered languages which can be recognized by higher-order pushdown automata and compared them with languages recognized by higher-order counter automata, which use higher-order stacks with only one stack symbol.

In Chapter 3 we defined two-player parity games which are played on the configuration graph of a higher-order pushdown system. We gave a k-ExpTime algorithm to compute global positional winning strategies for both players in a level-k higher-order pushdown parity game. These strategies were given by regular sets of stacks, where we used the definition of regularity for operations defined in [Car05, Fra05a]. The definition of regularity for operations relies on the use of symmetric operations, i.e., the symmetric destruction of level-k stacks by $\overline{copy_k}$. As byproducts of the global positional winning strategies we also got the winning regions of both players by regular sets of stacks in k-ExpTime for a level-k higher-order pushdown parity game. Note that deciding the winner in level-k higher-order pushdown games is already k-ExpTime hard which has been shown in [CW07]. In [CHM$^+$08] higher-order pushdown parity games were considered where the higher-order pushdown systems use the following: the unconditional destruction of stacks by $destr_k$ instead of $\overline{copy_k}$, and the definition of regularity of sets of higher-order stacks which we call regular for words [BM04]. It has been shown in [CHM$^+$08] that the winning region is regular for words in this case. This result is stronger than our result, as the definition of regularity for words

is more restrictive than the definition of regularity for operations. In [CHM+08] a finite description of a winning strategy was constructed for a given starting vertex and the player who wins from this vertex. The strategy was realized by a higher-order pushdown automaton that reads the moves of the play and outputs the next move. It is not known if positional winning strategies for higher-order pushdown parity games can be realized in the setting of regularity for words. In the last part of Chapter 3 we also considered winning conditions other than parity, i.e., Büchi and request-response winning conditions. We reduced both kinds of games to higher-order pushdown parity games. Note, that higher-order pushdown Büchi games are just a special case of higher-order pushdown parity games which use only two colors. It is also possible for other winning conditions to find a reduction. For this, the ideas for the respective reduction in the case of games on finite game graphs may be lifted to higher-order pushdown games. Nevertheless, it would also be interesting to solve higher-order pushdown Büchi games directly by some algorithm that uses a saturation method like in [HO07, HO08]. In this case where we can use the symmetric operations the saturation should not be based on the entries of the higher-order stacks but on the sequences of operations building them.

In Chapter 4 we took Church's Problem and extended it with a parameter $P \subseteq \mathbb{N}$. In this case we got a variation of Gale-Stewart games where in each round i first the current bit of the parameter is provided (i.e. 1 if $i \in P$ and 0 otherwise) then Player Input chooses a bit and afterwards Player Output chooses a bit. The resulting play can be seen as an infinite word over $\{0,1\}^3$. The winning condition for Player Output is that this word belongs to some language L respectively in terms of Church that it fulfills some MSO-formula φ over the structure $(\mathbb{N}, +1, P)$. We stated a theorem of Rabinovich [Rab06] and gave a new proof using techniques different than those of Rabinovich. The theorem says that if the MSO-theory of the structure $(\mathbb{N}, +1, P)$ is decidable then the winner of the parametrized regular game can be computed and a recursive winning strategy can be constructed from the winning condition. In the further part of this chapter we wanted to obtain sharper results on the format of the strategies, in dependence of the "complexity" of the parameter P. Thus, we introduced a deterministic machine (called parameter generator) that generates a parameter P and which uses in our cases higher-order stacks as memory structure. We showed that we

can guarantee the same "complexity" for the strategies that has been used for the structure of the parameter generator. The general approach to construct these strategies is based on a game product of a parameter generator and a parity automaton, that defines the winning condition of the parametrized regular game. Then, we applied the memoryless determinacy property of parity games. After stating this as an abstract result we gave a concrete and effective construction for the case where the parameter generator is a kind of higher-order pushdown automaton and for the case that it is a collapsible pushdown automaton. Our approach can be extended to other kinds of parameter generators which use different memory structures.

In Chapter 5 we considered the case where a higher-order pushdown automaton uses higher-order stacks which just contain one stack symbol. In this case we speak of a higher-order counter automaton that uses higher-order counters. We compared the languages which can be recognized by higher-order pushdown automata and higher-order counter automata. First, we showed for each language which is recognized by a level-k higher-order pushdown automaton that we could construct a level-$(k+1)$ higher-order counter automaton that recognizes the same language. Next, we gave examples for languages where there is no difference between the levels of higher-order pushdown automata and higher-order counter automata which recognize these languages. Furthermore, we obtained some partial results. We gave an automata theoretic proof for the difference of the recognition power of deterministic pushdown automata and deterministic counter automata. Finally, for each level k we proposed languages that can be recognized by some level-k higher-order pushdown automaton, conjecturing that they cannot be recognized by a level-k higher-order counter automaton.

We worked in this thesis with higher-order pushdown automata that use symmetric operations with the symmetric destruction of the level-k stack by $\overline{copy_k}$. So, we also used this definition in the case of higher-order counter automata. Another interesting problem when considering higher-order counter automata would be to compare those which use the symmetric destruction with the ones that use the unconditional destruction by $destr_k$. For the case of higher-order pushdown systems it has been shown in [Wöh05] that the automata which use the symmetric destruction can be simulated by the ones which use the unconditional destruction and vice versa. This simulation of the symmetric destruction

by the unconditional destruction works by enriching the stack alphabet with more symbols. This is no longer possible when we speak about counters, as they are not allowed to have more than one stack symbol. On the other hand counter automata using the symmetric destruction can still simulate counter automata with the unconditional destruction. So it would be interesting to see what difference this would make concerning the languages that these automata can recognize, as it seems that the higher-order counter automata with symmetric destruction are stronger, i.e., can recognize more languages.

Bibliography

[AdMO05] Klaus Aehlig, Jolie G. de Miranda, and C.H. Luke Ong. Safety is not a restriction at level 2 for string languages. *Foundations of Software Science and Computational Structures*, pages 490–504, 2005.

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR'97:Concurrency Theory, 8th International Conference, Warsaw, Poland*, volume 1243 of LNCS, pages 135–150. Springer Verlag, 1997.

[Ber79] Jean Berstel. *Transductions and context-free languages*, volume 4. Teubner Stuttgart, 1979.

[BL69] J. Richard Büchi and Lawrence H. Landweber. Solving Sequential Conditions by Finite-State Strategies. *Transactions of the AMS*, 138:295 – 311, 1969.

[BM04] Ahmed Bouajjani and Antoine Meyer. Symbolic Reachability Analysis of Higher-Order Context-Free Processes. In *FSTTCS*, LNCS, pages 135–147, 2004.

[Büc64] J. Richard Büchi. Regular canonical systems. *Archive for Mathematical Logic*, 6(3):91–111, 1964.

[Cac01] Thierry Cachat. Two-Way Tree Automata Solving Pushdown Games. In *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of LNCS, pages 303–317. Springer Verlag, 2001. Chapter 17.

[Cac02] Thierry Cachat. Symbolic Strategy Synthesis for Games on Pushdown Graphs. In *ICALP*, volume 2380 of LNCS, pages 704–715, 2002.

[Cac03] Thierry Cachat. Higher Order Pushdown Automata, the Caucal Hierarchy of Graphs and Parity Games. In *ICALP*, volume 2719 of LNCS, pages 556–569, 2003.

[Car05] Arnaud Carayol. Regular Sets of Higher-Order Pushdown Stacks. In *MFCS*, volume 3618 of LNCS, pages 168–179. Springer, 2005.

[Car06] Arnaud Carayol. *Automates infinis, logiques et langages*. PhD thesis, Universitè de Rennes 1, 2006.

[Cau02] Didier Caucal. On infinite graphs having a decidable monadic theory. In *MFCS*, volume 2420 of LNCS, pages 165–176. Springer, 2002.

[CHM+08] Arnaud Carayol, Matthew Hague, Antoine Meyer, C.-H. Luke Ong, and Olivier Serre. Winning Regions of Higher-Order Pushdown Games. In *LICS*, pages 193–204, 2008.

[Chu63] Alonzo Church. Logic, arithmetic, and automata. In *Proceedings of the international congress of mathematicians*, pages 23–35, Stockholm, 1963. Inst. Mittag-Leffler, Djursholm.

[CS08] Arnaud Carayol and Michaela Slaats. Positional strategies for higher-order pushdown parity games. In *MFCS*, volume 5162 of LNCS, pages 217–228. Springer, 2008.

[CW03] Arnaud Carayol and Stefan Wöhrle. The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In *FSTTCS*, volume 2914 of LNCS, pages 112–123. Springer, 2003.

[CW07] Thierry Cachat and Igor Walukiewicz. The Complexity of Games on Higher Order Pushdown Automata. *CoRR*, abs/0705.0262:1–12, 2007.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree Automata, Mu-Calculus and Determinacy. In *FoCS*, pages 368–377, 1991.

[Fra05a] Séverine Fratani. *Automates à piles de piles ... de piles*. PhD thesis, Université Bordeaux 1, 2005.

[Fra05b] Séverine Fratani. Regular sets over tree structures. http://www.labri.fr/perso/fratani/, 2005.

[GS53] David Gale and Frank M. Stewart. Infinite Games with Perfect Information. In *Contributions to the Theory of Games*, volume 2, pages 245–266. Ann. of Math. Stud, 1953.

[GTW01] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of LNCS. Springer, 2001.

[HMOS08] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible Pushdown Automata and Recursion Schemes. In *LICS*, pages 452–461. IEEE Computer Society, 2008.

[HO07] Matthew Hague and C.-H. Luke Ong. Symbolic Backwards-Reachability Analysis for Higher-Order Pushdown Systems. In *FoSSaCS*, volume 4423 of LNCS, pages 213–227, 2007.

[HO08] Matthew Hague and C.-H. Luke Ong. Symbolic Backwards-Reachability Analysis for Higher-Order Pushdown Systems. *Logical Methods in Computer Science*, 4(4):1–45, 2008.

[HST09a] Paul Hänsch, Michaela Slaats, and Wolfgang Thomas. Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies. Technical Report AIB-2009-18, RWTH Aachen, sep 2009.

[HST09b] Paul Hänsch, Michaela Slaats, and Wolfgang Thomas. Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies. In *FCT*, volume 5699 of LNCS, pages 181–192. Springer, 2009.

[Jur00] Marcin Jurdzinski. Small Progress Measures for Solving Parity Games. In *STACS*, volume 1770 of LNCS, pages 290–301, 2000.

[KV00] Orna Kupferman and Moshe Y. Vardi. An Automata-Theoretic Approach to Reasoning about Infinite-State Systems. In *CAV*, volume 1855 of LNCS, pages 36–52, 2000.

[Mas76] A. N. Maslov. Multi-level stack automata. *Problems Information Transmission*, 12:38–43, 1976.

[McN66] Robert McNaughton. Testing and Generating Infinite Sequences by a Finite Automaton. In *Information and Control*, volume 9, pages 521–530, 1966.

[MS85] David E. Muller and Paul E. Schupp. The Theory of Ends, Pushdown Automata, and Second-Order Logic. *Theoretical Computer Science*, 37:51–75, 1985.

[Par11] Pawel Parys. Collapse Operation Increases Expressive Power of Deterministic Higher Order Pushdown Automata. In *STACS*, volume 9 of *LIPIcs*, pages 603–614. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Scociety*, 141:1–35, 1969.

[Rab06] Alexander M. Rabinovich. Church Synthesis Problem with Parameters. In *CSL*, volume 4207 of LNCS, pages 546–561. Springer, 2006.

[Rab07] Alexander M. Rabinovich. The Church Synthesis Problem with Parameters. *Logical Methods in Computer Science*, 3(4):1–24, 2007.

[Rab09] Alexander M. Rabinovich. Decidable Extensions of Church's Problem. In *CSL*, volume 5771 of LNCS, pages 424 – 439, 2009.

[Rog87] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MIT press, 1987.

[Ser03] Olivier Serre. Note on Winning Positions on Pushdown Games with Omega-Regular Conditions. *Information Processing Letters*, 85(6):285–291, 2003.

[Sie75] Dirk Siefkes. The recursive sets in certain monadic second order fragments of arithmetic. *Archiv math. Logik*, 17(1):71–80, 1975.

[Tho78] Wolfgang Thomas. The theory of successor with an extra predicate. *Math. Ann.*, 237(2):121–132, 1978.

[Tho97] Wolfgang Thomas. Languages, Automata, and Logic. In *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.

[Tho03] Wolfgang Thomas. Constructing Infinite Graphs with a Decidable MSO-Theory. In *MFCS*, volume 2747 of LNCS, pages 113–124, 2003.

[Var98] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, volume 1443 of LNCS, pages 628–641, 1998.

[Wal96] Igor Walukiewicz. Pushdown Processes: Games and Model Checking. In *CAV*, volume 1102 of LNCS, pages 62–74, 1996.

[WHT03] Nico Wallmeier, Patrick Hütten, and Wolfgang Thomas. Symbolic synthesis of finite-state controllers for request-response specifications. In *CIAA*, volume 2759 of LNCS, pages 113–127. Springer, 2003.

[Wöh05] Stefan Wöhrle. *Decision problems over infinite graphs: Higher-order pushdown systems and synchronized products.* PhD thesis, RWTH Aachen, 2005.

[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.

Index

Curriculum Vitae

Persönliche Daten:

Name:	Slaats
Vorname:	Michaela
Geburtstag:	09. Juni 1982
Geburtsort:	Nettetal
Staatsangehörigkeit:	deutsch

Werdegang:

1988–1992:	Besuch der Städt. Grundschule Boisheim
1992–2001:	Besuch der Städt. Gesamtschule Nettetal
06/2001:	Abitur
10/2001–08/2007:	Studium der Informatik (Nebenfach Biologie) an der RWTH Aachen University
08/2007:	Diplom in Informatik Diplomarbeit: Infinite Games over Higher-Order Pushdown Systems
09/2007-11/2011:	Promotionsstudium in Informatik an der RWTH Aachen University
09/2009-08/2010:	Stipendium DFG-Graduiertenkolleg AlgoSyn
09/2010-11/2011:	Wissenschaftliche Angestellte am Lehrstuhl für Informatik 7, RWTH Aachen University