Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik VII

Diploma Thesis

Infinite Games over Higher-Order Pushdown Systems

Michaela Slaats

Supervisors: Dr. A. Carayol, Prof. Dr. Dr.h.c. W. Thomas

Abstract

In this thesis we deal with games over infinite graphs with regular winning conditions. A well studied family of such games are the pushdown games. An important result for these games is that the winning region can be described by regular sets of configurations. We extend this result to games defined by higher-order pushdown systems. The higher-order pushdown systems extend the usual pushdown systems by the use of higher-order stacks which are stacks of stacks of stacks etc.. We concentrate in this thesis just on level 2 stacks that are stacks that contain a sequence of level 1 stacks but the results can easily be lifted to level n. The operations to manipulate those stacks are the simple level 1 operations *push* and *pop* and also operations that can copy and destroy the topmost level 1 stacks inside the level 2 stack. For the definition of regularity over higher-order pushdown stacks we use a technique recently developed by Carayol in [Car05]. We will present an algorithm to compute the winning regions of reachability and parity games over higher-order pushdown graphs.

Zusammenfassung

In dieser Arbeit beschäftigen wir uns mit Spielen auf unendlichen Graphen mit regulären Gewinnbedingungen. Eine erste, gut untersuchte Familie solcher Spiele sind die Kellerspiele. Ein wichtiges Ergebnis bezüglich dieser Spiele ist, dass die Gewinnregionen durch reguläre Mengen von Konfigurationen beschrieben werden können. Wir erweitern dieses Ergebnis auf Spiele, die durch eine Erweiterung des Modells der Kellersysteme auf Kellersysteme höhere Ordnung definiert werden. Die Kellersysteme höherer Ordnung arbeiten über Kellern höherer Ordnung, die aus Kellern von Kellern von Kellern u.s.w. bestehen. Wir konzentrieren und hier nur auf Keller von Niveau 2 aber die Ergebnisse können leicht auf jedes Niveau n erweitert werden. Die Keller von Niveau 2 bestehen aus einer Sequenz von Kellern von Niveau 1. Die Operationen, um einen solchen Keller zu manipulieren, bestehen aus den *push* und *pop* Operationen, die von Niveau 1 bekannt sind und zusätzlich kann der oberste Keller von Niveau 1 in einem Niveau 2 Keller kopiert und gelöscht werden. Um über Kellern höherer Ordnung Regularität zu definieren, haben wir eine erst kürzlich von Carayol entwickelte Technik benutzt. Wir stellen hier einen Algorithmus vor, der die Gewinnregionen von Erreichbarkeits- und Paritätsspielen über Graphen, definiert durch Kellersysteme höhere Ordnung, berechnet.

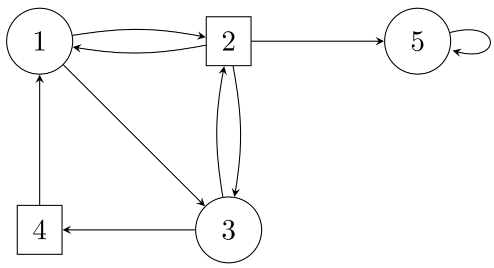
Contents

1 Introduction

1.1 Motivation

In this work we study infinite games over higher order pushdown systems. An infinite *game* can have different forms. In general a game is determined by a pair (G, φ) where G is the definition of the *game graph* and φ is the *winning condition*. Intuitively a game is played between two players, we call them player 0 and player 1. The game graph G also called the *arena* is a graph with a set of vertices Q that is partitioned into two sets each belonging to one player. A *play* is an infinite path through G starting from a designated vertex and proceeds always between the two players. The current vertex decides whose turn it is depending if it belongs to player 0 or 1. Both players have perfect information about the play that means they know the current position and the past of the play, furthermore there is no randomness in the game. Player 0 *wins* a play if the sequence of vertices visited in the play fulfills the winning condition φ. An example of a winning condition is that at least one vertex of a set F of vertices is reached in the game. This kind of games are called *reachability games*. Another kind of games where the infiniteness of the game comes into account are *parity games*. In a game with parity winning condition every vertex gets a color by a function Ω that is a value in $\{1, \ldots, n\}$ for a fixed $n \in \mathbb{N}$. The winning condition for a parity game is that the maximal color that in seen infinitely often during the infinite game is even. The *winning region* of a player contains those vertices from which he can always win independently of the moves of his opponent. A *winning strategy* is a specification of the behavior so that a player wins if he acts according to the strategy.

Example 1. *Lets consider the following game graph where the* ◯*-nodes belong to player 0 and the* □*-nodes belong to player 1.*



If we take as winning condition the reachability set $F = \{2, 4\}$ *then the winning region of player 0 consists of the vertices* $1, 2, 3, 4$ *and the winning region of player 1 is the vertex* 5*. A winning strategy of player 0 is to go from* 1 *to* 2 *and from* 3 *to* 4*.*

For the case that we consider a parity game and take as colors the names of the vertices the winning region of player 0 is the set $\{1, 3, 4\}$ *with the winning strategy to go from* 1 *to* 3 *and from* 3 *to* 4*. In this case we see the nodes* $1, 3$ *and* 4 *infinitely often and so the maximal infinitely often visited color is even. The winning region of player 1 are the vertices* $2, 5$ *with the winning strategy to go from* 2 *to* 5*.*

These games can be used for *model-checking*. The model-checking problem is to verify if a given formula holds in a given state of a given model. A good overview for the application of games for verification is given in [Wal04]. The formulas of the model-checking problem can for example be given by formulas of the *modal μ-calculus*. The modal μ-calculus is a fundamental logic to specify properties of transition systems and it was first introduced by D. Kotzen [Koz83]. The formulas of the μ-calculus have the nice property that they can be translated in linear time into parity conditions for games. From a graph representing the behaviour of the system and a μ-calculus formula, we can construct in linear time a parity game such that player 0 wins the game if and only if the behaviour of the system satisfy the property expressed by the μ-calculus formula.

To model the structure of a model-checking problem *pushdown systems* can be used. With a pushdown system an infinite graph can be constructed, it is called *pushdown graph*. This approach has the advantage that we have a finite representable system that can model infinite behavior. With pushdown graphs it is possible to model recursive procedures with bounded variables. More informations concerning model-checking with pushdown systems can be found in [EHRS00].

The pushdown graphs are defined by the reachable configurations of a pushdown system. To get a game graph the states of the pushdown system have to be partitioned into player 0 and player 1 states and parities have to be given to them. Those games over pushdown graphs are solvable in EXPTIME just like by the model-checking problem, see [Wal96]. In [BEM97] is a reachability analysis for the pushdown systems given and an application to model-checking is presented. Cachat gave in [Cac02a] a uniform construction to compute the winning configurations for reachability and Büchi pushdown games and showed so that the winning region is regular. Serre extend this result to the result that the winning region is regular for all ω-regular winning conditions in [Ser03].

An other problem in terms of model-checking is the *synthesis problem*. This is to construct a system from a given specification. If we see this problem again as a game we have the game given as a specification of a reactive system and player 0 serves as *controller* and player 1 as *environment*. The system has to satisfy the specification no matter what the environment does. If the specification is given as a winning condition in the game then the controller correspond just to the winning strategy in the game against the environment. In [Wal96] it is shown how to compute the winning strategy for pushdown games.

In generally coming back to pushdown graphs it remains to quote a property that allows us to get results over pushdown graphs by a logical approach. The pushdown graphs have the good property that they still have a decidable *MSO-theory*. This and other nice characterisations of the pushdown graphs are shown by Muller and Schupp in [MS85]. A basis of this result is Rabins Tree Theorem [Rab69]. To have a decidable MSO-theory is also a property of the *higher-order pushdown graphs* we like to examine here. This was shown by Carayol and Wöhrle in [CW03]. The higher-order pushdown graphs are defined by *higher-order pushdown systems* and the vertices and edges of the graph are in this case determined by the reachable configurations of the higher-order pushdown system. A higher-order pushdown system is analogical defined as a simple pushdown system but it works on *higher-order pushdown stacks* instead of the simple stacks. Higher-order pushdown stacks are stacks of stacks of stacks and so on. We restrict here just to level 2 but the most results can easily be extended to level n. A level 2 pushdown stack is just a stack that contains a nonempty sequence of level 1 stacks. In higher-order pushdown automata of level 2 we can do the simple *push* and *pop* operations on the topmost level 1 stack inside the level 2 stack and we can also copy and destroy the topmost level 1 stack in the level 2 stack. It was shown that with every level the class of graphs that can be produced by higher-order pushdown systems grows. The so defined hierarchy of graphs is equivalent to the *Caucal Hierarchy* which can be found in [CW03, Wöh05, Car06]. Additionally the higher order pushdown graphs correspond to higher order recursive procedures. A good sum up over this results can be found in [Tho03b]. In [Cac03b] it is shown that the parity games over higher-order pushdown graphs of level n can be solved in n-EXPTIME. Cachat and Walukiewicz showed in [CW07] an n-EXPTIME lower bound for deciding the winner in reachability games on higher-order pushdown graphs of level n.

The goal of this thesis is to show that the winning regions of reachability and parity games over higher-order pushdown graphs are regular in the sense of [Car05]. This result already follows from results over the MSO-theory which can be found in [Car06, Fra05b]. This approach provides an algorithm to compute the winning region but it is not a direct way to compute it and its complexity rest on the nesting of the quantifiers in the formula. To show the regularity of the winning region of games over higher-order pushdown graphs we first of all had to define what it means for a set of higher-order pushdown stacks to be regular. For that we used the definition of Carayol in [Car05] that tries just to extend the definition of regularity of words in a natural way. This definition of regularity over higher-order pushdown stacks provided a good starting point for this thesis.

1.2 Contribution of the Thesis

In this thesis we want to show by an automata based approach that the winning regions of reachability and parity games over higher-order pushdown graphs are regular, i.e. we restrict here to level 2 pushdown graphs but the results can easily be lifted to level n. For that we first define the model of higher-order pushdown automata and what it means in this framework to be regular.

It is already known that the winning region of games over higher-order pushdown automata is regular by an approach which bases on MSO-theory. However the use of MSO leads to an effective procedure that translates a formula into an automaton but this can just be done with a bad complexity. We want here to get an automata based approach to achieve a procedure with a better complexity.

The higher-order pushdown automata work with higher-order pushdown stacks that are defined as stacks of stacks. The automata can put elements of the stack alphabet into the the topmost level 1 stack and also deletes the topmost stack symbols again. Additionally it has the ability to copy the topmost stack and also to redo this by deleting it again. We define here this operations in a symmetric way that means that we have to add the symbol to the pop that we want to pop, for example pop_a instead of just *pop*. According to that the destroying of the topmost stack is just allowed if the two topmost stacks are equal. We need this restricted definition to get later a good definition of regularity that relies on some properties of this symmetric operations. Additionally to this operations we have the possibility by tests to check if the topmost level 1 or level 2 stack is empty, by this we avoid the definition of an additional stack symbol. Summing up we have the following set of operations over a stack alphabet Γ: $\{push_a, pop_a \mid a \in \Gamma\} \cup \{copy_1, \overline{copy_1}, T_{[\ ]_1}, T_{[\ ]_2}\}$. Together with the operations we define the set of instructions $\Gamma_2 = \{a, \bar{a} \mid a \in \Gamma\} \cup \{1, \bar{1}, \bot_1, \bot_2\}$ for an alphabet Γ that among other things act as a symbolic representation of the operations.

The definition of regularity of higher-order pushdown stacks is the following. We define a set of stacks to be regular if it can be produced by the application of a regular set of sequences of symmetric operations to the empty level 2 stack. This definition was introduced by Carayol in [Car05]. We also define some different automata models to accept those regular set of stacks. These automata have as alphabet the set of instructions Γ_2.

For the definition of the game graph we use regular sets of stacks. We wanted to use the most general definition and so we decided to have as vertices of the game just the stacks and not the configurations of a higher-order pushdown system that are composed of a stack and a state. So the graph is just defined by a regular set of stacks and a set of instruction sequences to define the edge relation and so the connection between the stacks. The partitioning is also given by regular sets of stacks.

To compute the winning regions of the higher-order pushdown games we use alternating automata over Γ_2. We use the alternation to model the behavior of player 1 and nondeterminism to model the behavior of player 0. To show the regularity of the winning regions we show in a second step that this kind of automata still accepts only regular sets of higher-order pushdown stacks. For the case of reachability games we can get this result from [Car06].

For parity games we have to change the definition of the alternating automata a little to imply the parity condition and so need to redo the proof that those automata still accept only regular sets of stacks. In one step of the proof we use a result of Vardi [Var98] that claims the equivalence of alternating two-way parity tree automata and nondeterministic one-way tree automata which run both over infinite complete trees.

The complexity of computing the winning regions of a higher-order pushdown game is k-exponential for level k of the pushdown game.

2 Basic principles

In this section we introduce the basic knowledge for this diploma thesis. The notations are according to the PhD of Carayol [Car06]. We first introduce the higher-order pushdown stacks of level 2 and show how they can be modified. Then we look at some of their properties and introduce higher-order pushdown automata of level 2 that work over these stacks. After that we give a definition of regularity for the stacks of level 2 and define automata models to recognize regular set of level 2 stacks. At the end of this chapter we make a short excursion to the monadic second-order theory that forestall our results.

2.1 Higher-order pushdown systems of level 2

The higher-order pushdown systems of level 2 work similar to the usual pushdown systems but they use as storage not simple level 1 stacks but stacks of level 2. The higher-order pushdown stacks of level 2 are stacks of stacks, i.e. they are stacks that contain as elements stacks of level 1.

We introduce the higher-order pushdown stacks of level 2 in section 2.1.1 and give in section 2.1.2 the operations to modify them. In section 2.1.3 we introduce instructions as a symbolic representation of these operations. In section 2.1.4 we characterize a special kind of instruction sequences that have the property that they can build up every stack in a unique way starting in the empty level 2 stack. In the last part 2.1.5 of this section 2.1 the higher-order pushdown automata are introduced.

2.1.1 Stacks of level 2

A *stack of level 1* over a finite alphabet Γ corresponds to a word of Γ^* and we write $[abc]_1$ for the stack that is conform to the word abc. The empty stack of level 1 corresponds to the empty word ε and is written as $[\,]_1$. The set of all stacks of level 1 over an alphabet Γ is written as $Stacks_1(\Gamma) = \Gamma^*$. The stacks are defined such that the topmost symbol of the stack is the last letter of the according word. We define a partial function $top : Stacks_1(\Gamma) \to \Gamma$ that is defined for every stack $w = [a_1 \ldots a_n]_1$ with $n \geq 0$ like follows:

$$top(w) = \begin{cases} a_n & \text{, if } n > 0 \\ \text{not defined} & \text{if } w = [\,]_1 \end{cases} .$$

A *stack of level 2* is a stack of stacks of level 1, i.e. of a non empty sequence of stacks of level 1. By this we have that $s = [s_1 \ldots s_n]_2$ is the level 2 stack that is composed of the sequence $s_1, \ldots, s_n$ of level 1 stacks. The empty stack of level 2 is the stack of level 2 that just contains the empty stack of level 1, i.e. $[\,]_2 = [[\,]_1]_2$. The set of all stacks of level 2 is defined by $Stacks_2(\Gamma) = (Stacks_1(\Gamma))^+ = (\Gamma^*)^+$.

The following example should show how such a stack of level 2 looks like. So let $s = [[abcabc]_1\,[aaaa]_1\,[bcbacba]_1]_2$ be shown in the illustration 1 in a more graphically way.

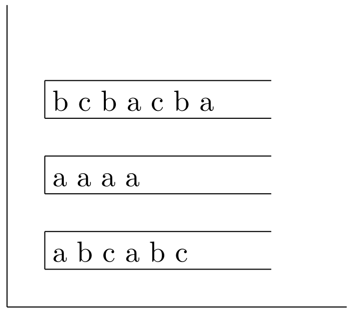


Figure 1: The illustration shows the stack $s = \left[[abcabc]_1 \, [aaaa]_1 \, [bcbacba]_1\right]_2$.

Additionally we define $Stacks_{\leq 2}(\Gamma) = Stacks_1(\Gamma) \cup Stacks_2(\Gamma)$. To get the topmost level 1 stack of a level 2 stack we add to the definition of *top* the following function $top_1 : Stacks_2(\Gamma) \to Stacks_1(\Gamma)$ defined by $top_1([s_1 \dots s_n]_2) = s_n$. To get the topmost symbol of the level 1 stack of a level 2 stack *top* is extended by $top([s_1 \dots s_n]_2) = top(s_n)$. For example we have $top([abc]_1) = c$, $top_1\left([[aaa]_1 \, [acb]_1]_2\right) = [acb]_1$ and $top\left([[aaa]_1 \, [bab]_1 \, [acb]_1]_2\right) = top([acb]_1) = b$.

2.1.2 Operations on stacks of level 2

In the following we introduce operations that are defined on stacks of level 2 that work over an alphabet Γ. An operation θ is a partial function from $Stacks_{\leq 2}$ to $Stacks_{\leq 2}$ where the level of the stack remains the same. The level of an operation θ, written as $|\theta|$, is defined as the smallest k such that $Dom(\theta) \cap Stacks_k(\Gamma) \neq \emptyset$. For the empty function $\emptyset$ the level is not defined and is set by convention to infinity, i.e. $|\theta| = +\infty$. For the concatenation of two operations θ and θ' holds that $|\theta \times \theta'| \geq \max\{|\theta|, |\theta'|\}$. To apply an operation θ of level 1 onto a stack of level 2 it holds that $\theta([s_1 \dots s_n]_2) = [s_1 \dots \theta(s_n)]_2$.

On level 1 we define the operations $push_x$ and pop_x for all $x \in \Gamma$. The operation $push_x$ adds the symbol x as topmost symbol to the stack and the operation pop_x deletes the topmost symbol of the stack if it is an x, otherwise it is not defined. Formal the operations are defined for every stack $s = [a_1 \dots a_n]_1$ of level 1 and for every $x \in \Gamma$ like follows:

$$
\begin{aligned}
push_x([s]_1) &= [sx]_1 \\
pop_x([a_1 \dots a_n]_1) &= \begin{cases} [a_1 \dots a_{n-1}] & \text{if } a_n = x \\ \text{not defined} & \text{otherwise.} \end{cases}
\end{aligned}
$$

Instead of the operation pop_x often the operation *pop* is defined that just deletes the topmost stack symbol if the stack is not empty. This does not change the expressivity of the model but we use here the pop_x because we need the symmetry of the operations (for that see section 2.1.4 on reduced sequences).

On level 2 there are the further operations $copy_1$, $destr_1$ and $\overline{copy_1}$. The operation $copy_1$ copies the topmost level 1 stack and the operation $destr_1$ deletes the topmost level 1 stack if there are at least 2 stacks in the level 2 stack. The operation $\overline{copy_1}$ is symmetric to the operation $copy_1$ and deletes the topmost level 1 stack but only if it is equivalent to the topmost but one level 1 stack. So this operation can only be applied if there are at least two stacks in the level 2 stack. The formal definition of the operations $copy_1$, $destr_1$ and $\overline{copy_1}$ looks like follows where $n \geq 1$ and $s_1, \ldots, s_{n+1}$ are stacks of level 1:

$$\begin{array}{lcl} copy_1\left([s_1 \ldots s_n]_2\right) & = & [s_1 \ldots s_n s_n]_2 \\ destr_1([s_1 \ldots s_n s_{n+1}]) & = & [s_1 \ldots s_n]_2 \\ \overline{copy_1}\left([s_1 \ldots s_n s_n]_2\right) & = & [s_1 \ldots s_n]_2 \,. \end{array}$$

The operations $destr_1$ and $\overline{copy_1}$ have the same expressivity like shown in [Car06] and [Wöh05]. In the rest of this thesis we only need the operation $\overline{copy_1}$ because we need the symmetry to define later regularity on higher-order pushdown stacks.

Example 2. *Let $\Gamma = \{a, b, c\}$ and $s = [[aa]\,[bc]]_2$ be a stack of level 2 then the application of different operations looks as follows:*

$$\begin{array}{llllll} & [[aa]\,[bc]]_2 & \overset{push_b}{\longrightarrow} & [[aa]\,[bcb]]_2 & \overset{copy_1}{\longrightarrow} & [[aa]\,[bcb]\,[bcb]]_2 \\ \overset{pop_b}{\longrightarrow} & [[aa]\,[bcb]\,[bc]]_2 & \overset{push_b}{\longrightarrow} & [[aa]\,[bcb]\,[bcb]]_2 & \overset{\overline{copy_1}}{\longrightarrow} & [[aa]\,[bcb]]_2 \\ \overset{destr_1}{\longrightarrow} & [[aa]]_2 & \overset{pop_a}{\longrightarrow} & [[a]]_2 & \overset{pop_a}{\longrightarrow} & [[\,]]_2 \end{array}$$

Beside the operations that has been introduced here to operate on stacks there are additionally test on stacks to test on the particular level $k \in [1, 2]$ if the stack is empty:

$$T_{[\,]_k}\left([s_1 \ldots s_n]_k\right) = \begin{cases} [\,]_k & \text{, if } [s_1 \ldots s_n]_k = [\,]_k \\ \text{not defined} & \text{, otherwise.} \end{cases}$$

The set of operations over stacks of level 1 respectively 2 over an alphabet Γ is defined like follows as Ops_1 respectively Ops_2:

$$\begin{array}{lcl} Ops_1 & = & \{pop_x, push_x | x \in \Gamma\} \cup \left\{T_{[\,]_1}\right\} \\ Ops_2 & = & Ops_1 \cup \{copy_1, \overline{copy_1}\} \cup \left\{T_{[\,]_2}\right\} \end{array}$$

For each level $k \geq 1$ a submonoid $Ops^\star$ can be defines as:

$$Ops_k^\star = \{\theta = \theta_1 \cdots \theta_n \mid n \geq 1, \forall i \in [1, n]\,, \theta_i \in Ops_k \text{ and } |\theta| \geq k\}\,.$$

The identity element of $Ops^\star$ is the identity function Id_k that if applied to a stack just gives back the same stack. The empty function $\emptyset$ is the absorbing element.

2.1.3 Instructions on stacks of level 2

In parallel to the operations on stacks we define instructions on stacks. For every operation we have an according symbol and so we define an alphabet of operations. In principle the instructions are just some kind of abbreviation to get a shorter representation for the following notations and automata models. They work as symbols for the operations.

We define the set of *instructions* Γ_2 for level 2 in bijection to Ops_2 by:

$$\begin{aligned} \Gamma_1 &= \Gamma \cup \overline{\Gamma} \cup \{\bot_1\} \\ \Gamma_2 &= \Gamma_1 \cup \left\{1, \overline{1}\right\} \cup \{\bot_2\}, \end{aligned}$$

where $\overline{\Gamma}$ is a set disjoint from Γ but in bijection with Γ. This means that there exists for every element $x \in \Gamma$ an according element $\overline{x} \in \overline{\Gamma}$. We introduce also for $\Gamma_k,\ k \in [1,2]$ the notations $\Gamma_k^0 = \Gamma \cup \overline{\Gamma} \cup \{l, \overline{l} \mid l \in [1, k-1]\}$ for the instructions that are according to the operations that explicit change the stack and $\Gamma_k^T = \{\bot_l \mid l \in [1,k]\}$ for the instructions that are according to the tests of emptiness of the topmost stack.

To get the connection between the operations Ops_2 and the instructions Γ_2 we define the bijective function $\mathcal{R} : \Gamma_2 \to Ops_2$ by:

$$\begin{array}{lll} \mathcal{R}(x) = push_x & \mathcal{R}_2(\overline{x}) = pop_x & \text{for } x \in \Gamma \\ \mathcal{R}(1) = copy_1 & \mathcal{R}_2\left(\overline{1}\right) = \overline{copy_1} & \\ \mathcal{R}(\bot_l) = T_{[\,]_l} & & \text{for } l \in [1,2]\,. \end{array}$$

For example the interpretation of the sequence $abc1\overline{c}b\bot_2 \in \Gamma_2^*$ of instructions by $\mathcal{R}$ is the sequence of operations $push_a push_b push_c copy_1 pop_c push_b T_{[\,]_2}$.

The operation $^{-}$ can also be enlarged to sequences of instructions. By this the instruction sequence $\overline{\rho}$ in principle undoes the sequence ρ. For this it has to be remarked that $pop_x push_x$ is not just ID_2 but also a test if the topmost stack symbol is x and that $\mathcal{R}(\overline{x}x)$ is only defined in this case. For this enlargement we now first define for all $x \in \Gamma_2^O$ that $\overline{\overline{x}} = x$ and for all $t \in \Gamma_2^T$ that $\overline{t} = t$. Then we have for all $\rho = \rho_1, \ldots, \rho_n \in \Gamma_2^*$ that $\overline{\rho} = \overline{\rho_n}, \ldots, \overline{\rho_1}$.

We define also a partial inverse function for the operations θ in $Ops_2^\star$ that is noted as θ^{-1}. It holds for all $x \in \Gamma$: $(push_x)^{-1} = pop_x$, $(pop_x)^{-1} = push_x$, $(copy_1)^{-1} = \overline{copy_1}$ and $(\overline{copy_1})^{-1} = copy_1$. But this function is not really the inverse for the monoid of $Ops_2^\star$. For this it suffice to remark that $pop_x push_x = Id_1$ does not hold in general.

Lemma 2.1.1. *For every $\rho \in \Gamma_2^*$ holds that $\mathcal{R}(\overline{\rho}) = \mathcal{R}(\rho)^{-1}$.*

Proof. It holds for all $\gamma \in \Gamma_2$ that $\mathcal{R}(\overline{\rho}) = \mathcal{R}(\rho)^{-1}$. The above claimed property follows by induction over the length of ρ. □

Example 3. *This is an illustration of the lemma 2.1.1.*

$$\begin{array}{ll} \rho & = abc1\overline{c}ba1\overline{a}\overline{b}cd \\ \overline{\rho} & = \overline{d}\overline{c}ba\overline{1}\overline{a}\overline{b}c\overline{1}\overline{c}\overline{b}\overline{a} \\ \mathcal{R}(\overline{\rho}) & = pop_d\, pop_c\, push_b\, push_a\, \overline{copy_1}\, pop_a\, pop_b\, push_c\, \overline{copy_1}\, pop_c\, pop_b\, pop_a \\ \mathcal{R}(\rho) & = push_a\, push_b\, push_c\, copy_1\, pop_c\, push_b\, push_a\, copy_1\, pop_a\, pop_b\, push_c\, push_d \\ \mathcal{R}(\rho)^{-1} & = pop_d\, pop_c\, push_b\, push_a\, \overline{copy_1}\, pop_a\, pop_b\, push_c\, \overline{copy_1}\, pop_c\, pop_b\, pop_a \end{array}$$

The following lemma says that if we delete the 1's in an instruction sequence ρ that does not contain $\overline{1}$ and $\bot_1$ then we get if we apply this modified instruction sequence $\tilde{\rho}$ onto the topmost level 1 stack of a stack s the same level 1 stack as if we apply ρ on s and take there the topmost level 1 stack.

Lemma 2.1.2. *For each sequence of instructions ρ in $\left(\Gamma_2 \setminus \left\{\overline{1}, \bot_2\right\}\right)^*$ and for every stack $s \in Stacks_2(\Gamma)$ holds that $\mathcal{R}(\rho)(s)$ is defined iff $\mathcal{R}(\tilde{\rho})(top_2(s))$ is defined where $\tilde{\rho} \in \Gamma_1^*$ is the sequence that is generated by deleting of all 1's of the sequence ρ. Additionally it holds that $top_2(\mathcal{R}(\rho)(s)) = \mathcal{R}(\tilde{\rho})(top_2(s))$ if both stacks are defined.*

Proof. The claim follows by induction over the length of ρ. □

Example 4. *We give here an example for lemma 2.1.2:*

$$\begin{aligned}
\rho &= abc1\bar{c}ba1\bar{a}\bar{b}cd \\
\mathcal{R}(\rho)([\ ]_2) &= [[abc], [abba], [abcd]]_2 \\
\tilde{\rho} &= ab\underline{c\bar{c}}\underline{b\underline{a\bar{a}}\bar{b}}cd = abcd \\
\mathcal{R}(\tilde{\rho})([\ ]_2) &= [[abcd]]_2 \\
s &= [[aa], [aba]]_2 \\
\mathcal{R}(\rho)(s) &= [[aa], [abaabc], [abaabba], [abaabcd]] \\
top(\mathcal{R}(\rho)(s)) &= [abaabcd] \\
\mathcal{R}(\tilde{\rho})(top_2(s)) &= [abaabcd]
\end{aligned}$$

2.1.4 Reduced sequences of instructions

To get a unique representation of a stack by an instruction sequence we need to define a special property on instruction sequences. This property we call reduced and it means that we do not allow "loops" in the instruction sequence, i.e. a reduced sequence of instruction does not allow to get the same stack more than once if we cut of the sequence at different steps. More formal this means that a reduced sequence ρ has the property that for every stack $s \in Stacks_2(\Gamma)$ there do not exist two sequences $\rho' \neq \rho'' \in \Gamma_2^*$ so that $\rho' \sqsubseteq \rho$, $\rho'' \sqsubseteq \rho$ and $\mathcal{R}(\rho')(s) = \mathcal{R}(\rho'')(s)$. Instead of this global property one could also check the following local property.

Definition 2.1.3. A sequence $\rho \in \Gamma_2^*$ is called *reduced*, if ρ neither contains a test $\bot_2$ nor a subsequence of the form $\gamma\overline{\gamma}$ for $\gamma \in \Gamma_2^O$.

By this local conditions it is easy to define a relation $\rightarrow_2 \subseteq \Gamma_2^* \times \Gamma_2^*$ that transforms a non-reduced sequence into a reduced sequence by the following rules:

$$\left\{(t, \epsilon), (\gamma\overline{\gamma}, \epsilon) \mid t \in \Gamma_2^T \text{ and } \gamma \in \Gamma_2^O\right\}$$

For every $\rho \in \Gamma_2^*$ exists a unique normal form $\rho^{\downarrow}$ that is exactly the reduced instruction sequence belonging to ρ because $\rightarrow_2$ is confluent and noetherian.

Example 5. *Let the non-reduced sequence* $\rho = aba\bar{a}1\bar{b}a\bar{a}b\bar{1}\bar{b}ab1a\bar{a}$ *be given. The according reduced sequence is* $\rho^{\downarrow} = aab1$.

$$\begin{aligned}\rho \;=\;& ab\underline{a\bar{a}}1\bar{b}a\bar{a}b\bar{1}\bar{b}ab1a\bar{a} \rightarrow_2 ab1\bar{b}\underline{a\bar{a}}b\bar{1}\bar{b}ab1a\bar{a} \rightarrow_2 ab1\underline{\bar{b}b}\bar{1}\bar{b}ab1a\bar{a} \rightarrow_2 \\ & ab\underline{1\bar{1}}\bar{b}ab1a\bar{a} \rightarrow_2 a\underline{b\bar{b}}ab1a\bar{a} \rightarrow_2 aab1\underline{a\bar{a}} \rightarrow_2 aab1 = \rho^{\downarrow}\end{aligned}$$

Remark that the relation $\rightarrow_2$ does not preserve the interpretation by $\mathcal{R}$ because in general we have that $\mathcal{R}(\rho)$ is different to $\mathcal{R}(\rho^{\downarrow})$. For example let $\rho = a\bar{b}b1$ then we have $\rho^{\downarrow} = a1$ if we take $s = [[a]]_2$ then we get that $\mathcal{R}(\rho)(s)$ is not defined and $\mathcal{R}(\rho^{\downarrow})(s) = [[aa][aa]]_2$.

Now we want to claim an important property of the reduced sequences and one feature that follows from it.

Theorem 2.1.4. *For all stacks* $u, v \in Stacks_2(\Gamma)$ *there exists an unique reduced sequence* $\rho_{u,v} \in \Gamma_2^*$ *so that* $v = \mathcal{R}(\rho_{u,v})(u)$.

The proof can be found in [Car06] page 82. A direct conclusion of this theorem is that for every stack $s \in Stacks_2(\Gamma)$ there exists an unique reduced sequence $\rho_s \in \Gamma_2^*$ that constructs s starting from the empty stack.

Definition 2.1.5. For every stack $s \in Stacks_2(\Gamma)$ the *reduced sequence of s* is the unique reduced sequence $\rho_s \in \Gamma_2^*$ so that $s = \mathcal{R}(\rho_s)([\,]_2)$.

Additionally we define for every stack $s \in Stacks_2(\Gamma)$, $Last(s) \in \Gamma_2^O \cup \{k, \varepsilon\}$ by:

$$Last(s) = \begin{cases} \rho_s(|\rho_s|) & \text{if } s \neq [\;]_2, \\ \varepsilon & \text{otherwise.} \end{cases}$$

2.1.5 Higher-order pushdown automata and their languages

In this subsection we shortly define higher-order pushdown automata for level 2 and their languages. We do not spent much time with them because we are more interested in the stacks as in the word languages that can be accepted with this kind of automata.

Definition 2.1.6. A *higher-order pushdown automata $\mathcal{A}$ of level 2* over the instructions Ops_2 is given by the tuple $(Q, \Sigma, \Gamma, \tau, I, F, \Delta)$ where

- Q is a finite set of states,
- Σ is the finite word alphabet,
- Γ is the finite stack alphabet,
- $\tau \in \Sigma$ is an extra symbol in Σ for silent transition so that τ does not count for the word that is accepted,
- $I \subseteq Q$ and $F \subseteq Q$ are the sets of initial respectively final states,
- and $\Delta \subseteq Q \times \Sigma \times Ops_2^{\star} \times Q$ is the transition relation.

A transition $(p, a, \theta, q) \in \Delta$ is written as $p \xrightarrow{a} (q, \theta)$. A configuration of $\mathcal{A}$ is a tuple in $Q \times Stacks_2(\Gamma)$ and the initial respectively final configurations are in $I \times Stacks_2(\Gamma)$ respectively $F \times Stacks_2(\Gamma)$. For every $a \in \Sigma$ there is a relation $\xrightarrow[\mathcal{A}]{a}$ over the configurations induced by:

$$(p, w) \xrightarrow[\mathcal{A}]{a} (q, w') \Leftrightarrow \exists (p, a, \theta, q) \in \Delta, w' = \theta(w).$$

This relation induces for all $u \in (\Sigma \backslash \{\tau\})^*$ a relation $\overset{u}{\underset{\mathcal{A}}{\Rightarrow}}$ defined by:

$$\begin{aligned} \overset{\varepsilon}{\underset{\mathcal{A}}{\Rightarrow}} &= (\xrightarrow[\mathcal{A}]{\tau})^* \\ \overset{ua}{\underset{\mathcal{A}}{\Rightarrow}} &= \overset{u}{\underset{\mathcal{A}}{\Rightarrow}} \cdot \xrightarrow[\mathcal{A}]{a} \cdot (\overset{\tau}{\underset{\mathcal{A}}{\Rightarrow}})^* \end{aligned}$$

where $u \in (\Sigma \backslash \{\tau\})^*$ and $a \in \Sigma \backslash \{\tau\}$.

The language that is accepted by the automaton $\mathcal{A}$ is noted as $\mathcal{L}(\mathcal{A})$ and defined as the set of words $u \in (\Sigma \backslash \{\tau\})^*$ so that there exists $i \in I$, $f \in F$ and $s \in Stacks_2(\Gamma)$ and $(i, [\ ]_2) \overset{u}{\underset{\mathcal{A}}{\Rightarrow}} (f, s)$.

Example 6. *Now we give as an example for a pushdown automata of level 2 the following automata* $\mathcal{A} = (Q, \Sigma, \Gamma, \tau, I, F, \Delta)$ *with* $\Sigma = \{a, b, \$, \tau\}$ *and* $\Gamma = \{a, b\}$ *that accepts the language* $L = \{w\$w \mid w \in \{a, b\}^*\}$.

We have $Q = \{i, p, q, f\}$, $I = \{i\}$ *and* $F = \{f\}$. *The set of transitions* Δ *is given by:*

$$\begin{array}{lll} i \xrightarrow{a} (i, push_a) & i \xrightarrow{b} (i, push_b) & i \xrightarrow{\$} (p, copy_1) \\ p \xrightarrow{\tau} (p, copy_1 pop_a) & p \xrightarrow{\tau} (p, copy_1 pop_b) & p \xrightarrow{\tau} (q, \bot_1) \\ q \xrightarrow{a} (q, push_a \overline{copy_1}) & q \xrightarrow{b} (q, push_b \overline{copy_1}) & q \xrightarrow{\tau} (f, \overline{copy_1}) \end{array}$$

The idea of this automaton is the following. First the automaton reads in state i the input word until the \$ and pushes it on the stack. If the \$ is reached we go into state p and copy the level 1 stack. After that in every step the topmost level 1 stack is copied and the topmost 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 τ *and no letters of the input word are read. Then in state q we read again an input letter, put it on 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 go to the final state.*

Now we give an example for the work of the automaton by showing it for the input word abb\$abb:

$$(i, [\ ]_2) \xrightarrow[\mathcal{A}]{a} (i, [[a]]_2) \xrightarrow[\mathcal{A}]{bb} (i, [[abb]]_2) \xrightarrow[\mathcal{A}]{\$} (p, [[abb][abb]]_2) \xrightarrow[\mathcal{A}]{\tau} (p, [[abb][abb][ab]]_2)$$

$$\xrightarrow[\mathcal{A}]{\tau^3} (q, [[abb][abb][ab][a][\ ]]_2) \xrightarrow[\mathcal{A}]{a} (q, [[abb][abb][ab][a]]_2) \xrightarrow[\mathcal{A}]{b} (q, [[abb][abb][ab]]_2)$$

$$\xrightarrow[\mathcal{A}]{b} (q, [[abb][abb]]_2) \xrightarrow[\mathcal{A}]{\tau} (f, [[abb]]_2)$$

2.2 Regular sets of higher-order pushdown stacks

In this section we want to introduce regular set of stacks of level 2. We know how regularity is defined for words. But how can this definition be enlarged to higher-order pushdown stacks in a natural way? A main difference between the definitions of regularity over words and the regularity over higher-order pushdown stacks is that by words we have a unique way to construct and represent the word but by higher-order pushdown stacks there a several ways to produce a stack by operation sequences starting in the empty level 2 stack. For example $push_a([\ ]_2) = push_a push_b pop_b([\ ]_2) = push_b copy_1 \overline{copy_1} pop_b push_a([\ ]_2) = [[a]]_2$. We will allow this not unique representation and this is one main point in the definition of regularity. If we restrict the operation sequence that builds up a stack by reduced sequences we get a normal form because for every stack the according reduced sequence is unique. This property holds only for the set of symmetric operation Ops_2 and so we define the regular sets of stacks just over the set of symmetric operation Ops_2.

The definition of regularity of higher-order stacks that is used here was introduced by Carayol in [Car05]. A nice property of those regular sets of stacks is that they form a Boolean algebra. This was also shown by Carayol in [Car06].

Regularity on level 1

The definition of regularity over stacks of level 1 is is very natural. If Γ is the stack alphabet then the regularity of the stacks over Γ is defined as the regularity of the free monoid Γ^*. That means the set of regular stack is defined as the ones that are consructable by a regular subset of $Ops_1^\star(\Gamma)$ applied to the empty level 1 stack $[\ ]_1$. This set is therefore defined as $Reg(\Gamma^*)$. This property was found by Büchi in [Büc64] and Benois in [Ben69].

Theorem 2.2.1 ([Büc64],[Ben69])**.** *For every finite alphabet* Γ *it holds that:*

$$Reg(Ops_1^\star(\Gamma))([\ ]_1) = Reg(\Gamma^*).$$

The proof of this theorem can be found in [Car06].

We write $Reg_1(\Gamma)$ for the set of *regular stacks of level 1.*

Regularity on level 2

For level 2 the *set of regular stacks* is defined as the sets of stacks which can be obtained by the application of a regular subset of $Ops_2^\star(\Gamma)$ to the empty level 2 stack $[\ ]_2$ or in other terms by the application of a regular set of instruction sequences applied to $[\ ]_2$. We denote the set of all regular stacks of level 2 by $Reg_2(\Gamma)$, it is formally defined by:

$$\begin{aligned} Reg_2(\Gamma) &= Reg(Ops_2^\star(\Gamma))([\ ]_2) \\ &= \mathcal{R}(Reg(\Gamma_2^*))([\ ]_2). \end{aligned}$$

By the definition of the regularity we can already see that it is strongly connected to the definition of automata over $Ops_2(\Gamma)$ and this can also be seen by the following proposition.

Proposition 2.2.2. For all pushdown automata $\mathcal{A}$ over $Ops_2(\Gamma)$ is the set of stacks of level 2 that appear in a final configuration of $\mathcal{A}$ that is reachable from an initial configuration in some set of $Reg_2(\Gamma)$.

Conversely for all sets R of $Reg_2(\Gamma)$ there exists a pushdown automata over $Ops_2(\Gamma)$ such that R is the set of stacks that appear in some final configuration of $\mathcal{A}$ that is reachable from an initial configuration.

Example 7. *If we look at the automaton $\mathcal{A}$ over $Ops_2(\Gamma)$ of example 6 then it is easy to see that the set of stacks of level 2 that can be reached in a final configuration from the initial configuration $(i, [\ ]_2)$ can be described by the following set R of $Reg(Ops_2^\star)$:*

$$\{push_a, push_b\}^* \cdot copy_1 \cdot (copy_1 \cdot \{pop_a, pop_b\})^* \cdot T_{[\,]_1} \cdot (\{push_a, push_b\} \cdot \overline{copy_1})^* \cdot \overline{copy_1}$$

This representation is not very informative and so we use instead the following finite representation:

$$\{push_a, push_b\}^*([\ ]_2).$$

2.3 Automata over Γ_2

In the following we introduce automata over Γ_2 and alternating automata over Γ_2 as well as some special cases of them. They accept regular sets of stacks of level 2 in a natural way by using the instructions Γ_2 as alphabet. Remark that we have introduced the instructions Γ_2 and divided them into the set of stack transforming instructions Γ_2^O and stack testing instructions Γ_2^T. We use here also the notation $Sing(P)$ for a set P to signify that P has a cardinality of at most 1.

Definition 2.3.1. An *automaton A over* Γ_2 is a tuple (Q, I, F, Δ) where Q is the set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and $\Delta \subseteq Q \times \Gamma_2^O \times Sing(\Gamma_2^T) \times Q$ is the transition relation.

A configuration is a tuple (p, s) in $Q \times Stacks_2(\Gamma)$. By $\mathcal{C}_A = Q \times Stacks_2(\Gamma)$ we denote the set off all configurations of A. A transition $(p, \gamma, T, q) \in \Delta$ is written as $p \xrightarrow{\gamma} q, T$ or as $p \xrightarrow{\gamma} q$ if $T = \emptyset$. If the automaton is in configuration (p, s) and takes the transition $p \xrightarrow{\gamma} q, T \in \Delta$ then it can reach the configuration (q, r) iff the stack $r = \mathcal{R}(\gamma)(s)$ is defined and $r \in Dom(\mathcal{R}(t))$ for all $t \in T$.

Remark that the test, if the stack is empty on some level, is done after the instruction γ is applied on the stack s.

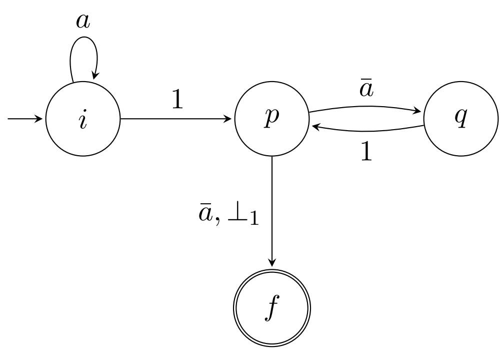
The automaton A induces for every $\gamma \in \Gamma_2^O$ a relation $\xrightarrow[A]{\gamma} \subseteq \mathcal{C}_A \times \mathcal{C}_A$ that is defined for all configurations (p, s) and (q, r) of $\mathcal{C}_A$ by $(p, s) \xrightarrow[A]{\gamma} (q, r)$ if there exists a transition $p \xrightarrow{\gamma} q, T \in \Delta$ so that $r = \mathcal{R}(\gamma)(s)$ and $r \in Dom(\mathcal{R}(t))$ for every $t \in T$.

A computation of A is a finite sequence $(p_0, s_0), \gamma_1, (p_1, s_1), \ldots, (p_{n-1}, s_{n-1}), \gamma_n, (p_n, s_n) \in \mathcal{C}_A(\Gamma_2^O \mathcal{C}_A)^*$ so that for every $l \in [0, n-1]$ holds that $(p_l, s_l) \xrightarrow[A]{\gamma_{l+1}} (p_{l+1}, s_{l+1})$. A computation of A accepts a stack $s \in Stacks_2(\Gamma)$ if $p_0 \in I$, $s_0 = [\ ]_2$, $p_n \in F$ and $s_n = s$. If there exists a computation of A that accepts a stack s we say that A accepts s. We write $\mathcal{S}(A)$ for the set of stacks of $Stacks_2(\Gamma)$ that are accepted by A.

Example 8. *We define now an automaton* $A = (Q, I, F, \Delta)$ *over* Γ_2 *where* $\Gamma = \{a\}$ *so that* A *accepts the stack language* $\{[[a^n][a^{n-1}]\cdots[a][\ ]]_2 \mid n > 0\}$. *The automaton* A *has the state set* $Q = \{i, p, q, f\}$ *with* $I = \{i\}$ *and* $F = \{f\}$. *The transition relation* Δ *is given by the following set of transitions:*

$$i \xrightarrow{a} i \qquad i \xrightarrow{1} p \qquad p \xrightarrow{\bar{a}} q \qquad q \xrightarrow{1} p \qquad p \xrightarrow{\bar{a}} f, \bot_1.$$

We can describe the automaton A *also by the following figure:*



For example the stack $[[aa][a][\ ]]_2$ *is accepted by the following computation:*

$$(i, [\ ]_2) \xrightarrow[A]{a} (i, [[a]]_2) \xrightarrow[A]{a} (i, [[aa]]_2) \xrightarrow[A]{1} (p, [[aa][aa]]_2)$$
$$\xrightarrow[A]{\bar{a}} (q, [[aa][a]]_2) \xrightarrow[A]{1} (p, [[aa][a][a]]_2) \xrightarrow[A]{\bar{a}} (f, [[aa][a][\ ]]_2)$$

It is obvious that the automata over Γ_2 accept exactly the set $Reg_2(\Gamma)$. The proof can be found in [Car06] page 98.

But the properties of this automata over Γ_2 do not satisfy our demands. To get better properties and good closure properties so we restrict the automata to accept just the computations that consist of reduced sequences of instructions.

Definition 2.3.2. An automata over Γ_2 is called *reduced* if for every computation $(p_0, [\ ]_2), \gamma_1, (p_1, s_1), \gamma_2, \ldots, \gamma_n, (p_n, s_n)$ of A the sequence $\gamma_1, \ldots, \gamma_n \in (\Gamma_2^O)^*$ is reduced.

Because of the uniqueness of the reduced sequence of a stack it is the case that a stack s is accepted by a computation :

$$(p_0, [\ ]_2) \xrightarrow[A]{\gamma_1} (p_1, s_1) \cdots (p_{n-1}, s_{n-1}) \xrightarrow[A]{\gamma_n} (p_n, s)$$

if $\gamma_1 \ldots \gamma_n$ is the reduced sequence of s.

Remark 2.3.3. By the definition of reduced sequence it is clear that in a reduced automaton over Γ_2 there could never appear the instruction $\overline{1}$. We can also show that the test $\bot_2$ is not necessary because the empty level 2 stack can just appear in the initial state and nowhere else and for this case we do not need the test.

For level 1 we do not loose expressivity by the restriction to only reduced automata. For the case of level 1 the reduced automata over Γ_1 are equal to the simple automaton over the alphabet Γ because by the definition 2.2.1 of Reg_1 by $Reg_1 = Reg(\Gamma^*)$ we know that we do not loose expressivity.

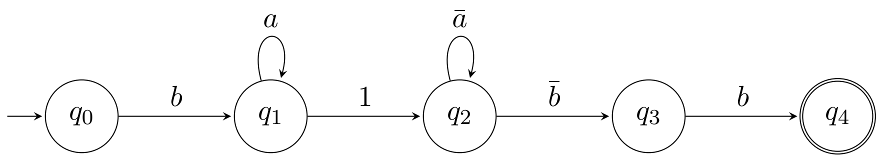
For level 2 we do no longer have this property and the class of languages that can be accepted by reduced automata over Γ_2 is a proper subset of the languages that can be accepted by the automata over Γ_2.

Theorem 2.3.4. *The reduced automata over Γ_2 are weaker as the automata over Γ_2.*

Proof. We have to show that there exists a set S of stacks in $Stacks_2(\Gamma)$ so that S can be accepted by an automaton over Γ_2 but there does not exists a reduced automaton over Γ_2 that recognizes S.

Let $S = \{[[ba^n][b]]_2 \,|\, n \geq 0\}$ and $\Gamma = \{a, b\}$. It is easy to see that S is accepted by the following automaton $A = (Q_A, I_A, F_A, \Delta_A)$ over Γ_2.

- $Q_A = \{q_0, q_1, q_2, q_3, q_4\}$
- $I_A = \{q_0\}$
- $F_A = \{q_4\}$
- Δ_A :



Now we have to show that S cannot be recognized by some reduced automaton over Γ_2. We show this by contradiction. Assume there exists a reduced automaton $B = (Q_B, I_B, F_B, \Delta_B)$ over Γ_2 that accepts S. We consider the stack $s = [[ba^n][b]]_2$ with $n = |Q_B|$ which belongs to S. The according reduced sequence to produce s is $ba^n1\overline{a}^n$. So in B there has to be a computation of the form:

$$(p_0, [\,]_2) \xrightarrow[B]{b} (p_1, s_1) \xrightarrow[B]{a} \cdots \xrightarrow[B]{a} (p_{n+1}, s_{n+1}) \xrightarrow[B]{1} (q_0, s_{n+2}) \xrightarrow[B]{\overline{a}} \cdots \xrightarrow[B]{\overline{a}} (q_n, s_{2n+2})$$

with $s_{2n+2} = s$.

Because of $|Q_B| = n$ there have to be some $i, j \in [0, n]$ with $i < j$ so that $q_i = q_j$. By this we get that the stack $s' = [[ba^n][ba^{j-i}]]$ is also accepted by B but $s' \notin S$. That is a contradiction to the assumption that $\mathcal{S}(B) = S$ holds. $\square$

By this proof we know that we cannot use the reduced automata over Γ_2 to capture $Reg_2(\Gamma)$. But we can enrich them by tests to give them the ability to recognize all languages of Reg_2. This automata with test are introduced a little later. The idea is that we can test the topmost level 1 stack of the current level 2 stack to have some properties, i.e. to belong to some language of Reg_1 and if the test is successfull we get back the identity of the stack. To get for an automata over Γ_2 the according reduced automata over Γ_2 with tests in Reg_1 we have to do several steps and go through different automata models.

We give here just a rough overview because it is already shown by Carayol in [Car06] in chapter 4.3. The proof work in a similar way as for the parity automata in chapter 4. To get an idea of the proof we now introduce first some other automata models over Γ_2.

Definition 2.3.5. An *alternating automaton over* Γ_2 is a tuple (Q, I, Δ), where Q is a finite set of states, $I \subseteq Q$ are the initial states and $\Delta \subseteq Q \times Sing(\Gamma_2^T) \times 2^{Q \times \Gamma_2^O}$ is the set of transitions.

A transition $\delta = (p, T, \{(q_1, \gamma_1), \ldots, (q_n, \gamma_n)\}) \in \Delta$ is noted as $p, T \to (q_1, \gamma_1) \wedge \ldots \wedge (q_n, \gamma_n)$. Intuitive the automaton A in configuration (p, s) with $p \in Q$ and $s \in Stacks_2(\Gamma)$ should, if s satisfies the tests in T, go into the n executions in parallel. The i^{th} execution starts in the configuration $(q_i, \mathcal{R}(\gamma_i)(s))$, if $\mathcal{R}(\gamma_i)(s)$ is defined.

We introduce some additional notations for the transition relation by defining for a transition $\delta = (p, T, A) \in \Delta$ the following three functions, $Head(\delta) = p$, $Test(\delta) = T$ and $Act(\delta) = A$.

An execution ε of A is a tuple (T, C), where T is a finite tree labeled with Γ_2 and C is a mapping from V_T in $Q \times Stacks_2(\Gamma)$. For all nodes $u \in V_T$ with the image (p, s) in C there exists a transition $\delta_u = p, T \to (q_1, \gamma_1) \wedge \ldots \wedge (q_n, \gamma_n) \in \Delta$ so that

- for all $t \in T$, $s \in Dom(\mathcal{R}(t))$,
- for all $i \in [1, n]$ it exists $v_i \in V_T$ so that $C(v_i) = (q_i, \mathcal{R}(\gamma_i)(s))$ and $u \xrightarrow[T]{\gamma_i} v_i$.

The set Φ_ε notes the mapping from V_T into Δ so that for every $u \in V_T$ the transition δ_u that is used at node u in ε is associated.

We say an execution $\varepsilon = (T, C)$ starts in $s \in Stacks_2(\Gamma)$ with state $q \in Q$ (resp. with transition $\delta \in \Delta$) if $C(root(T)) = (q, s)$ (resp. $\Phi_\varepsilon(root(T)) = \delta$)).

The automaton A accepts $s \in Stacks_2(\Gamma)$ if there exists an execution of A starting in s with $i \in I$. We write $\mathcal{S}(A)$ for the set of stacks of $Stacks_2(\Gamma)$ that are accepted by A. In analog we write for all $q \in Q$ (resp. $\delta \in \Delta$) the set $\mathcal{S}_q(A)$ (resp. $\mathcal{S}_\delta(A)$) of stacks $s \in Stacks_2(\Gamma)$ so that there exists a computation of A starting in state q (resp. with transition δ).

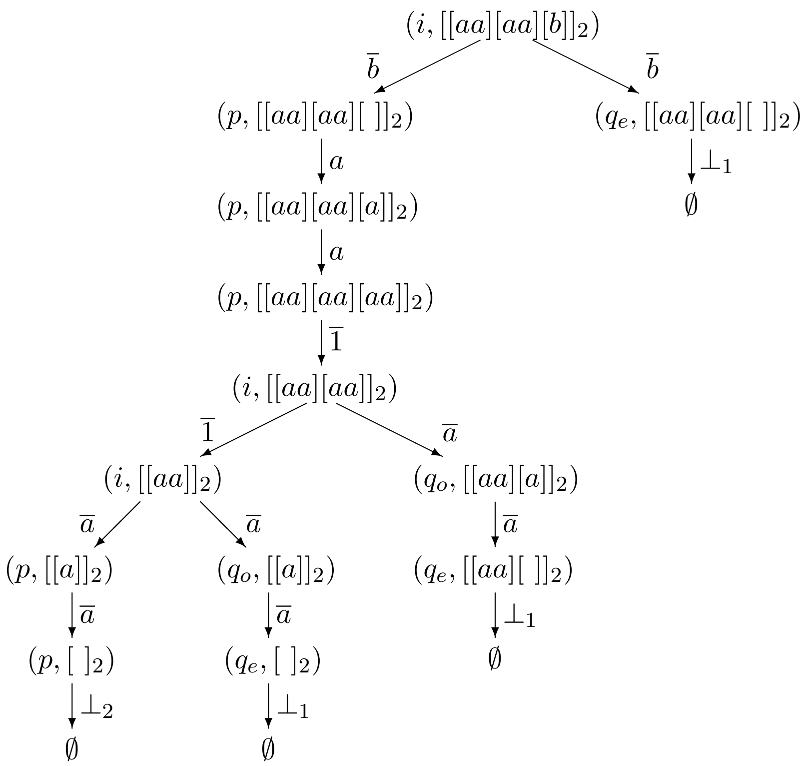
By Alt_2 we name the set of languages over stacks of level 2 that can be accepted by an alternating automata over Γ_2.

For the automata over Γ_2 we started the execution in the empty stack and proceed to a stack that we then accept. For alternating automata over Γ_2 it is somehow the other way around. There we start in the stack that we want to accept and then launch into the execution and go some path through other stacks and then stop. To get a little more familiar with this kind of automata consider the following example.

Example 9. *Consider the language* $\{[[x_1]\dots[x_n]] \mid n > 0, x_i \in \{a,b\}^*, |x_i|_a$ *is even* $\forall i \in [1,n]\}$*. We now want to give an alternating automaton* $A = (Q, I, \Delta)$ *over* Γ_2 *to accept this language. The automaton* A *has the state set* $Q = \{i, p, q_e, q_o\}$*, the initial state* $I = \{i\}$ *and the set of transitions* Δ*:*

$$\begin{array}{ll}
i \rightarrow (p,a) \wedge (q_e,\overline{b}) & i \rightarrow (p,b) \wedge (q_e,\overline{b}) \\
i \rightarrow (p,a) \wedge (q_e,\overline{a}) & i \rightarrow (p,b) \wedge (q_o,\overline{a}) \\
i \rightarrow (p,\overline{a}) \wedge (q_o,\overline{a}) & i \rightarrow (p,\overline{b}) \wedge (q_e,\overline{b}) \\
i \rightarrow (i,\overline{1}) \wedge (q_e,\overline{b}) & i \rightarrow (i,\overline{1}) \wedge (q_o,\overline{a}) \\
p \rightarrow (p,a) & q_e \rightarrow (q_e,\overline{b}) \\
p \rightarrow (p,b) & q_e \rightarrow (q_o,\overline{a}) \\
p \rightarrow (p,\overline{a}) & q_o \rightarrow (q_o,\overline{b}) \\
p \rightarrow (p,\overline{b}) & q_o \rightarrow (q_e,\overline{a}) \\
p \rightarrow (i,\overline{1}) & q_e, \bot_1 \rightarrow \emptyset \\
p, \bot_2 \rightarrow \emptyset &
\end{array}$$

For example the stack $[[aa][aa][b]]_2$ *is accepted by the following execution:*



The idea for the automaton is that in one branch we check if the current topmost level 1 stack contains an even number of a's and in the other branch the topmost level 1 stack is rebuild so that it can be destroyed by the $\overline{copy_1}$ *and the we use the alternation again.*

Further examples can be found in [Car06], e.g. on page 102.

These alternating automata over Γ_2 can also be *reduced* in this case the execution tree T has to be deterministic and for each stack $s \in Stacks_2(\Gamma)$ there is at most one node $v_s \in T$ so that $C(v_s) = (q, s)$ for some $q \in Q$.

Another special case of alternating automata over Γ_2 are the prune alternating automata over Γ_2. An alternating automata is called *prune* iff for all $\delta \in \Delta$ holds that $|Act(\delta)| \leq 1$. In this case the execution tree degenerates to a simple linear computation.

Now we want to define the above mentioned automata with tests. For that we first introduce some notations for the tests.

Definition 2.3.6. For all $l \geq 1$ and $k \geq l$ the *test* operation of level k is associated with a language $L \subseteq Stacks_l(\Gamma)$ and noted as $Test_L^k$. It is defined for all stacks $s \in Stacks_k(\Gamma)$ by:

$$Test_L^k(s) = \begin{cases} s & \text{if } top_l(s) \in L, \\ \text{not defined} & \text{otherwise.} \end{cases}$$

The according instruction to the operation $Test_L^k$ is T_L^k, i.e. $\mathcal{R}(T_L^k) = Test_L^k$.

An automaton over Γ_k with tests in a finite set $\mathcal{L}$ of $L \subseteq Stacks_l(\Gamma)$ has the instructions $\mathcal{T}_{\mathcal{L}}^k = \{T_L^k \mid L \in \mathcal{L}\}$. The domain for every $T \subseteq \mathcal{L}_{\mathcal{L}}^k$, $Dom(T) \subseteq Stacks_l$ is defined by:

$$Dom(T) = \begin{cases} \bigcap_{l \in [1,n]} L_l & \text{if } T = \{T_{L_1}^k, \ldots, T_{L_n}^k\} \text{ with } n > 0 \\ Stacks_l(\Gamma) & \text{if } T = \emptyset \end{cases}$$

Definition 2.3.7. An *automaton A over Γ_2 with tests* in a finite set $\mathcal{L}$ of subsets of $Stacks_l(\Gamma)$ for $l \leq 2$ is a tuple (Q, I, F, Δ) where Q is the finite set of states, $I \subseteq Q$ and $F \subseteq Q$ are the initial respectively final states, and $\Delta \subseteq Q \times \Gamma_2^O \times Sing(\Gamma_2^T) \times 2^{\mathcal{T}_{\mathcal{L}}^2} \times Q$ is the set of transitions.

The transition $(p, \gamma, T, T', q) \in \Delta$ is noted as $p \xrightarrow{\gamma} q, T, T'$. Intuitive it means that if the automaton is in a configuration (p, s) and takes the transition $p \xrightarrow{\gamma} q, T, T'$ to get in configuration (q, s') then $s' = \mathcal{R}(\gamma)(s)$ has to be defined and s' has to be in $Dom(T)$ and $Dom(T')$.

The tests $Test_L^k$ can also be added to the other kind of automatas as for example the reduced automata over Γ_2, the alternating automata over Γ_2 or the reduced and prune alternating automata over Γ_2. The definition of those automata models is similar to the one above. The tests T_L^k are just added beside the emptiness tests and their satisfiability has to be checked.

Now after that we have introduced the different kinds of automata we will need, we want to sketch the proof that we can find for every automaton over Γ_2 an according reduced automaton over Γ_2 with tests in Reg_1 so that they accept the same set of stacks. The first step is the equation between automata over Γ_2

and alternating automata over Γ_2. This can easily be achieved by just reversing the transitions and instructions. Then the next step we have to show that every alternating automaton over Γ_2 can be transformed into a reduced alternating automaton over Γ_2. This proof is quite complex and the transformation has exponential costs in the number of states of the alternating automaton. From the reduced alternating automata over Γ_2 we proceed to reduced and prune automata over Γ_2 with tests in Alt_1. To do so we have to guess the path in the computation tree of the reduced alternating automaton that leads to the empty level 2 stack. This path we keep and all other paths can be substituted by tests in Alt_1, because those other paths fulfill the requirements of lemma 2.1.2 and we can use it, i.e. it is enough to test the topmost level 1 stack and not the hole level 2 stack. Now it remains to show that $Alt_1 = Reg_1$. Here we can in the first step reuse the proof to get from the alternating automata over Γ_2 to reduced alternating automata over Γ_2 because this proof holds for all levels. In the second step we have again the problem to get a prune and reduced alternating automaton but here we do not need tests. This proof is very similar to the one we have for the case of parity automata in section 4.4.3.

This all together delivers the result that all these automata except exactly regular sets of stacks of level 2, i.e. Reg_2.

Corollary 2.3.8. For an alphabet Γ it holds that the alternating automata over Γ_2 accept the same regular sets of stacks as the reduced automata over Γ_2 with tests in Reg_1, i.e. it holds $Alt_2(\Gamma) = Reg_2(\Gamma)$.

2.4 Results over higher-order pushdown systems in MSO-logic

In this section we want to give some background information about higher-order pushdown systems respectively the graphs they produce and the use of the monadic second-order logic, short MSO, in the setting of what we like to show here in this thesis.

The pushdown games are particular cases of pushdown graphs. Theses graphs can be defined by MSO-interpretations in the full binary tree. A first well known property of the full binary tree is that every MSO-definable set of vertices is a regular set of words. For that we take the full binary tree with its canonical naming of vertices that means a vertex is named by the label of the path going from the root to this vertex.

The winning region of a game with a regular winning condition is definable by an MSO-formula this is proved by Walukiewicz in [Wal96] and for a simpler proof in the case where the game graph is deterministic see [Cac03a]. Combining these two results, it easily follows that the winning region of pushdown game with a regular winning condition is a regular set of configurations [1].

This approach has been lifted to higher-order pushdown games. In [CW03], the authors show that any higher-order pushdown graph can be interpreted in a particular graph called $GStacks_2$ which correspond to the graph of level 2

[1] A configuration in $Q \times \Gamma^*$ of a pushdown system can be represented by a word in $\Gamma^* Q$ composed of the stack and the state.

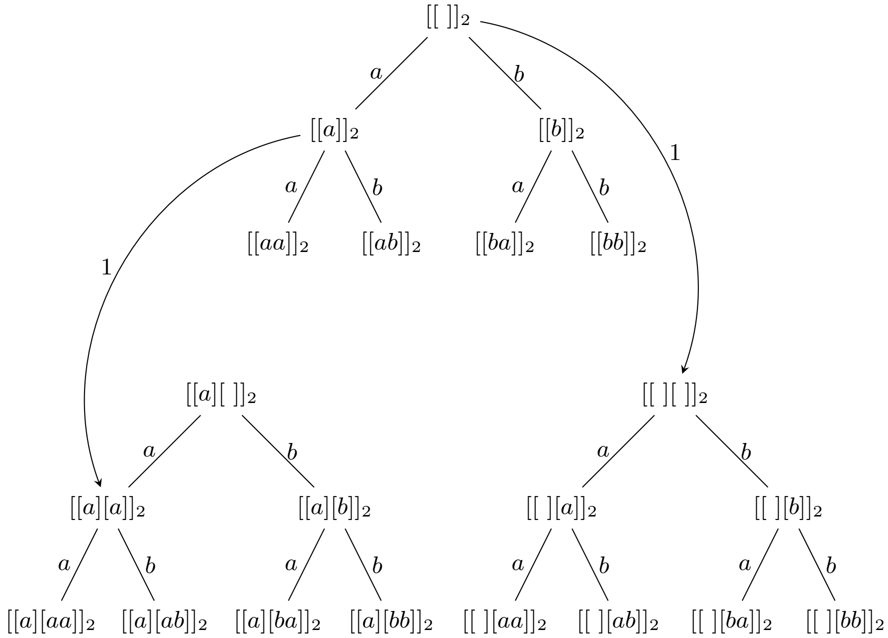


Figure 2: The graph of $GStacks_2$ for $\Gamma = \{a, b\}$.

stacks. There every node of the graph is a level 2 stack and the edges represent the push and copy operations. This graph is depicted in figure 2.

It has also been shown in [Car05] that any MSO-definable set of this structure is a regular set of level 2 stacks. Again with this two results, we obtain that the winning region of the higher-order pushdown game with a regular winning condition is a regular set of configurations. This approach can be extended at every level.

With this background one might think that the rest of this diploma thesis is useless because we already know that the winning region is regular. But the main contribution of this thesis is to provide an automata based method to compute the winning regions which does not rely on MSO logic and has minimal complexity.

3 Reachability games

In this section we want to introduce games over higher-order pushdown graphs. For that we now first define those graphs by using regular sets of stacks. We give a regular set of stacks that contains just the stacks that form the the set of the vertices of the game graph, two regular sets of stacks to define the two partitions of the game graph and a set of instruction sequences that connect the stacks respectively vertices of the game graph. We could have also used a higher-order pushdown system and define the game graph by the configurations that are reachable from the initial configuration of this system but the approach we use here is more general. Additionally it has the advantage that the game graph just contains the stacks instead of the configurations and so we can avoid the states.

In this chapter we will use as winning condition the reachability condition. In the next chapter we will look at the parity condition. The goal set of the reachability condition is here also given by a regular set of stacks that should be a subset of the set that defines the game graph.

After we have defined the setting for the reachability games in section 3.1 we show that the winning regions of those games are regular in section 3.3. We do this by an automata based approach and use the before introduced alternating automata over Γ_2. We use these automata to compute the winning region of Player 0 and use then the result of Carayol that the alternating automata over Γ_2 just recognize regular sets of stacks of level 2.

To compute the winning region with the alternating automata over Γ_2 we first have to enrich them by allowing them to work over instruction sequences instead of instructions and we also allow tests in Reg_2. But this is just done to make the proof easier to understand it does not enrich the expressivity of the model. This part is shown in section 3.2.

3.1 Definition of games

Definition 3.1.1. A *game graph* G over Γ_2 is a tuple $(S, S_0, S_1, \mathit{In})$, where S, S_0 and S_1 are regular sets of stacks of level 2 with $S_0 \cap S_1 = \emptyset$ and $S = S_0 \cup S_1$ and In is a set of instruction sequences in Γ_2, i.e. $\mathit{In} = \{\rho_1, \ldots, \rho_m\}$ with $\rho_i \in \Gamma_2^*$ for $i \in [1, m]$.

The vertices of the game graph are the stacks $s \in S$. They induce together with the set of instruction sequences In a labeled edge relation $E \subseteq \mathit{Stacks}_2(\Gamma) \times \mathit{In} \times \mathit{Stacks}_2(\Gamma)$ that is defined by $s \xrightarrow{\rho} s' \in E$ iff $\mathcal{R}(\rho)(s) = s'$ for $\rho \in \mathit{In}$ and $s, s' \in S$. The stacks $s \in S_0$ are the vertices of Player 0 and according to that the stacks $s \in S_1$ are the vertices of Player 1.

A play is an infinite sequence of stacks of level 2 $\eta = s_0 s_1 s_2 \ldots \in S^\omega$ so that $s_i \xrightarrow{\rho_i} s_{i+1} \in E$ for some $\rho_i \in \mathit{In}$ for all $i \geq 0$.

Definition 3.1.2. A *reachability game* R over Γ_2 is a tuple $R = (G, F)$, where $G = (S, S_0, S_1, \mathit{In})$ is a game graph over Γ_2 and $F \subseteq S \subseteq \mathit{Stacks}_2(\Gamma)$ is a regular set of stacks, the goal set.

A game $\eta \in Stacks_2(\Gamma)^\omega$ is *won* by Player 0 if $f \in Occ(\eta)$ for some $f \in F$, where $Occ(\eta)$ is the set of stacks that appear in the game, or if Player 1 ends up in a deadlock. The set $Win \subseteq Stacks_2(\Gamma)^\omega$ describes the set of games that are won by Player 0.

Definition 3.1.3. A *strategy* ξ for Player 0 is a function assigning to every sequence of vertices $\overrightarrow{s}$ ending in a vertex $s \in S_0$ a vertex $\xi(s)$ so that $s \xrightarrow{\rho} \xi(s) \in E$ for some $\rho \in In$ holds. A strategy is *winning* iff it guarantees a win for Player 0 whenever he follows the strategy.

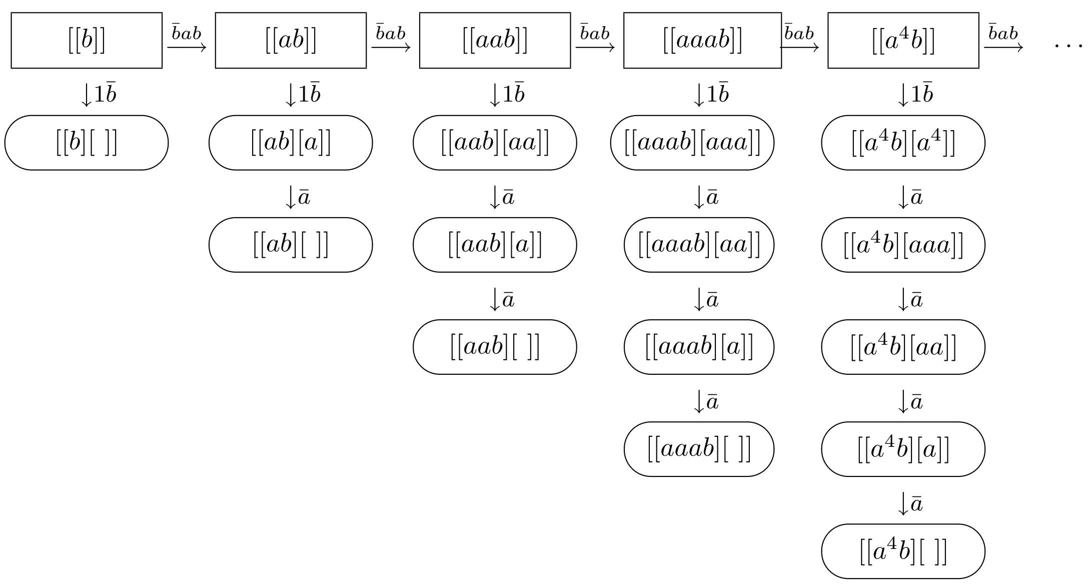
Definition 3.1.4. For a game (G, φ) is the *winning region* of Player 0 defined as $W_0 = \{s \in Stacks_2(\Gamma) \mid \text{Player 0 has a winning strategy starting from } s\}$.

Example 10. *Let* $\Gamma = \{a, b\}$, *we define the game* $R_1 = (G_1, F_1)$ *by* $G_1 = (S, S_0, S_1, In)$ *with*

- $S = \mathcal{R}(a^*b + a^*b1\bar{b}\bar{a}^*)([\ ]_2) = \{[[a^*b]]_2\} \cup \{[[a^ib][a^j]]_2 \mid i \geq j\}$,
- $S_0 = \mathcal{R}(a^*b1\bar{b}\bar{a}^*))([\ ]_2) = \{[[a^ib][a^j]]_2 \mid i \geq j\}$,
- $S_1 = \mathcal{R}(a^*b)([\ ]_2) = \{[[a^*b]]_2\}$,
- $In = \{\bar{b}ab, 1\bar{b}, \bar{a}\}$ *and*
- $F_1 = \mathcal{R}(a^+b1\bar{b}\bar{a}^*\bot_1)([\ ]_2) = \{[[a^nb][\ ]]_2 \mid n > 0\}$.

Here we characterize every regular set of stacks first formal by a regular sequence of instructions and then we give also an informal description of the set by describing the stacks directly.

By G_1 *we get the following game graph where the stacks that are surrounded by circles belong to Player 0 and the one surrounded by squares belong to Player 1:*



The winning region of Player 0 in the game R_1 consists of the regular set of stacks $W_0 = \mathcal{R}(a^+b1\bar{b}\bar{a}^*)([\ ]_2) = \{[[a^ib][a^j]]_2 \mid i \geq j, j > 0\}$. In this region he has the strategy to "go down" that means to pop the a's and so reach a stack of the winning set. The winning region of Player 1 is the regular set of stacks $W_1 = \mathcal{R}(a^*b + b1\bar{b})([\ ]_2) = \{[[a^*b]]_2\} \cup \{[[b][\ ]]_2\}$. The Player 1 has the strategy to choose always the instruction sequence $\bar{b}ab$ and so avoid the winning region of Player 0. In this case the play goes on to infinity and there is never a stack of the winning set reached. For the case that the game starts in the stack $[[b]]_2$ Player 1 has additionally the possibility to go by the instruction sequence $1\bar{b}$ to the stack $[[b][\ ]]_2$ and win there because Player 0 can not choose any successor and he has not reached a stack of the goal set F_1.

3.2 Preliminaries

We introduce in this section 3.2 some extensions of known automata models. Additionally we show some important lemmas that we need in the next section 3.3 to proof the regularity of the winning regions of the reachability games. We will need some of this lemmas also later in the section 4 to proof the regularity of the winning regions of the parity games.

We introduce now the alternating automaton over instruction sequences over Γ_2. It works just similar as the alternating automaton over Γ_2 but it can make several operations in one step but the automaton can just take those instruction sequences that he has given in the set *In*. We need it because in the games the players can go from one stack to another by an instruction sequence of a set of instruction sequences *In* and we somehow want to model the behavior of the players.

The alternating automata over instruction sequences are used later to compute the winning region of Player 0. After we have introduced the alternating automata over instructions sequences over Γ_2 we show that it is equivalent to the alternating automata over Γ_2. This result is obvious because we just have to introduce new states. We need this equivalence to show that these automata recognize only sets of $Reg_2(\Gamma)$.

Definition 3.2.1. An *alternating automaton A over instruction sequences $In = \{\rho_1, \ldots, \rho_m\}$* with $\rho_i \in \Gamma_2^*$ for all $i \in [1, m]$ is a tuple (Q, I, Δ), where Q is a finite set of states, $I \subseteq Q$ is the set of initial states and $\Delta \subseteq Q \times Sing\left(\Gamma_2^T\right) \times 2^{Q \times In}$ is the set of transitions.

A transition $\delta = (p, T, \{(q_1, \rho_{j_1}), \ldots, (q_n, \rho_{j_n})\}) \in \Delta$ is noted as $p, T \rightarrow (q_1, \rho_{j_1}) \wedge \ldots \wedge (q_n, \rho_{j_n})$. In a configuration (p, s) with $p \in Q$ and $s \in Stacks_2(\Gamma)$ the automaton A goes in parallel into the n different executions, if s fulfills the tests of T, where the i-th execution starts with the configuration $(q_i, \mathcal{R}(\rho_{j_i})(s))$, if $\mathcal{R}(\rho_{j_i})(s)$ is defined.

An execution $\mathcal{E}$ of A is a tuple (T, C), where T is a finite tree that is labeled by *In* and C is a mapping from V_T into the set $Q \times Stacks_2(\Gamma)$. For a vertex $u \in V_T$ that is mapped by C onto (p, s) there exists a transition $\delta_u = p, T \rightarrow (q_1, \rho_{j_1}) \wedge \ldots \wedge (q_n, \rho_{j_n}) \in \Delta$, so that:

- for all $t \in T$, $s \in Dom(\mathcal{R}(t))$,

- for all $i \in [1,n]$ exists a $v_i \in V_t$, so that $C(v_i) = (q_i, \mathcal{R}(\rho_{j_i})(s))$ and $u \stackrel{\rho_{j_i}}{\rightarrow} v_i$.

We say an execution $\mathcal{E} = (T, C)$ starts in a stack $s \in Stacks_2(\Gamma)$ with the state $q \in Q$ if $C(root(T)) = (q, s)$. The automaton accepts a stack $s \in Stacks_2(\Gamma)$ if an execution of A starts in a state $i \in I$. With $\mathcal{S}(A)$ we denote the set of all stacks that are accepted by A.

We will give here at the moment no example but you can see the first part of the example 14 for the proof of regularity of the winning region as an example for this kind of automaton.

Lemma 3.2.2. *The alternating automata over instruction sequences over* Γ_2 *are equivalent to the alternating automata over* Γ_2.

Proof. It is clear that each alternating automaton over Γ_2 is an alternating automaton over instruction sequences over Γ_2 where the instruction sequences are just the single instructions in Γ_2.

For the converse let $A_S = (Q_S, F_S, \Delta_S)$ be an alternating automaton over the instruction sequences $In = \{\rho_1, \dots, \rho_n\}$ with $\rho_i = \rho_{i_1}, \dots, \rho_{i_{n_i}} \in \Gamma_2^*$, $\rho_{i_j} \in \Gamma_2$ for all $i \in [1,n]$ and $j \in [1, n_i]$. Then we construct an alternating automaton $A_N = (Q_N, F_N, \Delta_N)$ over Γ_2 with $\mathcal{S}(A_S) = \mathcal{S}(A_N)$ like follows.

The idea is to split the transitions $p, T \rightarrow (p_1, \rho_{k_1}) \wedge \dots \wedge (p_m, \rho_{k_m}) \in \Delta_S$ with $\rho_{k_l} \in In$ for all $l \in [1,m]$ into the transitions

$$\begin{array}{ccccccc} & \nearrow & (pp_{1,k_{1_1}}, \rho_{k_{1_1}}) & \rightarrow & \cdots & \rightarrow & (p_1, \rho_{k_{1_{k_1}}}) \\ p, T & \rightarrow & (pp_{2,k_{2_1}}, \rho_{k_{2_1}}) & \rightarrow & \cdots & \rightarrow & (p_2, \rho_{k_{2_{k_2}}}) \\ & \vdots & & & & & \\ & \searrow & (pp_{m,k_{m_1}}, \rho_{k_{m_1}}) & \rightarrow & \cdots & \rightarrow & (p_m, \rho_{k_{m_{k_m}}}). \end{array}$$

So we define A_N by:

- $Q_N = Q_S \cup \{pq_{i_1}, \dots, pq_{i_{n_i-1}} | \forall p, q \in Q_S, i \in [1,n]\}$,
- $I_N = I_S$,
- Δ_N: for $p, T \rightarrow (p_1, \rho_{k_1}) \wedge \dots \wedge (p_m, \rho_{k_m}) \in \Delta_S$ add the following transitions:

$$\begin{array}{l} \{p, T \rightarrow (pp_{1,k_{1_1}}, \rho_{k_{1_1}}) \wedge \dots \wedge (pp_{m,k_{m_1}}, \rho_{k_{m_1}})\} \quad \cup \\ \{pp_{i,k_{i_j}}, \emptyset \rightarrow (pp_{i,k_{i_{j+1}}}, \rho_{k_{i_{j+1}}}) \,|\, \forall i \in [1,m], j \in [1, n_i - 2]\} \quad \cup \\ \{pp_{i,k_{n_i-1}}, \emptyset \rightarrow (p_i, \rho_{k_{i_{n_i}}}) \,|\, \forall i \in [1,m]\}. \end{array}$$

The complexity analysis for this transformation has the result that $|Q_N| \in O(|Q_S|^2 \cdot (\sum_{i=1}^{n} |\rho_i| + 1))$ and $|\Delta_N| \in O(|\Delta_S| \cdot \max_{i \in [1,n]} |\rho_i|)$. □

Example 11. *We give here an example for the definition of an alternating automaton over instruction sequences over* Γ_2 *and show also how lemma 3.2.2 works.*

Let $A = (Q, I, \Delta)$ *be an alternating automaton over the instruction sequences* $In = \{aa\overline{1}, \overline{aa}\}$. *We define* A *with the state set* $Q = \{i, p, p', q\}$ *the initial state set* $I = \{i\}$ *and the transitions* $\Delta = \{i, \perp_1 \rightarrow (p, aa\overline{1});\ p \rightarrow (p, aa\overline{1}) \wedge$

$(q, \overline{aa})$; $q \to (q, \overline{aa})$; $q, \bot_1 \to \emptyset$; $p \to (p', \overline{a})$; $p' \to (p', \overline{a})$; $p', \bot_2 \to \emptyset\}$. The automaton A recognizes the regular set of stacks defined by $\mathcal{R}(a^*(1\overline{aa})^*\bot_1)([\,]_2) = \{[[(aa)^n][(aa)^{n-1}]\ldots[\,]_2 \mid n \geq 1\}$.

The according alternating automaton $A' = (Q', I', \Delta')$ over Γ_2 is by lemma 3.2.2 defined by:

- $Q' = \{i, p, p', q, ip_{a_1}, ip_{a_2}, pp_{a_1}, pp_{a_2}, pq_{\overline{a}}, qq_{\overline{a}}\}$
- $I' = \{i\}$
- $\Delta' = \{i, \bot \to (ip_{a_1}, a)\ ;\ ip_{a_1} \to (ip_{a_2}, a)\ ;\ ip_{a_2} \to (p, \overline{1})\ ;$
  $p \to (pp_{a_1}, a) \wedge (pq_{\overline{a}}, \overline{a})\ ;\ pp_{a_1} \to (pp_{a_2}, a)\ ;\ pp_{a_2} \to (p, \overline{1})\ ;$
  $pq_{\overline{a}} \to (q, \overline{a})\ ;\ p \to (p', \overline{a})\ ;\ p' \to (p', \overline{a})\ ;\ p', \bot_2 \to \emptyset\ ;$
  $q \to (qq_{\overline{a}}, \overline{a})\ ;\ qq_{\overline{a}} \to (q, \overline{a})\ ;\ q, \bot_1 \to \emptyset\}$

We can again like for all other types of automata that are introduced here add tests to the alternating automata over instruction sequences like introduced in definition 2.3.6 and 2.3.7. These tests do not give more expressivity to the alternating automata like we show in the following proof. We add them just to make the proof of the regularity of the winning region easier to understand.

Lemma 3.2.3. *For all alternating automata A (over instruction sequences) over Γ_2 with tests in a finite set of languages $\mathcal{L} \subset Alt_2$, there exists an alternating automaton (over instruction sequences) over Γ_2 so that $\mathcal{S}(B) = \mathcal{S}(A)$. Additionally if each $L \in \mathcal{L}$ is accepted by an alternating automaton A_L (over instruction sequences) over Γ_2 then the size of $|Q_B|$ is bounded by $exp[0](|Q_A| + \sum_{L\in\mathcal{L}} |Q_L|)$.*

Proof. Let $A = (Q_A, I_A, \Delta_A)$ be an alternating automaton (over instruction sequences) over Γ_2 with tests in $\mathcal{L} \subset Alt_2$. And let all those languages $L \in \mathcal{L}$ be accepted by an alternating automaton $A_L = (Q_L, I_L, \Delta_L)$ (over instruction sequences) over Γ_2. Without loss of generality we can assume that the state sets of those automata are pairwise disjoint.

We construct now an alternating automaton $B = (Q_B, I_B, \Delta_B)$ (over instruction sequences) over Γ_2 that accepts $\mathcal{S}(A)$, with:

- $Q_B = Q_A \cup \bigcup_{L\in\mathcal{L}} Q_L$
- $I_B = I_A$
- $\Delta_B = \{p, T \to R \wedge \bigwedge_{T_L^2 \in T'} (i_L, \varepsilon) \mid \delta = p, T, T' \to R \in \Delta_A \text{ and } i_L \in I_L\}$
  $\cup\ \bigcup_{L\in\mathcal{L}} \Delta_L$

By construction B is equivalent to A. Remark that $|Q_B|$ is linear in $|Q_A| + \sum_{L\in\mathcal{L}} |Q_L|$. □

Example 12. *We show now an example for the lemma 3.2.3. For that let $\Gamma = \{a, b\}$ and $A = (Q, I, \Delta)$ be an alternating automaton over Γ_2 with tests in $\mathcal{L} \subseteq Alt_2$ recognizing the regular set of stacks $L = \{[[w_1][w_2]\ldots[w_n]]_2 \mid w_1 \in \Gamma^*, n > 1, \forall i \in [1, n-1]\ w_{i+1} \sqsubseteq w_i, |w_i|_a \text{ even }\}$. For simplicity we take for A a prune automaton, i.e. without universal branching and the language $\mathcal{L} =$*

$\{L_{even}\}$ with $L_{even} = \{[w]_1 \mid w \in \Gamma^*, |w|_a$ even$\}$ is just in Alt_1. We define A by $Q = \{i, f\}$, $I = \{i\}$, $\Delta = \{i \to (i, a);\ i \to (i, b);\ i, T_{L_{even}} \to (i, \overline{1});\ i, T_{L_{even}} \to (f, \overline{1});\ f \to (f, \overline{a});\ f \to (f, \overline{b});\ f, \bot_2 \to \emptyset\}$. For L_{even} we define the alternating automaton $B = (Q', I', \Delta')$ over Γ_1 by $Q' = \{p_e, p_o\}$, $I' = \{p_e\}$ and $\Delta' = \{p_e \to (p_e, \overline{b});\ p_e \to (p_o, \overline{a});\ p_o \to (p_o, \overline{b});\ p_o \to (p_e, \overline{a});\ p_e, \bot_1 \to \emptyset\}$.

With lemma 3.2.3 we can define now an alternating automaton C over Γ_2 without tests. So let $C = (Q_C, I_C, \Delta_C)$ be defined by:

- $Q_C = \{i, f, p_e, p_o\}$
- $I = \{i\}$
- $\Delta_C = \{i \to (i, a);\ i \to (i, b);$
  $i \to (i, \overline{1}) \wedge (p_e, \varepsilon);$
  $i \to (f, \overline{1}) \wedge (p_e, \varepsilon);$
  $f \to (f, \overline{a});\ f \to (f, \overline{b});\ f, \bot_2 \to \emptyset$
  $p_e \to (p_e, \overline{b});\ p_e \to (p_o, \overline{a});\ p_e, \bot_1 \to \emptyset$
  $p_o \to (p_o, \overline{b});\ p_o \to (p_e, \overline{a})\}$

In figure 3.2 we illustrate the three automata.

Now we introduce an other automata model respectively give it a new name. We mentioned at the end of section 2.2 the prune alternating automata, i.e. an alternating automata without alternation. This automata are now redefined and named back-automata because they are very similar to the automata over Γ_2 but they start like the alternating automata in the stack we want to accept and accept if the execution is finite. We need them for some proof where we want to check some simple properties of stacks. It is clear that they can accept less then the alternating automata over Γ_2.

Definition 3.2.4. The *back-automaton* A over Γ_2 is a tuple (Q, I, Δ) where Q is a finite set of states, $I \subseteq Q$ is the set of initial states and $\Delta \subseteq (Q \times Sing\left(\Gamma_2^T\right) \times Q \times \Gamma_2^O) \cup (Q \times Sing(\Gamma_2^T) \times \emptyset)$ is the set of transitions.

A configuration of A is a tuple (p, s) of $Q \times Stack_2(\Gamma)$. Let $\mathcal{C}_A = Q \times Stacks_2(\Gamma)$ be the set of all configurations of A. We can write a transition $(p, T, q, \gamma) \in \Delta$ also as $p, T \xrightarrow{\gamma} q$ respectively $p \xrightarrow{\gamma} q$ if $T = \emptyset$. An automaton goes from a configuration (p, s) into a configuration (q, s') by the application of the transition $p, T \xrightarrow{\gamma} q \in \Delta$ if $s' = \mathcal{R}(\gamma)(s)$ is defined and $s \in Dom(\mathcal{R}(t))$ for $T = \{t\}$.

The automaton A induces for all $\gamma \in \Gamma_2^O$ a relation $\xrightarrow[A]{\gamma} \subseteq \mathcal{C}_A \times \mathcal{C}_A$. This relation is for all configurations (p, s) and (q, s') of $\mathcal{C}_A$ defined as $(p, s) \xrightarrow[A]{\gamma} (q, s')$, if a transition $p, T \xrightarrow{\gamma} q \in \Delta$ exists with $s' = \mathcal{R}(\gamma)(s)$ and $s \in Dom(\mathcal{R}(t))$ for all $t \in T$.

An execution of A is a sequence $(p_0, s_0), \gamma_1, \ldots, (p_{n-1}, s_{n-1}), \gamma_n, (p_n, s_n) \in \mathcal{C}_A\left(\Gamma_2^O \mathcal{C}_A\right)^*$ so that for all $l \in [0, n-1]$, $(p_l, s_l) \xrightarrow[A]{\gamma_l} (p_{l+1}, s_{l+1})$ and $p_n, T \to \emptyset \in \Delta$. An execution of A accepts a stack $s \in Stacks_2(\Gamma)$ if $p_0 \in I$ and $s_0 = s$. The set of all stacks in $Stacks_2(\Gamma)$ that are accepted by A is written as $\mathcal{S}(A)$.

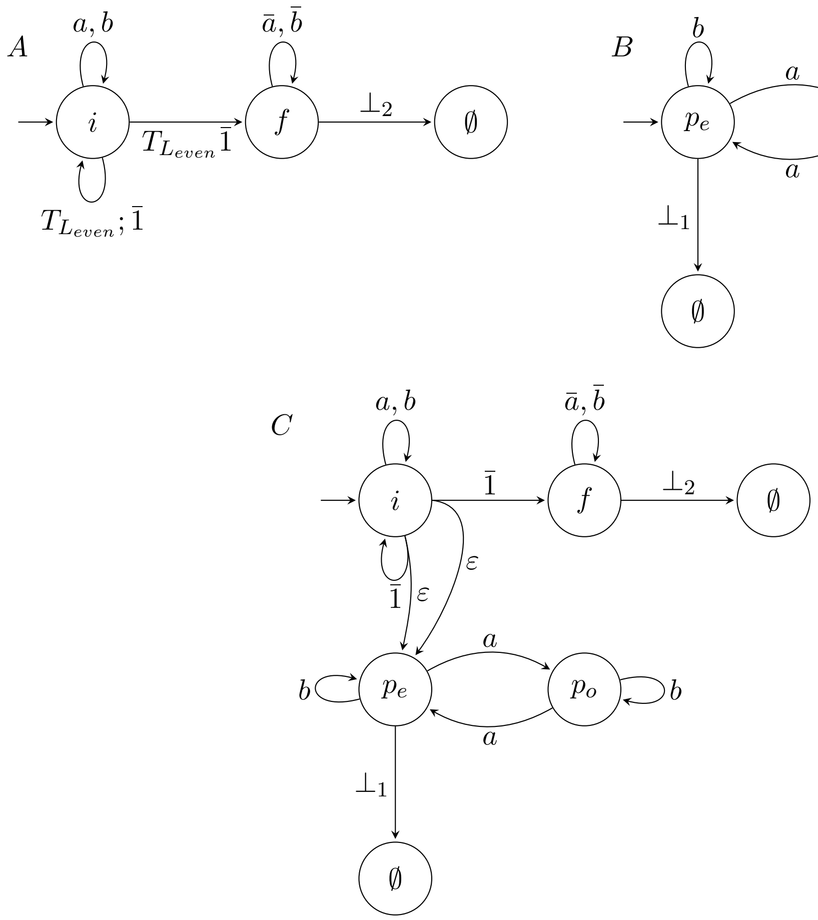


Figure 3: This figure shows the tree automata of example 12.

Example 13. *We will give here an example for a back-automaton. Consider the language* $\{[[a^n][a^{n-1}]\cdots[a][\ ]]_2 \mid n > 0\}$ *of example 8. We introduce a back-automaton* $A = (Q, I, \Delta)$ *over* Γ_2 *to accept this language. The automaton* A *has the state set* $Q = \{i, p, q, q'\}$*, the initial state* $I = \{i\}$ *and the set of transitions* Δ*:*

$$i, \bot_1 \longrightarrow (p, a) \quad p \longrightarrow (q, \overline{1}) \quad q \longrightarrow (p, a)$$

$$q \longrightarrow (q', \overline{a}) \quad q' \longrightarrow (q', \overline{a}) \quad q', \bot_2 \longrightarrow \emptyset$$

For example the stack $[[aa][a][\ ]]_2$ *is accepted by the following execution:*

$$\begin{array}{ccccccc} & (i, [[aa][a][\ ]]_2) & \xrightarrow[A]{a} & (p, [[aa][a][a]]_2) & \xrightarrow[A]{\overline{copy_1}} & (q, [[aa][a]]_2) \\ \xrightarrow[A]{a} & (p, [[aa][aa]]_2) & \xrightarrow[A]{\overline{copy_1}} & (q, [[aa]]_2) & \xrightarrow[A]{\overline{a}} & (q', [[a]]_2) \\ \xrightarrow[A]{\overline{a}} & (q', [\ ]_2) & \longrightarrow & \emptyset & & \end{array}$$

We use now the back-automaton over Γ_2 to define a language so that it contains for a instruction sequence all stacks where the application of the sequence is not defined. We will need this later for the proof of regularity of the winning region of reachability games and again also for the parity games. We first tried to define languages so that we divide the set of all stacks over Γ_2 into disjunct subsets $\widetilde{L}_{i_1,\ldots,i_k}$ so that we know for each language exactly those instruction sequences $\{i_1, \ldots, i_k\}$ of *In* that are defined on the stacks of the language but this attempt causes an additional exponential blowup that we can avoid with the following construction.

Lemma 3.2.5. *For every instruction sequence* ρ *in* Γ_2^* *we can define a language* L_ρ *by an back-automaton* A_ρ*, so that* A_ρ *accepts exactly the stacks* $s \in Stacks_2(\Gamma)$ *such that* $\mathcal{R}(\rho)(s)$ *is not defined.*

Proof. Let $\rho = \gamma_1 \ldots \gamma_k$ and let $A_\rho = (Q, I, \Delta)$. We define A_ρ such that it guesses the instruction γ_i, $i \in [1, k]$ in ρ for which holds that $\mathcal{R}(\gamma_1 \ldots \gamma_{i-1})(s) = s'$ is defined and $\mathcal{R}(\gamma_i)(s')$ is not defined. There are two possibilities, either $\mathcal{R}(\gamma_i) = pop_a$ for an $a \in \Gamma$ or $\mathcal{R}(\gamma_i) = \overline{copy_1}$. For the first case we need to do an other *pop* that is defined to prove that γ_i is not defined and for the second case we need to know that the last instruction of the reduced sequence of the current stack is not equal 1. To prove this we need to store the last instruction of the reduced sequence of the current stack in the states. So A_ρ looks as follows:

- $Q = \{q_1, \ldots, q_n, q\} \times \Gamma_2$
- $I = \{q_1\} \times \Gamma_2$
- $\Delta = \{(q_j, \gamma) \to ((q_{j+1}, \gamma'), \gamma_j) \mid j \in [1, i-1], i \in [1, n]\}$
  $\cup \{(q_i, \gamma) \to ((q, \gamma'), \overline{\gamma''}) \mid \gamma'' \in \Gamma - \{\overline{\gamma_i}\}$, for $\mathcal{R}(\gamma_i) = pop_x, i \in [1, n]\}$
  $\cup \{q \to \emptyset\}$
  $\cup \{(q_i, \gamma) \to \emptyset \mid \gamma \neq 1$ and $\gamma_i = \overline{1}, i \in [1, n]\}$

Now we show that A_ρ accept exactly the stacks $s \in Stacks_2(\Gamma)$ such that $\mathcal{R}(\rho)(s)$ is not defined.

For that we first assume that A_ρ accepts a stack s and $\mathcal{R}(\rho)(s)$ is defined. In this case there is an accepting execution for s. In this execution the first transition rule is always applicable because the application of ρ is defined, but to get an accepting execution we need to apply one of the other transition rules say for example at step i. By definition the second rule is just applicable if in ρ the current operation ρ_i is a pop_x and another pop_y is defined for $x, y \in \Gamma$, $x \neq y$. This is not possible since there is always at most one *pop* defined for a stack and this is by assumption the ρ_i and so the second rule can not be taken. The last rule is only applicable if $\gamma_i = \overline{1}$ and the last instruction of the reduced sequence of the current stack is not 1. But because ρ is defined on s this is also not possible. By this we know that s has no accepting execution in A_ρ this is a contradiction to the assumption.

We proof now that for all stacks s so that $\mathcal{R}(\rho)(s)$ is not defined holds that $s \in \mathcal{L}(A_\rho)$. This follows by the construction.

The state size of the automaton is polynomial in the size of the length of the instruction sequence and the size of the stack alphabet. □

We need also to define two other languages. The first language IN_ρ consists of those stacks $s \in Stacks_2(\Gamma)$ such that the application of ρ to s is still in the language S. The second language OUT_ρ is the opposite, there the application of ρ to s is no longer in the language S respectively it is in $Stacks_2(\Gamma) - S = \overline{S}$. So we can use the same proof for both languages.

As we will use this lemma 3.2.6 for the proof of regularity of the wining region we want a minimal cost for the complexity. For that we can assume without loss of generality that the language S in the definition of the game graph is given by a deterministic automaton over Γ_2 so that it it cost no effort to compute the complement $\overline{S}$.

Lemma 3.2.6. *For every instruction sequence ρ in Γ_2^* and any regular language S over stacks of level 2 we can define a language IN_ρ (resp. OUT_ρ) by an back-automaton A_ρ (resp. $\bar{A}_\rho$), so that A_ρ (resp. $\bar{A}_\rho$) accepts exactly the stacks $s \in Stacks_2(\Gamma)$ such that $\mathcal{R}(\rho)(s) \in S$ (resp. $\mathcal{R}(\rho)(s) \in \bar{S}$).*

Proof. Let $\rho = \gamma_1 \dots \gamma_k$ and let $A_\rho = (Q, I, \Delta)$ (resp. $\bar{A}_\rho = (Q, I, \Delta')$) be an back-automaton over Γ_2 with tests is Reg_2. So A_ρ looks as follows:

- $Q = \{q_0, \dots, q_k, \}$
- $I = \{q_0\}$
- $\Delta = \{q_j \to (q_{j+1}, \gamma_{j+1}) \mid j \in [0, k-1]\} \cup \{q_k, T_S \to \emptyset\}$

And $\bar{A}_\rho$ has the same state set but the transitions:

$$\Delta' = \{q_j \to (q_{j+1}, \gamma_{j+1}) \mid j \in [0, k-1]\} \cup \{q_k, T_{\bar{S}} \to \emptyset\}$$

It is clear by construction that the two automata fulfill the requirements and that their size is linear in the size of the instruction sequence and the size of the automaton of the automaton for S respectively $\overline{S}$. □

3.3 Proof of regularity

Now we have defined all preliminaries and all help lemmas we need to proof the regularity of the winning region of reachability games over Γ_2.

The idea of this proof is first to define an alternating automaton A over instruction sequences $In = \{\rho_1, \ldots, \rho_n\}$ in Γ_2 that accepts only those stacks that are in the winning region of Player 0. The second step is to show that these automata accept only regular sets of stacks, i.e. sets in $Reg_2(\Gamma)$.

To define the alternating automaton we go on like this. We first test in every step of the automaton if the current stack belongs to Player 0 or 1. If it belongs to Player 0 the automaton can choose non-deterministical an instruction sequence of In respectively a transition that executes this sequence. The choose is done in such a way that the automaton guesses the best way to get to a stack of the final set F if we are in the winning region of Player 0. If the current stack belongs to Player 1 we use the universality of the automaton and follow all possible sequences of instructions that stay in the game arena S and in the end reach for all of them a stack in F. For this we need the tests because not for all instructions the application to the current stack is defined and the application of some may lead outside of S. We have to guess the instruction sequences that are defined and stay in S and take them and for the other ones we have guessed to be not defined or not in S we have to verify this by a test. To do so we have introduced lemmas 3.2.5 and 3.2.6. These tests can be included into the execution as an extra branch like we have shown in lemma 3.2.3. If we reach in the game a stack of F the automaton has to guess this and also verify it with a test.

Theorem 3.3.1. *For a two player reachability game $R = (G, F)$ over Γ_2 with $G = (S, S_0, S_1, In)$ and a regular set of stacks $F \subseteq S \subseteq Stacks_2(\Gamma)$ the winning region for Player 0 is regular.*

Proof. Let $A = (Q_A, I_A, \Delta_A)$ be an alternating automaton over instruction sequences $In = \{\rho_1, \ldots, \rho_n\}$ over Γ_2 with tests in $\mathcal{L} \in Alt_2$, where $\mathcal{L}$ is defined as follows:

$$\mathcal{L} = \{S_0, S_1, F, L_\rho, IN_{\rho'}, OUT_{\rho''} \mid \text{for all } \rho, \rho', \rho'' \in In\},$$

where all L_ρ are defined like in lemma 3.2.5 and $IN_{\rho'}$, $OUT_{\rho''}$ like in lemma 3.2.6.

The definition of A looks then as follows:

- $Q_A = \{p\}$, (we do not need states here),
- $I_A = \{p\}$,
- Δ_A:
  - if the current stack is in S_0:

$$\{p, T_{S_0} \rightarrow (p, \rho_i) \mid i \in [1, n]\}$$

– if the current stack is in S_1:

$$p, \{T_{S_1}, T_{L_{\rho_{l_1}}}, \dots, T_{L_{\rho_{l_k}}}, T_{IN_{\rho_{j_1}}}, \dots, T_{IN_{\rho_{j_f}}}, T_{OUT_{\rho_{g_1}}}, \dots, T_{OUT_{\rho_{g_h}}} \mid l_x, j_y, g_z \in [1,n], \forall x \in [1,k], y \in [1,f], z \in [1,h], k+f+h = n \text{ and } \{\rho_{l_1}, \dots, \rho_{l_k}, \rho_{j_1}, \dots, \rho_{j_f}, \rho_{g_1}, \dots, \rho_{g_h}\} = \mathit{In}\} \rightarrow \bigwedge_{\rho \in [\rho_{j_1}, \rho_{j_f}]} (p, \rho)$$

– always:

$$\{p, T_F \rightarrow \emptyset\},$$

if this transition is applicable it has to be taken.

Now we have to show that $L(A) = W_0$ holds:

$\boxed{\Rightarrow:}$

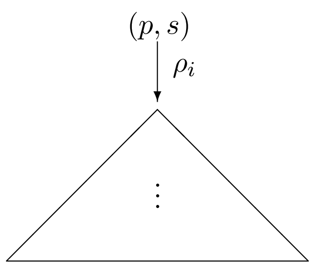
If $s \in L(A)$ then there exists an accepting execution $\varepsilon = (T, C)$ of A where T is a finite tree labeled by *In* and C is a mapping from V_T to $Q_A \times \mathit{Stacks}_2(\Gamma)$ respectively we can forget here about the states because we have just one. We show $s \in W_0$ by induction over the height n of the execution tree.

$\mathbf{n = 0}$: Because $s \in L(A)$ we have that in $root(T)$ the transition $p, T_{\mathcal{S}(A_F)} \rightarrow \emptyset$ is applicable and so it follows that $s \in F$ and by this $s \in W_0$. Or in the automaton the transition $p, \{T_{S_1}, T_{L_{\rho_{l_1}}}, \dots, T_{L_{\rho_{l_k}}}, T_{OUT_{\rho_{g_1}}}, \dots, T_{OUT_{\rho_{g_h}}} \mid l_x, g_z \in [1,n], \forall x \in [1,k], z \in [1,h], k+h = n$ and $\{\rho_{l_1}, \dots, \rho_{l_k}, \rho_{g_1}, \dots, \rho_{g_h}\} = \mathit{In}\} \rightarrow \emptyset$ is applicable. In this case Player 1 ends up in a deadlock, i.e. there is no instruction sequence defined and staying in S which he could choose. So $s \in W_0$, too.

$\mathbf{n > 0}$: We now have to distinguish between two cases:

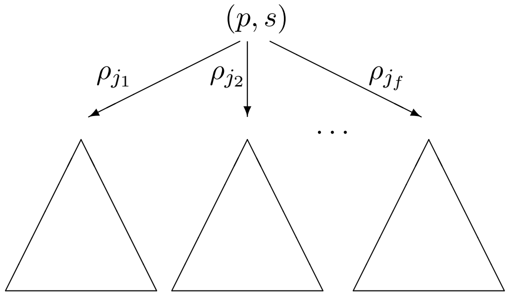
1. $s \in S_0$ and
2. $s \in S_1$.

1. $\mathbf{s \in S_0}$**:** In this case we have the following execution:



Here the automaton chooses non-deterministic the right transition $p, T_{S_0} \rightarrow (p, \rho_i)$ so that $\mathcal{R}(\rho_i)(s)$ is defined and "nearer" to a stack in F. The subtree of T with the root $(p, \mathcal{R}(\rho_i)(s))$ has to be accepted also by the execution. So the height of the execution tree is reduced and we can apply the induction hypothesis, that says that $\mathcal{R}(\rho_i)(s)$ belongs to W_0. So s has to belong also to W_0 because Player 0 can by choosing ρ_i get from s into W_0.

2. $\mathbf{s \in S_1}$**:** In this case we have the following execution:



So there is only the transition $p, \{T_{S_1}, T_{L_{\rho_{l_1}}}, \dots, T_{L_{\rho_{l_k}}}, T_{IN_{\rho_{j_1}}}, \dots, T_{IN_{\rho_{j_f}}}, T_{OUT_{\rho_{g_1}}}, \dots, T_{OUT_{\rho_{g_h}}} \mid l_x, j_y, g_z \in [1,n], \forall x \in [1,k], y \in [1,f], z \in [1,h], k + f + h = n$ and $\{\rho_{l_1}, \dots, \rho_{l_k}, \rho_{j_1}, \dots, \rho_{j_f}, \rho_{g_1}, \dots, \rho_{g_h}\} = In\} \rightarrow \bigwedge_{\rho \in [\rho_{j_1}, \rho_{j_f}]} (p, \rho)$ applicable because if we guessed right the instruction sequences $\rho_{l_1}, \dots, \rho_{l_k}$ have to be exactly the ones that are not defined on the current stack which is verified by the tests $T_{L_{\rho_{l_1}}}, \dots, T_{L_{\rho_{l_k}}}$, the instruction sequences $\rho_{j_1}, \dots, \rho_{j_f}$ are those where their application to the current stack is still in S which is verified by the tests $T_{IN_{\rho_{j_1}}}, \dots, T_{IN_{\rho_{j_f}}}$ and the instruction sequences $\rho_{g_1}, \dots, \rho_{g_h}$ are those where the application to the current stack lead outside of S which is verified by the tests $T_{OUT_{\rho_{g_1}}}, \dots, T_{OUT_{\rho_{g_h}}}$.

So we apply in the transition just the instruction sequences $\rho_{j_1}, \dots, \rho_{j_f}$ and get the successor stacks $\mathcal{R}(\rho_{j_i})(s) = s_{j_i}$ for $i \in [1, f]$. All the according subtrees with root (p, s_{j_i}) have to be accepted by the execution and so by induction hypothesis all s_{j_i} for $i \in [1, f]$ are in W_0. So whatever Player 1 chooses of this defined and in S staying instruction sequences it leads always to a stack in W_0. If Player 1 would choose in the game an instruction sequence that is not defined on the current stack or that leads outside of the game graph he would fail and Player 0 would win. So it follows that s is in W_0.

$\boxed{\Leftarrow:}$

For all $s \in W_0$ we have to show that $s \in L(A)$. If $s \in W_0$ then it holds either that $s \in F$ or in a deadlock of Player 1 or Player 0 can choose his instruction sequences such that he can force the play to reach a stack in F or a deadlock of Player 1 if started in s.

Is $s \in F$ then in automaton A the transition $p, T_F \rightarrow \emptyset$ is applicable and so $s \in L(A)$. Is s in a deadlock of Player 1 in the automaton the transition $p, \{T_{S_1}, T_{L_{\rho_{l_1}}}, \dots, T_{L_{\rho_{l_k}}}, T_{OUT_{\rho_{g_1}}}, \dots, T_{OUT_{\rho_{g_h}}} \mid l_x, g_z \in [1,n], \forall x \in [1,k], z \in [1,h], k + h = n$ and $\{\rho_{l_1}, \dots, \rho_{l_k}, \rho_{g_1}, \dots, \rho_{g_h}\} = In\} \rightarrow \emptyset$ is applicable and $s \in L(A)$.

Otherwise Player 0 can choose his instructions so that he can reach a stack in F or a deadlock of Player 1 whatever Player 1 does. In this case the automaton A mimic the bePlayer of Player 0. If it is Player 0's turn and he chooses the instruction sequence ρ_i the automaton has to take the transition $p, T_{S_0} \rightarrow (p, \rho_i)$. If it is Player 1 turn the automaton has to mimic all possible instruction sequences that are defined on s and stay in S by taking the according transition.

If the automaton mimic the play like this for showing that $s \in L(A)$ it suffice to show that the execution of the automaton is finite.

Assumption: There exits an infinite execution.
Because $|In|$ is finite the branching factor of T is finite, too. So by König's lemma there has to be an infinite path starting in s. But if the automaton models the game like described above and $s \in W_0$ holds there has to be a stack $s' \in F$ that is reached in the infinite path in the game and so also in the execution of the automaton. In this case the transition $p, T_F \to \emptyset$ is applicable and so the path stops. This is a contradiction to the assumption that the execution is infinite.

We have shown that we can compute the winning region of a reachability game over Γ_2 by an alternating automaton A over instruction sequences $In = \{\rho_1, \ldots, \rho_n\}$ over Γ_2 with tests in $\mathcal{L} \in Alt_2$. By lemma 3.2.3 we know that we can get rid of the test by doing them as extra branches in the execution tree and get so an alternating automaton B over instruction sequences. By lemma 3.2.2 we can construct an alternating automaton C over Γ_2 that is equivalent to B. By corollary 2.3.8 we know that the alternating automata over Γ_2 recognize only the regular sets of stacks. So we got that the winning region of reachability games over Γ_2 is regular.

The state complexity of this result is double exponential in the size of the set In of the instruction sequences. The most parts of the proof have just linear or polynomial effort but the last result $Alt_2 = Reg_2$ has double exponential effort because the step alternating to reduced alternating that has exponential effort has to be made two times, one time for every level. □

Example 14. *We give now an example for the construction of the alternating automaton to compute the winning region of Player 0 for the reachability game defined in example 10. So let $A = (\{q\}, \{q\}, \Delta)$ be an alternating automaton over instruction sequences with tests in $\mathcal{L} \subseteq Alt_2$. Here we have $\mathcal{L}$ given by:*

$$\mathcal{L} = \{S_0, S_1, F_1, L_\rho, IN_{\rho'}, OUT_{\rho''} \mid \textit{for all } \rho, \rho', \rho'' \in In\},$$

where all L_ρ are defined like in lemma 3.2.5 and $IN_{\rho'}$, $OUT_{\rho''}$ like in lemma 3.2.6. The transitions Δ are given by:

$$\begin{aligned} q, T_{S_0} &\to (p, \overline{a}) \\ q, \{T_{S_1}, T_{\overline{a}}, T_{IN_{\overline{b}ab}}, T_{IN_{1\overline{b}}}\} &\to (q, \overline{b}ab) \wedge (q, 1\overline{b}) \\ q, T_{F_1} &\to \emptyset \end{aligned}$$

Example 15. *In the following we want to show another example for a reachability game and the alternating automaton to compute the winning region. For this let the game graph $G_3 = (S, S_0, S_1, In)$ be defined by:*

- $S = \mathcal{R}(((a + aa + b + bb)(1 + a1 + b1))^* p_1 + ((a + aa + b + bb)(1 + a1 + b1))^* (a + aa + b + bb) p_0)([\ ]_2)$
- $S_0 = \mathcal{R}(((a + aa + b + bb)(1 + a1 + b1))^* (a + aa + b + bb) p_0)([\ ]_2)$
- $S_1 = \mathcal{R}(((a + aa + b + bb)(1 + a1 + b1))^* p_1)([\ ]_2)$
- $In = \{\overline{p_1} a p_0, \overline{p_1} aa p_0, \overline{p_0} a 1 p_1, \overline{p_1} b p_0, \overline{p_1} bb p_0, \overline{p_0} 1 b p_0, \overline{p_0} 1 p_1\}$

and the goal set is $F_3 = \mathcal{R}((b1+bbb1+aab1)(aa1+bb1)^*p_1)([\,]_2)$*, in particularly in every level 1 stack there must be an even number of a and an odd number of b. In figure 4 is shown a part of the game graph. We need the* p_0 *and* p_1 *just to make sure which instruction sequences are performed by which player, besides they can be ignored.*

We define the alternating automaton $A_{R_3} = (\{q\}, \{q\}, \Delta_{R_3})$ *with tests in* $\mathcal{L} \subseteq Alt_2$ *to compute the winning region. For this let* $\mathcal{L}$ *be defined by:*

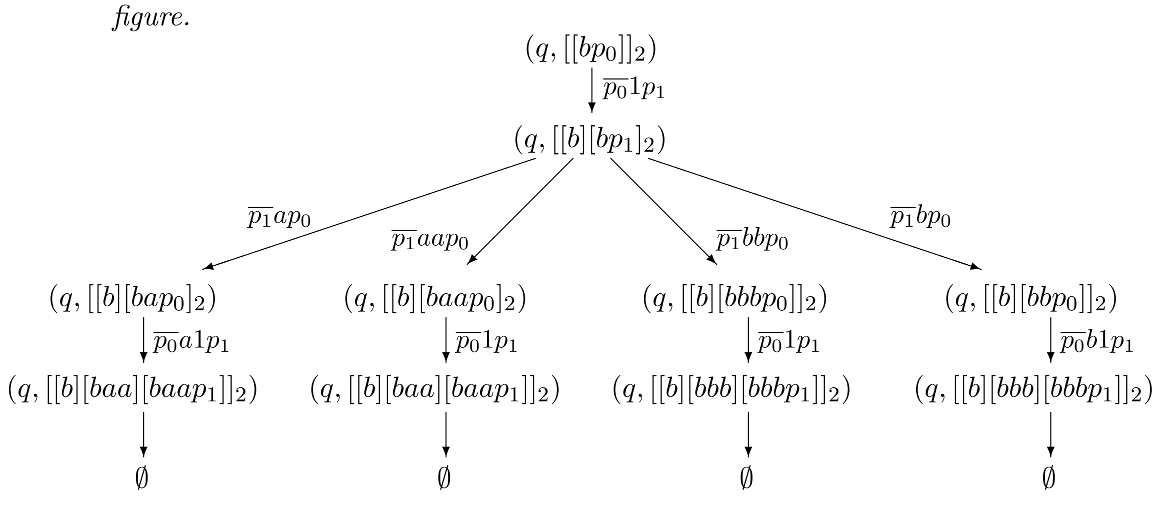
$$\mathcal{L} = \{S_0, S_1, F_3, L_\rho, IN_{\rho'}, OUT_{\rho''} \mid \textit{for all } \rho, \rho', \rho'' \in In\},$$

where all L_ρ *are defined like in lemma 3.2.5 and* $IN_{\rho'}$*,* $OUT_{\rho''}$ *like in lemma 3.2.6. The transitions* Δ_{R_3} *of* A_{R_3} *are defined by:*

$$\begin{aligned} q, T_{S_0} &\rightarrow (\overline{p_0}1p_1) \\ q, T_{S_0} &\rightarrow (\overline{p_0}a1p_1) \\ q, T_{S_0} &\rightarrow (\overline{p_1}b1p_1) \\ q, T_{F_3} &\rightarrow \emptyset \end{aligned}$$

$$q, \{T_{S_1}, T_{\overline{p_0}a1p_1}, T_{\overline{p_0}b1p_1}, T_{\overline{p_0}1p_1}, T_{IN_{\overline{p_1}ap_0}}, T_{IN_{\overline{p_1}aap_0}}, T_{IN_{\overline{p_1}bp_0}}, T_{IN_{\overline{p_1}bbp_0}}\}$$
$$\rightarrow (q, \overline{p_1}ap_0) \wedge (q, \overline{p_1}aap_0) \wedge (q, \overline{p_1}bp_0) \wedge (q, \overline{p_1}bbp_0)$$

We show now an example execution of the automaton A_{R_3} *on the stack* $[[bp_0]]_2$ *that is in the winning region. In the second step we could have already chosen the transition* $q, T_{F_3} \rightarrow \emptyset$ *but we go on to show a transition of Player 1. We also do not note the tests beside the transitions to avoid a too complex figure.*



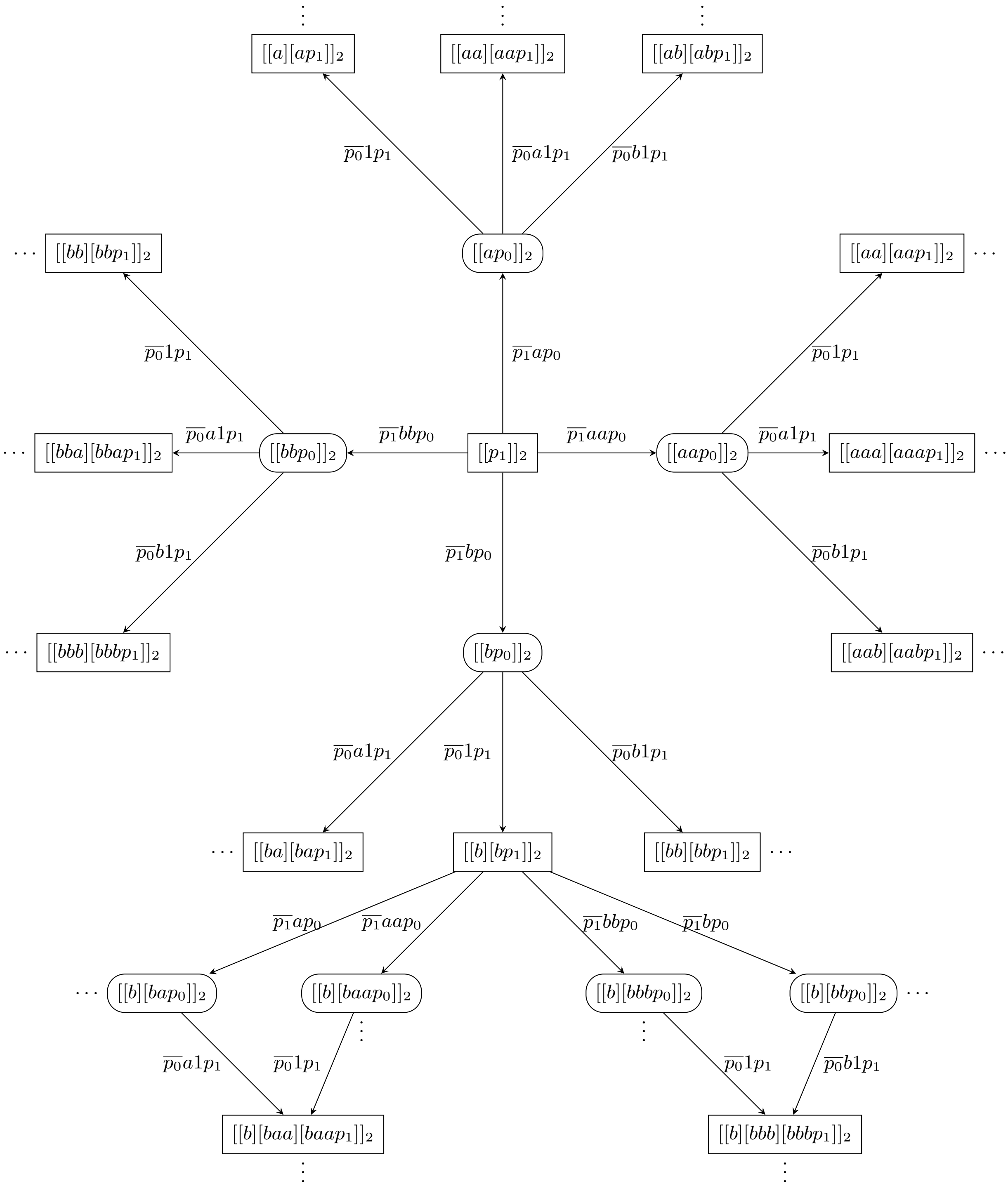


Figure 4: This figure shows a part of the game graph of example 15.

4 Parity Games

In this section we introduce parity games over higher-order pushdown graphs that are defined by regular sets of stacks of level 2 and instruction sequences over Γ_2. We show that the winning region of Player 0 for this games is again a regular set of stacks of level 2. The idea for this proof is very similar as for reachability games but here the plays become really infinite because of the parity condition.

The plan for this chapter is the following. We first introduce parity games and a new automata model, the alternating parity automata over Γ_2, which work similar as the alternating automata over Γ_2. The main difference is that they include the parity condition to accept a stack and so the execution tree becomes infinite. We need this new automata model to proof that the winning region of the parity games is regular. The proof works in two main steps. First we use the alternating parity automaton to model the game and compute the stacks that are in the winning region of the parity game and then we show that these automata accept only regular set of stacks. These main steps are outlined in figure 5.

The first step is straight forward and very similar to the case of reachability games. The automaton has to guess the best instruction sequence for Player 0 to fulfill the parity condition and it has to check by the use of the universality for all possible instruction sequences of Player 1 that the parity condition is satisfied. If the automaton accepts a stack then it has to be in the winning region of the game and vice versa.

The second step is much more complicated. To show that the alternating parity automata over Γ_2 accept just regular set of stacks of level 2 we first need to proof the equivalence to reduced alternating parity automata. For this purpose we use a result over tree automata of Vardi [Var98]. After that the proof again is very similar to the case of reachability games respectively to the case of alternating automata over Γ_2. We first make the reduced parity automata prune and add tests to them in Alt_1 respectively in a version of alternating with parity condition over Γ_1, we call it $PAlt_1$. In this step we loose the parity condition in level 2 but we still have it for level 1. Then we have to show that $PAlt_1$ is equal to Reg_1. This is again very similar to the case without parity.

4.1 Preliminaries

In this part we define parity games and alternating parity automata to have the basic structures that we need for this section.

Definition 4.1.1. A *parity game* P over Γ_2 is a tuple $P = (G, \Omega)$, where $G = (S, S_0, S_1, In)$ is a game graph over Γ_2 and $\Omega : Stacks_2(\Gamma) \to \{0, \ldots, m\}$ is a coloring, where we can assume that for every color $i \in [1, m]$ there is a regular set of stacks C_i that contains exactly those stacks with color i.

A play $\eta \in Stacks_2(\Gamma)^\omega$ is won by Player 0, if $max(Inf(\Omega(\eta)))$ is even or the game ends for Player 1 in a deadlock. The set $Win \subseteq Stacks_2(\Gamma)^\omega$ specifies the set of plays, that are won by Player 0.

parity game over instruction sequences over Γ_2

$\Downarrow$

alternating parity automaton over Γ_2

$\Downarrow$

reduced alternating parity automaton over Γ_2

$\Downarrow$

reduced automaton over Γ_2 with tests in Reg_1

Figure 5: The picture shows the plan for the chapter 4.

The game graph is defined as for reachability games in definition 3.1.1.

Example 16. *Let $\Gamma = \{a, b\}$, we define the parity game $P_1 = (G_2, \Omega)$ by $G_2 = (S, S_0, S_1, In)$ with:*

- $S = \mathcal{R}(a^*b + a^*b1\bar{b}(\bar{a}1)^*)([\ ]_2)$,
- $S_0 = \mathcal{R}(a^*b1\bar{b}(\bar{a}1)^*)([\ ]_2)$,
- $S_1 = \mathcal{R}(a^*b)([\ ]_2) = \{[[a^*b]]_2\}$,
- $In = \{\bar{b}ab, \bar{b}\bar{a}b, 1\bar{b}, b\bar{1}, \bar{1}a, \bar{a}1\}$,

And the coloring Ω is given by:

- $C_1 = \mathcal{R}(a^*b + a^*b1\bar{b})([\ ]_2)$ *and*
- $C_2 = \mathcal{R}(a^*b1\bar{b}(\bar{a}1)^+)([\ ]_2)$.

By G_2 we get the game graph depictured in Figure 6 where the stacks that are surrounded by circles belong to Player 0 and the one surrounded by squares belong to Player 1.

The stacks in the topmost two rows have the color 1 and the stacks in all the rows below have the color 2.

The winning region of Player 0 in the game P_1 consists of the regular set of stacks $W_0 = \mathcal{R}(a^+b1\bar{b}(\bar{a}1)^*)([\ ]_2)$. In this region he has the strategy to "toggle" between the stacks with color 2 and choose never the instruction sequence $b\bar{1}$. The winning region of Player 1 is the regular set of stacks $W_1 = \mathcal{R}(a^*b + b1\bar{b})([\ ]_2)$. The Player 1 has the strategy to choose either always the instruction sequence $\bar{b}ab$ and so stay in parity 1 or he "toggles" between some of his stacks with color 1. In this case only the priority 1 is seen infinitely often.

To prove that the winning region of the parity games over Γ_2 are regular we need to make some adjustments on the different types of automata that we already know. In principle we just have to add an accepting component to take the parity condition into account.

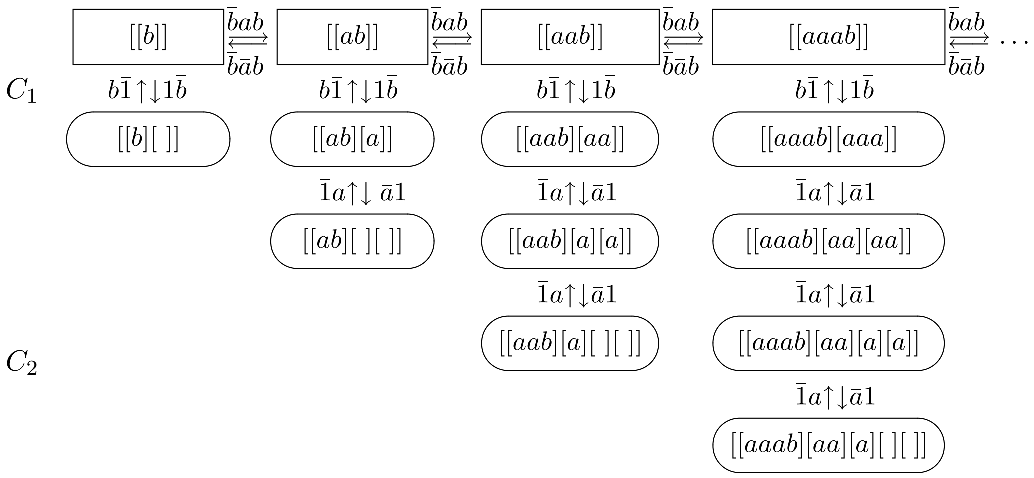


Figure 6: Game graph of Example 16.

Because of this we define the alternating parity automaton over Γ_2 and alternating parity automata over instruction sequences over Γ_2. It can be adapted from lemma 3.2.2 that these two types of alternating parity automata are equivalent.

Definition 4.1.2. An *alternating parity automaton over* Γ_2 is defined by a tuple (Q, I, Δ, Ω), where Q is a finite set of states, $I \subseteq Q$ are the initial states, $\Delta \subseteq Q \times Sing(\Gamma_2^T) \times 2^{Q\times\Gamma_2^O}$ is the set of transitions and $\Omega : Q \times \{0, \ldots, m\}$ is a coloring.

A transition $\delta = (p, T, \{(q_1, \gamma_1), \ldots, (q_n, \gamma_n)\}) \in \Delta$ is noted as $p, T \rightarrow (q_1, \gamma_1) \wedge \ldots \wedge (q_n, \gamma_n)$. Intuitive the automaton A in configuration (q, s) with $q \in Q$ and $s \in Stacks_2(\Gamma)$ should, if s satisfies the tests in T, go into the n executions in parallel. The i^{th} execution starts in the configuration $(q_i, \mathcal{R}(\gamma_i)(s))$, if $\mathcal{R}(\gamma_i)(s)$ is defined.

An execution ε of A is a tuple (T, C), where T is an infinite tree labeled with Γ_2 and C is a mapping from V_T in $Q \times Stacks_2(\Gamma)$. For all nodes $u \in V_T$ with the image (p, s) in C there exists a transition $\delta_u = p, T \rightarrow (q_1, \gamma_1) \wedge \ldots \wedge (q_n, \gamma_n) \in \Delta$ so that

- for all $t \in T$, $s \in Dom(\mathcal{R}(t))$,
- for all $i \in [1, n]$ it exists $v_i \in V_T$ so that $C(v_i) = (q_i, \mathcal{R}(\gamma_i)(s))$ and $u \xrightarrow[T]{\gamma_i} v_i$.

The set Φ_ε notes the mapping from V_T into Δ so that for every $u \in V_T$ the transition δ_u that is used at node u in ε is associated.

We say an execution $\varepsilon = (T, C)$ starts in $s \in Stacks_2(\Gamma)$ with state $q \in Q$ (resp. with transition $\delta \in \Delta$) if $C(root(T)) = (q, s)$ (resp. $\Phi_\varepsilon(root(T)) = \delta$).

The automaton A accepts $s \in Stacks_2(\Gamma)$ if there exists an execution of A starting in s with $i \in I$ and for every path in T if holds that:

- either the path is finite and ends with a transition $p, T \to \emptyset$,
- or the path is infinite and fulfills the acceptance condition that the maximal infinitely often seen color is even, i.e. for a path $\eta = q_0 q_1 q_2 \ldots \in Q^\omega$ holds that $max(Inf(\Omega(\eta)))$ is even.

Definition 4.1.3. An *alternating parity automaton A over instruction sequences $In = \{\rho_1, \ldots, \rho_m\}$* with $\rho_i \in \Gamma_2^*$ for all $i \in [1, m]$ Γ_2 is a tuple (Q, I, Δ, Ω), where Q is a finite set of states, $I \subseteq Q$ are the initial states, $\Delta \subseteq Q \times Sing(\Gamma_2^T) \times 2^{Q \times In}$ is the set of transitions and $\Omega : Q \times \{0, \ldots, l\}$ is a coloring.

A transition $\delta = (p, T, \{(q_1, \rho_{j_1}), \ldots, (q_n, \rho_{j_n})\}) \in \Delta$ is noted as $p, T \to (q_1, \rho_{j_1}) \wedge \ldots \wedge (q_n, \rho_{j_n})$. Intuitive the automaton A in configuration (q, s) with $q \in Q$ and $s \in Stacks_2(\Gamma)$ should, if s satisfies the tests in T, go into the n executions in parallel. The i^{th} execution starts in the configuration $(q_i, \mathcal{R}(\rho_{j_i})(s))$, if $\mathcal{R}(\rho_{j_i})(s)$ is defined.

An execution ε of A is a tuple (T, C), where T is an infinite tree labeled with In and C is a mapping from V_T in $Q \times Stacks_2(\Gamma)$. For all nodes $u \in V_T$ with the image (p, s) in C there exists a transition $\delta_u = p, T \to (q_1, \rho_{j_1}) \wedge \ldots \wedge (q_n, \rho_{j_n}) \in \Delta$ so that

- for all $t \in T$, $s \in Dom(\mathcal{R}(t))$,
- for all $i \in [1, n]$ it exists $v_i \in V_T$ so that $C(v_i) = (q_i, \mathcal{R}(\rho_{j_i})(s))$ and $u \xrightarrow{\rho_{j_i}} v_i$.

The set Φ_ε notes the mapping from V_T into Δ so that for every $u \in V_T$ the transition δ_u that is used at node u in ε is associated.

We say an execution $\varepsilon = (T, C)$ starts in $s \in Stacks_2(\Gamma)$ with state $q \in Q$ (resp. with transition $\delta \in \Delta$) if $C(root(T)) = (q, s)$ (resp. $\Phi_\varepsilon(root(T)) = \delta$)).

The automaton A accepts $s \in Stacks_2(\Gamma)$ if there exists an execution of A starting in s with $i \in I$ and for every path in T if holds that:

- either the path is finite and ends with a transition $p, T \to \emptyset$,
- or the path is infinite and fulfills the acceptance condition that the maximal infinitely often seen color is even, i.e. for a path $\eta = q_0 q_1 q_2 \ldots \in Q^\omega$ holds that $max(Inf(\Omega(\eta)))$ is even.

Lemma 4.1.4. *The alternating parity automata over instruction sequences over Γ_2 are equivalent to the alternating parity automata over Γ_2.*

Proof. Let $A_S = (Q_S, F_S, \Delta_S, \Omega_S)$ be an alternating parity automata over instruction sequences $In = \{\rho_1, \ldots, \rho_n\}$, with $\rho_i = \rho_{i_1}, \ldots, \rho_{i_{n_i}} \in \Gamma_2^*$, $\rho_{i_j} \in \Gamma_2$ for all $i \in [1, n]$ and $j \in [1, n_i]$. Then we construct an alternating parity automaton $A_N = (Q_N, F_N, \Delta_N, \Omega_N)$ over Γ_2 with $\mathcal{S}(A_S) = \mathcal{S}(A_N)$ as follows.

The idea is to split the transitions $p, T \to (p_1, \rho_{k_1}) \wedge \ldots \wedge (p_m, \rho_{k_m}) \in \Delta_S$ with $\rho_{k_l} \in In$, for all $l \in [1, m]$ into the transitions

$$\begin{array}{ccccccc} & \nearrow & (pp_{1,k_{1_1}}, \rho_{k_{1_1}}) & \to & \cdots & \to & (p_1, \rho_{k_{1_{k_1}}}) \\ p, T & \to & (pp_{2,k_{2_1}}, \rho_{k_{2_1}}) & \to & \cdots & \to & (p_2, \rho_{k_{2_{k_2}}}) \\ & \vdots & & & & & \\ & \searrow & (pp_{m,k_{m_1}}, \rho_{k_{m_1}}) & \to & \cdots & \to & (p_m, \rho_{k_{m_{km}}}). \end{array}$$

- $Q_N = Q_S \cup \{pq_{i_1}, \ldots, pq_{i_{n_{i-1}}} | \forall p, q \in Q_S, i \in [1, n]\}$
- $I_N = I_S$
- Δ_N: for $p, T \to (p_1, \rho_{k_1}) \wedge \ldots \wedge (p_m, \rho_{k_m}) \in \Delta_S$ add the following transitions:

$$\begin{array}{l} \{p, T \to (pp_{1,k_{1_1}}, \rho_{k_{1_1}}) \wedge \ldots \wedge (pp_{m,k_{m_1}}, \rho_{k_{m_1}})\} \quad \cup \\ \{pp_{i,k_{i_j}}, \emptyset \to (pp_{i,k_{i_{j+1}}}, \rho_{k_{i_{j+1}}}) \,|\, \forall i \in [1, m], j \in [1, n_i - 2]\} \quad \cup \\ \{pp_{i,k_{n_i-1}}, \emptyset \to (p_i, \rho_{k_{i_{n_i}}}) \,|\, \forall i \in [1, m]\} \end{array}$$

- $\Omega_N(q) = \Omega_S(q)$ for all $q \in Q_S$ and
  $\Omega_N(pq_{i_j}) = \Omega_S(p)$ for all $p \in Q_S$, $i \in [1, n]$ and $j \in [1, n_i]$

The complexity for this is so that $|Q_N| \in O(|Q_S|^2 \cdot (\sum_{i=1}^{n} |\rho_i| + 1))$ and $|\Delta_N| \in O(|\Delta_S| \cdot \max_{i \in [1,n]} |\rho_i|)$. □

We will later again need the case that the automaton is reduced, so we now introduce here already the definition.

Definition 4.1.5. An alternating parity automaton over Γ_2 is *reduced*, if for all executions $\varepsilon = (T, C)$ of A,

- the tree T is deterministic,
- for every stack $s \in Stacks_2(\Gamma)$ it exists at most one node $u_s \in V_T$ so that $C(u_s) = (q, s)$ for some $q \in Q$.

Remark that for the case of reduced parity automata we have only infinite paths in the execution if the stack grows to infinity otherwise the paths become finite.

4.2 Proof of regularity

In this section we start with the proof that the winning region of a two player parity game is regular.

The idea for the proof is similar to the one for the reachability games. But now we need to use a alternating parity automaton over sequences of instructions over Γ_2 instead of a normal alternating automaton.

Again it holds that a stack is accepted by the automaton if and only if it is in the winning region of Player 0 in the game. For the acceptance we now have to test which coloring the current stack has and “store” it in the current state. So the automaton can check if Player 0 wins with his parity acceptance condition. Because the winning depends on an infinite condition we can start with some arbitrary color resp. state.

For the test of the color for the current stack we use again the tests in the automaton. We additionally need again to test if the current stack belongs to Player 0 or Player 1 and the tests if some instruction sequence is not defined on the current stack and if it leads outside S or stay inside of it.

In this proof we only show that the winning region is computable by a parity automaton. So we have to prove in the next sections that the alternating parity automata over Γ_2 accept only regular sets of stacks.

Theorem 4.2.1. *For a two player parity game $P = (G, \Omega)$ over Γ_2 with $G = (S, S_0, S_1, In)$ and a coloring $\Omega : Stacks_2(\Gamma) \to \{0, \dots, m\}$ the winning region of Player 0 is computable and regular.*

Proof. Let $A = (Q_A, I_A, \Delta_A, \Omega')$ be an alternating parity automaton over sequences of instructions $In = \{\rho_1, \dots, \rho_n\}$ with tests in $\mathcal{L} \in Alt_2$, where $\mathcal{L}$ is defined as follows:

$$\mathcal{L} = \{S_0, S_1, C_1, \dots, C_m, L_\rho, IN_{\rho'}, OUT_{\rho''} \mid \text{for all } \rho, \rho', \rho'' \in In\},$$

where all L_ρ are defined like in lemma 3.2.5 and $IN_{\rho'}$, $OUT_{\rho''}$ like in lemma 3.2.6.

The definition of A looks then as follows:

- $Q_A = \{p_0, \dots, p_m\}$,
- $I_A = \{p_0\}$,
- Δ_A:
  - if the current stack is in S_0 and so belongs to Player 0:

$$\{p_i, \{T_{S_0}, T_{C_j}\} \to (p_j, \rho_l) \mid i, j \in [0, m], l \in [1, n]\}$$

  - if the current stack is in S_1 and so belongs to Player 1:

$$p_i, \mathcal{T} \to \bigwedge_{\rho \in [\rho_{j_1}, \rho_{j_f}]} (p_u, \rho)$$

    where

$$\begin{aligned} \mathcal{T} \;=\; & \{T_{S_1}, T_{C_u}, T_{L_{\rho_{l_1}}}, \dots, T_{L_{\rho_{l_k}}}, \\ & T_{IN_{\rho_{j_1}}}, \dots, T_{IN_{\rho_{j_f}}}, T_{OUT_{\rho_{g_1}}}, \dots, T_{OUT_{\rho_{g_h}}} \mid \\ & u \in [1, m], l_x, j_y, g_z \in [1, n], \forall x \in [1, k], \\ & y \in [1, f], z \in [1, h], k + f + h = n \text{ and} \\ & \{\rho_{l_1}, \dots, \rho_{l_k}, \rho_{j_1}, \dots, \rho_{j_f}, \rho_{g_1}, \dots, \rho_{g_h}\} = In\} \end{aligned}$$

- $\Omega' : Q_A \to \{0, \dots, m\}$ is defined by:

$$\Omega'(p_i) = i \text{ for all } i \in [0, m]$$

Now we have to show that $s \in \mathcal{S}(A)$ iff $s \in W_0$.

$\boxed{\Rightarrow}$:

Let $s \in \mathcal{S}(A)$. Then there exists an accepting execution for s of A starting in s.

- Case 1: $s \in S_0$
  In this case the execution starts with a transition $p_0, \{T_{S_0}, T_{C_j}\} \rightarrow (p_j, \rho_l)$ for $l \in [1, n]$ where the automaton chooses nondeterministical the right transition, so that $\mathcal{R}(\rho_l)(s)$ is defined and the execution tree rooted by $(p_j, \mathcal{R}(\rho_l)(s))$ is still accepting. With the test T_{C_j} we know that s is colored by j in the game and we color here in the automaton the next state p_j by j, so we delay the coloring one step back, but that does not make any difference.

  By a kind of induction hypothesis we can argue that if $\mathcal{R}(\rho_l)(s)$ is in W_0, because the tree rooted by $(p_j, \mathcal{R}(\rho_l)(s))$ is still accepting, then s has also to be in W_0 because in the game Player 0 can choose the instruction ρ_l to get by $\mathcal{R}(\rho_l)(s)$ into W_0.

- Case 2: $s \in S_1$
  In this case the execution starts with a transition of the form $p_0, \mathcal{T} \rightarrow \bigwedge_{\rho \in [\rho_{j_1}, \rho_{j_f}]}(p_u, \rho)$ with $\mathcal{T}$ defined as above. There is only one transition that can be applied because if we guessed right the instruction sequences $\rho_{l_1}, \ldots, \rho_{l_k}$ have to be exactly the ones that are not defined on the current stack which is verified by the tests $T_{L_{\rho_{l_1}}}, \ldots, T_{L_{\rho_{l_k}}}$, the instruction sequences $\rho_{j_1}, \ldots, \rho_{j_f}$ are those such that their application to the current stack is still in S which is verified by the tests $T_{IN_{\rho_{j_1}}}, \ldots, T_{IN_{\rho_{j_f}}}$ and the instruction sequences $\rho_{g_1}, \ldots, \rho_{g_h}$ are those where the application to the current stack lead outside of S which is verified by the tests $T_{OUT_{\rho_{g_1}}}, \ldots, T_{OUT_{\rho_{g_h}}}$.
  So we apply in the transition just the instruction sequences $\rho_{j_1}, \ldots, \rho_{j_f}$ and get the successor stacks $\mathcal{R}(\rho_{j_i})(s) = s_{j_i}$ for $i \in [1, f]$. Again we know by the test T_{C_j} the color of s and transmit this information to the states. All the according subtrees with root (p_u, s_{j_i}) have to be accepted by the execution and so by induction hypothesis all s_{j_i} for $i \in [1, f]$ are in W_0. So whatever Player 1 chooses it leads always to a stack in W_0 and so it follows that s is also in W_0.

$\boxed{\Leftarrow}$:

Let now $s \in W_0$. Then Player 0 has a winning strategy starting in s so that the game $\eta = \eta_0\eta_1\eta_2 \ldots \in Stacks_2(\Gamma)^\omega$ with $\eta_0 = s$ fulfills the parity condition so that the maximal color that is seen infinitely often is even or the game ends for Player 1 in a deadlock.

The automaton can mimic the strategy of Player 0 by choosing the transition $p_g, \{T_{S_0}, T_{C_j}\} \rightarrow (p_j, \rho_l)$ if Player 0 chooses the instruction ρ_l. And for the case that Player 1 has to choose an instruction the automaton has to branch into all for Player 1 possible instructions by taking the according transition. If the game ends for Player 1 in a deadlock then in the automaton we have a transition $p_g, \{T_{S_1}, T_{C_j}, T_{\rho_1}, \ldots, T_{\rho_n}\} \rightarrow \emptyset$ and the automaton accept the path.

Assume now that $s \in W_0$ but $s \notin \mathcal{S}(A)$ and the automaton mimics the game and the chooses of Player 0 as described above. If $s \notin \mathcal{S}(A)$ then there is at least one path in the execution tree of A, so that the accepting condition is not fulfilled or the path breaks up because no transition fits. But that the path breaks up can just happen for the case that for a configuration (p, s'), $p \in Q_A$, $s' \in S_0$ there is no transition applicable what is equivalent to the case, that there is no instruction in *In* that is defined for s'. However in the game there is the same problem and so Player 0 would not be able to choose an instruction that is defined on s' and so would loose. That is a contradiction to the assumption that $s \in W_0$.

Assume we have an infinite path in the execution that does not fulfill the acceptance condition. We know by the construction of the automaton that the coloring of the stacks in the game is adapted by the coloring of the states in the automaton. So if in the execution the coloring of the states does not fulfill the parity condition then the coloring of the stacks in the game can also not fulfill the parity condition. That is a contradiction to the assumption that $s \in W_0$.

Now we have proven that we can compute the winning region of a parity game by an alternating parity automaton over instruction sequences over Γ_2 with tests in Alt_2.

By the following lemma 4.2.2 we show that we can do the tests as extra branches in the automaton and so we have an equivalent alternating parity automaton over instruction sequences over Γ_2.

From the lemma 4.1.4 we know that the alternating parity automata over instruction sequences are equivalent to the alternating parity automata over Γ_2, so we restrict the following proof to this automata.

By theorem 4.4.9 that follows in the next section we will know that the alternating parity automata accept only regular sets of stacks.

Comments to the complexity of this proof are given in the section 4.5 of this chapter. □

Example 17. *We give now an example for the construction of the alternating parity automaton to compute the winning region of Player 0 for the parity game defined in example 16. So let $A = (\{q_1, q_2\}, \{q_1\}, \Delta, \Omega)$ be an alternating parity automaton over instruction sequences with tests in $\mathcal{L} \subseteq Alt_2$. Here we have $\mathcal{L}$ given by:*

$$\mathcal{L} = \{S_0, S_1, C_1, C_2, L_\rho, IN_{\rho'}, OUT_{\rho''} \mid \textit{for all } \rho, \rho', \rho'' \in In\},$$

where all L_ρ are defined like in lemma 3.2.5 and $IN_{\rho'}$, $OUT_{\rho''}$ like in lemma 3.2.6. The transitions Δ are given by:

$$\begin{aligned}
q_1, \{T_{S_0}, T_{C_1}\} \to (q_1, b\bar{1}) \quad &, \quad q_1, \{T_{S_0}, T_{C_1}\} \to (q_1, \bar{a}1),\\
q_2, \{T_{S_0}, T_{C_1}\} \to (q_1, b\bar{1}) \quad &, \quad q_2, \{T_{S_0}, T_{C_1}\} \to (q_1, \bar{a}1),\\
q_1, \{T_{S_0}, T_{C_2}\} \to (q_2, \bar{1}a) \quad &, \quad q_1, \{T_{S_0}, T_{C_2}\} \to (q_2, \bar{a}1),\\
q_2, \{T_{S_0}, T_{C_2}\} \to (q_2, \bar{1}a) \quad &, \quad q_2, \{T_{S_0}, T_{C_2}\} \to (q_2, \bar{a}1),
\end{aligned}$$

$$\begin{aligned}
q_1, \{T_{S_1}, C_1, L_{\bar{b}\bar{a}b}, L_{\bar{a}1}, L_{\bar{1}a}\} \quad &\to \quad (q_1, \bar{b}ab) \wedge (q_1, 1\bar{b}),\\
q_1, \{T_{S_1}, C_1, L_{\bar{a}1}, L_{\bar{1}a}\} \quad &\to \quad (q_1, \bar{b}ab) \wedge (q_1, \bar{b}\bar{a}b) \wedge (q_1, 1\bar{b})
\end{aligned}$$

The coloring Ω *is defined by* $\Omega(q_1) = 1$ *and* $\Omega(q_2) = 2$*.*

Lemma 4.2.2. *For all alternating parity automata* A *(over instruction sequences) over* Γ_2 *with tests in a finite set of languages* $\mathcal{L} \subset Alt_2$*, there exists an alternating parity automaton (over instruction sequences) over* Γ_2 *so that* $\mathcal{S}(B) = \mathcal{S}(A)$*. Additionally if each* $L \in \mathcal{L}$ *is accepted by an alternating automaton* A_L *(over instruction sequences) over* Γ_2 *then the size of* $|Q_B|$ *is bounded by* $exp[0](|Q_A| + \sum_{L\in\mathcal{L}} |Q_L|)$*.*

Proof. Let $A = (Q_A, I_A, \Delta_A, \Omega)$ be an alternating parity automaton (over instruction sequences) over Γ_2 with tests in $\mathcal{L} \subset Alt_2$. And let all those languages $L \in \mathcal{L}$ be accepted by an alternating automaton $A_L = (Q_L, I_L, \Delta_L)$ (over instruction sequences) over Γ_2. Without loss of generality we can assume that the state sets of those automata are pairwise disjoint.

We construct now an alternating parity automaton $B = (Q_B, I_B, \Delta_B, \Omega')$ (over instruction sequences) over Γ_2 that accepts $\mathcal{S}(A)$, with:

- $Q_B = Q_A \cup \bigcup_{L\in\mathcal{L}} Q_L$
- $I_B = I_A$
- $\Delta_B = \{p, T \to R \wedge \bigwedge_{T_L^2 \in T'} (i_L, \varepsilon) \mid \delta = p, T, T' \to R \in \Delta_A \text{ and } i_L \in I_L\}$ $\cup \bigcup_{L\in\mathcal{L}} \Delta_L$
- $\Omega'(p) = \Omega(p)$ for all $p \in Q_A$ and
  $\Omega'(p) =$ maximal even number for all $p \in Q_L$ for all $L \in \mathcal{L}$

By construction B is equivalent to A. □

4.3 Reduction: alternating parity to reduced alternating parity

The proof that the alternating parity automata are equal to the reduced parity automata is very complicated. We first started with Büchi games instead of parity games and tried to do a proof in a similar way as for the simple case without the parity condition like in section 4.3.2.1 of [Car06], but the construction was hard to understand and the inclusion of the accepting component would have been additionally complicated because we collect states and need to remember if we really see a final state infinitely often in a path like in [MS95].

So we decided to use a result of Vardi in [Var98] that is also outlined in a better understandable way in [Cac02b]. This proof works with the parity condition instead of the Büchi condition and so we also changed our contribution to the parity winning condition. To use the result of Vardi we need to introduce two other automata models, the alternating two-way tree automaton and the nondeterministic one-way tree automata, and establish a connection between this automata and the alternating parity automata respectively the reduced alternating parity automata. The main difference between this automata models is that the alternating two-way and the one-way tree automata run over infinite W-trees and the (reduced) alternating parity automata work on stacks.

To use the result of Vardi and get the connection between this different automata models we now first define this automata models by a short adaption of the definition in [Car06]. After that we proof the equivalence of alternating parity automata over Γ_2 and reduced alternating parity automata over Γ_2 by using of the result of Vardi.

4.3.1 Alternating two-way parity tree automata

If we have a finite set W of directions then the W-tree is a prefix closed set $T \subseteq W^*$. Prefix closed means that if we have $x.d \in T$ with $x \in W^*$ and $d \in W$ then x is also in T. The elements of T are called nodes and the root is denoted by the empty word ε. For every $x.d \in T$ and $d \in W$ the node x is the unique parent of $x.d$ and $x.d$ is a child of x. The direction of a node $x.d$ ($\neq \varepsilon$) is d. The full infinite tree is $T = W^*$. A path (branch) of a tree T is a sequence $\beta \in T^\omega$ so that $\beta = u_0 u_1 u_2 \ldots u_n$ or $\beta = u_0 u_1 u_2 \ldots$, so that $u_0 = \varepsilon$ and $\forall i < n$, $\exists d \in W, u_{i+1} = u_i.d$. A path can be finite or infinite.

Given two finite alphabets W and Σ, a Σ-labeled W-tree is a pair $\langle T, l\rangle$ where T is a W-tree and $l : T \to \Sigma$ maps each node of T to a letter in Σ.

For a finite set X $\mathcal{B}(X)$ is the set of positive Boolean formulas over X, i.e. with the operations $\wedge$, $\vee$, $false$ and $true$. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}(X)$, we say that Y satisfies θ iff assigning $true$ to elements in Y and $false$ to elements in $X \backslash Y$ makes θ true.

To navigate through the tree let $ext(W) := W \uplus \{\varepsilon, \uparrow\}$ be the extension of W. The symbol $\uparrow$ means "to go to parent node" and ε means "stay on the present node". For simplification we define $\forall u \in W^*, d \in W$ that $u.\varepsilon = u$ and $ud \uparrow = u$. The node $\varepsilon \uparrow$ is not defined.

Definition 4.3.1. An *alternating two-way parity tree automaton* over Σ-labeled W-trees is a tuple $A = (Q, \Sigma, \delta, q_I, Acc)$ where

- Q is a finite set of states,
- Σ is the finite input alphabet,
- $\delta : Q \times \Sigma \to \mathcal{B}^+(ext(W) \times Q)$ is the transition function,
- q_I is the initial state, and
- Acc is the acceptance condition.

The run of an alternating two-way tree automaton A over a labeled tree $\langle W^*, l\rangle$ is another labeled tree $\langle T_r, r\rangle$ in which every node is labeled by an element of $W^* \times Q$. The label of a node and its successors have to satisfy the transition function. A run $\langle T_r, r\rangle$ is a Σ_r-labeled Γ-tree, for some (almost arbitrary) set Γ of directions, where $\Sigma_r := W^* \times Q$ and $\langle T_r, r\rangle$ satisfies the following conditions:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_I)$
2. Consider $y \in T_R$ with $r(y) = (x, q)$ and $\delta(q, l(x)) = \theta$. Then there is a (possible empty) set $Y \subseteq ext(W) \times Q$, such that Y satisfies θ, and for all $\langle d, q'\rangle \in Y$, there is $\gamma \in \Gamma$ such that $y.\gamma \in T_r$ and the following holds: $r(y.\gamma) = (x.d, q')$.

A run is accepting if all its infinite path satisfy the acceptance condition *Acc*. The finite paths of a run end with a transition $\theta = true$ are viewed as a successful termination.

In our case we can substitute the acceptance condition *Acc* by the coloring $\Omega : Q \to \{0, \ldots, m\}$ and the condition that the maximal infinitely often seen color is even. In this case we say alternating two-way parity tree automaton.

4.3.2 Nondeterministic one-way parity tree automaton

Definition 4.3.2. A *nondeterministic one-way parity tree automaton* over a Σ-labeled W-trees is a tuple $A = (Q, \Sigma, \delta, Q_I, \Omega)$ where

- Q is a finite set of states,
- Σ is the finite input alphabet,
- $\Delta \subseteq Q \times \Sigma \times Q^{|W|}$ is the transition relation,
- Q_I is the set of initial states, and
- $\Omega : Q \to \{0, \ldots, m\}$ is a coloring.

For a tree t is ρ a run of A on t if

- $\rho(\varepsilon) \in Q_I$
- $\rho(x), t(x), \rho(x1), \ldots, \rho(x|W|)) \in \Delta$ for every x in the tree

A run is accepted, if all its infinite path satisfy the parity acceptance condition that the maximal infinitely often seen color in the path is even.

4.3.3 Proof of theorem 4.3.3

Vardi proofed in [Var98] that for every alternating two-way parity tree automaton A there exists an equivalent nondeterministic one-way parity tree automaton ε, so that $L(A) = L(\varepsilon)$. To get the result that we need we have to proof the following theorem.

Theorem 4.3.3. *Alternating two-way parity tree automata are equivalent to a nondeterministic one-way parity tree automata.*

$$\Longrightarrow$$

Alternating parity automata over Γ_2 are equivalent to reduced alternating parity automata over Γ_2.

Remark 4.3.4. To proof this we have to show:

1. For an alternating parity automaton A over Γ_2 we can construct an equivalent alternating two-way parity tree automaton B over a $(\Gamma_2^Q \cup \{\bullet, \diamond\})$-labeled Γ_2^Q-tree.

Automata on stacks		Tree automata
alt. parity aut. over Γ_2	$\underset{1}{\Longrightarrow}$	alt. two-way parity tree aut.
$\Downarrow$ Thm. 4.3.13		$\Updownarrow$ Vardi
reduced alt. parity aut. over Γ_2	$\overset{2}{\Longleftarrow}$	nondet. one-way parity tree aut.

Figure 7: This figure shows the the connections between the different automata models. We have to show 1 and 2 of remark 4.3.4 to get the result (Theorem 4.3.13) we need.

2. For a nondeterministic one-way parity tree automaton B' over a $(\Gamma_2^O \cup \{\bullet, \diamond\})$-labeled Γ_2^O-tree we can construct an equivalent reduced alternating parity automata A' over Γ_2.

In figure 7 we outline the connections between the different automata models and the proofs. We want to show that we can construct for every alternating parity automaton over Γ_2 an equivalent reduced alternating parity automaton. We do not give the direct proof but use the result of Vardi. For this we have to show that we can construct for every alternating parity automaton over Γ_2 an equivalent alternating two-way parity tree automaton and for every nondeterministic one-way parity tree automaton an equivalent reduced alternating parity automaton over Γ_2. For that we have to define for every stack $s \in Stacks_2(\Gamma)$ a $(\Gamma_2^O \cup \{\bullet, \diamond\})$-labeled Γ_2^O-tree $\langle (\Gamma_2^O)^*, l_s \rangle_s$ and then equivalent means that the particular stack automaton accepts s iff the particular tree automaton accepts $\langle (\Gamma_2^O)^*, l_s \rangle_s$.

To do those proofs we first look in the following remark a bit closer at the connections between the different automata models to get an impression to which incidents we should pay attention.

Remark 4.3.5. Similarities and differences between alternating parity automata over Γ_2 and alternating two-way parity tree automata:

- in the alternating two-way parity tree automata we have as input a Σ-labeled W-tree
- in the alternating parity automata over Γ_2 we have as input a stack $s \in Stacks_2(\Gamma)$
- the execution of the alternating two-way parity tree automata is a tree labeled by elements of $W^* \times Q$
- the execution $\varepsilon = (T, C)$ of the alternating parity automata over Γ_2 is a Γ_2-labeled tree together with a mapping $C : V_T \to Q \times Stacks_2(\Gamma)$
- the $\uparrow$ of the alternating two-way parity tree automata which has the meaning to go to the parent node, corresponds to the $\overline{\gamma}$ of the alternating automata over Γ_2, if γ was the last instruction taken before

- from a node $x \in W^*$ in the tree of an alternating two-way parity tree automaton we can go with all $d \in W$ to a child $x.d \in W^*$ in the tree, with ε we can stay in the node x and with $\uparrow$ to the parent node of x

- from a node $s \in Stacks_2(\Gamma)$ in the execution tree of an alternating automaton over Γ_2 we can go with an instruction $\gamma \in \Gamma$ to a child in the execution tree only if $\mathcal{R}(\gamma)(s)$ is defined and if γ' is the last instruction taken before γ and if $\gamma = \overline{\gamma'}$ then we go back to a stack we have seen before

$\Rightarrow$ that leads to a big difference: the trees the alternating two-way parity tree automata run over are complete and the trees of the executions of the alternating automata over Γ_2 are not

We now have to add a lemma that we need later in the proof and what just claim the obvious property that we can restrict the alternating parity automaton to have just one initial state instead of a set of initial states.

Lemma 4.3.6. *For every alternating parity automaton A over Γ_2 exists an equivalent alternating parity automaton A^+ that has just one initial state instead of a set of initial states.*

Proof. Let $A = (Q_A, I_A, \Delta_A, \Omega)$ be an alternating automaton over Γ_2. Then we define $A^+ = (Q_{A^+}, \{q_I\}, \Delta_{A^+}, \Omega)$ with

- $Q_{A^+} = Q_A \cup \{q_I\}$

- $\Delta_{A^+} = \Delta_A$ and if $I_A = \{i_1, \ldots, i_k\}$ then add for transitions $(i_j, T \to (q_{j_1}, \gamma_{j_1}) \wedge \ldots \wedge (q_{j_{n_j}}, \gamma_{j_{n_j}})) \in \Delta_A$, $j \in [1, k]$ the transitions $(q_I, T \to (q_{j_1}, \gamma_{j_1}) \wedge \ldots \wedge (q_{j_{n_j}}, \gamma_{j_{n_j}}))$, $j \in [1, k]$ to Δ_{A^+}.

The automaton now has to guess the right transition instead of the right initial state. The construction can be done in $\mathcal{O}(|A|)$ and the number of states of Q_{A^+} is in $\mathcal{O}(|Q_A|)$. □

We now define for every stack $s \in Stacks_2(\Gamma)$ a $(\Gamma_2^O \cup \{\bullet, \diamond\})$-labeled Γ_2^O-tree $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$. We will need this tree for both proof of remark 4.3.4. For the proof from alternating parity automaton over Γ_2 to alternating two-way parity automaton we actually need a little other tree that takes also the tests of emptiness into account. But we will first show the proof without the tests and then give a overview to the proof with tests and define there also the according tree.

In the alternating parity automata over Γ_2 we have a stack $s \in Stacks_2(\Gamma)$ as input and in an alternating two-way parity automaton respectively the one-way tree automata we have a Σ-labeled W-tree that means a tuple $\langle W^*, l \rangle$, where l is a labeling function $l : W^* \to \Sigma$. We take now $W = \Gamma_2^O$ and $\Sigma = \Gamma_2^O \cup \{\bullet, \diamond\}$.

The idea is to define a set of trees which all contain all stacks in $Stacks_2(\Gamma)$. In especial for a particular stack $s \in Stacks_2(\Gamma)$ we take s as the root and add as childs those nodes which are reachable by one instruction. This we do for every level where we also make sure that we stay reduced in all paths so that every stack is only seen once in the tree. But this tree as you see in Example 18 for $s = [[a][a]]_2$ is not complete. We can also not store the stacks directly in the tree but by using the instructions Γ_2^O as directions we can conclude from the instruction sequence leading to a node to the stack it should represent. But if we take Γ_2^O as W we have the problem that the application of the instructions onto your start stack s are not always defined or that we may "go up" in the stacks that means they do some non reduced instruction sequence. So we use the labeling function to code the right part of the Γ_2^O-tree respectively we lable a node by the last taken instruction and the part we "do not want" we label by $\diamond$. We need the last taken instruction to detect later when we go up in the tree. We call this to s constructed tree $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$.

Definition 4.3.7. Let s be a stack in $Stacks_2(\Gamma)$. Then we define the $(\Gamma_2^O \cup \{\bullet, \diamond\})$*-labeled* Γ_2^O*-tree* $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$ with the labeling:

$$\begin{aligned} l_s(\varepsilon) &= \bullet \\ l_s(x.d) &= \begin{cases} d & \text{iff } (*) \\ \diamond & \textit{else} \end{cases} \end{aligned}$$

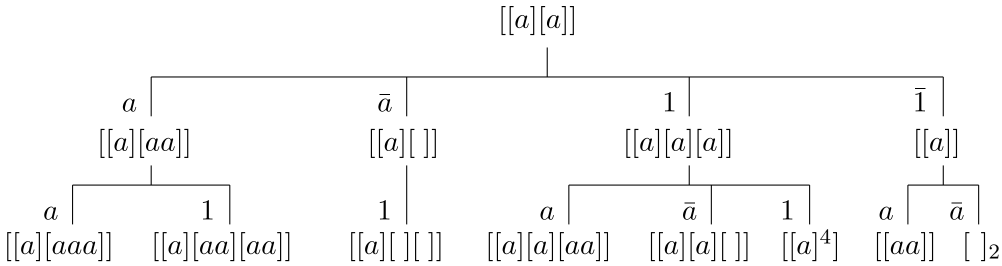
$(*)$:

- $x.d \in (\Gamma_2^O)^*$ is a reduced sequence, i. e. not the reduced sequence that constructs the current stack from the empty stack but the instruction sequence that constructs the current stack from the initial stack s, without visiting a stack twice and
- $\mathcal{R}(xd)(s) = s'$ is defined.

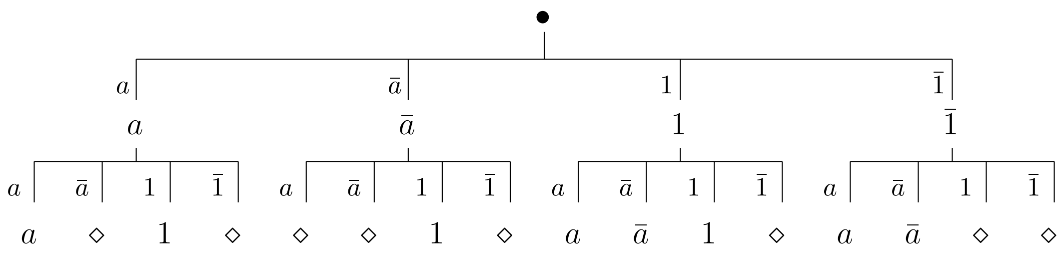
Remark that the tree $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$ is not a regular tree.

Example 18. *To get an intuition for this definition we give the following illustration for some particular stack* $s = [[a][a]]_2$.

First we show how the tree of reduced instruction sequences for $[[a][a]]_2$ *looks. This tree contains all stacks that are reachable from the stack* $[[a][a]]_2$*, which are i.e. all stacks in* $Stacks_2(\Gamma)$*. For each node in the tree we can do at most all instructions of* Γ_2^O *but we can only follow those which are defined and which respect the reduced sequence and so this tree is infinite but not complete. We show here just the first three levels of the tree.*



Now we show how we code this tree by the labeling we defined in definition 4.3.7 into the full $\Gamma_2 \cup \{\bullet, \diamond\}$*-labeled* Γ_2 *tree* $\langle \Gamma_2, l \rangle_{[[a][a]]_2}$*:*



Now we can start with the first part of the proof. We split the proof into two parts. In the first part we do not care about the test of emptiness of the stacks, i.e. $\bot_1$ and $\bot_2$ and make a detailed proof. In the second part we add the test again but do not proof that the construction work because it is obvious by the first part.

Theorem 4.3.8. *For an alternating parity automaton A over Γ_2 we can construct an equivalent alternating two-way parity tree automaton B over a $(\Gamma_2^O \cup \{\bullet, \diamond\})$-labeled Γ_2^O-tree.*

Proof of 4.3.8 without tests. Let $A = (Q_A, I_A, \Delta_A, \Omega)$ be an alternating parity automaton over Γ_2. We assume by theorem 4.3.6 that A has just one initial state q_I'.

Now we can define the alternating two-way parity tree automaton $B = (Q_B, \Gamma_2^O \cup \{\bullet, \diamond\}, \delta, q_I, \Omega')$ over the above defined $(\Gamma_2^O \cup \{\bullet, \diamond\})$-labeled Γ_2^O-trees.

- $Q_B = Q_A$
- $q_I = q_I'$
- δ:
  - let the following transitions δ_i, for $i \in [1, k]$ be all transitions in Δ_A with $Head(\delta_i) = q$ for all $q \in Q_A$:

$$\begin{aligned} \delta_1 &= q \rightarrow (q_{1,1}, \gamma_{1,1}) \wedge \ldots \wedge (q_{1,n_1}, \gamma_{1,n_1}) \\ \delta_2 &= q \rightarrow (q_{2,1}, \gamma_{2,1}) \wedge \ldots \wedge (q_{2,n_2}, \gamma_{2,n_2}) \\ &\vdots \\ \delta_k &= q \rightarrow (q_{k,1}, \gamma_{k,1}) \wedge \ldots \wedge (q_{k,n_k}, \gamma_{k,n_k}) \end{aligned}$$

  - then we have in δ the transition: $\delta(q, l_1) = \tilde{\delta_1^{l_1}} \vee \ldots \vee \tilde{\delta_k^{l_1}}$
  - for all labelings $l_1 \in \Gamma_2^O \cup \{\bullet\}$, for $l_1 = \diamond$ we do not add transitions
  - where for all $i \in [1, k]$: $\tilde{\delta_i^{l_1}} = (d_{i,1}, q_{i,1}) \wedge \ldots \wedge (d_{i,n_i}, q_{i,n_i})$
  - with

$$d_{i,j} = \begin{cases} \gamma_{i,j} & \text{if } l_1 \neq \overline{\gamma_{i,j}} \wedge l_1 \neq \diamond \\ \uparrow & \text{if } l_1 = \overline{\gamma_{i,j}} \wedge l_1 \neq \diamond \end{cases}$$

- $\Omega'(q) = \Omega(q)$, for all $q \in Q_A = Q_B$

Now we have to show that $s \in \mathcal{S}(A)$ iff $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s \in \mathcal{S}(B)$.
$\boxed{\Rightarrow}$:
Let $s \in \mathcal{S}(A)$.

A run of A is an execution $\varepsilon_A = (T_A, C_A)$. Assume ε_A is an accepting run of A. Then we construct an according accepting run T_B of B.

First we can assume without loss of generality, that $V_T = \Psi^*$ so that for all $u, v \in V_T$ holds, if $u \xrightarrow[T_A]{a} v$ then $v = ua$ for $a \in \Psi$. This can be done just by renaming the tree T_A.

Then we define a run T_B of B accepting $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$. For that let T_B be a mapping $T_B : V_T \to Q_B \times (\Gamma_2^O)^*$ defined by

$$T_B(u) = (q, \rho) \quad \text{if} \quad C_A(u) = (q, s')$$

where ρ is the smallest reduced sequence transforming s into s' and $u \in V_T$.

Now we have to proof that T_B is really an accepting run of B. We do this is two parts:

1. Show that T_B is a run of B.
2. Show that T_B is accepting.

$\boxed{\text{For 1.}}$ We have to check for all $u \in V_T$ that for u in T_B a transition of B is applied.

Let for $u \in V_T$ in ε hold that $C(u) = (q, s')$ and the transition $\delta_i \in \Delta_A$, with $\delta_i = q \to (q_1, \gamma_1) \wedge \ldots \wedge (q_n, \gamma_n)$ is applied. We know by the previous definition that $T_B(u) = (q, \rho)$, where ρ is the smallest reduced sequence transforming s into s'. We also have by definition of $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$ that

$$l_s(\rho) = \begin{cases} \bullet & \text{if } \rho = \varepsilon \\ \rho(|\rho|) & \text{otherwise.} \end{cases}$$

The case $l_s(\rho) = \diamond$ does not appear because we already know that $\mathcal{R}(\rho)(s) = s'$ is defined by definition. Let $l_s(\rho) = l$. By the definition of the construction of B there is a transition $\delta(q, l) = \tilde{\delta_1^l} \vee \ldots \vee \tilde{\delta_i^l} \vee \ldots \vee \tilde{\delta_k^l}$, where we now concentrate just on the $\tilde{\delta_i^l} = (d_{i,1}, q_{i,1}) \wedge \ldots \wedge (d_{i,n_i}, q_{i,n_i})$. We can do so because just one of the $\tilde{\delta_j^l}$, $j \in [1, k]$ has to be fulfilled to allow to take the transition.

We now have to show for every $j \in [1, n_i]$ that the pair (d_{i_j}, q_{i_j}) leads to a successor state of u in T_B. By definition of d_{i_j} we have to distinguish two cases:

1. $d_{i_j} = \gamma_{i_j}$, if $l \neq \overline{\gamma_{i_j}}$, and
2. $d_{i_j} = \uparrow$, if $l = \overline{\gamma_{i_j}}$.

The first case is straight forward and we have in T_A the successor state $v = ua \in V_T$ where $C(v) = (q_{i_j}, \mathcal{R}(\gamma_{i_j})(s'))$ and in T_B we have $T_B(v) = (q_{i_j}, \rho\gamma_{i_j})$.

In the second case we have $d_{i_j} = \uparrow$ we have to know the predecessor node v of u that means $u = va$ for an $a \in \Psi$ because we need to go up in the tree again by $\uparrow$ in a copy of v, say v'. In T_A we have for $v' = ub$ with $C(v') = (q_{i_j}, \mathcal{R}(\overline{\gamma_{i_j}})(s'))$ and $\mathcal{R}(\overline{\gamma_{i_j}})(s')$ is the same stack as s'' for $C(v) = (q', s'')$. So we have in T_B,

$T_B(v') = (q_{i_j}, \rho')$ where ρ' is the smallest reduced sequence to transform s into s''.

For 2. It follows directly that if A accepts the execution ε then B has also to accept T_B because the acceptance condition just depends on the states and the states are not changed in the labeling of the trees.

$\Leftarrow$:

Let $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s \in \mathcal{S}(B)$. We now have to show that $s \in \mathcal{S}(A)$.

Let T_B be an accepting run tree of B on $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$. We again define w.l.o.g. a renaming of the nodes in T_B by $V_T = \Psi^*$ so that for all $u, v \in V_T$ holds, if $u \xrightarrow[T_B]{a} v$ then $v = ua$ for $a \in \Psi$. It also holds that for all $u \in V_T$, $T_B(u) \in (Q_B \times (\Gamma_2^O)^*)$. We now want to construct an accepting execution $\varepsilon = (T_A, C)$ of A on s.

For that we have to associate to any node u of V_T a stack s_u so that

$$s_u = \mathcal{R}(\rho)(s) \quad \text{if} \quad T_B(u) = (q, \rho).$$

It is clear that s_u is defined for all $u \in V_T$ because otherwise the input tree $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$ would be labeled at u by $\diamond$ and for this we do not have a suitable transition by construction so that we does not get an complete run.

Now we have for the execution of A the stacks that are associated to the nodes of T_B but we also need the instruction that lead from one stack (resp. node) to a direct successor stack (resp. node). For any two nodes u, v in $Dom(T_B)$ such that v is a successor of u (i. e. $v = ua$ for an $a \in \Psi$) there exists an unique instruction $\gamma_{u \to v} \in \Gamma_2^O$ such that $\mathcal{R}(\gamma_{u \to v})(s_u) = s_v$. We have this property because of the definition of the input tree and of the transition relation of B, where the directions correspond to the instructions.

We define now $Dom(T_A) = V_T$ and the labeling of T_A by:

$$u \xrightarrow[T_A]{\gamma_{u \to v}} v \quad \text{if} \quad u, v \in V_T, v = ua \text{ for an } a \in \Psi$$

and the mapping C is defined for all $u \in V_T$ by:

$$C(u) = (q, s') \quad \text{if} \quad T_B(u) = (q, \rho) \text{ and } s' = \mathcal{R}(\rho)(s) = s_u.$$

Now we have to show that $\varepsilon = (T_A, C)$ is really an accepting execution for A. We do this is two parts:

1. Show that $\varepsilon = (T_A, C)$ is an execution of A.

2. Show that $\varepsilon = (T_A, C)$ is accepting.

For 1. We have to check for all $u \in V_T$ that a transition of A is applied. Let for $u \in V_T$ in T_B with $T_B(u) = (q, \rho)$ the transition $\delta(q, l_1) = \tilde{\delta_1^{l_1}} \vee \ldots \vee \tilde{\delta_k^{l_1}}$ with $l_1 = \rho(|\rho|)$ be applied. That means there exists an $i \in [1, k]$ so that $\tilde{\delta_i^{l_1}} = (d_{i,1}, q_{i,1}) \wedge \ldots \wedge (d_{i,n_i}, q_{i,n_i})$ is the part of the transition that is really taken. The $d_{i,j}$ correspond either to a $\gamma_{i_j} \in \Gamma_2^O$ or to $\uparrow$ for all $j \in [1, n_i]$. For the $\uparrow$ we know by $l_1 = \rho(|\rho|)$ that they correspond to $\overline{l_1}$. So we have in Δ_A by construction the transition $q \rightarrow (\gamma'_{i,1}, q_{i,1}) \wedge \ldots \wedge (\gamma'_{i,n_i}, q_{i,n_i})$ where

$$\gamma'_{i,j} = \begin{cases} d_{i,j} & \text{if } d_{i,j} = \gamma_{i,j} \\ \overline{l_1} & \text{if } d_{i,j} =\uparrow . \end{cases}$$

We now have to show for every $j \in [1, n_i]$ that the pair $(\gamma'_{i,j}, q_{i,j})$ leads to a successor state, say v, of u in T_A and $u \xrightarrow[T_A]{\gamma'_{i,j}} v$.

In T_B we have for all j that we have the successor state v with $T_B(v) = (q_{i,j}, \rho\gamma_{i,j})$ if $d_{i,j} \neq\uparrow$ and so we got by definition in ε that $u \xrightarrow[T_A]{\gamma_{i,j}} v = u \xrightarrow[T_A]{\gamma'_{i,j}} v$ and also $C(v) = (q_{i,j}, s_v)$. If now $d_{i,j} =\uparrow$ then $T_B(v) = (q_{i,j}, \rho')$ for $\rho = \rho' l_1$ and we got by definition that in ε that $u \xrightarrow[T_A]{\overline{l_1}} v = u \xrightarrow[T_A]{\gamma'_{i,j}} v$ and also $C(v) = (q_{i,j}, s_v)$.

For 2. It follows directly that if B accepts T_B then A has also to accept ε because the accepting condition just depends on the states and the states are not changed in the labeling of the trees.

The construction can be done in $\mathcal{O}(|A|)$ and the number of states of Q_B is in $\mathcal{O}(|Q_A|)$. $\square$

Proof of 4.3.8 with tests. If we now add the emptiness tests back to the alternating parity automaton, we have to additionally store in the input tree for the alternating two-way parity automaton, if the "current stack" fulfills the emptiness test of level 1 or level 2. For that we use $\top$ to indicate that the topmost level 1 stack is not empty, $\bot_1$ if the topmost level 1 stack is empty, $\bot_2$ if the stack is the empty level 2 stack and $\diamond$ if we are in the non defined region.

We take now $W = \Gamma_2^O$ and $\Sigma = (\Gamma_2^O \cup \{\bullet, \diamond\}) \times \{\top, \bot_1, \bot_2, \diamond\}$. So in Σ we store the last instruction we have taken to construct the current stack and we store if we have an empty stack or not.

$$\begin{aligned} l_s(\varepsilon) &= (\bullet, e) \qquad \text{iff } (*) \\ l_s(x.d) &= \begin{cases} (d, e) & \text{iff } (**) \\ (\diamond, \diamond) & \text{otherwise} \end{cases} \end{aligned}$$

$(*)$:

- $e = \bot_2$ if $s = [[\ ]]_2$
- $e = \bot_1$ if $s = [[w_1], \ldots, [w_n], [\ ]]_2$ for some $w_1, \ldots, w_n \in \Gamma^*$, $n > 0$
- $e = \top$ otherwise

$(**)$:

- $x.d \in (\Gamma_2^O)^*$ is a reduced sequence, i. e. not the reduced sequence that constructs the current stack from the empty stack but the instruction sequence that constructs the current stack from the initial stack s, without visiting a stack twice
- $\mathcal{R}(xd)(s) = s'$ is defined,

- $e = \bot_2$ if $s' = [[\ ]]_2$
- $e = \bot_1$ if $s' = [[w_1], \ldots, [w_n], [\ ]]_2$ for some $w_1, \ldots, w_n \in \Gamma^*$, $n > 0$
- $e = \top$ otherwise

In the definition of the alternating two-way parity tree automaton we now just add additional conditions if there is a test in the transition of the alternating automaton.

Now we can define similarly to the case without tests the alternating two-way parity tree automaton $B = (Q_B, (\Gamma_2^O \cup \{\bullet, \diamond\}) \times \{\top, \bot_1, \bot_2, \diamond\}, \delta, q_I, \Omega)$ over the above defined $(\Gamma_2^O \cup \{\bullet, \diamond\}) \times \{\top, \bot_1, \bot_2, \diamond\}$-labeled Γ_2^O-trees.

- $Q_B = Q_A$
- $q_I = q'_I$
- δ^2:
  - for $q, T \to (q_1, \gamma_1) \wedge \ldots \wedge (q_n, \gamma_n) \in \Delta_A$
  - then we have in δ for all $l_1 \in \Gamma_2^O \cup \{\bullet\} \times \{\top, \bot_1, \bot_2\}$ the transition:
    * Case 1: $T = \emptyset \Longrightarrow q, l_1 \to (d_1, q_1) \wedge \ldots \wedge (d_n, q_n)$
    * Case 2: $T = \bot_1 \Longrightarrow q, l_1 \to (d_1, q_1) \wedge \ldots \wedge (d_n, q_n)$ only if $l_1 = (\gamma, \bot_1)$ or $l_1 = (\gamma, \bot_2)$ for a $\gamma \in \Gamma_2^O$
    * Case 3: $T = \bot_2 \Longrightarrow q, l_1 \to (d_1, q_1) \wedge \ldots \wedge (d_n, q_n)$ only if $l_1 = (\gamma, \bot_2)$ for a $\gamma \in \Gamma_2^O$
  - with
$$d_{i,j} = \begin{cases} \gamma_{i,j} & \text{if } l_1 \neq \overline{\gamma_{i,j}} \wedge l_1 \neq \diamond \\ \uparrow & \text{if } l_1 = \overline{\gamma_{i,j}} \wedge l_1 \neq \diamond \end{cases}$$
  - but only if $l_1 \neq \diamond$ otherwise we do not add this transition.
- $\Omega'(q) = \Omega(q)$, for all $q \in Q_A = Q_B$

It is clear that this construction works, because it is similar to the construction without tests and so we will not do again the proof. The complexity is also not changed by adding the tests and stays the same as before. □

Example 19. *We give now an example for the construction in the proof of theorem 4.3.8 without tests. The alternating parity automaton A over Γ_2 is chosen very simple because we are here not interested in the set of stacks it recognizes but just in the construction. So let $A = (Q_A, I_A, \Delta_A, \Omega_A)$ be defined by:*

- $Q_A = \{i, p, q\}$
- $I_A = \{i\}$

[2]We look here just at the single transitions of Δ_A for δ we have to put all by nondeterminism possible transitions for some $q \in Q_A$ into one transition by using disjunctions. So we use first the transformation below and do the the disjunction.

- $\Delta_A = \{i \to (p, \overline{a}) \wedge (q, b);$
  $p \to (q, b);$
  $p \to (p, 1) \wedge (p, a);$
  $q \to (p, \overline{b})\}.$

- $\Omega_A(i) = 1,\ \Omega_A(p) = 1,\ \Omega_A(q) = 2$

Then we get by theorem 4.3.8 the equivalent alternating two-way parity tree automaton $B = (Q_B, (\Gamma_2^O \cup \{\bullet, \diamond\}), \delta, q_I, \Omega_B)$ *with:*

- $Q_B = \{i, p, q\}$
- $q_I = i$
- δ :[3]
  - $\delta(i, \bullet) = (\overline{a}, p) \wedge (b, p)$
  - $\delta(p, a) = \delta(p, b) = \delta(p, 1) = (b, q) \vee (1, p) \wedge (a, p)$
  - $\delta(p, \overline{a}) = (b, q) \vee (1, p) \wedge (\uparrow, p)$
  - $\delta(p, \overline{b}) = (\uparrow, q) \vee (1, p) \wedge (a, p)$
  - $\delta(q, \overline{a}) = \delta(q, \overline{b}) = \delta(q, 1) = (\overline{b}, p)$
  - $\delta(q, b) = (\uparrow, p)$
- $\Omega_B(i) = 1,\ \Omega_B(p) = 1,\ \Omega_B(q) = 2$

Now that we have proven the first part of theorem 4.3.3 we start the proof of the second part. For that we have to show that a nondeterministic one-way tree automaton B' over a $(\Gamma_2^O \cup \{\bullet, \diamond\})$-labeled Γ_2^O-tree with parity acceptance condition we can construct an equivalent reduced alternating parity automata A' over Γ_2. Here we restrict to the case without the tests of emptiness again. The proof is already very complicated without the tests but it is again not very complicated to add them again.

Theorem 4.3.9. *For a nondeterministic one-way parity tree automaton* B' *over a* $(\Gamma_2^O \cup \{\bullet, \diamond\})$*-labeled* Γ_2^O*-tree with parity acceptance condition we can construct an equivalent reduced alternating parity automata* A' *over* Γ_2.

We want to construct A' so that

$$B' \text{ accepts a tree } \left\langle (\Gamma_2^O)^*, l_s \right\rangle_s \text{ iff } A' \text{ accepts } s$$

The nondeterministic one-way parity tree automaton B' works over the full $(\Gamma_2^O \cup \{\bullet, \diamond\})$-labeled Γ_2^O-tree but the reduced alternating automaton A' just work over the part of this tree that is not labeled by $\diamond$. The tree $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$ is defined so that if a node is labeled by $\diamond$ then all its successors are also labeled by $\diamond$ which means the whole subtree.

So we define $T_\diamond$ to be the full Γ_2^O-tree where all nodes are labeled with the predicate $\diamond$. We can proof that $T_\diamond$ is regular.

[3]We note here just those transitions where the label is possible, e.g. the $\overline{1}$ is never taken as instruction in Δ_A and so it can not appear as label in δ.

Lemma 4.3.10. *For all $q \in Q_{B'}$, decide if B' has an accepting run on $T_\diamond$ starting in q.*

Proof. Idea: Construct from the transition relation of B' a finite game, where Player 0 chooses a transition and Player 1 pick one of the possible successors. The winning condition of the game is the parity accepting condition with Ω like in the tree automaton B'. For this we only need to consider those transitions of B' that contain as label $\diamond$. Then we got that Player 0 wins the game from state q if and only if in B' is an accepting run on $T_\diamond$ starting with state q. □

With this theorem we can define the state set $Q_\diamond$ as those states that initiate an accepting run on $T_\diamond$.

We introduce here two different proofs for the second part of theorem 4.3.3. In the first one we add tests in Reg_2 to get the information, which instructions are defined on the current stack. In the second version we abstain from this tests and get the information just by guessing and verification.

In the following we introduce two lemmas. We need them to get the information which instructions are defined on the current stack. By this lemmas we define later the languages for the tests in Reg_2.

Lemma 4.3.11. *For every instruction $\gamma \in \Gamma_2^O$ we can define a language $\mathcal{L}_\gamma \in Stacks_2$ so that $s \in \mathcal{L}_\gamma$ iff $\mathcal{R}(\gamma)(s)$ is defined.*

Proof. We need this lemma in principle just for the *pop*-instructions and the $\overline{copy_1}$ because the *push*-instructions and the $copy_1$ are defined on all stacks.

Construct a back-automaton $A = (Q, I, \Delta)$ over Γ_2 with $\mathcal{S}(A) = \mathcal{L}_\gamma$.

- $Q = \{i, f\}$
- $I = \{i\}$
- $\Delta = \{i \xrightarrow{\gamma} f\} \quad \cup \quad \{f \to \emptyset\}$

It remains to show that $\mathcal{L}_\gamma = \mathcal{S}(A)$ holds:

⇒ : Let $s \in \mathcal{L}_\gamma$. Then we have by definition, that $\mathcal{R}(\gamma)(s) = s'$ is defined. Then there is a computation of A, $(i, s_0) \underset{A}{\xrightarrow{\gamma}} (f, s_1) \to \emptyset$, so that $s = s_0$ and $s_1 = s'$. So we have $s \in \mathcal{S}(A)$.

⇐ : Let $s \in \mathcal{S}(A)$. Then there exists a unique computation $(i, s_0) \underset{A}{\xrightarrow{\gamma}} (f, s_1) \underset{A}{\to} \emptyset$ for $s = s_0$ in A. So it follows that $\mathcal{R}(\gamma)(s) = s_1$ is defined and so $s \in \mathcal{L}_\rho$.

The effort of this construction is just constant, because we have always just to perform one step. □

Lemma 4.3.12. *For every stack alphabet Γ there exists a set of languages $L_{i,j}$ for $i \in [0, n]$ for $\Gamma = \{\gamma_1 \dots, \gamma_n\}$ and $j \in \{0, 1\}$, so that the set of all stacks is divided into disjunct sets, so that $\mathcal{R}(\overline{\gamma_i})(s)$ is defined only for γ_i resp. if $i = 0$ then the topmost level 1 stack is empty and additionally if $j = 1$ then $\mathcal{R}(\overline{1})(s)$ is defined for all $s \in L_{i,j}$ resp. if $j = 0$ $\mathcal{R}(\overline{1})(s)$ is not defined.*

Proof. By lemma 4.3.11 we have for every instruction $\gamma \in \Gamma_2^O$ a language exists with $s \in \mathcal{L}_\gamma$ iff $\mathcal{R}(\gamma)(s)$ is defined. We know that these languages are closed under boolean combinations, so we define $L_{i,j}$ like follows:

$$\begin{array}{lrcl} \text{if } i > 0: & & & \\ & L_{i,1} & = & \mathcal{L}_{\overline{\gamma_i}} \cap \mathcal{L}_{\overline{1}} \\ & L_{i,0} & = & \mathcal{L}_{\overline{\gamma_i}} - \mathcal{L}_{\overline{1}} = \mathcal{L}_{\overline{\gamma_i}} \cap \overline{\mathcal{L}_{\overline{1}}} \\ \text{if } i = 0: & & & \\ & L_{0,1} & = & \mathcal{L}_0 \cap \mathcal{L}_{\overline{1}} \\ & L_{0,0} & = & \mathcal{L}_0 - \mathcal{L}_{\overline{1}} = \mathcal{L}_0 \cap \overline{\mathcal{L}_{\overline{1}}} \end{array}$$

where we define L_0 by the following back-automaton $A = (\{i\}, \{i\}, \{i, \bot_1 \to \emptyset\})$ that accepts those stacks that have as topmost level 1 stack the empty stack.

For $|\Gamma| = n$ we have $(n+1) \cdot 2$ different languages $L_{i,j}$ and for every language we need at most an intersection and a complementation of a language $\mathcal{L}_i$, $i \in \{0, \ldots, n, \overline{1}\}$. Because by lemma 4.3.11 we know that for this languages we need just a constant effort we stay here linear in the size of Γ. □

Proof of theorem 4.3.9 with tests

Proof. We have to show that for every nondeterministic one-way parity tree automaton we can construct an equivalent reduced alternating parity automaton over Γ_2. In this proof we add to the reduced alternating parity automaton over Γ_2 the tests in $\mathcal{L} \subseteq Reg_2$. Like shown in lemma 4.2.2 this changes not the expressivity of the automaton. With this tests which use the lemmas 4.3.11 and 4.3.12 we know always which instructions are defined on the current stack. We need this extra information to know which branches of B' we have to follow and which leads to a non defined stack. So let $\mathcal{L} = \{L_{i,j} \mid i \in [0, n], j \in \{0, 1\}\}$ where the $L_{i,j}$ are defined by lemma 4.3.12.

Let $B' = (Q_{B'}, (\Gamma_2^O \cup \{\bullet, \diamond\}), \Delta_{B'}, q_{I_{B'}}, \Omega)$[4] be a nondeterministic one-way parity tree automaton. Then we define the reduced alternating parity automaton $A' = (Q_{A'}, I_{A'}, \Delta_{A'}, \Omega')$ with tests in $\mathcal{L}$ like follows:

- $Q_{A'} = Q_{B'} \times (\Gamma_2^O \cup \{\bullet\})$
- $I_{A'} = \{q_{I_{B'}}\}$
- $\Delta_{A'}$:
  - if $(q, \gamma, q_{\gamma_1}, \ldots, q_{\gamma_m}) \in \Delta_{B'}, \gamma \neq \diamond$
  - then we have in $\Delta_{A'}$ the transitions:

$$(q, \gamma), T_{L_{i,j}} \to \bigwedge_{\gamma_k \in R} ((q_{\gamma_k}, \gamma_k), \gamma_k)$$

for all $i \in [0, n], j \in [0, 1]$ and $q_k \in Q_\diamond$ for all $\gamma_k \in \Gamma_2^O - R$

[4]For the nondeterministic one way parity tree automaton B' we assume an ordering of the set $\Gamma_2^O = \{\gamma_1, \ldots, \gamma_m\}$ so that the i-th son of a node is reached by the direction γ_i that corresponds in the reduced alternating parity automaton to the instruction γ_i.

– where

$$R = \begin{cases} \Gamma \cup \{\overline{\gamma_i}, 1, \overline{1}\} \setminus \{\overline{\gamma}\} & \text{if } i \neq 0, j = 1 \\ \Gamma \cup \{\overline{\gamma_i}, 1\} \setminus \{\overline{\gamma}\} & \text{if } i \neq 0, j = 0 \\ \Gamma \cup \{1, \overline{1}\} \setminus \{\overline{\gamma}\} & \text{if } i = 0, j = 1 \\ \Gamma \cup \{1\} \setminus \{\overline{\gamma}\} & \text{if } i = 0, j = 0, \end{cases}$$

- $\Omega'((q, \gamma)) = \Omega(q)$, for all $q \in Q_{A'}$, $\gamma \in (\Gamma_2^O \cup \{\bullet\})$

The idea is to follow in A' just those instructions that are defined on the current stack and form a reduced sequence. If an instruction is defined on the current stack we know by the tests $T_{L_{i,j}}$ and if it is reduced we know by storing the last instruction in the states. For all other instructions in Γ_2 we have to check that the according state of B' is in $Q_\diamond$ because by the definition of the tree $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$ the subtree contains from there only $\diamond$ and by that we know that it is accepted.

We do not proof that this construction work, but we will proof that the following construction without the tests works and from this it is easy to see that this construction will also work.

□

Proof of theorem 4.3.9 without tests

Now we will proof the theorem 4.3.9 without additional tests. In this case we have to guess which instructions are defined on the current stack and we have to know if we are still on a reduced instruction sequence. We add for this two additional informations to the states. We add the last taken instruction and also a number 0 or 1. The 1 indicates that we are on the path that leads to the empty level 2 stack and the 0 is for all other nodes. Because we have to guess here which instructions are defined we have to make many case differentiations respectively by this differentiations we know for some instructions already if they are defined or not.

Proof of 4.3.9 without tests: If we want to transform the nondeterministic one-way parity tree automaton B' into a reduced alternating automaton A' over Γ_2 we have to know which branches of the transition of B' leads to a non defined stack, so that we have to check, if we are in a state equivalent to a state $q \in Q_\diamond$ or if it leads to a defined stack and we have to follow this branch.

For the operations $push_a$ for all $a \in \Gamma$ and for $copy_1$ the following stack is in general always defined but not always in the reduced tree we work on, where the instruction sequence has to be reduced to lead to a defined stack respectively to a node in the tree that is not labeled by $\diamond$. But we can easily check that by remembering the last taken instruction in the state.

For the operations pop_a for all $a \in \Gamma$ there is just exactly one operation defined namely the one where the letter corresponds to the topmost symbol of the topmost stack. But there can be again the case that this instruction leads to node in the tree that is labeled by $\diamond$. By remembering the last instruction in the state we know if it is a *push* that no *pop* leads to a defined stack. If the last instruction is not a *push* we can guess the *pop* we are allowed to perform.

The operation $\overline{copy_1}$ is just defined if the last operation of the reduced sequence of the current stack is $copy_1$. Because we are in the reduced case the $\overline{copy_1}$ can just appear in the branch, that leads from the initial stack to the empty stack. By a special state set (marked by the 1) we guess this way from the start stack to the empty stack and guess there also if the instruction $\overline{copy_1}$ is defined. In all paths diverging from the path leading to the empty stack we have the property that the last taken instruction is also the last instruction of the reduced sequence that constructs the stack.

Let $B' = (Q_{B'}, (\Gamma_2^O \cup \{\bullet, \diamond\}), \Delta_{B'}, q_{I_{B'}}, \Omega)$ be a nondeterministic one-way parity tree automaton. Then we define the reduced alternating parity automaton $A' = (Q_{A'}, I_{A'}, \Delta_{A'}, \Omega')$ like follows:

- $Q_{A'} = Q_{B'} \times (\Gamma_2^O \cup \{\bullet\}) \times \{0, 1\}$
- $I_{A'} = \{(q_{I_{B'}}, \bullet, 1)\}$
- $\Delta_{A'}$:
  - if $(q, l, q_{\gamma'_1}, \ldots, q_{\gamma'_n}) \in \Delta_{B'}$ where $\Gamma_2 = \{\gamma'_1, \ldots, \gamma'_n\}$ then
  - Case 1: we are diverged from the path to the empty stack, the last taken instruction was $l = \gamma$ and we are in the state $(q, \gamma, 0)$:
    - Case 1.1: and $\perp_1$ holds:

$$(q, \gamma, 0), \{\perp_1\} \rightarrow \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', 0), \gamma') \in \Delta_{A'}$$
$$\text{and } \forall \gamma'' \notin D \; q_{\gamma''} \in Q_\diamond$$

with:

$$\begin{array}{lll} \text{if } \gamma = \overline{\gamma_i} & : & D = \Gamma - \{\gamma_i\} \cup \{1\} \\ \text{if } \gamma = 1 & : & D = \Gamma \cup \{1\} \end{array}$$

    - Case 1.2: and $\perp_1$ holds not:

$$(q, \gamma, 0), \emptyset \rightarrow \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', 0), \gamma') \in \Delta_{A'}$$
$$\text{and } \forall \gamma'' \notin D \; q_{\gamma''} \in Q_\diamond$$

with:

$$\begin{array}{lll} \text{if } \gamma = \gamma_i & : & D = \Gamma \cup \{1\} \\ \text{if } \gamma = \overline{\gamma_i} & : & D = \Gamma - \{\gamma_i\} \cup \{\overline{\gamma_m}, 1\} \\ \text{if } \gamma = 1 & : & D = \Gamma \cup \{\overline{\gamma_m}, 1\} \end{array}$$

  - Case 2: we are on the path to the empty stack, the last taken instruction was $l = \gamma$ and we are in the state $(q, \gamma, 1)$:
    - Case 2.1: and $\perp_1$ holds:

Case 2.1.1: next operation not $\overline{1}$:

$$(q, \gamma, 1), \{\perp_1\} \rightarrow \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma') \in \Delta_{A'}$$
$$\text{and } \forall \gamma'' \notin D \; q_{\gamma''} \in Q_\diamond$$
$$\text{and } \exists \gamma'' \in D \; j_{\gamma''} = 1 \wedge \forall \gamma''' \in D \wedge \gamma''' \neq \gamma'' \rightarrow j_{\gamma'''} = 0$$

with:

$$\begin{array}{lll} \text{if } \gamma = \overline{\gamma_i} & : & D = \Gamma - \{\gamma_i\} \cup \{1\} \\ \text{if } \gamma = 1 & : & D = \Gamma \cup \{1\} \\ \text{if } \gamma = \overline{1} & : & D = \Gamma \end{array}$$

Case 2.1.2: next operation is $\overline{1}$:

$$\begin{aligned}&(q,\gamma,1),\{\bot_1\} \rightarrow \textstyle\bigwedge_{\gamma'\in D}((q_{\gamma'},\gamma',j_{\gamma'}),\gamma') \in \Delta_{A'}\\ &\quad\text{and } \forall\gamma'' \notin D\ q_{\gamma''} \in Q_\diamond\\ &\quad\text{and } j_{\overline{1}} = 1 \wedge \forall\gamma''' \in (D - \{\overline{1}\})\ j_{\gamma'''} = 0\end{aligned}$$

with:

$$\begin{array}{lcl}\text{if } \gamma = \overline{\gamma_i} & : & D = \Gamma - \{\gamma_i\} \cup \{1, \overline{1}\}\\ \text{if } \gamma = \overline{1} & : & D = \Gamma \cup \{\overline{1}\}\end{array}$$

* Case 2.2: and $\bot_1$ holds not:

Case 2.2.1: next operation not $\overline{1}$:

$$\begin{aligned}&(q,\gamma,1),\emptyset \rightarrow \textstyle\bigwedge_{\gamma'\in D}((q_{\gamma'},\gamma',j_{\gamma'}),\gamma') \in \Delta_{A'}\\ &\quad\text{and } \forall\gamma'' \notin D\ q_{\gamma''} \in Q_\diamond\\ &\quad\text{and } \exists\gamma'' \in D\ j_{\gamma''} = 1 \wedge \forall\gamma''' \in D \wedge \gamma''' \neq \gamma'' \rightarrow j_{\gamma'''} = 0\end{aligned}$$

with:

$$\begin{array}{lcl}\text{if } \gamma = \gamma_i & : & D = \Gamma \cup \{1\}\\ \text{if } \gamma = \overline{\gamma_i} & : & D = \Gamma - \{\gamma_i\} \cup \{\overline{\gamma_m}, 1\}\\ \text{if } \gamma = 1 & : & D = \Gamma \cup \{\overline{\gamma_m}, 1\}\\ \text{if } \gamma = \overline{1} & : & D = \Gamma \cup \{\overline{\gamma_m}\}\end{array}$$

Case 2.2.2: next operation is $\overline{1}$:

$$\begin{aligned}&(q,\gamma,1),\emptyset \rightarrow \textstyle\bigwedge_{\gamma'\in D}((q_{\gamma'},\gamma',j_{\gamma'}),\gamma') \in \Delta_{A'}\\ &\quad\text{and } \forall\gamma'' \notin D\ q_{\gamma''} \in Q_\diamond\\ &\quad\text{and } j_{\overline{1}} = 1 \wedge \forall\gamma''' \in (D - \{\overline{1}\})\ j_{\gamma'''} = 0\end{aligned}$$

with:

$$\begin{array}{lcl}\text{if } \gamma = \gamma_i & : & D = \Gamma \cup \{1, \overline{1}\}\\ \text{if } \gamma = \overline{\gamma_i} & : & D = \Gamma - \{\gamma_i\} \cup \{\overline{\gamma_m}, 1, \overline{1}\}\\ \text{if } \gamma = \overline{1} & : & D = \Gamma \cup \{\overline{\gamma_m}, \overline{1}\}\end{array}$$

– Case 3: if we reach the empty level two stack, i.e. $\bot_2$ holds:

$$\begin{aligned}&(q,\gamma,1),\{\bot_2\} \rightarrow \textstyle\bigwedge_{\gamma'\in D}((q_{\gamma'},\gamma',0),\gamma') \in \Delta_{A'}\\ &\quad\text{and } \forall\gamma'' \notin D\ q_{\gamma''} \in Q_\diamond\end{aligned}$$

with:

$$\begin{array}{lcl}\text{if } \gamma = \overline{\gamma_i} & : & D = \Gamma - \{\gamma_i\} \cup \{1\}\\ \text{if } \gamma = \overline{1} & : & D = \Gamma\end{array}$$

- $\Omega' = \Omega$ intersected with the condition that $Q_{B'} \times \Gamma_2^O \times 1$ is not seen infinitely often on a path

Now we have to show that $\langle(\Gamma_2^O)^*, l_s\rangle_s \in \mathcal{S}(B')$ iff $s \in \mathcal{S}(A')$.

$\boxed{\Rightarrow}$:

Let $\langle(\Gamma_2^O)^*, l_s\rangle_s \in \mathcal{S}(B')$.

Then there exists a run of B' on the full $(\Gamma_2^O)^*$-tree labeled by l_s. The run ρ is a labeling of $(\Gamma_2^O)^*$ by states of $Q_{B'}$, i.e. $\rho : (\Gamma_2^O)^* \to Q_{B'}$.

Let now $T_{B'}$ be a by B' accepted tree. Then we define a tree $T'_{B'}$ by a renaming of the nodes of $T_{B'}$ that are not labeled by $\diamond$ by $V_T = \Psi^*$, so that for all $u, v \in V_T$ holds, if $u \overset{a}{\underset{T'_{B'}}{\to}} v$ then $v = ua$ for $a \in \Psi$. We also define $T'_{B'} \in Q_{B'} \times (\Gamma_2^O)^*$, where for all $u \in V_T$ we have $T'_{B'}(u) = (\rho(u), \eta)$ and η is the instruction sequence leading from the root to the node u in $\langle(\Gamma_2^O)^*, l_s\rangle_s$.

We can restrict $T_{B'}$ to $T'_{B'}$ because for all nodes that are in $T_{B'}$ but not in $T'_{B'}$ we know that they are labeled by $\diamond$. We know by the definition of l_s that if we get into a node u in $\langle(\Gamma_2^O)^*, l_s\rangle_s$ that is labeled by $\diamond$ that all successor nodes are also labeled by $\diamond$ and by lemma 4.3.10 we get that we just have to check that $\rho(u) \in Q_\diamond$ to know if the subtree below u is accepted.

We now want to construct an accepting execution $\varepsilon = (T_{A'}, C)$ of A' on s. For that we have to associate to any node u of V_T a stack s_u, so that

$$s_u = \mathcal{R}(\eta)(s) \quad \text{if} \quad T'_{B'}(u) = (q, \eta).$$

Now we need the instruction that lead from one stack (resp. node) to a direct successor stack (resp. node). For any two nodes u, v in $Dom(T'_{B'})$ such that v is a successor of u (i.e. $v = ua$ for an $a \in \Psi$) there exists an unique instruction $\gamma_{u\to v} \in \Gamma_2^O$ such that $\mathcal{R}(\gamma_{u\to v})(s_u) = s_v$. We have this property because of the definition of the input tree and of the transition relation of B', where the directions correspond to the instructions.

We define now $Dom(T_{A'}) = V_T$ and the labeling of $T_{A'}$ by:

$$u \overset{\gamma_{u\to v}}{\underset{T_{A'}}{\longrightarrow}} v \quad \text{if} \quad u, v \in V_T, v = ua \text{ for an } a \in \Psi$$

and the mapping C is defined for all $u \in V_T$ by:

$$C(u) = ((q, \eta(|\eta|), i), s_u) \quad \text{if} \quad T'_{B'}(u) = (q, \eta), s_u = \mathcal{R}(\eta)(s) \text{ and}$$
$$i = \begin{cases} 1 & \text{if we are on the path to } [\,]_2 \\ 0 & \text{otherwise} \end{cases}$$

Now we have to show that $\varepsilon = (T'_{B'}, C)$ is really an accepting execution of A' for s. We do this in two parts:

1. Show that $\varepsilon = (T'_{B'}, C)$ is an execution of A' for s.
2. Show that $\varepsilon = (T'_{B'}, C)$ is accepting.

$\boxed{\text{For 1.}}$ We have to check for all $u \in V_T$ that a transition of A' is applied. Let for $u \in V_T$ in $T'_{B'}$ with $T'_{B'}(u) = (q, \eta)$ the transition $(q, l, q_{\gamma'_1}, \ldots, q_{\gamma'_n}) \in \Delta_{B'}$ with $l = \eta(|\eta|)$ be applied. Then we have to distinguish several cases like in the transition relation.

1. If we are in a node u that is not on the path leading to the empty level 2 stack:

   1.1 and the topmost level 1 stack of s_u is empty:

   1.1.1 and $l = \overline{\gamma_i}$:
   then the transition $(q, \overline{\gamma_i}, 0), \{\bot_1\} \to \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', 0), \gamma')$ is applied and for all $\gamma'' \in \Gamma_2^O - D$ $q_{\gamma''}$ has to be in $Q_\diamond$ for $D = \Gamma - \{\gamma_i\} \cup \{1\}$.
   We have to show that for every $\gamma' \in D$ the tuple $((q_{\gamma'}, \gamma', 0), \gamma')$ leads to a successor state, say v, of u in $T_{A'}$ and $u \xrightarrow[T_{A'}]{\gamma'} v$. With the instructions in D we have exactly those instructions, that are reduced and defined on the current stack, because we can always apply all *push*-instructions, but have because of the reducedness to forbid γ_i because $\overline{\gamma_i}$ was the last instruction that was taken. We can not do a *pop* because the topmost stack is empty. The 1 is always defined and the $\overline{1}$ can not appear because we guessed that we are not on the path to the empty stack.
   In $T'_{B'}$ we have for all $\gamma' \in D$ that for the successor state v, $T'_{B'}(v) = (q_{\gamma'}, \eta\gamma')$ and so we got by the definition of ε that $u \xrightarrow[T_{A'}]{\gamma'} v$ and $C(v) = ((q_{\gamma'}, \gamma', 0), s_v)$.

   1.1.2 and $l = 1$:
   similar to case 1.1.1 but without forbidding the γ_i in D.

   1.2 and the topmost level 1 stack of s_u is not empty:

   1.2.1 and $l = \gamma_i$:
   this case is very similar to the case 1.1.2 but without performing the $\bot_1$-test, but here the $\overline{\gamma_i}$ is forbidden in D because it is not reduced.

   1.2.2 and $l = \overline{\gamma_i}$
   then the transition $(q, \gamma, 0), \emptyset \to \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', 0), \gamma')$ is applied and for all $\gamma'' \in \Gamma_2^O - D$ $q_{\gamma''}$ has to be in $Q_\diamond$ for $D = \Gamma - \{\gamma_i\} \cup \{\overline{\gamma_m}, 1\}$.
   We have to show that for every $\gamma' \in D$ the tuple $((q_{\gamma'}, \gamma', 0), \gamma')$ leads to a successor state, say v, of u in $T_{A'}$ and $u \xrightarrow[T_{A'}]{\gamma'} v$. With the instructions in D we have exactly those instructions, that are reduced and defined on the current stack, because we can always apply all *push*-instructions, but have because of the reducedness to forbid γ_i because $\overline{\gamma_i}$ was the last instruction that was taken. We can do a *pop* but in the transition relation we have to guess which one it is. Now we know the current stack and its top symbol, say the top symbol of s_u is γ_m so the successor state of u is for the instruction $\overline{\gamma_m}$ defined. The 1 is always defined and the $\overline{1}$ can not appear because we guessed that we are not on the path to the empty stack.

1.2.3 and $l = 1$
this case is similar to the case 1.2.1 but without forbidding the γ_i in D.

2. If we are in a node u that is on the path leading to the empty level 2 stack:

2.1 and the topmost level 1 stack of s_u is empty:

2.1.1 and the next operation is not $\overline{1}$:

2.1.1.1 and $l = \overline{\gamma_i}$:
then the transition $(q, \gamma, 1), \{\bot_1\} \rightarrow \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ is applied and for all $\gamma'' \in \Gamma_2^O - D$ $q_{\gamma''}$ has to be in $Q_\diamond$ for $D = \Gamma - \{\gamma_i\} \cup \{1\}$ and additionally the further way to $[\ ]_2$ has to be guessed by setting the $j_{\gamma'}$'s.
We have to show that for every instruction $\gamma' \in D$ the tuple $((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ leads to a successor state, say v, of u in $T_{A'}$ and $u \xrightarrow[T_{A'}]{\gamma'} v$. With the instructions in D we have exactly those instructions, that are reduced and defined on the current stack, because we can always apply all *push*-instructions, but have because of the reducedness to forbid γ_i because $\overline{\gamma_i}$ was the last instruction that was taken. We can not do a *pop* because the topmost stack is empty. The 1 is always defined and the $\overline{1}$ can not appear because we guessed so. But we also guessed that we are on the path to the empty level 2 stack and we now also have to guess which instruction is the next on this path and label it by 1 and the other by 0.
In $T'_{B'}$ we have for all $\gamma' \in D$ that for the successor state v, $T'_{B'}(v) = (q_{\gamma'}, \eta\gamma')$ and so we got by the definition of ε that $u \xrightarrow[T_{A'}]{\gamma'} v$ and $C(v) = ((q_{\gamma'}, \gamma', j_{\gamma'}), s_v)$.

2.1.1.2 and $l = 1$:
similar to case 2.1.1.1 but without forbidding the γ_i in D.

2.1.1.3 and $l = \overline{1}$:
similar to case 2.1.1.2 but without the 1 in D.

2.1.2 and the next operation is $\overline{1}$:

2.1.2.1 and $l = \overline{\gamma_i}$:
then the transition $(q, \gamma, 1), \{\bot_1\} \rightarrow \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ is applied and for all $\gamma'' \in \Gamma_2^O - D$ $q_{\gamma''}$ has to be in $Q_\diamond$ for $D = \Gamma - \{\gamma_i\} \cup \{1\}$ and $j_{\overline{1}} = 1$ and all other $j_{\gamma'} = 0$.
We have to show that for every instruction $\gamma' \in D$ the tuple $((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ leads to a successor state, say v, of u in $T_{A'}$ and $u \xrightarrow[T_{A'}]{\gamma'} v$. With the instructions in D we have exactly those instructions, that are reduced and defined on

the current stack, because we can always apply all *push*-instructions, but have because of the reducedness to forbid γ_i because $\overline{\gamma_i}$ was the last instruction that was taken. We can not do a *pop* because the topmost stack is empty. The 1 is always defined and the $\overline{1}$ appears because we guessed so . The $\overline{1}$ can just appear on the path to the empty level 2 stack so its transition get the 1 to indicate that and all other get the 0.
In $T'_{B'}$ we have for all $\gamma' \in D$ that for the successor state v, $T'_{B'}(v) = (q_{\gamma'}, \eta\gamma')$ and so we got by the definition of ε that $u \xrightarrow[T_{A'}]{\gamma'} v$ and $C(v) = ((q_{\gamma'}, \gamma', j_{\gamma'}), s_v)$.

2.1.2.2 and $l = \overline{1}$:
similar to case 2.1.2.1 but without forbidding the γ_i and without the 1 in D, because the last instruction was the $\overline{1}$.

2.2 and the topmost level 1 stack of s_u is not empty:

2.2.1 and the next operation is not $\overline{1}$:

2.2.1.1 and $l = \gamma_i$:
then the transition $(q, \gamma, 1), \emptyset \rightarrow \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ is applied and for all $\gamma'' \in \Gamma_2^O - D$ $q_{\gamma''}$ has to be in $Q_\diamond$ for $D = \Gamma \cup \{1\}$ and additionally the further way to $[\,]_2$ has to be guessed by setting the $j_{\gamma'}$'s.
We have to show that for every instruction $\gamma' \in D$ the tuple $((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ leads to a successor state, say v, of u in $T_{A'}$ and $u \xrightarrow[T_{A'}]{\gamma'} v$. With the instructions in D we have exactly those instructions, that are reduced and defined on the current stack, because we can always apply all *push*-instructions, but have because of the reducedness to forbid $\overline{\gamma_i}$ because γ_i was the last instruction that was taken. The 1 is always defined and the $\overline{1}$ appears not because we guessed so.
In $T'_{B'}$ we have for all $\gamma' \in D$ that for the successor state v, $T'_{B'}(v) = (q_{\gamma'}, \eta\gamma')$ and so we got by the definition of ε that $u \xrightarrow[T_{A'}]{\gamma'} v$ and $C(v) = ((q_{\gamma'}, \gamma', j_{\gamma'}), s_v)$.

2.2.1.2 and $l = \overline{\gamma_i}$:
then the transition $(q, \gamma, 1), \emptyset \rightarrow \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ is applied and for all $\gamma'' \in \Gamma_2^O - D$ $q_{\gamma''}$ has to be in $Q_\diamond$ for $D = \Gamma - \{\gamma_i\} \cup \{\overline{\gamma_m}, 1\}$ and additionally the further way to $[\,]_2$ has to be guessed by setting the $j_{\gamma'}$'s.
We have to show that for every instruction $\gamma' \in D$ the tuple $((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ leads to a successor state, say v, of u in $T_{A'}$ and $u \xrightarrow[T_{A'}]{\gamma'} v$. With the instructions in D we have exactly those instructions, that are reduced and defined on

the current stack, because we can always apply all *push*-instructions, but have because of the reducedness to forbid γ_i because $\overline{\gamma_i}$ was the last instruction that was taken. We can do a *pop* but in the transition relation we have to guess which one it is. Now we know the current stack and its top symbol, say the top symbol of s_u is γ_m so the successor state of u is for the instruction $\overline{\gamma_m}$ defined. The 1 is always defined and the $\overline{1}$ appears not because we guessed so.
In $T'_{B'}$ we have for all $\gamma' \in D$ that for the successor state v, $T'_{B'}(v) = (q_{\gamma'}, \eta\gamma')$ and so we got by the definition of ε that $u \xrightarrow[T_{A'}]{\gamma'} v$ and $C(v) = ((q_{\gamma'}, \gamma', j_{\gamma'}), s_v)$.

2.2.1.3 and $l = 1$:
similar to case 2.2.1.2 but without forbidding the γ_i in D.

2.2.1.4 and $l = \overline{1}$:
similar to case 2.2.1.3 but without the 1 in D.

2.2.2 and the operation is $\overline{1}$:

2.2.2.1 and $l = \gamma_i$:
then the transition $(q, \gamma, 1), \emptyset \rightarrow \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ is applied and for all $\gamma'' \in \Gamma_2^O - D$ $q_{\gamma''}$ has to be in $Q_\diamond$ for $D = \Gamma - \{\gamma_i\} \cup \{\overline{\gamma_m}, 1\}$ and $j_{\overline{1}} = 1$ and all other $j_{\gamma'} = 0$.
We have to show that for every instruction $\gamma' \in D$ the tuple $((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ leads to a successor state, say v, of u in $T_{A'}$ and $u \xrightarrow[T_{A'}]{\gamma'} v$. With the instructions in D we have exactly those instructions, that are reduced and defined on the current stack, because we can always apply all *push*-instructions. We can not do a *pop* because the last instruction was a *push* so the only defined *pop* is not reduced. The 1 is always defined and the $\overline{1}$ appears because we guessed so. The $\overline{1}$ can just appear on the path to the empty level 2 stack so its transition get the 1 to indicate that and all other get the 0.
In $T'_{B'}$ we have for all $\gamma' \in D$ that for the successor state v, $T'_{B'}(v) = (q_{\gamma'}, \eta\gamma')$ and so we got by the definition of ε that $u \xrightarrow[T_{A'}]{\gamma'} v$ and $C(v) = ((q_{\gamma'}, \gamma', j_{\gamma'}), s_v)$.

2.2.2.2 and $l = \overline{\gamma_i}$:
similar to case 2.2.2.1 but now we again have to forbid γ_i in D because of the reducedness and we have again to guess the right defined *pop* like in case 2.2.1.2 and add it to D.

2.2.2.3 and $l = \overline{1}$:
similar to the case 2.2.2.2 but without the 1 and without forbidding the γ_i in D.

2.3 and we reached the empty level 2 stack, i.e. $s_u = [\ ]_2$:

2.3.1 and $l = \overline{\gamma_i}$:

then the transition $(q,\gamma,1),\{\bot_2\} \to \bigwedge_{\gamma'\in D}((q_{\gamma'},\gamma',0),\gamma')$ is applied and for all $\gamma'' \in \Gamma_2^O - D$ $q_{\gamma''}$ has to be in $Q_\diamond$ for $D = \Gamma - \{\gamma_i\} \cup \{1\}$.
We have to show that for every $\gamma' \in D$ the tuple $((q_{\gamma'},\gamma',j_{\gamma'}),\gamma')$ leads to a successor state, say v, of u in $T_{A'}$ and $u \xrightarrow[T_{A'}]{\gamma'} v$. With the instructions in D we have exactly those instructions, that are reduced and defined on the current stack, because we can always apply all *push*-instructions, but have because of the reducedness to forbid $\overline{\gamma_i}$ because γ_i was the last instruction that was taken. We can not do a *pop* or an $\overline{1}$ because the current stack is $[\ ]_2$. The 1 is always defined.
In $T'_{B'}$ we have for all $\gamma' \in D$ that for the successor state v, $T'_{B'}(v) = (q_{\gamma'},\eta\gamma')$ and so we got by the definition of ε that $u \xrightarrow[T_{A'}]{\gamma'} v$ and $C(v) = ((q_{\gamma'},\gamma',j_{\gamma'}),s_v)$.

2.3.2 and $l = \overline{1}$:
similar to case 2.3.1 but without forbidding the γ_i and without the 1.

For all cases we have for the $\gamma'' \notin D$ that we know that they lead in $T_{B'}$ to a node labeled by $\diamond$ and by lemma 4.3.10 it is clear that it suffice to check that those states $q_{\gamma''}$ are in $Q_\diamond$.

For 2. We have to check that ε is really accepting. The accepting condition of A' has two parts, first it is the same as the one of B' and it just depends on the states. If we set $q \in Q_{B'}$ equal to the states $(q \times \Gamma_2^O \times \{0,1\}) \in Q_{A'}$ then A' accepts the same as B'. In the second part of the accepting condition it is checked that we guessed the right path to $[\ ]_2$, because in this case it is reached and we have just finitely many 1-states in all paths. If we guessed the wrong path and never reach $[\ ]_2$ then the condition would be not fulfilled and we would not accept.

$\Leftarrow$:
Let now $s \in \mathcal{S}(A')$.

A run of A' is an execution $\varepsilon = (T_{A'},C)$. Then we construct an according run ρ on the tree $\langle(\Gamma_2^O)^*,l_s\rangle_s$.

The tree $T_{A'}$ is just a subtree of the run tree of B'. We can w.l.o.g. rename it by $V_T = \Psi^*$ so that for all $u,v \in V_T$ holds, if $u \xrightarrow[T'_{B'}]{a} v$ then $v = ua$ for $a \in \Psi$.

Then we define ρ on $\langle(\Gamma_2^O)^*,l_s\rangle_s$ like follows. Because we work at B' on the full tree instead just on the "defined and reduced part" we need a new extra state $q_\diamond$ and all nodes in the tree $\langle(\Gamma_2^O)^*,l_s\rangle_s$ that are labeled by $\diamond$ are in ρ labeled by $q_\diamond$, i.e. for all $u \in \langle(\Gamma_2^O)^*,l_s\rangle_s$ with $l_s(u) = \diamond$ holds $\rho(u) = q_\diamond$. And we need to add a transition between those states by $(q_\diamond,\diamond,q_\diamond,\ldots,q_\diamond)$. We additionally add to the accepting condition *Acc* that all path that contain infinitely many $q_\diamond$ are accepted. This we can do because we know by the definition of the transition relation $\Delta_{A'}$ that for all instructions that are not defined on the current stack the according state has to be in $Q_\diamond$ and so lead in B' to an accepting subtree.

So we define for all $u \in (\Gamma_2^O)^*$ that $\rho(u) = q_u$ if $l_s(u) \neq \diamond$ and $C(u) = ((q_u, l_s(u), j_u), s_u)$ and for $l_s(u) = \diamond$ we have $\rho(u) = q_\diamond$.

Now we have to proof that ρ is really an accepting run of B'. We do this in two parts:

1. Show that ρ is a run of B'.

2. Show that ρ is accepting.

For 1. We have to check for all $u \in \left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$ that a transition of B' is applied.

If now $l_s(u) = \diamond$ we defined above that the transition $(q_\diamond, \diamond, q_\diamond, \ldots, q_\diamond)$ is applied. Because if we get into an non defined area of $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$, i.e. labeled by $\diamond$, then all successors are also labeled by $\diamond$ and so the transition leads for all $\gamma \in \Gamma_2^O$ to a successor state in $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$ that is labeled by $\diamond$.

Otherwise we have $l_s(u) \neq \diamond$ and in this case we have $u \in V_T$ and in ε that $C(u) = ((q_u, l_s(u), j_u), s_u)$, where j_u is 0 if we are not on the path to $[\ ]_2$ and 1 otherwise and s_u is the current stack. Depending on which instructions are defined, if $\perp_1$ is fulfilled and if we are on the path to $[\ ]_2$ or not the transition δ_u of $\Delta_{A'}$ is chosen. Say $\delta_u = (q_u, l_s(u), j_u) \to \bigwedge_{\gamma' \in D}((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ and for all $\gamma \notin D$ we know the successor state in B' is in $Q_\diamond$ and we defined it here to be $q_\diamond$. For the other $\gamma' \in D$ we have to show that $((q_{\gamma'}, \gamma', j_{\gamma'}), \gamma')$ leads to a successor state of u in $\left\langle (\Gamma_2^O)^*, l_s \right\rangle_s$ that is not labeled by $\diamond$ but that is clear since in D there are exactly the instructions, that are defined on s_u. So we have if v is the successor of u for γ', i.e. $v = ua \in V_T$ for $a \in \Psi$ with $C(v) = ((q_{\gamma'}, \gamma', j_{\gamma'}), \mathcal{R}(\gamma')(s_u))$ that $\rho(v) = q_{\gamma'}$.

For 2. It follows directly that if A' accepts ε then B' has to accept ρ because the accepting condition depends on the states to which in A' just a little more information was added. And for the new paths in the run we defined that they are accepted.

The size of the state set Q'_A is in $\mathcal{O}(|Q_{B'}| \cdot |\Gamma_2| \cdot 2)$ and so linear in the number of the states. □

Now that we have shown the theorem 4.3.3 we can claim the following:

Theorem 4.3.13. *For every alternating parity automaton over Γ_2 there exists an reduced alternating parity automaton over Γ_2.*

Proof. Vardi proofed in [Var98] that for every alternating two-way automaton A there exists an equivalent nondeterministic one-way tree automaton ε, so that $L(A) = L(\varepsilon)$ and with the theorem 4.3.3 the claim follows.

The two parts of theorem 4.3.3 have just linear complexity but the construction of Vardi has an exponential effort. So we got altogether an exponential complexity for the computation of the reduced alternating parity automaton over Γ_2 that is equivalent to the given alternating parity automaton over Γ_2. □

4.4 Reduction: reduced alternating parity to reduced automata with tests

In this section transform the reduced alternating parity automata over Γ_2 first into a reduced and prune alternating automata over Γ_2 with tests in $PAlt_1$[5] and then into a reduced automata over Γ_2 with tests in $PAlt_1$ and show afterwards that $PAlt_1$ is equal to Reg_1. With this and the previous results we can prove that the alternating parity automata over Γ_2 recognize only regular sets of stacks. Finally we receive by the shown results that the winning region of parity games over Γ_2 is regular.

We define now first what it means to be prune for an alternating automaton.

Definition 4.4.1. An alternating parity automaton $A = (Q_A, I_A, \Delta_A, \Omega)$ over Γ_2 is called *prune* if for all $\delta \in \Delta_A$ holds that $|Act(\delta)| \leq 1$.

4.4.1 From reduced to reduced and prune

Our next goal is it to get a prune and reduced alternating automaton over Γ_2 with tests in Alt_1 respectively in $PAlt_1$ out of a reduced alternating parity automaton over Γ_2. The idea for the construction is to guess the path to the empty level 2 stack and "cut of" all branches that lead not to it respectively substitute them by tests in $PAlt_1$. We can do so because in the paths that depart from the path to $[\ ]_2$ the instructions $\overline{1}$ and $\perp_2$ can not appear which is the main property we need for the reduction from level 2 to level 1. The path to $[\ ]_2$ has also the property that it is finite and so we get rid of the parity condition in level 2. For level 1 we still have it in the tests in $PAlt_1$.

More specific to get pruned transitions we define for every transition $\delta = p, T \rightarrow R \in \Delta_A$ and for every $\gamma \in \Gamma_2^O$ the prune transition $\tilde{\delta}_\gamma = p, T \rightarrow R \cap (Q_A \times \{\gamma\})$ and "the rest" as $\delta_\gamma = \bullet \rightarrow R \cap (Q_A \times (\Gamma_2^O - \{\gamma\}))$. In the reduced and prune alternating automaton we guess by the transitions $\tilde{\delta}_\gamma$ the way to $[\ ]_2$ and test by some languages that are constructed by some automaton $A_{\delta,\gamma}$ if for "the rest"-transitions the current stack is also in the language of the original automaton. To do this we can as mentioned before restrict the automaton to Γ_1.

We know for every stack $s = [s_1, \ldots, s_n]_2$ with $Last(s) = \gamma$ (this is to be sure that it is $\overline{\gamma}$ that is the next instruction toward the empty stack of level 2) that $s \in \mathcal{S}_{\delta_{\overline{\gamma}}}(A)$ iff $s_n \in \mathcal{S}(A_{\delta,\overline{\gamma}})$, where $\mathcal{S}_{\delta_{\overline{\gamma}}}(A)$ means the stacks, that are accepted by A when starting with transition $\delta_{\overline{\gamma}}$. We get those automata by the following two lemmas.

Lemma 4.4.2. *For all reduced alternating parity automata A over Γ_2, for all $\gamma \in \Gamma_2^O$ and for all transitions $\delta \in \Delta_A$ so that $\overline{\gamma} \notin \pi_2(Act(\delta))$ and $Test(\delta) = \emptyset$ there exists an alternating parity automaton $A_{\delta,\overline{\gamma}}$ over Γ_1, so that for all stacks $s = [s_1, \ldots, s_n]_2$ with $Last(s) = \gamma$ holds:*

$$s \in \mathcal{S}_\delta(A) \Leftrightarrow s_n \in \mathcal{S}(A_{\delta,\overline{\gamma}}).$$

[5] We will note the languages that can be accepted by an alternating parity automaton over Γ_1 by $PAlt_1$. We will show in section 4.4.3 that $PAlt_1 = Reg_1$.

Additionally $|A_{\delta_0,\overline{\gamma}}|$*is in* $\mathcal{O}(A)$ *and* $|Q_{A_{\delta_0,\overline{\gamma}}}|$ *is in* $\mathcal{O}(Q_A)$*.*

Proof. Let now $A = (Q_A, I_A, \Delta_A, \Omega)$ be an reduced alternating parity automaton over Γ_2, γ a symbol of Γ_2^O and δ_0 a transition of Δ_A so that $\overline{\gamma} \notin \pi_2(Act(\delta_0))$ and $Test(\delta_0) = \emptyset$.

We can show that we can forget about the transitions in Δ_A that contain the instructions $\overline{1}$ and $\bot_2$ without changing $\mathcal{S}_{\delta_0}(A) \cap \{s \in Stacks_2(\Gamma) \mid Last(s) = \gamma\}$.

Let $s = [s_1, \ldots, s_n]_2$ have the reduced sequence ρ_s with $\rho_s(|\rho_s|) = \gamma$ and so it holds $Last(s) = \gamma$. Intuitive we have because we forbid the $\overline{\gamma}$ in δ_0 and because the automaton is reduced, that $[\ ]_2$ can from s never be reached and so in the execution of s the $\overline{1}$ and the $\bot_2$ can never appear. Now we want to proof this more formal.

Suppose that s is accepted by the execution $\varepsilon_A = (T_A, C_A)$ of A when started with δ_0. We have to show that $\overline{1}$ does not appear in the labeling of T_A.

Let $u \in V_{T_A}$ with $C_A(u) = (p, s')$. Then there exists $\rho' \in (\Gamma_2^O)^*$ so that $root(T_A) \xrightarrow[A]{\rho'} u$. It holds per definition that $s' = \mathcal{R}(\rho')(s)$, and so $s = \mathcal{R}(\rho_s\rho')([\ ]_2)$. The sequence $\rho_s\rho'$ is reduced because ρ_s is reduced by definition and ρ' is reduced because A is reduced and the concatenation of both is reduced because of the definition of δ, i.e. $\overline{\gamma} \notin \pi_2(Act(\delta_0))$ and so $\rho'(1) \neq \overline{\gamma} = \overline{(\rho_s(|\rho_s|))}$. By remark 4.1.11 of [Car06] page 84 which says that in a reduced sequence the $\overline{1}$ and the $\bot_2$ never appear, we know that $\overline{1}$ can not appear in ρ'. The same holds for $\bot_2$.

Now we construct an alternating parity automaton $A_{\delta_0,\overline{\gamma}}$ defined by the tuple $(Q_A \cup \{i_0\}, \{i_0\}, \Delta_{A_{\delta_0,\overline{\gamma}}}, \Omega')$ where i_0 is a new state that does not appear in Q_A and Ω' is equal to Ω but there is also a color for i_0 defined, which color it is means less since it only appears once. The set $\Delta_{A_{\delta_0,\overline{\gamma}}}$ is obtained from Δ_A by replacing all 1 by ε. So define $\Delta_{A_{\delta,\overline{\gamma}}}$ by:

$$\begin{array}{ll} & \{p, T \to (p_1, \tilde{\gamma_1}) \wedge \ldots \wedge (p_n, \tilde{\gamma_n}) \mid p, T \to (p_1, \gamma_1) \wedge \ldots \wedge (p_n, \gamma_n) \in \Delta_A\} \\ \cup & \{i_0, T \to (p_1, \tilde{\gamma_1}) \wedge \ldots \wedge (p_n, \tilde{\gamma_n}) \mid \delta_0 = p, T \to (p_1, \gamma_1) \wedge \ldots \wedge (p_n, \gamma_n)\} \end{array}$$

where for all $\gamma \in \Gamma_1^0$, $\tilde{\gamma} = \gamma$ and $\tilde{1} = \varepsilon$.

By this construction we get an alternating parity automaton over Γ_1 with ε-transitions. By construction and by lemma 2.1.2 we have that for all stacks $s = [s_1, \ldots, s_n]_2$ with $Last(s) = \gamma$, $s \in \mathcal{S}_{\delta_0}(A)$ iff $s_n \in \mathcal{S}(A_{\delta_0,\gamma})$.

The automaton $A_{\delta_0,\overline{\gamma}}$ is exponential in the size of the automaton A, if we eliminate the ε-transitions otherwise it is just linear. □

In the previous lemma we excluded the case, that $s = [\ ]_2$ and so $Last(s)$ is not defined. For that case the following lemma is introduced, where the proof is adapted of the previous lemma and the complexity is again linear.

Lemma 4.4.3. *For all reduced alternating parity automata* A *over* Γ_2 *and for all transitions* $\delta \in \Delta_A$ *there exists an alternating parity automaton* $A_{\delta,\varepsilon}$ *over* Γ_1*, so that:*

$$[]_2 \in \mathcal{S}_\delta(A) \Leftrightarrow []_1 \in \mathcal{S}(A_{\delta,\varepsilon}).$$

Additionally $|A_{\delta,\varepsilon}|$*is in* $\mathcal{O}(A)$ *and* $|Q_{A_{\delta,\varepsilon}}|$ *is in* $\mathcal{O}(Q_A)$*.*

Now we can construct a reduced and prune alternating automaton over Γ_2 with test in $PAlt_1$ that is equivalent to a reduced alternating parity automaton over Γ_2.

Theorem 4.4.4. *For every reduced alternating parity automata A over Γ_2 there exists an equivalent reduced and prune alternating automaton B over Γ_2 with tests in $\mathcal{L} \subset PAlt_1$ such that every accepting execution of B ends in $[\ ]_2$.*

Proof. Let $A = (Q_A, I_A, \Delta_A, \Omega)$ be a reduced alternating parity automaton over Γ_2. We construct an equivalent reduced and prune alternating automaton B over Γ_2 with tests in $\mathcal{L} \subset PAlt_1$ such that every accepting execution of B ends in $[\ ]_2$.

For that we now first define the language $\mathcal{L}$. Therefore we need the already known splitting of the transitions $\delta \in \Delta_A$ into $\tilde{\delta_\gamma}$ and δ_γ defined by:

$$\begin{aligned} &\text{for all } \delta = p, T \to R \in \Delta_A \text{ and all } \gamma \in \Gamma_2^O \\ &\tilde{\delta_\gamma} = p, T \to R \cap (Q_A \times \{\gamma\}) \text{ and} \\ &\delta_\gamma = \bullet \to R \cap (Q_A \times (\Gamma_1^0 - \{\gamma\})), \end{aligned}$$

where $\bullet$ is a new symbol that does not appear in Q_A.

By lemma 4.4.2 we get the alternating parity automaton $A_{\delta,\gamma}$ over Γ_1 for the automaton $(Q_A, I_A, \Delta_A \cup \{\delta_\gamma\}, \Omega)$, for every transition $\delta \in \Delta_A$ and every $\gamma \in \Gamma_2^O$. Additionally we get by lemma 4.4.3 for every transition $\delta = p, T \to R \in \Delta_A$ by defining $\delta_\varepsilon = \bullet \to R$ an alternating parity automaton $A_{\delta,\varepsilon}$ over Γ_1. If we put together the languages of all this automata we get $\mathcal{L} \subset PAlt_1$ the language of the tests we need:

$$\mathcal{L} = \{\mathcal{S}(A_{\delta,\gamma}), \mathcal{S}(A_{\delta,\varepsilon}) \mid \delta \in \Delta_A \text{ and} \gamma \in \Gamma_2^O\}.$$

We define the reduced and prune alternating automaton $B = (Q_B, I_B, \Delta_B)$ over Γ_2 with test in $\mathcal{L}$ with:

- $Q_B = (Q_A \cup \{\bullet\}) \times (\Gamma_1^0 \cup \{\varepsilon, 1\})$
- $I_B = I_A \times (\Gamma_1^0 \cup \{\varepsilon, 1\})$
- $$\begin{aligned} \Delta_B = &\{(p,\gamma), T, T_{\mathcal{S}(A_{\delta,\overline{\gamma}})} \to ((q,\gamma'),\overline{\gamma}) \mid \delta \in \Delta_A \text{ and } \tilde{\delta_{\overline{\gamma}}} = p, T \to (q,\overline{\gamma})\} \\ \cup\ &\{(p,\gamma), T, T_{\mathcal{S}(A_{\delta,\overline{\gamma}})} \to ((\bullet,\gamma'),\overline{\gamma}) \mid \delta \in \Delta_A \text{ and } \tilde{\delta_{\overline{\gamma}}} = p, T \to \emptyset\} \\ \cup\ &\{(p,\varepsilon), \bot_2, T_{\mathcal{S}(A_{\delta,\varepsilon})} \to \emptyset \mid \delta \in \Delta_A \text{ and } Head(\delta) = p\} \\ \cup\ &\{(\bullet,\gamma) \to ((\bullet,\gamma'),\overline{\gamma})\} \\ \cup\ &\{(\bullet,\varepsilon), \bot_2 \to \emptyset\} \end{aligned}$$

  where in all sets holds that, $\gamma \in \Gamma_1^0 \cup \{1\}$ and $\gamma' \in \Gamma_1^0 \cup \{1, \varepsilon\}$ with $\gamma' \neq \overline{\gamma}$.

The idea of the construction is the following. In the second component of the states we guess the last instruction of the reduced sequence of the current stack say s, i.e. $Last(s) = \gamma$, and then do the opposite instruction $\overline{\gamma}$ to go toward $[\ ]_2$. By the test $T_{\mathcal{S}(A_{\delta,\overline{\gamma}})}$ we check that the other paths, which we do not longer follow, are accepting. We have then two cases how to reach $[\ ]_2$. The first is that the automaton A reaches $[\ ]_2$ and wants to go on. In this case we have to guess by the third transition set above that we reach $[\,]_2$ and check this by the test $\bot_2$ and then accept. The other case is that A does normally not reach $[\,]_2$. In this case we stop in A at some point $\emptyset$. In this case we go in B into the new state $\bullet$ (see second transition set). From there we stay in $\bullet$ and go on by guessing the instructions to reach $[\,]_2$ (see fourth transition set). If we reach $[\,]_2$ which we check by the test $\bot_2$ we accept (see last transition set).

It is clear by construction, that B is a reduced and prune alternating automaton and that every accepting execution of B ends in $[\,]_2$. Now we have to proof that $\mathcal{S}(A) = \mathcal{S}(B)$.

$\boxed{\mathcal{S}(A) \subseteq \mathcal{S}(B)\text{:}}$

So let $\varepsilon_A = (T_A, C_A)$ be an accepting execution of A for a stack s. Now we have to show that there is also an accepting execution for s in B. For that we define $\varepsilon_B = (T_B, C_B)$ like follows. Let ρ_s be the reduced sequence of s and let $\overline{\rho_s} = \rho_1 \dots \rho_n$ then we define $V_{T_B} = u_0, \dots, u_n$ and $u_{i-1} \xrightarrow[T_B]{\rho_i} u_i$ for $i \in [1, n]$ and $C_B(u_i) = ((p, \overline{\rho_{i+1}}), s')$ where $C_A(u_i) = (p, s')$ and $\mathcal{R}(\rho_1 \dots \rho_i)(s) = s'$, for the case that $[\,]_2$ is reached in A. For the case that in A the execution goes just j steps into the direction of $[\,]_2$ we have for $i \leq j$, $C_B(u_i) = ((p, \overline{\rho_{i+1}}), s')$ where $C_A(u_i) = (p, s')$ and for $i > j$, $C_B(u_i) = ((\bullet, \overline{\rho_{i+1}}), s')$, where $\mathcal{R}(\rho_1 \dots \rho_i)(s) = s'$. By the lemmas 4.4.2 and 4.4.3 we get for the path that we substitute in B by tests, that the test are fulfilled if and only if the according paths in A are accepting.

$\boxed{\mathcal{S}(A) \supseteq \mathcal{S}(B)\text{:}}$

We have now to show that for all accepting executions ε_B of B there exists an accepting execution in A.

If we therefore have for $\varepsilon_B = (T_B, C_B)$ that if $C_B(root(V_{T_B})) = ((p, \gamma), s)$ then we have in $\varepsilon_A = (T_A, C_A)$, $C_A(root(V_{T_A}) = (p, s)$, if $Last(s) = \gamma$. If we go in ε_B by a transition from one state in T_B to the successor state then we have in ε to go to the same state but additionally we have to add the subtrees that are in B substituted by the test $T_{\mathcal{S}(A_{\delta,\overline{\gamma}})}$. We get the correctness of this by the lemmas 4.4.2 and 4.4.3.

The size of Δ_B is polynomial in the size of Δ_A, $|\mathcal{L}| \leq 3 \cdot |\Delta_A|$ and for all $L \in \mathcal{L}$ there exists an alternating automaton A_L over Γ_1 with $|Q_{A_L}|$ and $|A_L|$ that are bounded by $exp[0](|Q_A|)$ and $exp[0](|A|)$.

□

4.4.2 From reduced and prune to non alternating

Now it is left to show two things. First we show that each reduced and prune alternating automaton over Γ_2 with test in $\mathcal{L} \subset PAlt_1$ is equivalent to a reduced automaton over Γ_2 with test in $\mathcal{L} \subset PAlt_1$. Then we have also to show that the alternating parity automata over Γ_1 accept only Reg_1.

Theorem 4.4.5. *For each reduced and prune alternating automaton over Γ_2 with test in $\mathcal{L} \subset PAlt_1$ such that every accepting execution of B ends in $[\,]_2$ there is an equivalent reduced automaton over Γ_2 with test in $\mathcal{L} \subset PAlt_1$.*

Proof. Let $A = (Q_A, I_A, \Delta_A)$ be a reduced and prune alternating automata over Γ_2 with test in $\mathcal{L}$ such that every accepting execution of B ends in $[\ ]_2$. Then we construct an equivalent reduced automaton $B = (Q_B, I_B, F_B, \Delta_B)$ over Γ_2 with test in $\mathcal{L}$ like follows:

- $Q_B = Q_A \times (\Gamma_2^O \cup \{\varepsilon\})$
- $I_B = \{(q, \varepsilon) \mid q, \bot_2, T' \to \emptyset \in \Delta_A\}$
- $F_B = I_A \times (\Gamma_2^O \cup \{\varepsilon\})$
- $\Delta_B = \{(q, \gamma) \xrightarrow{\gamma'} (p, \gamma'), T, T' \mid p, T, T' \to (q, \overline{\gamma'}) \in \Delta_A \text{ and } \gamma' \neq \overline{\gamma}\}$

We need the last instruction in the states as additional information to make sure that we stay reduced. The general idea is just to reverse the transitions including the instructions.

By construction and by lemma 2.1.1 we have that $\mathcal{S}(A) = \mathcal{S}(B)$.

The number of states $|Q_B|$ is polynomial in the number of states $|Q_A|$ together with the size of Γ_2^O. □

4.4.3 $PAlt_1 = Reg_1$

To show that the alternating parity automata over Γ_1 accept exactly the regular sets of stacks of level 1 (Reg_1) we do a similar construction as for the theorem 4.4.4. By adapting a result from Serre [Ser03] it is also possible to show that the winning region of a higher-order pushdown parity game of level 1 can be represented by a NFA of exponential size in the size of the higher-order pushdown automaton of level 1 that defines the game.

We do here the proof in our terms. For that we can assume we have a reduced alternating parity automaton A over Γ_1, because Theorem 4.3.13 for alternating parity automata over Γ_2 does also hold for Γ_1. Now we need to guess the path from the input level 1 stack s to the empty level 1 stack $[\ ]_1$ which is just a sequence of pops. So we define for every transition $\delta = p, T \to R \in \Delta_A$ the transitions $\tilde{\delta} = p, T \to \cap(Q_A \times \overline{\Gamma})$ and $\delta' = p \to R \cap (Q_A \times \Gamma)$. Because the automaton we assume is reduced we know that all executions that start with a transition δ' are completely labeled by Γ, i.e. by *push*'s.

By the construction of the reduced and prune automaton we indeed not add all transitions $\tilde{\delta}$. We first have to check that the language of the according transition δ', i.e. $\mathcal{S}(A_{\delta'})$ is not empty. It is obvious that $\mathcal{S}(A_{\delta'})$ is either $\emptyset$ or $Stacks_1(\Gamma)$, because as remarked above in the execution there are just *push*-instructions performed and so it is clear that the accepting does not depend on the stack but just on the transitions. We will proof this in the next lemma.

Lemma 4.4.6. *For every reduced alternating parity automaton A over Γ_1 defined by $(Q_A, I_A, \Delta_A, \Omega)$ and for all transitions $\delta \in \Delta_A$, so that $Test(\delta) = \emptyset$ and $\pi_2(Act(\delta)) \cap \overline{\Gamma} = \emptyset$, it holds that $\mathcal{S}_\delta(A)$ is either equals $\emptyset$ or equals $Stack_1(\Gamma)$. can be decided in $\mathcal{O}(|A|)$.*

Proof. Let now $A = (Q_A, I_A, \Delta_A, \Omega)$ be an reduced alternating parity automaton over Γ_1 that satisfies the required conditions. Because we are only interested in $\mathcal{S}_\delta(A)$ and for δ we know that $Test(\delta) = \emptyset$ and $\pi_2(Act(\delta)) \cap \overline{\Gamma} = \emptyset$, it is clear that for all $\delta \in \Delta_A$, $Test(\delta) = \emptyset$ and $\pi_2(Act(\delta)) \subseteq \Gamma$.

We show now that if $\mathcal{S}_\delta(A)$ is not empty then it is equal to $Stacks_1(\Gamma)$. Assume there exists a stack s that is accepted by A by the execution $\varepsilon_A = (T_A, C_A)$ and the execution ε_A starts in δ with s. By the definition of an execution it holds that for all $u \in V_{T_A}$ the stack $\pi_2(C_A(u))$ is equal to $\mathcal{R}(\rho_u)(s)$ where $\rho_u \in (\Gamma_1^0)^*$ has the form such that $root(T_A) \xrightarrow[T_A]{\rho_u} u$. But as A is reduced and the execution starts with the transition δ where $\pi_2(Act(\delta)) \cap \overline{\Gamma} = \emptyset$ we know that $\rho_u \in \Gamma^*$.

Let now $s' \in Stacks_1(\Gamma)$. We define the execution $\varepsilon'_A = (T_A, C'_A)$ that accepts s' like follows. For all $u \in V_{T_A}$ we define $C'_A(u) = (\pi_1(C_A(u)), \mathcal{R}(\rho_u)(s'))$. So we just adopt the states of C_A and change only the stacks. We can do this because just *push* instructions are performed which are defined on every stack. That ε' is accepting is therefore clear, because the accepting depends just on the states which are not changed. It is also obvious that ε' starts in s' by transition δ and so $s' \in \mathcal{S}_\delta(A)$.

For the test of emptiness of $\mathcal{S}_\delta(A)$ it suffice to do a test of emptiness in a nondeterministic top-down tree automata with parity acceptance condition Ω, because as we already seen in the transitions of the restricted automaton for $\mathcal{S}_\delta(A)$ there are no *pop* instructions. This can be done for an automaton with n states and k colors in $\mathcal{O}(n^k)$ like we see in [KV98].

It seems here that we get by this another exponential blow-up but in fact the step to go from an alternating parity automaton of level 1 to a reduced alternating parity automaton of level 1 is exponential in the number of state but linear for the number of parity. So the test of emptiness is simply exponential in the size of the alternating parity automaton of level 1 and we do not get an additional exponential blow-up. □

We show now first that we can make every reduced alternating parity automaton over Γ_1 prune and then we can, by a similar proof as for level 2, show that we can construct for every the reduced and prune alternating automata over Γ_1 an equivalent reduced automaton over Γ_1.

Theorem 4.4.7. *For all reduced alternating parity automata A over Γ_1 it exists a reduced automaton C over Γ_1 so that $\mathcal{S}(A) = \mathcal{S}(C)$. Additionally $|Q_C| \leq |\Gamma_1| \cdot (|Q_A| + 1)$.*

Proof. Let $A = (Q_A, I_A, \Delta_A, \Omega)$ be a reduced alternating parity automaton over Γ_1. We can assume without loss of generality that $|\pi_2(Act(\delta)) \cap \overline{\Gamma}| \leq 1$ because there can never be two different *pop*s be defined on the same stack, i.e. for all $x \neq y \in \Gamma$, $Dom(\mathcal{R}(\overline{x})) \cap Dom(\mathcal{R}(\overline{y})) = \emptyset$.

To get the reduced automaton C we first define a reduced and prune alternating automaton $B = (Q_B, I_B, \Delta_B)$ over Γ_1 which performs only *pop* operations and tests of emptiness of level 1. For that we define for every transition $\delta = p, T \to R \in \Delta_A$ the transitions $\tilde{\delta} = p, T \to R \cap (Q_A \times \overline{\Gamma})$ and

$\delta' = \bullet \to R \cap (Q_A \times \Gamma)$, where $\bullet$ is a new symbol that is not in Q_A. It is clear that the δ' fulfill the condition of lemma 4.4.6 and if we define for every δ' the automaton $A_{\delta'}$ by just adding δ' to Δ_A, we can apply the lemma 4.4.6 to filter out the right transitions $\tilde{\delta}$.

So we define B by $Q_B = Q_A$, $I_B = I_A$ and Δ_B by:

$$\Delta_B = \{\tilde{\delta} \mid \delta \in \Delta_A \text{ and } \mathcal{S}_{\delta'}(A_{\delta'}) \neq \emptyset\}.$$

By construction B is prune and if A is reduced then also B has to be reduced. It is also clear that for every stack in $\mathcal{S}(B)$ the execution is finite, because by the transitions in Δ_B we just destruct the input stack and accept. It remains to show that $\mathcal{S}(A) = \mathcal{S}(B)$. By construction it holds that $\mathcal{S}(A) \subseteq \mathcal{S}(B)$. For the other direction we have to show that for all accepting executions of B there exists also an accepting execution of A. This is clear since for all $\tilde{\delta} \in \Delta_B$, $\mathcal{S}_{\delta'}(A)$ is equal to $Stacks_1(\Gamma)$, by lemma 4.4.6.

By the following lemma 4.4.8 we get that for every reduced and prune automaton over Γ_1 which performs only *pop*s and tests of emptiness we can construct a reduced automaton C over Γ_1 that is equivalent to B. □

Lemma 4.4.8. *For every reduced and prune alternating automaton A over Γ_1 which performs only pops and tests of emptiness there exists a reduced automaton B over Γ_1 so that $\mathcal{S}(A) = \mathcal{S}(B)$ and $|Q_B|$ is polynomial in $|Q_A|$ and Γ_1.*

Proof. Let $A = (Q_A, I_A, \Delta_A)$ be a reduced and prune alternating automaton over Γ_1 which performs only *pop*s and tests of emptiness. We construct a reduced automaton $B = (Q_B, I_B, F_B, \Delta_B)$ over Γ_1 that is equivalent to A, like follows:

- $Q_B = (Q_A \cup \{\bullet\}) \times (\Gamma_1^0 \cup \{\varepsilon\})$, where $\bullet \notin Q_A$
- $I_B = \{(q, \varepsilon) \mid q, \bot_1 \to \emptyset \in \Delta_A\} \cup \{(\bullet, \varepsilon)\}$
- $F_B = I_A \times (\Gamma_2^O \cup \{\varepsilon\})$
- $$\begin{aligned}\Delta_B = &\{(q,\gamma) \xrightarrow{\gamma'} (p,\gamma'), T \mid p, T \to (q, \overline{\gamma'}) \in \Delta_A \text{ and } \gamma' \neq \overline{\gamma} \neq \emptyset\}\\ \cup\ &\{(\bullet,\gamma) \xrightarrow{\gamma'} (p,\gamma'), T \mid p, T \to \emptyset \in \Delta_A \text{ and } \gamma' \neq \overline{\gamma} \neq \varepsilon\}\\ \cup\ &\{(\bullet,\gamma) \xrightarrow{\gamma'} (\bullet,\gamma') \mid \gamma' \neq \overline{\gamma} \neq \varepsilon\}\end{aligned}$$

We need the last instruction in the states as additional information to make sure that we stay reduced. The general idea is just to reverse the transitions including the instructions. For the case that in A we do not go the hole way to the empty level 1 stack, we need the last two sets, that guess the way to the empty stack from the stack where A stops respectively from the empty stack to this stack.

By construction and by lemma 2.1.1 we have that $\mathcal{S}(A) = \mathcal{S}(B)$. □

Example 20. *We give now an example for the construction of lemma 4.4.7. For that let* $\Gamma = \{a, b\}$ *and* $A = (Q_A, I_A, \Delta_A, \Omega)$ *be a reduced alternating parity automaton over* Γ_1 *with* $Q_A = \{i, p, q\}$, $I = \{i\}$, *the following set of transitions* Δ_A*:*

$$\begin{array}{lll} \delta_1 = i \to (i, \overline{a}) \wedge (p, b) & \delta_2 = i \to (i, \overline{b}) \wedge (q, a) & \delta_3 = i, \bot_1 \to \emptyset \\ \delta_4 = p \to (p, a) & \delta_5 = q \to (q, b) \wedge (q, a) & \end{array}$$

and the coloring $\Omega(i) = 1$, $\Omega(p) = 0$, $\Omega(q) = 1$. *The automaton* A *recognizes the regular set of stacks* $\{[a^n]_1 \mid n \geq 0\}$.

Now we define first the pruned ($\tilde{\delta}$) *and the rest* (δ') *transitions.*

$$\begin{array}{lll} \tilde{\delta_1} = i \to (i, \overline{a}) & \tilde{\delta_2} = i \to (i, \overline{b}) & \tilde{\delta_3} = i, \bot_1 \to \emptyset \\ \tilde{\delta_4} = p \to \emptyset & \tilde{\delta_5} = q \to \emptyset & \end{array}$$

$$\begin{array}{lll} \delta'_1 = \bullet \to (p, b) & \delta'_2 = \bullet \to (q, a) & \delta'_3 = \bullet, \bot_1 \to \emptyset \\ \delta'_4 = \bullet \to (p, a) & \delta'_5 = \bullet \to (q, b) \wedge (q, a) & \end{array}$$

It is easy to see that if the transition δ_2 *and then* δ_5 *is applied the state* q *appears infinitely often in a path of the execution and so the maximal color that is seen infinitely often is odd. By lemma 4.4.6 we know that* $\mathcal{S}_{\delta'_2}(A_{\delta'_2}) = \mathcal{S}_{\delta'_5}(A_{\delta'_5}) = \emptyset$ *and* $\mathcal{S}_{\delta'_1}(A_{\delta'_1}) = \mathcal{S}_{\delta'_3}(A_{\delta'_3}) = \mathcal{S}_{\delta'_4}(A_{\delta'_4}) \neq \emptyset$.

Now with this knowledge we can define the reduced and prune alternating automaton $B = (Q_B, I_B, \Delta_B)$[6] *with* $Q_B = \{i\}$, $I_B = \{i\}$ *and* $\Delta_B = \{i \to (i, \overline{a});\ i, \bot_1 \to \emptyset\}$.

It remains to construct the reduced automaton $C = (Q_C, I_C, F_C, \Delta_C)$ *over* Γ_1 *that is equivalent to* B *by:*

- $Q_C = Q_B \times (\Gamma_1^O \cup \{\varepsilon\})$
- $I_C = \{(i, \varepsilon)\}$
- $F_C = \{i\} \times (\Gamma_1^O \cup \{\varepsilon\})$
- $\Delta_C = \{(i, \varepsilon) \xrightarrow{a} (i, a);\ (i, a) \xrightarrow{a} (i, a)\}$

4.4.4 Conclusion

Theorem 4.4.9. *For every alternating parity automaton* A *over* Γ_2 *there exists an equivalent reduced automaton* A' *over* Γ_2 *with tests in* $\mathcal{L} \subset Reg_1$.

Proof. By theorem 4.3.13 we know that there is a reduced alternating parity automaton B that is equivalent to A and by theorem 4.4.4 we have that there is an reduced automaton A' over Γ_2 with tests in $\mathcal{L} \subset PAlt_1$ that is equivalent to B and by theorem 4.4.7 we know that $PAlt_1 = Reg_1$. □

parity game over instruction sequences over Γ_2

(1) $\Downarrow$ 4.2.1

alternating parity automaton over instruction sequences over Γ_2
with tests in Alt_2

(2) $\Downarrow$ 4.2.2

alternating parity automaton over instruction sequences over Γ_2

(3) $\Downarrow$ 4.1.4

alternating parity automaton over Γ_2

(4) $\Downarrow$ 4.3.13

reduced alternating parity automaton over Γ_2

(5) $\Downarrow$ 4.4.4

reduced and prune alternating automaton over Γ_2 with tests in $PAlt_1$

(6) $\Downarrow$ 4.4.5

reduced automaton over Γ_2 with tests in $PAlt_1$

(7) $\Downarrow$ 4.4.7

reduced automaton over Γ_2 with tests in Reg_1

Figure 8: The picture shows the single steps of the theorem 4.2.1.

4.5 Overview and complexity

The Figure 8 shows an overview over the single proofs that lead to the claim that the winning region of parity games over instruction sequences over Γ_2 is regular.

Remark 4.5.1 (Remark to complexity)**.** If we take a look at the different steps of the proof of Theorem 4.2.1 we see that we get a double exponential complexity in the number of states regarding the size of the stack alphabet. In especial just the steps 4 and 7 in Figure 8 are exponential all other steps are just polynomial.

In particular just the step to get from alternating to reduced alternating cost an exponential blow-up and it has to be done for every level. So if we would take a game over stacks of level k we would have to iterate the steps 4 and 5 k times and get so a k-exponential complexity.

[6] We throw out directly the state q because its transition are thrown out by the construction and the state p and the transition $p \to \emptyset$ because it is no longer reachable from the initial state i with a transition.

5 Conclusion

5.1 Summary

In this thesis we have dealt with the problem to proof the regularity of the winning regions of reachability and parity games defined over higher-order pushdown graphs and also to find an algorithm to compute the winning regions. We restrict in this thesis just to higher-order pushdown graphs of level 2 but the results can be lifted straightforward to level n.

We have introduced first in Chapter 2 the fundamentals we are working with. For this we have defined the higher-order pushdown stacks of level 2 whas ich are stacks of level 1 stacks. Additionally we defined operations to manipulate the stacks, i.e. the operations are used to build up a stack. We have the operations *push* and *pop* to add respectively delete symbols of the stack alphabet in the topmost level 1 stack and we have the operations *copy* and $\overline{copy}$ to copy respectively delete the hole topmost level 1 stack. Additionally we have defined instructions as a symbolic representation for the operations to get a short form for notations. After the definition of the higher-order pushdown automata of level 2 we defined the term of regularity for higher-order pushdown stacks. For this purpose we have used a recently developed technique of Carayol [Car05]. The main idea of it is to call a set of stacks regular if it can be produced by a regular set of operation sequences applied to the empty level 2 stack. For this it has to be exposed that the construction of a stack by a regular instruction sequence is not unique. We can define a unique reduced instruction sequence to construct a stack but in this case the sets of stacks that can be defined by regular reduced operation sequences are a really subset of the regular sets of stacks that we have defined before. This is shown in Theorem 2.3.4. So it is important to allow the variety of operation sequences to produce one stack. In Chapter 2 we have define some different kinds of automata models that run over the instructions and accept only regular sets of stacks.

In Chapter 3 we have shown the regularity of the winning region of higher-order pushdown games of level 2 with reachability winning condition. For that we first have defined the higher-order pushdown games by regular sets of stacks together with a set of instruction sequences. By this we have achieved a very general definition and the vertices of the game graph consist only of the stacks without additional states. The goal set of the reachability game is defined by a regular set of stacks, too. To show the regularity of the winning region, which is a set of stacks again, we define first an alternating automaton over the instructions of level 2 that recognizes exactly the stacks in the winning region. After that we have used the result of Carayol that this kind of automaton recognizes only regular sets of stacks.

In chapter 4 we have shown that the winning regions of higher-order parity games of level 2 are regular. The approach is similar as for reachability games. First we have defined an alternating automaton that accepts exactly the stacks in the winning region and show then that these automata accept only regular sets of stacks of level 2. For this we define alternating parity automata over the instruction of level 2 that include the parity condition as accepting condition and which execution is an infinite tree. To show that these alternating parity automata accept only regular sets of stacks we reduce them to reduced automata over instructions of level 2 with tests in the regular sets of stacks of level 1. This is done in many substeps. The most important and complex step is to show that the alternating parity automata are equivalent to the reduced alternating parity automata. We have used a result of Vardi [Var98] that quotes the equality between alternating two-way parity tree automata and nondeterministic one-way parity tree automata that run over infinite trees. After this part of the proof the rest is again similar to the proofs of the reachability games.

The complexity to compute the winning region of the higher-order pushdown games of level 2 is double exponential for reachability games as well as for parity games. This is caused by the fact that the reduction from alternating (parity) to reduced alternating (parity), which has an exponential effort, has to be performed two times, once for every level. So for a game over higher-order pushdown stacks of level n we would get a n-exponential complexity.

5.2 Further research

One approach for further research is to lift the results from level 2 to level n pushdown stacks but this would not mean too much work. The most proofs can be easily expanded to level n because they work naturally for every level or induction can be applied by using the idea for level 2. We have not done the proofs for all level here to stay more concrete and get a better intuition for the proofs.

A more interesting topic for research would be regular winning strategies for higher-order pushdown games. A winning strategy is a specification of the behavior such that a player wins if he acts according to the strategy. In particularly a strategy for player 0 is a function that assigns to each play prefix ending in a state of player 0 an edge to determine the successor vertex. A strategy is positional if the choose of the next vertex depends only on the current vertex and not on the hole play prefix. We want to show that the strategies for higher-order pushdown games are also regular and we want these strategies to be positional. By Fratani [Fra05a] it is already known that these regular strategies exists but we want here additionally to compute them.

Regularity in the case of strategies means that we can divide the vertices of a game respectively the stacks they represent into regular sets of stacks. For each set we know which edge respectively instruction to choose. Beside the proof that the strategy for higher-order pushdown stacks is regular we want again to construct an automaton to compute this strategy. In particular we want to define a strategy by a higher-order pushdown automaton. To compute the winning strategies we want to use again the result of Vardi in [Var98] respectively a part of the proof where a strategy is coded in a one-way tree automaton.

References

[Alb02] L. Alberucci. Strictness of the modal μ-calculus hierarchy. In *Automata Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of LNCS, pages 185–203. Springer Verlag, 2002. Chapter 11.

[ATM03] R. Alur, S. La Torre, and P. Madhusudan. Modular strategies for recursive game graphs. In *Tools and algorithms for the construction and analysis of systems*, volume 2619 of LNCS, pages 363–378, 2003.

[AVW03] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theor. Comput. Sci.*, 303(1):7–34, 2003.

[BEM97] A. Bouajjani, J. Esparza, and O. 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, July 1997.

[Ben69] M. Benois. Parties rationelles du group libre. Technical Report 269: 1188-1190, Comptes-Rendus de l'Acamdémie des Sciences de Paris, 1969.

[Büc64] J. Büchi. Regular canonical systems. *Arch. Math. Logik Grundlag.*, 6:91–111, 1964.

[Cac02a] T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *Automata, Languages and Programming : 29th International Colloquium, ICALP 2002, Malaga, Spain*, volume 2380 of LNCS, pages 704–715. Springer Verlag, 2002.

[Cac02b] T. 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, 2002. Chapter 17.

[Cac03a] T. Cachat. *Games on pushdown graphs and extensions.* PhD thesis, RWTH-Aachen, 2003.

[Cac03b] T. Cachat. Higher-order-pushdown automata, the caucal hierarchy of graphs and parity games. In *ICALP'03*, volume 2719 of LNCS, pages 556–569, 2003.

[Car05] A. Carayol. Regular sets of higher-order pushdown stacks. In *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005*, volume 3618 of LNCS, pages 168–179. Springer Verlag, 2005.

[Car06] A. Carayol. *Automates infinis, logiques et langages.* PhD thesis, Université de Rennes 1, 2006.

[CW03] A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of LNCS, pages 112–123. Springer Verlag, 2003.

[CW07] T. Cachat and I. Walukiewicz. The complexity of games on higher order pushdown automata. http://hal.archives-ouvertes.fr/docs/00/14/42/26/PDF/hpds-compl.pdf, May 2007.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of CAV 2000*, volume 1855 of LNCS, pages 232–247. Springer, 2000.

[Fra05a] S. Fratani. *Automates à piles de piles . . . de piles.* PhD thesis, Université Bordeaux 1, 2005.

[Fra05b] S. Fratani. Regular sets over tree structures. http://www.labri.fr/perso/fratani/, 2005.

[FS06] S. Fratani and G. Sénizergues. Iterated pushdown automata and sequences of rational numbers. In *Annals of Pure and Applied Logic*, volume 141, pages 363–411, September 2006.

[JW96] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monatic second order logic. In *Proceedings of the 7th International Conference on Concurrency Theory, CONCUR'96*, volume 1119 of LNCS, pages 263–277. Springer, 1996.

[Koz83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KV98] O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th ACM Symposion on Theory of Computing, STOC'98*, Dallas, 1998.

[KV00] O. Kupferman and M.Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Proceedings of the 12th International Conference on Computer Aided Verification, CAV 2000*, volume 1855 of LNCS, pages 36–52, 2000.

[MS85] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[MS95] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of rabin, mcnaughton and safra. *Theoretical Computer Science*, 141:69–107, 1995.

[Niw97] D. Niwinski. Fixde point characterization of infinite behavior of finite-state systems. *Theoretical Computer Science*, 189:1–69, 1997.

[Rab69] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Scociety*, 141:1–35, July 1969.

[Rab06] A. Rabinovich. Church synthesis problem with parameters. In *Computer Science Logic*, volume 4207 of LNCS, pages 546–561. Springer Verlag, 2006.

[Roh02] P. Rohde. Expressive power of monadic second-order logic and modal μ-calculus. In *Automata Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of LNCS, pages 239–257. Springer Verlag, 2002. Chapter 14.

[Ser03] O. Serre. Note on winning positions on pushdown games with omega-regular conditions. *Information Processing Letters*, 85(6):285–291, 2003.

[Tho02] W. Thomas. Infinite games and verification. In *Computer Aided Verification*, volume 2404 of LNCS, pages 58–64. Springer, 2002.

[Tho03a] W. Thomas. Automata and reactive systems. http://www-i7.informatik.rwth-aachen.de/scripte/ars-english.pdf, 2003.

[Tho03b] W. Thomas. Constructing infinite graphs with a decidable mso-theory. In *Proc. of MFCS'03*, volume 2747 of LNCS, pages 113–124. Springer Verlag, 2003.

[Tho06] W. Thomas. Vorlesung infinite-state system verification. Technical report, Chair of Computer Science 7, RWTH-Aachen, April 2006.

[Var98] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *Automata, Languages and Programming*, volume 1443 of LNCS, pages 628–641. Springer Berlin / Heidelberg, 1998.

[Wal96] I. Walukiewicz. Pushdown processes: games and model checking. In *CAV'96*, volume 1102 of LNCS, pages 62–74, 1996. Full version in Information and Computation 164, pp. 234-263, 2001.

[Wal02a] I. Walukiewicz. Automata and games for synthesis. In *Algebraic Methodology and Software Technology: 9th International Conference, AMAST 2002*, volume 2422 of LNCS, pages 15–20. Springer, 2002.

[Wal02b] I. Walukiewicz. Monadic second order logic on tree-like structures. *Theoretical Computer Science*, 275(1-2):311–346, 2002.

[Wal04] I. Walukiewicz. A landscape with games in the background. In *Proceedings of LICS'04*, pages 356–366, 2004.

[Wöh05] S. Wöhrle. *Decision problems over infinite graphs: Higher-order pushdown systems and synchronized products.* PhD thesis, RWTH Aachen, 2005.

[Zap02] J. Zappe. Modal μ-calculus and alternating tree automata. In *Automata Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of LNCS, pages 171–184. Springer Verlag, 2002. Chapter 10.

Index