Simplification Problems for Automata and Games

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

vorgelegt von

Diplom-Informatiker
Stefan Repke (geb. Schulz)
aus Cottbus

Berichter: Privatdozent Dr. Christof Löding
Universitätsprofessor Dr. Dr.h.c. Wolfgang Thomas
Dr. habil. Thomas Colcombet

Tag der mündlichen Prüfung: 13. Mai 2014

Zusammenfassung

Die Vereinfachung von Automaten beschäftigt sich mit der Frage, ob ein gegebener Automat in ein simpleres Format überführt werden kann; d.h., ob im gegebenen Fall ein bestimmtes Merkmal des Automatenmodells verzichtbar ist. Wenn ja, so soll ein vereinfachter Automat generiert werden. Die zugrundeliegende Motivation ist, dass einfachere Automaten bessere theoretische Eigenschaften besitzen. Vereinfachung können auch für Spiele betrachtet werden, die durch Automaten definiert sind. Dabei lautet die Frage, ob Gewinnstrategien durch Automaten dargestellt werden können, deren Format simpler ist als die Spielspezifikation. Diese Arbeit widmet sich zwei Aspekten der Vereinfachung: Regularitätsproblemen und vorausschauender Delegierung.

Beim Regularitätsproblem geht es darum, zu entscheiden, ob ein gegebener deterministischer Kellerautomat (DPDA) eine reguläre Sprache erkennt. Diese Frage wurde vor Jahrzehnten gelöst, wohingegen das erweiterte Problem für ω-DPDAs (welche unendliche Eingaben verarbeiten) seitdem offen geblieben ist. Wir erweitern eine Methode, die es erlaubt, bekannte Ergebnisse über DPDAs zu verallgemeinern auf ω-DPDAs mit schwacher Akzeptanz. Für diese schwachen ω-DPDAs zeigen wir die Entscheidbarkeit des Regularitätsproblems und zusätzlich des Äquivalenzproblems. Für ω-DPDAs im Allgemeinen geben wir eine Kongruenzrelation an, die Regularität charakterisiert. Die Entscheidbarkeit bleibt hingegen offen.

Eine verallgemeinerte Variante des Regularitätsproblems beschäftigt sich mit Pushdown-Spielen (PDGs), d.h., Zwei-Spieler-Spiele welche durch ω-DPDAs beschrieben werden und für welche Gewinnstrategien ebenfalls durch DPDAs dargestellt werden können. Regularität bedeutet entsprechend, dass ein PDG eine reguläre Gewinnstrategie besitzt; d.h., sie ist darstellbar durch einen endlichen Automaten. Wir zeigen dass diese Form des Regularitätsproblems bereits unentscheidbar ist für PDGs mit Sicherheitsbedingung.

Ein anderer Aspekt der Vereinfachung beschäftigt sich mit vorausschauender Delegierung für nichtdeterministische endliche Automaten (FSAs). Das Problem ist hier, zu entscheiden, ob Transitionen deterministisch gewählt werden können, wenn man eine beschränkte Vorschau auf die Eingabe zugesteht. Eine solche Auswahlfunktion nennt man Delegierer. Nicht-triviale Komplexitätsergebnisse über die Existenz von Delegierern sind bisher nur bekannt für eine eingeschränkte Unterklasse von FSAs. Wir erbringen entsprechende Ergebnisse für (uneingeschränkte) FSAs und zeigen, dass es PSPACE-vollständig ist, zu entscheiden, ob ein Delegierer für eine beliebige Vorschaubeschränkung existiert und einen solchen zu bestimmen. Weiterhin zeigen wir mithilfe von Zwei-Spieler-Spielen, dass die Existenz eines Delegierers in Polynomialzeit entschieden werden kann, wenn die Vorschaubeschränkung konstant ist.

Abstract

Automata simplification asks whether a given automaton can be converted into one of a simpler format, i.e., whether a certain feature of the automaton model is avoidable in the given case. If yes, an automaton of this simpler format shall be synthesized. The idea is that simpler automata models have better closure and algorithmic properties. Simplification can also be studied for games that are defined by automata. Then, the question is whether winning strategies can be implemented by automata of a format that is simpler than the one used for the game specification. In this thesis, we continue the research on two different aspects of simplifications, namely regularity problems and lookahead delegation.

The regularity problem is to decide whether a given deterministic pushdown automaton (DPDA) recognizes a regular language; which means that the pushdown stack is unnecessary as an equivalent finite state automaton can be found. This problem was solved decades ago for DPDAs whereas its pendant for ω-DPDAs (working on infinite input sequences) remained open since then. We adapt a technique that allows us to lift some known result from DPDAs to the restricted class of ω-DPDAs with weak acceptance conditions. For those weak ω-DPDAs, we show the regularity problem and further the equivalence problem to be decidable. For the general case of ω-DPDAs, we define a congruence relation that characterizes regularity, whereas the decidability remains open, unfortunately.

A generalized variant of the regularity problem concerns pushdown games (PDGs), i.e., two-player games which are specified by ω-DPDAs and which omit a winning strategy representable by a DPDA, too. In this setting, regularity means that there is a 'regular' winning strategy, i.e., one representable by a finite state automaton. We show the regularity problem to be undecidable even for PDGs with safety acceptance.

The other aspect of simplification deals with lookahead delegation for nondeterministic finite state automata (FSAs). The problem is to decide whether transitions can be chosen deterministically when a bounded lookahead on the input word is allowed. Such a choice function is called delegator. Nontrivial complexity results on the existence of delegators are only known for a restricted subclass of FSAs, yet. We contribute the corresponding results for (unrestricted) FSAs. We provide polynomial space upper and lower bounds for deciding the existence of a delegator with some bounded lookahead (and synthesizing it). By using two-player games as a tool, we further prove that the existence can be decided in polynomial time if the length of the lookahead is fixed.

Contents

Chapter 1

Introduction

Automata are one of the most classic concepts in theoretical computer science. They first occurred in the context of neurophysiology [MP43] and soon turned out to be a valuable model of computation, too. Especially **deterministic finite state automata (DFSAs)** are natural as they coincide with many other formalism, and they enjoy outstanding algorithmic and closure properties (see [HU79] for an overview). In computer science, automata can be found in a variety of applications, like in compilers or other tasks of text processing and in verification.

The more complex the applications became the more useful extensions arose for classic DFSAs; e.g., nondeterminism, pushdown stack, or infinite input words to name just a few standard extensions:

- A **(nondeterministic) finite state automaton (FSA)** might choose among different transitions for a given input letter. This allows the automaton to 'guess' a run with respect to the remaining input.
- The finite memory of a **pushdown automaton (PDA)** is equipped with an addition **stack**, which is an unbounded LIFO data structure. It essentially allows a PDA to test the input for recursive structures.
- Infinite words, so-called **ω-words**, can be handled by automata with a different acceptance conditions. For **parity acceptance**, a number, called **color**, is assigned to each state. An ω-word is accepted by an **ω-automaton** if it induces a state sequence where an even number is the lowest one that occurs infinitely often.

All these extensions can be combined; e.g. to obtain a **deterministic pushdown ω-automaton (ω-DPDA)**. Some of them do not increase the expressiveness (e.g., a DFSA does not gain any expressiveness when equipped with nondeterminism and ε-transitions), whereas in general, they do. On the other hand, such an increase leads to poorer closure and algorithmic

properties. This is the motivation to consider simplification problems, i.e., to asks whether certain extensions can be avoided for a given automaton. In positive cases, an automaton of a simpler format shall be computed.

Many questions of simplification have been studied so far; some of them being:

- "Does a DPDA recognize a regular language?" [Ste67, Val75],
- "Can the ω-acceptance of an ω-DFSA be reduced?" [Lan69, CM99],
- "Can the ω-acceptance of an ω-DPDA be reduced?" [Lin77, CG78],
- "Does a regular language belong to a certain subfamily of regular languages?" [Str94], and
- "Can a nondeterministic FSA choose transitions deterministically when using lookahead?" [RS07].

In this thesis, we continue the research on two of these problems: the first one which is called **regularity problem**, and the last one called **lookahead delegation**. Details on them are discussed in the following.

Regularity Problems

Based on the regularity problem for DPDAs, we further consider two generalized variants of the problem, namely for ω-DPDAs and pushdown games. We first motivate the basic problem which explains why determinism is required.

Pushdown automata are popular as they can express the nested structure of program code, as well as recursive nature of program runs, or they can process data trees in a linearized form, like XML. PDAs have been studied decades ago and their closure and algorithmic properties turned out to be much worse in comparison to FSAs (see [HU79]). To demonstrate the complexity, one can look at the two most fundamental simplification problems which ask whether the language is empty, or whether it is universal (i.e., every word is accepted), respectively. For nondeterministic PDAs, emptiness is decidable in polynomial time [EHRS00] whereas universality is undecidable [HU79, Theorem 8.11]. This undecidability is handed down to many other important problems; like equivalence ("Do two PDAs recognize the same language?") and regularity ("Does a PDA recognize a regular language?"); the latter one being a simplification problem.

The situation becomes slightly better when turning to deterministic PDAs (DPDAs). Although, they still lack of some important closure properties, the determinism allows the universality problem to be reduced to the emptiness problem which makes it decidable in polynomial time, too. Other related problems are known to be decidable, but with a high complexity: there exists an algorithm testing regularity in doubly exponential time [Ste67, Val75] and decades later, an involving algorithm was found that decides equivalence with non-elementary running time [Sén01, Sén02]. For the restricted class of deterministic one-counter automata, where the stack can only be used as a counter, both problems were recently shown to be NL-complete [BG11, BGJ13].

For many modern applications, it is not enough that a program processes a single request and terminates afterwards. Instead, it is required to run ad infinitum while constantly answering user requests. This idea is captured by ω-automata, which are particularly used in model checking (see [BK08]). Many theoretical properties of automata on infinite words can be derived from the case of finite words with more difficult proofs (see [PP04]). When ω-DPDAs (and ω-PDAs) were studied in the 1970's [CG77a, CG77b, CG78], the universality problem turned out to remain decidable for ω-DPDAs, whereas regularity was posed as an open problem in [CG78], and also the equivalence problem is open up to now.

In this thesis, we tackle the regularity problem for ω-DPDAs. We show the decidability of both the regularity and the equivalence problem for the restricted class of **weak ω-DPDAs**. An ω-automaton (with parity acceptance) is called **weak** if the colors that occur during a run never increase. This allows the run to change only a bounded number of times between being accepting or rejecting (i.e., even and odd) until eventually, the color stabilizes. This coincides with the boolean combination of reachability and safety conditions. We obtain our results by extending a normal form for weak ω-DFSAs which establishes an interesting connection between languages of finite and infinite words [Löd01]: the same language of infinite words is recognized from two states if, and only if, the same language of finite words is recognized from the same two states (when considering the ω-DFSA as DFSA where states with even color are defined as accepting). This normal form was used in [Löd01] to lift known minimization results from DFSAs to weak ω-DFSAs. With some rather technical effort, we extend this normal form to the framework of pushdown automata which then allows us to easily lift regularity and equivalence decision procedures from DPDAs to ω-DPDAs. These results were presented in a shortened version [LR12].

Unfortunately, we did not solve the open problems for (strong) ω-DPDAs. Nevertheless, we contribute a theoretical result which might be a step towards the solution. Based on the language recognized by some ω-DPDA, we define a congruence relation that has finite index

(i.e., finitely many classes) if, and only if, the language is regular. Similar results for languages of finite words are the well-known Myhill-Nerode congruence which characterizes regularity, or another congruence which characterizes whether a language can be recognized by a so-called visibly PDA [AKMV05]. The problem of deciding whether the index of our congruence is finite remains as open as for the regularity problem, though.

Infinite two-player games is an important model closely related to ω-automata. A classic motivation to study such games is Church's synthesis problem [Chu57, Chu63]. For a system specification given as infinite sequences of pairs (input and output), synthesis means to find a controller that produces for each input sequence an output sequence according to the specification. In [BL69], a solution is presented where the specification is assumed to be an ω-DFSA and it was shown that then, a controller can be represented by a finite state device, too.

From the perspective of games, Church's problem is a special case of finding a winning strategy. An infinite two-player game is played on a directed graph where each vertex is associated with one player. Starting from a certain initial vertex, the player who is in charge of it has to pick one of its outgoing edges and the play proceeds analogously from this new vertex. In total, a play corresponds to an infinite sequence of vertices. The winning condition (for a certain player) is usually defined by a parity condition on the vertices. A winning strategy advices a player to pick edges such that he wins each resulting play, no matter how his opponent plays.

A classic approach to finitely represent the (possibly infinitely many) vertices of a game graph is to use a deterministic automaton and its configuration graph. We then understand a play as an ω-word rather than an infinite sequence of vertices. Consequently, strategies can be specified by deterministic automata that read the play as an input word. For many automaton models, it is possible to synthesize a winning strategy ('controller') that uses the same automaton model as the one that describes the game ('specification'); e.g., for finite state games [BL69] and for pushdown games [Wal01].

Such results establish games as useful tools, but they can also be subject to simplification problems themselves. One problem might be to decide whether a winning strategy of a simpler automaton model exists. An example of this form is the question studied in [SV02, SS07] that is to decide whether XML documents can be validated against a given DTD by using only constant memory, i.e., by a DFSA. This problem can be reduced to the simplification problem of whether a certain pushdown game admits a finite state winning strategy (see Example 3.1.2 for details). Note that this connection does not allow undecidability results for simplification problems to be transfered back to the validation problem due to the direction of the reduction.

Our contributions to simplification for pushdown games involve firstly, the 'regularity problem' for winning strategies and secondly, we define games that simulate acceptance by automata and express certain related properties. Regarding winning finite state strategies (FSSs) for pushdown games, we show that a winning FSS exists as soon as one can win against a reachability winning condition, whereas the existence becomes undecidable when a safety winning condition is considered. Further, we present a game construction that is intended to simulate the input and the acceptance of a given ω-DPDA. We show that this so-called **classification game** connects regularity for languages and games in the sense that the language is regular if, and only if, there exists a winning FSS (for the player who is in charge of the acceptance). The game is further useful to decide simplification problems on the acceptance component of a ω-DPDA; i.e., deciding whether a language can already be recognized with simpler acceptance, like reachability, safety, Büchi, or co-Büchi. The novelty of our classification game yields a simpler, less technical, and more general solution to these problems that are known to be decidable [CG78, Lin77].

Lookahead Delegation

The objective of simplification is not necessarily to represent an automaton in a simpler (i.e., restricted) format by introducing new states. A different approach is followed in **lookahead delegation** as studied in [RS07]: the problem is to decide whether a nondeterministic automaton can choose transitions deterministically when it is allowed to look some letters ahead on the input word. The given automaton itself remains unchanged in the sense that no new states or transitions are added.

This requirement arose from the original motivation based on distributed web services, called e-services [ACKM04], which can be formally understood as DFSAs in this context. The problem of e-service composition [BCG+03, GHIS04, DIS05, MW08] asks whether a target specification (given as a DFSA) can be composed of multiple available e-services (given as tuple of DFSAs). If this is possible, then a **lookahead delegator** shall be synthesized which assigns each input letter to one of the DFSAs that has to process it. As established in [GHIS04], the delegator has to make its decisions based on the current state of each DFSA, the current input symbol, and a bounded lookahead on the further input. It was shown that deciding the existence of a delegator with a given amount of lookahead is EXPTIME-complete [GHIS04, MW08].

The reformulation of the delegation problem where a single (nondeterministic) FSA is given instead of multiple DFSAs is obtained by considering the (fully asynchronous) product of all DFSAs (the specification as well as the available e-services). In fact, this formulation with a single automaton is more general although it looks simpler. The central decision problem is

hence to decide the existence of a lookahead delegator for a given automaton. Three variants of that problem were introduced in [RS07]:

- First, where the amount of lookahead is fixed (hard-coded into the algorithm).
- Second, where the amount of lookahead is given (as a part of the input).
- And finally, where the amount of lookahead is arbitrary but bounded.

Further, the complexity of these three problems was studied in [RS07] and the following non-trivial upper bounds are given only for a restricted class of FSAs called unambiguous FSAs: PTIME, co-NP, and PSPACE, respectively. The latter problem regarding the existence of a bounded lookahead delegator was posed as an open problem for the case of FSAs [RS07] and for the case of a tuple of DFSAs [DIS05].

In this thesis, we tie in with the previous results and contribute bounds for the case of general FSAs: The first problem is in PTIME, which generalizes the bound from [RS07] and corrects another result.[1] The main idea of our construction is to use a game that simulates the delegation process such that a winning strategy for the delegating player directly corresponds to a delegator in the automaton. By a non-trivial abstraction of the game representation, such a strategy can be computed in polynomial time. We further prove that the latter two problems are PSPACE-complete. For the upper bound, we present a different algorithm that can store a lookahead of exponential length in polynomial space by using the binary encoding of this length. We then prove that a maximal useful lookahead is exponential in the size of the automaton which makes the last problem decidable in polynomial space with the above algorithm. The latter result solves the open problems due to [DIS05, RS07]. These results have been presented in excerpts in [LR13]. Finally, we extend the delegation problem to pushdown automata and show the three problems to be undecidable for PDAs in general. Only for the restricted class of visibly PDAs, we prove the decidability of the first two problems.

Outline

This thesis is structured as follows. We introduce in Chapter 2 the fundamental concepts that we need throughout our work; like automata and languages, games and strategies, and computational machines. In Chapter 3, we present our results regarding the regularity problem for deterministic pushdown automata. We start with the most complex variant of the problem

[1] In [RS07, Theorem 5] is stated that the delegation problem with fixed lookahead is PSPACE-hard for FSAs. The proof uses a reduction from an inclusion problem of the form "Is the language recognized by a given FSA a subset of another fixed language?". However, this problem is not PSPACE-hard.

that asks for the existence of 'regular' strategies for pushdown games in Section 3.1. We then study in Section 3.2 how the classification game can be used to simplify ω-DPDAs. A normal form for weak ω-DPDAs is introduced in Section 3.3 to solve the problems of regularity and equivalence. In Section 3.4, the chapter concludes by a congruence relation that characterizes regularity for (non-weak) ω-DPDAs. In Chapter 4, we consider lookahead delegation for nondeterministic automata. First, we present decidability results for finite state automata in Section 4.1, where we give a rather complete picture of the complexities for various formulations of the delegation problem. In Section 4.2, we extend our studies to pushdown automata and show decidability in a restricted case but undecidability in general. Finally, we give a conclusion in Chapter 5, where we also point out aspects that remain open or can be the subject of further research.

Chapter 2

Preliminaries

This chapter is devoted to the introduction of classic concepts and results that we work with at several places throughout this thesis. After some preliminary notation, we define automata which form the basis of our studies. For technical reasons, we start with the most general definition of pushdown automata. Then, simpler automata models are obtained as restricted special cases. We proceed with the definition of games and strategies, and how they can both be represented by automata. We conclude by a short introduction of register machines.

Sets. We define the following integer intervals: the natural numbers $\mathbb{N} = \{0, 1, \ldots\}$, the positive natural numbers $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$, the interval $[n] = \{0, \ldots, n-1\}$ of the first n naturals, and the binary numbers $\mathbb{B} = [2]$. We write $\omega = \mathbb{N}$ and use ω to denote countable infinity. We denote the cardinality of a set S by $|S|$.

Words and Languages. Let Σ be an **alphabet**, i.e., a finite set of so-called symbols or letters. Then, Σ^* (Σ^ω) denotes the set of **(ω-)words** over Σ, i.e., finite (countably infinite) sequences of Σ-symbols. A subset of Σ^* (Σ^ω) is called **(ω-)language**. For a (finite) word $w = a_1 \cdots a_n \in \Sigma^*$ with $a_1, \ldots, a_n \in \Sigma$, we define $|w| = n \in \mathbb{N}$ as its length and $w^{\mathrm{R}} = a_n \cdots a_1 \in \Sigma^*$ as its **reversal**. The **empty word** ε is the unique word of length $|\varepsilon| = 0$.

For languages $L, L' \subseteq \Sigma^*$, we define the following languages:

$$
\begin{aligned}
L^+ &= \left\{ w_1 \cdots w_n \;\middle|\; n \in \mathbb{N}_+ \text{ and } w_i \in L \text{ for all } i \in \{1, \ldots, n\} \right\}, \\
L^* &= L^+ \cup \{\varepsilon\}, \\
L \cdot L' &= \left\{ w \cdot w' \;\middle|\; w \in L \text{ and } w' \in L' \right\}, \\
L + L' &= L \cup L', \\
\overline{L} &= \Sigma^* \setminus L.
\end{aligned}
$$

For a language $L \subseteq \Sigma^*$ and ω-languages $L', L'' \subseteq \Sigma^\omega$, we analogously define the following ω-languages:

$$\begin{aligned} L^\omega &= \left\{ w_1 \cdot w_2 \cdots \;\middle|\; w_i \in L \text{ for all } i \in \mathbb{N}_+ \right\}, \\ L \cdot L' &= \left\{ w \cdot \alpha \;\middle|\; w \in L \text{ and } \alpha \in L' \right\}, \\ L' + L'' &= L' \cup L'', \\ \overline{L'} &= \Sigma^\omega \setminus L'. \end{aligned}$$

We often identify a single word $w \in \Sigma^*$ (or an ω-word $\alpha \in \Sigma^\omega$) with the singleton language $\{w\} \subseteq \Sigma^*$ (respectively $\{\alpha\} \subseteq \Sigma^\omega$), e.g., in combination with the above operators (like in the following). An ω-word $\alpha = uv^\omega$ that repeats some (nonempty) infix v ad infinitum after some prefix u is called **ultimately periodic**.

2.1 Automata

Automata are devices used to accept or reject input words by processing them letter-wise such that in each step, some memory is updated. We use the notation as it can be found in modern literature (e.g., [HMU01]).

Pushdown Automata

Pushdown automata are named after their memory structure. Besides the classic finite memory, they are equipped with a **stack** which is a last-in-first-out (LIFO) data structure represented by a word only growing to the left. When an input letter is processed, it is important in which state the automaton is and which information is present at the top of the stack, i.e., the leftmost symbol of the stack word. Based on this triple of letter, state, and stack top, the automaton can choose a so-called transition which describes how the memory shall be updated, i.e., the transition defines a new state and a (possibly empty) sequence of stack symbols to be placed on top of the stack as a replacement for the previous top symbol. An input word hence induces a sequence of memory updates. This process usually starts at a certain configuration of the memory, and depending on where it leads to, the automaton accepts or rejects the input.

Machines, Configurations, and Runs. Before issuing the acceptance behavior, we have a closer look at the update process which forms the core of the automaton called **machine**.

To extend the finite input alphabet Σ with the empty word, we write $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. Given a finite stack alphabet Γ and a stack bottom symbol $\bot \notin \Gamma$, the stack content is a word from

$\Gamma^* \cdot \bot$, i.e., the bottom symbol occurs at the rightmost position, and only there, while the stack is growing to the left. Similarly, we abbreviate $\Gamma_\bot = \Gamma \cup \{\bot\}$. We usually use capital letters for stack symbols ($A, B, C, \ldots \in \Gamma_\bot$) and stack words ($U, V, W, \ldots \in \Gamma_\bot^*$), whereas lowercase letters denote input symbols ($a, b, c, \ldots \in \Sigma$) and input words ($u, v, w, \ldots \in \Sigma^*$).

Definition 2.1.1. A **pushdown machine** (or **PDM** for short) $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, \bot)$ consists of

- a finite **state** set Q, and an **initial state** $q_0 \in Q$,
- a finite **input alphabet** Σ, a finite **stack alphabet** Γ, a stack **bottom symbol** $\bot \notin \Gamma$, and
- a finite **transition relation** $\Delta \subseteq Q \times \Gamma_\bot \times \Sigma_\varepsilon \times Q \times \Gamma_\bot^*$ such that the bottom symbol $\bot$ occurs at the bottom of the stack and only there, i.e., for each transition $(p, A, a, q, W) \in \Delta$ holds $W \in \Gamma^*\bot$ if $A = \bot$ and $W \in \Gamma^*$ if $A \neq \bot$. Note that the transitions are **nondeterministic**.

$\lhd$

A **configuration** consists of a state $q \in Q$ and some stack content $W \in \Gamma^*\bot$. We denote a configuration as a word $qW \in Q\Gamma^*\bot$ instead of a tuple. A transition $(p, A, a, q, V) \in \Delta$ reads the input letter a and replaces the leftmost part of a configuration pAW by qVW for some stack suffix $W \in \Gamma_\bot^*$ (which is empty iff $A = \bot$). We denote this as $pAW \xrightarrow{a} qVW$. A finite sequence $p_0W_0, \ldots, p_nW_n$ of configurations is a **run** of $\mathcal{M}$ on some input word $w \in \Sigma^*$ if $p_0W_i \xrightarrow{a_1} p_1W_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} p_nW_n$ such that $w = a_1 \cdots a_n$ where $a_0, \ldots, a_n \in \Sigma_\varepsilon$. We then write $p_0W_0 \xrightarrow{w} p_nW_n$. Further, an infinite sequence $p_0W_0, p_1W_1, \ldots$ of configurations is called a run of $\mathcal{M}$ on some input ω-word $\alpha \in \Sigma^\omega$ if for each finite prefix w of α, there is a prefix of the sequence that is a run of $\mathcal{M}$ on w. Note that in case of $a_i = \varepsilon$, the machine might be able to perform a so-called **ε-transition** without reading an input letter. However, the definitions do not allow a run on an ω-word to contain an infinite subsequence of ε-transitions.

Unless stated otherwise, we assume a run to start at the **initial configuration** $q_0\bot$ consisting of the initial state q_0 and the empty stack $\bot$.

Automata, Acceptance, and Languages. We can now lift machines to automata by introducing acceptance mechanisms. For finite input words, the resulting runs are finite, too, which allows us to accept words by the last state that a run leads to. A **pushdown automaton (PDA)** $\mathcal{A} = (\mathcal{M}, F) = (Q, \Sigma, \Gamma, \Delta, q_0, \bot, F)$ consists of a PDM $\mathcal{M}$ as above and a set of **accepting** states $F \subseteq Q$. A run is **accepting** if it leads from the initial configuration to a configuration where the state is accepting. The set of words having an accepting run is the language $L_*(\mathcal{A}) \subseteq \Sigma^*$ of $\mathcal{A}$. A language that is accepted by some PDA is called **pushdown language (PDL)**.

For ω-words, each run is also infinite and has no last state. The acceptance condition we use considers the states that occur infinitely often. A **pushdown ω-automaton (ω-PDA)** $\mathcal{A}' = (\mathcal{M}, \Omega) = (Q, \Sigma, \Gamma, \Delta, q_0, \bot, \Omega)$ consists of a PDM $\mathcal{M}$ as above and a **coloring** function $\Omega : Q \to \mathbb{N}$ that assigns **colors** to the states. A run is **accepting** if it fulfills the **(min-)parity acceptance condition** with respect to Ω: the lowest color that occurs infinitely often in the run has to be even. Analogously, the set of ω-words having an accepting run is the language $L_\omega(\mathcal{A}') \subseteq \Sigma^*$ of $\mathcal{A}'$. A language is called **pushdown ω-language (ω-PDL)** if it is accepted by some ω-PDA.

Restrictions

There are several ways to restrict pushdown automata which lead to other useful automata models like finite state automata. Some classic restrictions are introduced in the following. For the following, fix some PDM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, \bot)$.

Restricting Stack Usage. We call $\mathcal{M}$ a **one-counter machine (OCM)** if it has a unary stack alphabet, i.e., $|\Gamma| = 1$. With this restriction, it can use the stack only as a counter for increasing, decreasing, and testing for zero.

If the stack alphabet is empty, then $\mathcal{M}$ is called a **finite state machine (FSM)**. In this case, we can completely omit all components from the notation that are related to the stack since the stack content is always $\bot$.

We say that $\mathcal{M}$ is **visibly** (or a **VPDM**) if the input alphabet is partitioned into three subsets $\Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_s$ (see [AM04]). When the input letter is from Σ_c, then it is a **call** and one symbol has to be pushed onto the stack, whereas a letter from Σ_r indicates a **return** and the topmost symbol has to be popped from the stack (if not empty). Finally, Σ_s indicates a **skip** where the stack remains unchanged, which is also required for ε-transitions. Formally, the following holds for all transitions:

$$(q, A, a, q', W') \in \Delta \quad \text{implies that } a \neq \varepsilon \text{ and } W' = \begin{cases} A & \text{if } a \in \Sigma_s, \\ A'A & \text{if } a \in \Sigma_c \text{ for some } A' \in \Gamma, \\ \varepsilon & \text{if } a \in \Sigma_r \text{ and } A \neq \bot, \\ \bot & \text{if } a \in \Sigma_r \text{ and } A = \bot. \end{cases}$$

Restricting Transitions. If $\mathcal{M}$ has no ε-transitions, we call it **ε-free** (in the literature, this is also called **real-time**).

We call $\mathcal{M}$ **total** if for each combination of state, stack symbol, and input letter, there is at least one applicable transition.

We call $\mathcal{M}$ a **deterministic** PDM (**DPDM** for short) if for each combination of state, stack symbol, and input letter, there is at most one applicable transition, i.e., for all $p \in Q$, $A \in \Gamma_\bot$, and $a \in \Sigma$,

$$\left|\left\{(q, W) \mid (p, A, a, q, W) \in \Delta\right\}\right| + \left|\left\{(q, W) \mid (p, A, \varepsilon, q, W) \in \Delta\right\}\right| \leq 1.$$

In this case, we identify Δ with the (partial) **transition function** $\delta : Q \times \Gamma_\bot \times \Sigma_\varepsilon \to Q \times \Gamma_\bot^*$ where $\delta(p, A, a) = (q, W)$ if $(p, A, a, q, W) \in \Delta$. We further extend δ to the (partial) **transition function δ^*** for words: for a finite word w, let $\delta^*(w)$ be the unique configuration qW such that $q_0\bot \xrightarrow{w} qW$ and no further ε-transition is possible from qW. To be this definition well-defined, we forbid infinite sequences of ε-transitions for DPDMs since the run, which is uniquely determined by the input, would get stuck on such a sequence and cannot process further inputs. Note that in the deterministic setting, ε-sequences of unbounded length are useful in order to retrieve information somewhere lower in the stack within one step. Such sequences are unproblematic as the stack height is arbitrary but finite. As soon as an arbitrarily long ε-sequence is never popping the stack symbol it started at, it results in a loop. However, such ε-loops can be useful in combination with nondeterminism. We explain in the following how they can be detected and removed efficiently for DPDMs.

Removal of Infinite ε-Sequences. Suppose that from a configuration qAW of a DPDM $\mathcal{A}$, the unique ε-sequence leads to another configuration of lower stack height: $qAW \xrightarrow{\varepsilon} pW$ for some state $p \in Q$. Such behavior cannot be harmful up to this point as the (finite) stack height is decreased eventually. We hence understand an infinite ε-sequence to start at a configuration where the stack height does not drop below the initial value. This means consequently that the entire stack content below the top symbol has no impact anymore. It can hence be ignored.

We can effectively detect infinite ε-sequences by the following algorithm that is run for each pair $(q, A) \in Q \times \Gamma_\bot$ of state and top stack symbol. Let $\mathcal{M}_\varepsilon$ be like $\mathcal{M}$ where only ε-transitions are retained. We can compute in polynomial time [EHRS00] an FSA (i.e., an FSM with accepting states) that recognizes the set $\mathrm{post}^*_{\mathcal{M}_\varepsilon}(qA)$ of all configurations reachable from qA in $\mathcal{M}_\varepsilon$. The considered pair induces an infinite ε-sequence if, and only if,

a) the initial stack top is never dropped, i.e., $p \notin \mathrm{post}^*_{\mathcal{M}_\varepsilon}(qA)$ for all $p \in Q$, and

b) there is a further ε-transition for each reached pair of state and stack top, i.e., for each $(p, B) \in Q \times \Gamma_\bot$ that has no outgoing ε-transition, there should be no word $pBW \in \mathrm{post}^*_{\mathcal{M}_\varepsilon}(qA)$ for some $W \in \Gamma_\bot^*$.

Whether both conditions hold can be checked by membership tests. In case of a positive answer, it suffices to redirect the ε-transition for the pair (q, A) to some sink state with a self-loop for every input letter[1].

Restricting Parity Acceptance. When restricting the color set $\Omega(Q)$ of $\mathcal{A}$ to $\{0,1\}$ or $\{1,2\}$, we end up with **Büchi** or **co-Büchi** acceptance, respectively.

We call $\mathcal{A}$ **weak** if colors never increase during a run. When additionally restricting the color set of $\mathcal{A}$ to $\{0,1\}$ or $\{1,2\}$, we end up with **reachability** and **safety** acceptance, respectively.

However, note that some restrictions of the ω-acceptance can be bypassed with nondeterminism if the automaton can simply guess a suitable acceptance behavior. The ω-automata that we deal with in this thesis are usually deterministic.

All the above restrictions and terminology concerning the type of the underlying PDM or the acceptance of the PDA carry over to the automata and language classes they characterize. A complete list of abbreviations for the above models is given in Section 2.3. As usual, a language or ω-language is called **regular** if it can be accepted by an FSA, or ω-FSA, respectively. An example of an ε-free OCA is depicted in Figure 3.8 at page 58, and examples of ε-free FSAs can be found in Figures 4.1a and 4.3 at pages 62 and 73.

Determinization

When applying restrictions to an automata model, it is obvious that the restrictions characterize a subclass of languages which usually is a proper subclass. Due to the various possible combinations of restrictions, we do not give a complete picture that relates the restricted language classes. Instead, we point out some prominent cases where determinism does not affect the expressiveness.

A classic method for determinization is the **powerset construction** which constructs a deterministic automaton each state of which indicates all possible states that the nondeterministic automaton could be in. This idea is not applicable to pushdown automata as one also has to consider different possible stack contents which are possibly of different height. The powerset construction only extends to PDAs that are visibly because then, the stack height is determined by the input and is hence the same for all possible runs. With this property, one can similarly introduce new stack symbols that indicate sets of possible stack symbols.

Proposition 2.1.2 ([AM04]). *For each VPDA, there exists an ε-free DVPDA over the same partition of the alphabet that accepts the same language.*

[1] One might need several sink states depending on the context that $\mathcal{M}$ is used in.

When restricting the cardinality of the stack alphabets to one, only the stack height is of importance as the stack symbols do not differ any more. In this case, the powerset construction can be applied in the usual way where the stack alphabet is unchanged.

Proposition 2.1.3. *For each VOCA, there exists an ε-free DVOCA over the same partition of the alphabet that accepts the same language.*

Further, when restricting the stack alphabet to be empty, one is dealing with the class of regular languages.

Corollary 2.1.4. *For each FSA, there exists an ε-free DFSA that accepts the same language.*

The latter determinization results can be lifted from regular languages to regular ω-languages. The corresponding proof is much more involved in the case of infinite words (using [McN66, Saf88]; see [Löd98] for details).

Proposition 2.1.5. *For each ω-FSA, there exists an ε-free ω-DFSA that accepts the same language.*

2.2 Games

Throughout this thesis, we directly consider simplification problems for games, but also use games as a tool to solve other problems. Our terminology concerning two-player games is based on [Grä11].

Formally, a **(parity) game** $\mathcal{G} = (V, V_0, E, \Omega)$ is played between two players and consists of

a) a directed graph (V, E) with **vertices** V and **edges** $E \subseteq V \times V$,

b) a partition of the vertices into $V_0 \subseteq V$ for **Player 0** and $V_1 = V \setminus V_0$ for **Player 1**, and

c) a coloring function $\Omega : V \to \mathbb{N}$ with bounded codomain.

A finite or infinite path in $\mathcal{G}$ is called a **play**, i.e., a sequence $v_0 v_1 v_2 \cdots$ of vertices such that there is an edge $(v_i, v_{i+1}) \in E$ between each two consecutive vertices. It starts in some vertex $v_0 \in V$ (that is usually fixed) and then, for each i, the successor v_{i+1} of v_i is chosen by Player 0 if $v_i \in V_0$ or by Player 1 if $v_i \in V_1$. Player 0 **wins** an infinite play $\alpha \in V^\omega$ if α fulfills the min-parity condition w.r.t. Ω. Otherwise, Player 1 wins. We note here that usually a player loses if there are no outgoing edges from his position. As this behavior bypasses the actual winning condition, we require that the game graph only consists of **non-terminal** vertices, i.e., each vertex must

have at least one outgoing edge. One can obtain this from the standard setting by introducing sink vertices where the corresponding player loses.

For $\sigma \in \{0,1\}$, a **strategy** for Player σ is a function $s : V^* V_\sigma \to V$ that chooses a successor vertex $s(v_0 \cdots v_i) = v_{i+1}$ with $(v_i, v_{i+1}) \in E$ for each finite play $v_0 \cdots v_i \in V^* V_\sigma$ that ends in a vertex of Player σ. We say that s is **winning** from a vertex v_0 if every infinite play $v_0 v_1 \cdots$ induced by this strategy is won by Player σ, i.e., $v_0 v_1 \cdots$ fulfills the parity condition if $s(v_0 \cdots v_i) = v_{i+1}$ for every $v_i \in V_\sigma$. Finally, we say that Player σ **wins** $\mathcal{G}$ from a vertex v_0 if he has a winning strategy from there. The **winning region** of Player σ consists of all vertices v_0 with the above property. A game is called **determined** if for every vertex, one player can win, i.e., the set of vertices is partitioned into the winning regions of the two players.

A strategy s is called **positional** if its choice only depends on the last vertex, i.e., $s(v_0 \cdots v_i) = s(u_0 \cdots u_j)$ holds for any two finite plays $v_0 \cdots v_i, u_0 \cdots u_j \in V^* V_\sigma$ with $v_i = u_j$. We then consider a strategy as a function $s : V_\sigma \to V$. A game is **positionally determined** if for every vertex, one of the players has a positional winning strategy. A fundamental result is that the games we consider here have this property.

Proposition 2.2.1 ([EJ91, Mos91, Zie98]). *Parity games are positionally determined.*

The various restrictions of parity acceptance carry over to parity games. A portion of a safety game together with a positional winning strategy is depicted in Figure 4.2 at page 67. For safety games, winning strategies can be computed easily with an attractor construction (see [Grä11]).

Proposition 2.2.2. *A positional winning strategy for a given safety game can be computed in time linear in the number of vertices.*

Games and Automata

Note that game graphs can be infinite. However, we need to represent them finitely in order to run algorithms on them. One well-known way of doing so is by automata: an ω-automaton can be used to define the graph, the partition, and the winning condition. In detail, a **pushdown game (PDG)** $\mathcal{G} = (\mathcal{A}, Q_0)$ consists of an ω-DPDA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot, \Omega)$ and a set $Q_0 \subseteq Q$. The game is played on the configuration graph of $\mathcal{A}$ and can be identified with the game $\mathcal{G} = (V, V_0, E, \Omega')$ where

a) the vertices are configurations: $V = Q\Gamma^*\bot$ and $V_0 = Q_0\Gamma^*\bot \subseteq V$,

b) the edges are given by transitions: $(qW, q'W') \in E$ if $qW \xrightarrow{a} q'W'$ for some $a \in \Sigma$ and such that no further ε-transition is possible, and

c) the coloring Ω' is induced by Ω: $\Omega'(qW) = \Omega(q)$.

As mentioned before, ε-transitions increase the expressiveness of DPDAs. However, we additionally require for PDGs that $\mathcal{A}$ is **ε-weak**, meaning that the color does never increase during an ε-transition: $\Omega(p) \geq \Omega(q)$ whenever $\delta(p, A, \varepsilon) = (q, V)$ for some $A \in \Gamma_\perp$, $V \in \Gamma_\perp^*$. We explain later why this is important. A weak ω-PDA is ε-weak by definition. Every other ω-PDA can easily be ε-weakened by additionally storing the lowest color during an ε-sequence. The game is supposed to start at the configuration $v_0 = \delta^*(\varepsilon)$, i.e., at the target of a possible ε-sequence from the initial configuration $q_0\perp$.

Since the underlying automaton is deterministic, we can identify a (finite or infinite) play $v_0 v_1 \cdots$ with a (respectively, finite or infinite) word $a_0 a_1 \cdots$ over Σ that induces the play as follows: $v_i = \delta^*(a_0 \cdots a_{i-1})$ for each position i. Input letters are also called **actions** in this context. After a finite play $a_0 \cdots a_i$, the next action a_{i+1} is chosen by Player 0 iff $v_i \in Q_0\Gamma^*\perp$. Finally, an infinite play $a_0 a_1 \cdots$ is won by Player 0 iff $a_0 a_1 \cdots \in L_\omega(\mathcal{A})$. Note that the winning condition ignores the colors of intermediate states occurring during ε-transitions which is not necessary since $\mathcal{A}$ is assumed to be ε-weak, i.e., the last state of an ε-sequence hence has the lowest color.

A consequence of the action-based approach is that we can also identify strategies with functions $f : \Sigma^* \to \Sigma$ that read and output actions instead of vertices. We are especially interested in representing such strategies by automata. A **pushdown strategy (PDS)** $\mathcal{S} = (\mathcal{M}', \sigma)$ consists of a DPDM $\mathcal{M}' = (Q', \Sigma, \Gamma', \delta', q_0', \perp')$ and a function $\sigma : Q' \to \Sigma$. The strategy it defines is $f(w) = \sigma(q)$ where q is the state of the configuration $qW = \delta'^*(w)$ reached by w.

Again, the restrictions of automata and machines (and the corresponding terminology) carry over to pushdown games and strategies; e.g., FSGs and FSSs for finite state games and strategies, respectively. An example of a safety DOCA that defines a safety OCG is depicted in Figure 3.1 at page 23.

Usually, games which are represented by automata can be solved algorithmically. The format of a winning strategy is usually connected to the format of the game representation.

Proposition 2.2.3. *In particular,*

a) the winner of an FSG has a winning FSS [BL69], and

b) the winner of a PDG has a winning PDS [Wal01].

It is further possible to synthesize such strategies.

In this thesis, we only need the connections listed above, although, there exist many more (see e.g. [RT07, Fri10, COT12]).

Another interesting property is that the winning region can also be represented by automata. The winning region forms a language over the alphabet $Q \cup \Gamma_\perp$ since the game vertices are the configurations of the PDA $\mathcal{A}$. The regularity of the winning region was proven for Büchi conditions in [Cac02] and independently for parity conditions in [Ser03]. The next result combines some known tools to obtain a DFSA that recognizes the winning region (read in reverse for the sake of complexity).

Lemma 2.2.4. *For a given PDG $\mathcal{G} = (Q, \Sigma, \Gamma, \delta, q_0, \perp, \Omega, Q_0)$, one can generate a DFSA $\mathcal{A}'' = (Q'', \Sigma'', \delta'', q_0'', F'')$ such that for each configuration $qW \in Q\Gamma^*\perp$,*

$$qW \in L_*(\mathcal{A}'')^{\mathrm{R}} \;\Leftrightarrow\; \textit{Player 0 wins } \mathcal{G} \textit{ from } qW.$$

$\mathcal{A}''$ is of size $|Q''| \in 2^{\mathcal{O}(|Q|)}$ and the computation takes time $2^{\mathcal{O}(|Q|\cdot c)}$ where $c = |\Omega(Q)|$ is the number of colors used by $\mathcal{A}$.

Proof. From [Ser03], we know that each winning region in a pushdown game is a regular set of configurations. To generate a DFSA of exponential size, we use an algorithm given in [HO09]. It yields a so-called 'alternating multi-automaton' $\mathcal{A}'$. Before we formally introduce this automaton model, we want to mention the key properties of $\mathcal{A}'$. It is an alternating automaton that has same states as $\mathcal{G}$ (plus two extra states). In $\mathcal{A}'$, a run from a state $q \in Q$ on a word $W \in \Gamma^*\perp$ is accepting iff Player 0 can win from qW in $\mathcal{G}$.

Formally, $\mathcal{A}' = (Q', \Sigma', \Delta', F')$ is a special case of an alternating automaton and consists of the following components:

a) states $Q' = Q \uplus \{p_1', p_2'\}$, final states $F' \subseteq Q'$,

b) input alphabet $\Sigma' = \Gamma_\perp$, and

c) alternating transition relation $\Delta' \subseteq Q' \times \Sigma' \times 2^{Q'}$.

To define the acceptance of $\mathcal{A}'$, let $q' \xrightarrow{\varepsilon} \{q'\}$, and $q' \xrightarrow{AW} Q_1' \cup \ldots \cup Q_n'$ iff $(q', A, \{q_1', \ldots, q_n'\}) \in \Delta'$ and $q_i' \xrightarrow{W} Q_i'$ for all $i \in \{1, \ldots, n\}$. In this setting, no designated initial states are needed because we want to ask whether a run starting from $q \in Q$ is accepted. Then, the language accepted by $\mathcal{A}'$ is

$$L_*(\mathcal{A}') = \left\{ qW \in Q\Gamma^*\perp \;\middle|\; q \xrightarrow{W} P' \text{ for some } P' \subseteq F' \right\}.$$

A DFSA for the reversal language $L_*(\mathcal{A}')^{\mathrm{R}}$ of an alternating automaton $\mathcal{A}'$ can be obtained by a reversal powerset construction [CKS81]. In the following, we provide such a construction on $\mathcal{A}'$ to obtain the desired DFSA $\mathcal{A}''$ with an exponential blowup. Let $\mathcal{A}'' = (Q'', \Sigma'', \delta'', q_0'', F'')$ be a DFSA with

a) states $Q'' = 2^{Q'} \uplus \{q_\top'', q_\bot''\}$, initial state $q_0'' = F'$, accepting states $F'' = \{q_\top''\}$,

b) input alphabet $\Sigma'' = \Sigma' = \Gamma_\bot$, and

c) transition function $\delta'' : Q'' \times \Sigma'' \to Q''$ as follows (where $A \in \Gamma_\bot$, $q \in Q$):

$$\begin{aligned}
\delta''(P', A) &= \left\{ q' \in Q' \;\middle|\; (q', A, P'') \in \Delta \text{ for some } P'' \subseteq P' \right\}, \\
\delta''(P', q) &= \begin{cases} q_\top'', & \text{if } q \in P', \\ q_\bot'', & \text{if } q \notin P', \end{cases} \\
\delta''(q'', x) &= q_\bot'', \quad \text{for all } q'' \in \{q_\top'', q_\bot''\} \text{ and } x \in \Sigma''.
\end{aligned}$$

By construction, we have that:

$$\begin{aligned}
qW \in L_*(\mathcal{A}'')^{\mathrm{R}} &\Leftrightarrow (qW)^{\mathrm{R}} = (W^{\mathrm{R}} q) \in L_*(\mathcal{A}'') \\
&\Leftrightarrow q_\top'' = \delta''^*(W^{\mathrm{R}} q) \\
&\Leftrightarrow q \in \delta''^*(W^{\mathrm{R}}) \\
&\Leftrightarrow q \xrightarrow{W} P' \text{ for some } P' \subseteq F' \\
&\Leftrightarrow qW \in L_*(\mathcal{A}') \\
&\Leftrightarrow \text{Player 0 can win } \mathcal{G} \text{ from } qW.
\end{aligned}$$

The claimed running time results from the composition of the two algorithms. □

Concerning the running time, note that c is constant for games with Büchi condition. This also affects weak conditions as they can be rewritten as Büchi condition by redefining even colors to 0 and odd colors to 1.

2.3 Abbreviations

Many restrictions of pushdown machines, (ω-)automata, (ω-)languages, games, and strategies have been defined up to now. In the following, we give an overview of all abbreviations used for various applications of pushdown machines (square brackets indicate optional restriction, curly brackets are choices):

prefix ·	infix	· postfix
[D][V] ·	{PD,OC,FS} ·	{M,A,L}
ω-[D][V] ·	{PD,OC,FS} ·	{A,L}
[V] ·	{PD,OC,FS} ·	{G,S}

The meaning is as follows (letters occurring in the abbreviation are emphasized).

- The prefix indicates optional restrictions: **D**eterministic, **V**isibly.
- The infix indicates stack restrictions: **P**ush**D**own, **O**ne-**C**ounter, **F**inite **S**tate.
- The postfix indicates the device: **M**achine, **A**utomaton, **L**anguage, **G**ame, **S**trategy.

2.4 Register Machines

We conclude the preliminaries with a short introduction of certain computational machines. Beside the well known model of Turing machines being the reference for decidability, there exist many other Turing-complete models, i.e., models that can simulate Turing machines (and vice versa).

A **2-register machine (2RM)** is an input-free deterministic machine equipped with two counters. Each of the counters can be increased, decreased, and tested for zero. This concept is similar to a two-counter machine, i.e., the product of two DOCMs. The input alphabet does not matter. It can hence be considered as some singleton set and is omitted from the notation. Formally, a 2RM $\mathcal{M} = (Q, \delta, q_0, q_F)$ consists of

a) a finite set Q of **states**, **initial state** $q_0 \in Q$, **halting state** $q_F \in Q$, and

b) an input-free deterministic **transition function** $\delta : \big(Q \setminus \{q_F\}\big) \times \{0,1\}^2 \to Q \times \{-1,0,+1\}^2$.

The **configurations** of $\mathcal{M}$ form the set $Q \times \mathbb{N}^2$ where δ leads from a configuration (p, n_0, n_1) to another $(q, n_0 + d_0, n_1 + d_1)$ iff $\delta\big(p, \operatorname{sgn}(n_0), \operatorname{sgn}(n_1)\big) = (q, d_0, d_1)$, i.e., depending on the current state and whether the registers are zero, δ leads to another state, and increases or decreases the registers. W.l.o.g., We require that register values are never negative, i.e., $n_i + d_i \in \mathbb{N}$ for all $i \in \{0,1\}$.

The **halting problem** denotes the problem to decide whether the unique run of $\mathcal{M}$ starting from configuration $(q_0, 0, 0)$ leads to the halting state. This problem is undecidable for 2RMs as they can simulate Turing machines and vice versa (see [HMU01, Section 8.5.4]).

Chapter 3

Regularity Problems for Pushdown Games and ω-Automata

In this chapter, we study **regularity problems** for several aspects related to pushdown machines (automata, ω-automata, games, and strategies), that is to decide whether the pushdown stack is unnecessary. The essence of the question is whether a finite state representation exists for something given by a pushdown representation. Usually, this yields a simplification of the considered problem since pushdown representations are more expressive at the price of worse decidability properties in comparison to finite state representations. Depending on the aspect, the problems read as follows:

a) Can the language of a given pushdown automaton (PDA) be recognized by a finite state automaton (FSA)?

b) Can the language of a given pushdown ω-automaton (ω-PDA) be recognized by a finite state ω-automaton (ω-FSA)?

c) Does the winner of a given pushdown game (PDG) also have a winning finite state strategy (FSS)?

Generally, we can restrict the problem to deterministic pushdown machines (DPDMs). On the one hand, the **universality problem** ("Does an automaton accept every word?") is already undecidable for nondeterministic pushdown automata [HU79]. This is one of the two 'trivial' simplification problems and a special case of the regularity problem. As universality is undecidable for PDA, regularity has to be undecidable for pushdown automata, too. On the other hand, we need determinism in pushdown games to obtain meaningful definitions and to use the classic results on strategies.

The benefits of DPDAs are that decision procedures exist for testing regularity [Ste67, Val75] and equivalence [Sén01, Sén02]. The latter result is mentioned for being a breakthrough in

this field. These properties of DPDAs are the motivation to study the situation for infinite words. The regularity and equivalence problem for ω-DPDAs remained open, whereas some basic properties were lifted from finite to infinite words [CG77a, CG77b, CG78].

We give solutions to the regularity problems for pushdown games (Section 3.1) and weak pushdown ω-automata (Section 3.3), and a partial solution for general pushdown ω-automata (Section 3.4). In between, in Section 3.2, we develop a connection between the problems for games and ω-automata. Excerpts of these results were presented in [LR12].

3.1 Finite State Strategies for Pushdown Games

In this section, we study the regularity problem for pushdown games, which is to decide whether Player 0 can win a given pushdown game with a finite state strategy. According to our definition of pushdown games and strategies (cf. pages 16 and 17), we understand a play as an ω-word that results from the letters chosen by the players and that uniquely describes a run of the deterministic automaton defining the game. Analogously, an 'automaton strategy' is reading a finite play prefix as a word such that the reached state indicates the choice of the strategy.

In this section, we only consider winning conditions that are weak (or subclasses thereof) since we already show undecidability for safety winning conditions.

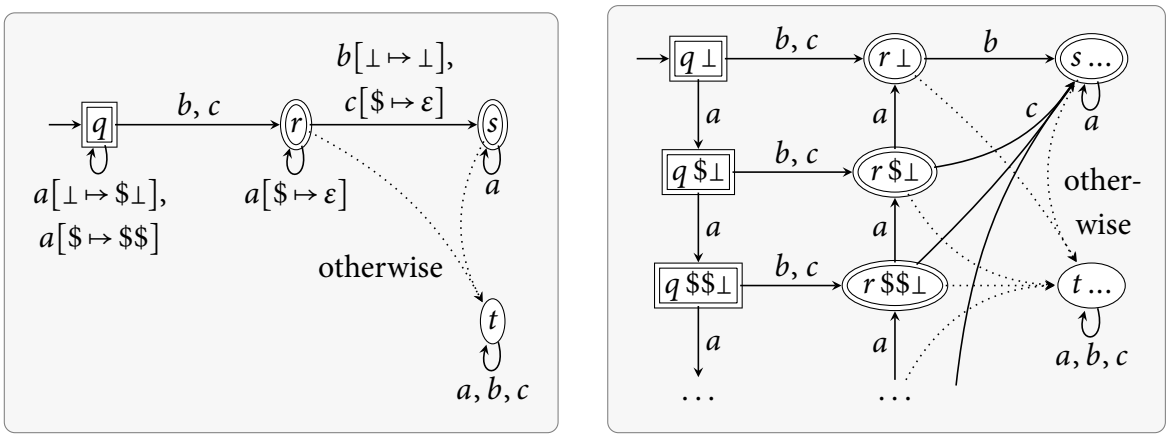
Example 3.1.1. Let $\mathcal{G} = (\mathcal{A}, Q_0)$ be a safety OCG based on the safety ω-DOCA $\mathcal{A}$ as depicted in Figure 3.1. The winning condition is given by the language

$$L_\omega(\mathcal{A}) \;=\; \left\{ a^\omega + a^n(b+c)(a^n b + a^m c)a^\omega \;\middle|\; n > m \right\}.$$

In the game, Player 1 starts by giving a sequence of letters a succeeded by a letter b or c. Then, Player 0 can either give an a-sequence of the same length proceeded by ba^ω, or he gives a shorter a-sequence succeeded by ca^ω.

In $\mathcal{G}$, Player 0 has a winning FSS although the ω-language of $\mathcal{A}$ is not regular. Such a strategy would be to answer a play starting with $(b+c)$ by ba^ω, whereas a play starting with $a^+(b+c)$ is answered by ca^ω. In fact, Player 0 has to remember only the first letter played by his opponent. ◁

As another example, we want to reconsider the claim about XML verification from the introduction on page 4.



(a) A safety ω-DOCA (the stack is ignored if no operation is given)

(b) The configurations yield a safety game graph

Figure 3.1: A safety OCG on the configuration graph of a safety ω-DOCA (circled states belong to Player 0, boxed to Player 1; doubly bordered states have color 2, otherwise color 1)

Example 3.1.2. We want to show that the problem studied in [SV02, SS07] can be reduced to the regularity problem for pushdown games. The problem asks whether it is possible to verify an XML documents against a DTD by using only a finite state device.

From a theoretical point of view, a DTD is a finite set of rules that describe a language of (unranked) trees where each node has a label of some finite alphabet and the root has a certain fixed label. For each label, a rule specifies the allowed label sequences of the children by a regular language. E.g., for the alphabet $\{r, a, b\}$ with root label r, the rules $r \to L_r$, $a \to L_a$ and $b \to L_b$ with $L_r = a^*$, $L_a = b$, $L_b = \varepsilon$ generate all trees such that

a) r is at the root,

b) below r, there is a finite sequence of a (maybe empty, then the root is the only node), and

c) below each a, there is one b.

The linearization of such a derivation tree t is a string $\mathrm{lin}(t)$ defined as follows. If the symbol at the root is a and below that there are the subtrees $t_1, \ldots, t_n$, then

$$\mathrm{lin}(t) = [_a \cdot \mathrm{lin}(t_1) \cdots \mathrm{lin}(t_n) \cdot]_a.$$

The question studied in [SV02, SS07] is the following: Given a DTD D, can the set of linearizations of its derivation trees be accepted by a finite automaton, provided that only

correct linearizations are given to this automaton. In other words, if Lin is the set of all linearizations (for the alphabet of the DTD D), and $\mathrm{lin}(D)$ is the set of all linearizations of derivation trees of D, then the question is whether there is a finite state automaton $\mathcal{A}$ such that

$$\mathrm{lin}(D) \;=\; L_*(\mathcal{A}) \cap \mathrm{Lin}\,.$$

This problem can be restated as a regularity problem for a safety pushdown game as follows. We can build a DPDA over the alphabet for linearizations that recognizes $\mathrm{lin}(D)$. Now, we add the symbols #, Y, N to the alphabet. The idea is that Player 1 plays symbols from the alphabet for linearizations, and at some point #. After #, Player 0 has to play Y (Yes) or N (No) to declare that the word played by Player 1 is in $\mathrm{lin}(D)$ or not. However, Player 0 only needs to make this decision in the case that Player 1 played a correct linearization (this can be encoded in the pushdown game). The game is made such that a wrong decision of Player 0 leads to a state that is losing (color 1). All other states have color 2.

Now, it is not difficult to see that Player 0 has a finite state strategy in this game if, and only if, a finite automaton $\mathcal{A}$ exists with the required property. Indeed, a finite state strategy for Player 0 can be used as $\mathcal{A}$ and vice versa, $\mathcal{A}$ can be used to define a finite state strategy. ◁

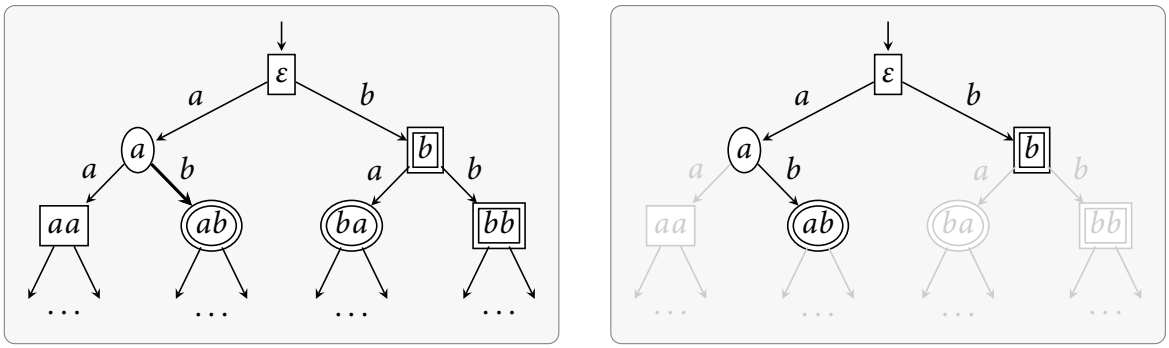
Note that we implicitly follow the perspective of Player 0 when talking about winning conditions because some restrictions of parity acceptance are not self-dual. E.g., a safety condition corresponds to a reachability condition from the perspective of his opponent, whereas the roles are exchanged in a reachability game. The results of this section show that the existence of a winning FSS is not symmetric: it depends on the perspective.

Reachability Winning Conditions

We start with the case of a reachability condition which turns out to be trivially winnable with an FSS.

Lemma 3.1.3. *In a reachability PDG, Player 0 has a winning finite state strategy as soon as he has a winning strategy.*

Proof. Let $\mathcal{G} = (Q, \Sigma, \Gamma, \delta, q_0, \bot, \Omega, Q_0)$ be a reachability PDG (with a weak coloring function Ω only using colors $\Omega(Q) = \{0,1\}$) and let f be a winning strategy for Player 0. We consider all possible play prefixes where Player 0 plays according to f until a state of color 0 is reached. Such a state exists on each play since f is winning for Player 0. When arranging these play prefixes as a tree as depicted in Figure 3.2, we obtain a finitely branching tree in which each branch is finite. Further, the tree must have finitely many nodes according to König's Lemma.



(a) Tree of play prefixes with a winning strategy for Player 0 (*a* to *ab*)

(b) Tree of play prefixes is finite after pruning

Figure 3.2: Pruning the tree of play prefixes of a reachability PDG in Lemma 3.1.3 (circled states belong to Player 0, boxed to Player 1; doubly bordered states have color 0, otherwise color 1)

The tree is hence finite and contains enough information for Player 0 to win. Using the nodes of the tree as states of a finite state machine directly yields a winning strategy. □

In the proof, we only used that each vertex of the game has a bounded number of successors and that the corresponding edges are labeled deterministically. The result can be lifted to more general classes of games as soon as this property is fulfilled. For PDGs, we know that the winner can be determined effectively [Wal01]. The next result is then a consequence of Lemma 3.1.3.

Corollary 3.1.4. *For a reachability pushdown game, it is decidable whether Player 0 has a winning finite state strategy.*

Safety Winning Conditions

To show the asymmetry of the problem, we prove that the existence of a winning FSS for a safety condition is undecidable. We do this by constructing a safety one-counter game that encodes the run of a 2-register machine (2RM; cf. Section 2.4). The goal idea is that Player 0 can win the game with an FSS if, and only if, the 2RM halts.

In our game, the counter can simulate only one of the two registers of a 2RM while the other one is completely ignored. Player 1 chooses the simulated register at the beginning in such a way that an FSS of Player 0 cannot recognize his choice. Player 1 achieves this by arbitrarily playing actions that increase and decrease the counter, followed by a distinct action

that determines which register is simulated depending on whether the counter is zero or not at this point. Afterwards (starting with empty counter), Player 0 has to play transitions of the 2RM while the game checks the correctness with respect to the state and the simulated register. Since an FSS of Player 0 cannot remember which register is simulated, a correctly played transition sequence corresponds to the run of the 2RM. By (a sequence of) transitions, we mean domain elements of the deterministic transition function of the 2RM, i.e., a triple of the state and the signs of the two register values.

A first approach would be to encode the halting problem, which can be achieved by additionally demanding Player 0 to reach a halting state. The idea of the resulting game is sketched in Figure 3.3a. The downside of this approach is that the winning condition is not captured by a safety condition since the acceptance can change twice: in the first part, an infinite play is good for Player 0, in the second part, it is bad, whereas reaching the halting state is good again.

To overcome this problem, we consider the slightly different approach sketched in Figure 3.3b, where a play is considered good as long as Player 0 can give an infinite sequence of 2RM transitions that is correct for the simulated register. Obviously, for each 2RM, Player 0 can win this game by just playing the transitions of the run as this is correct for both registers. We refer to the transitions (i.e., triples of state and signs of register values) of the run as **run signature**. But, in order to win with an FSS, this run signature must be representable with finite memory, which means that it has to be ultimately periodic, i.e., the sequence forms an infinite word uv^ω for some nonempty finite words u, v. Since 2RM are Turing complete, there are some without an ultimately periodic run signature.

Example 3.1.5. Consider a 2RM that swaps register values and increases the value after each swap. The run looks as follows (intermediate configurations are left out):

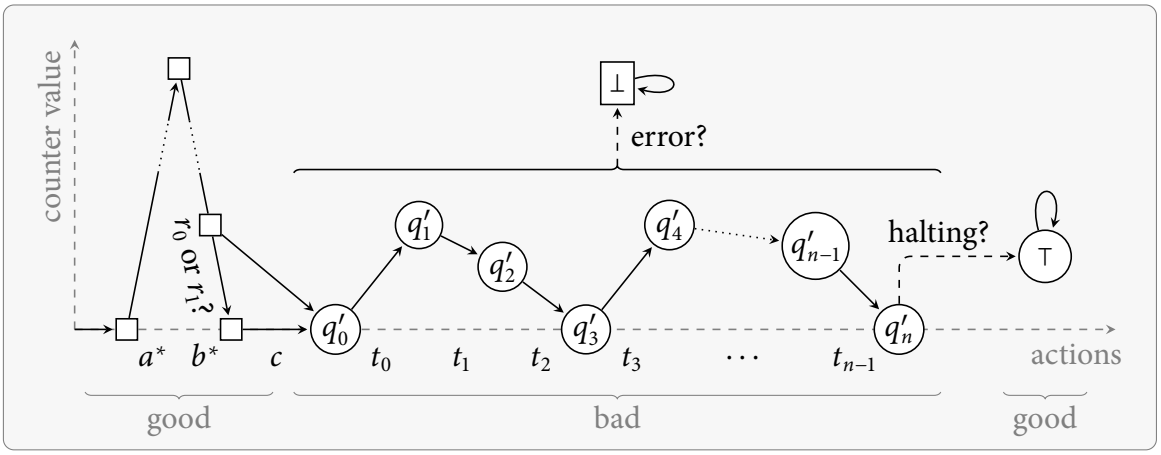
$$(q,0,0),\ldots,(q,0,1),\ldots,(q,2,0),\ldots,(q,0,3),\ldots,(q,4,0),\ldots,(q,0,5),\ldots$$

In the run signature, the element $(q,0,1)$ occurs infinitely often (induced by the configurations $(q,0,1),(q,0,3),(q,0,5),\ldots$) but the distance in between two consecutive occurrences is unbounded. It is hence not ultimately periodic. ◁

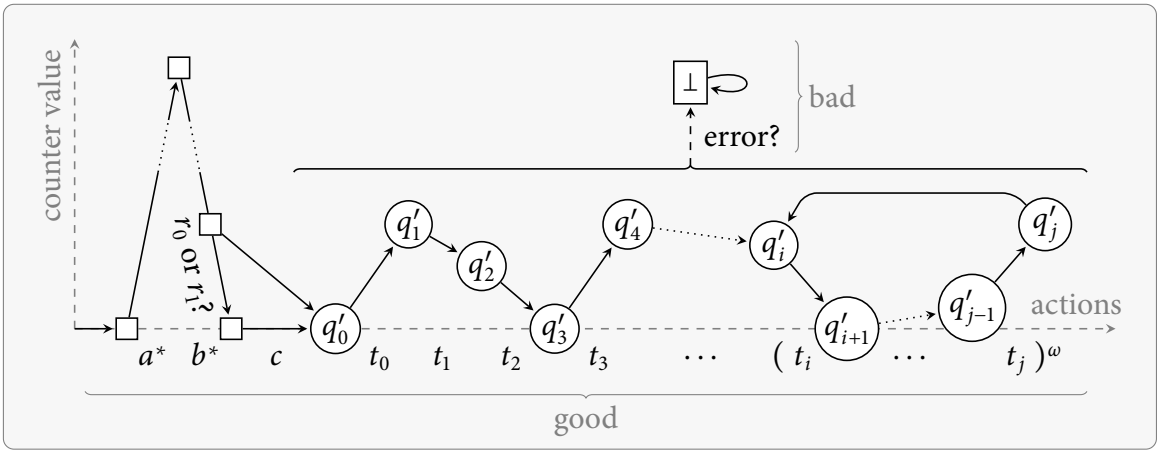
Before we proceed with the game result, we show this property of the run signature to be undecidable as well.

Lemma 3.1.6. *It is undecidable whether the run signature of a 2RM is ultimately periodic.*

Proof. We show the claimed undecidability by a Turing reduction of the halting problem. To this end, we construct two different 2RM $\mathcal{M}_0, \mathcal{M}_1$ from a given 2RM $\mathcal{M}$. Both these machines



(a) Reduction of the halting problem of a 2RM to a weak OCG



(b) Reduction of the ultimately periodic run problem of a 2RM to a safety OCG

Figure 3.3: Simulating the run of a 2RM by a weak OCG

first simulate $\mathcal{M}$ until it reaches a halting state. Then, $\mathcal{M}_0$ continues with a computation that has an ultimately periodic run, whereas $\mathcal{M}_1$ continues with one not being ultimately periodic. By construction, $\mathcal{M}$ reaches the halting state if, and only if, the property of having an ultimately periodic run differs for $\mathcal{M}_0$ and $\mathcal{M}_1$. □

Theorem 3.1.7. *For an ε-free safety OCG, it is undecidable whether Player 0 has a winning finite state strategy.*

Proof. By using Lemma 3.1.6, it suffices to construct a safety OCG $\mathcal{G}$ that simulates a given 2RM $\mathcal{M}$ such that Player 0 can win $\mathcal{G}$ with an FSS iff $\mathcal{M}$ has an ultimately periodic run signature. As explained before, $\mathcal{G}$ is sketched in Figure 3.3b and is divided into two phases.

First, by increasing and decreasing the counter, Player 1 determines by an empty counter at a certain position whether register 0 or 1 is simulated by the counter. In the second phase, Player 0 has to play an infinite transition sequence of $\mathcal{M}$ that is correct with respect to the state and the simulated register. A play is won by Player 0 iff Player 1 never leaves the first phase or the second phase is reached and Player 0 can give an infinite sequence that is a correct simulation of $\mathcal{M}$. This corresponds to a safety winning condition.

For a given 2RM $\mathcal{M} = (Q', \delta', q_0', F')$, we formally construct the following ε-free safety OCG $\mathcal{G} = (Q, \Sigma, \Gamma, \delta, q_0, \bot, \Omega, Q_0)$ with

a) states $Q = Q_0 \cup Q_1$ where $Q_0 = \{q_\bot\} \cup \big(Q' \times \mathbb{B}\big)$ and $Q_1 = \{q_0, q_{1,0}, q_{1,1}, q_\top\}$,

b) safety coloring $\Omega : Q \to \{1, 2\}$ where $\Omega(q) = 1$ iff it is the bad state $q = q_\bot$,

c) alphabets $\Sigma = \{a, b, c\} \cup \big(Q' \times \mathbb{B}^2\big)$ and $\Gamma = \{\$\}$,

d) transitions:

i) first phase (for $A \in \Gamma_\bot$ and $i \in \mathbb{B}$):

$$\delta(q_0, A, a) = (q_0, \$A),$$

$$\delta(q_0, A, b) = (q_0, W) \qquad \text{where } W = \begin{cases} \bot & \text{if } A = \bot, \\ \varepsilon & \text{if } A = \$, \end{cases}$$

$$\delta(q_0, A, c) = (q_{1,j}, A) \qquad \text{where } j = \begin{cases} 0 & \text{if } A = \bot, \\ 1 & \text{if } A = \$, \end{cases}$$

$$\delta(q_{1,i}, A, b) = (q_{1,i}, W) \qquad \text{where } W = \begin{cases} \bot & \text{if } A = \bot, \\ \varepsilon & \text{if } A = \$, \end{cases}$$

$$\delta(q_{1,i}, \bot, c) = \big((q_0', i), \bot\big),$$

ii) second phase (for $\delta'(p', s_0, s_1) = (q', d_0, d_1)$, $A \in \Gamma_\bot$, and $i \in \mathbb{B}$):

$$\delta\big((p', i), A, (p', s_0, s_1)\big) = \big((q', i), W\big)$$

$$\text{where } A = \begin{cases} \bot & \text{if } s_i = 0, \\ \$ & \text{if } s_i = 1, \end{cases} \quad \text{and } W = \begin{cases} \varepsilon & \text{if } d_i = -1, \\ A & \text{if } d_i = 0, \\ \$A & \text{if } d_i = +1, \end{cases}$$

iii) all other cases:

$$\delta(p, A, x) = (q, A) \qquad \text{where } q = \begin{cases} q_\perp & \text{if } p \in Q_0, \\ q_\top & \text{if } p \in Q_1. \end{cases}$$

In detail, the construction works as follows. The game is supposed to start at the vertex $q_0\perp$. In the first phase, Player 1 can play actions $a^n b^m c b^\ell c$ with $n \leq m + \ell$. This leads to the vertex $(q_0', i)\perp$ with $i = 0$ if $n = m$ or $i = 1$ otherwise. In the second phase, the game simulates register i of $\mathcal{M}$, i.e., the counter mimics all increase and decrease operations and verifies zero-tests only for register i. Player 0 continues by playing an infinite sequence of transitions of $\mathcal{M}$, i.e., a sequence over $Q' \times \{0,1\}^2$. Player 0 loses if, and only if, his transition sequence is inappropriate with respect to the simulated register and state.

If $\mathcal{M}$ has an ultimately periodic run signature, then Player 0 has a winning FSS by playing exactly this signature since all transitions are correct no matter which register is simulated. For the converse, assume that Player 0 has a winning FSS $\mathcal{S}$ with s states. When Player 1 starts with an action sequence a^s, then a state repetition occurs in $\mathcal{S}$. Let $\delta^*_{\mathcal{S}}(a^x) = \delta^*_{\mathcal{S}}(a^y)$ with $0 \leq x < y \leq s$. By continuing from this strategy state with actions $b^x c b^y c$, we see that $\mathcal{S}$ remains in the same state $\delta^*_{\mathcal{S}}(a^x b^x c b^y c) = \delta^*_{\mathcal{S}}(a^y b^x c b^y c)$, whereas different registers are simulated in both cases. Since $\mathcal{S}$ is winning although it cannot know which register is simulated, it must produce an ultimately periodic transition sequence which is correct with respect to the state and both registers and which hence is the run signature of $\mathcal{M}$. □

Note that the game in the previous undecidability proof does not have the visibly property (i.e., where input symbols determine the type of the stack operation) although in the first phase, symbol a always induces a push, b a pop, and c and internal stack operation. The visibility property is violated in the second phase where the type of the stack operation is not only determined by the transition played by Player 0, but also by the register that is simulated.

Nevertheless, it is possible to recycle the idea of the previous proof for visibly pushdown games by introducing additional stack symbols. We will add a new dummy symbol which prevents an FSS of Player 0 from seeing what is happening on the stack, i.e., whether important or useless actions are performed.

Theorem 3.1.8. *For a safety (ε-free) VPDG, it is undecidable whether Player 0 has a winning finite state strategy.*

Proof. The main idea is based on the construction of Theorem 3.1.7. But due to the visibility property, we now have to prevent Player 0 from knowing the number of counting symbols on

the stack. We overcome this by introducing an additional dummy symbol # beside the counting symbol \$ to the stack alphabet such that for the simulation of the register, we only count \$ and ignore # on the stack. The first phase of the game proceeds as before. The difference occurs in the second phase when Player 0 has to play $\mathcal{M}$ transitions since he should not directly control the stack height. Instead, we put Player 1 in charge of verifying the precondition of the transition and updating the register value on the stack. Player 1 should only enter the verification mode if he is sure that the transition is not applicable at this point. Otherwise, he has to update the register according to the chosen transition where he uses the technique of the first phase to prevent Player 0 from seeing what is happening on the stack. Player 0 continues afterwards by playing the next transition.

For a given 2RM $\mathcal{M} = (Q', \delta', q_0', F')$, we formally construct the following safety VPDG $\mathcal{G} = (Q, \Sigma, \Gamma, \delta, q_0, \bot, \Omega, Q_0)$ with

a) states $Q = Q_0 \cup Q_1$ with $Q_0 = \{q_\bot\} \cup \big(Q' \times \mathbb{B}\big)$ and $Q_1 = \{q_0, q_\top\} \cup \big(Q' \times \mathbb{B}^2 \times \{-1, 0, +1\}\big) \cup \big(Q' \times \mathbb{B} \times \{-1, 0, +1\}\big) \cup \{\mathit{check}_0, \mathit{check}_1\}$,

b) safety coloring $\Omega : Q \to \{1, 2\}$ where $\Omega(q) = 1$ iff it is the bad state $q = q_\bot$,

c) alphabets $\Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_s$ with $\Sigma_c = \{a\}$, $\Sigma_r = \{b\}$, $\Sigma_s = \{c\} \cup \big(Q' \times \mathbb{B}^2\big)$ and $\Gamma = \{\$, \#\}$,

d) transitions:

i) first phase (for $A \in \Gamma_\bot$ and $i \in \mathbb{B}$):

$$\delta(q_0, A, a) = (q_0, \#A),$$
$$\delta(q_0, \#, b) = (q_0, \varepsilon)$$
$$\delta(q_0, A, c) = \big((q_0', i, 0), A\big) \qquad \text{where } i = \begin{cases} 0 & \text{if } A = \bot, \\ 1 & \text{if } A = \#, \end{cases}$$

ii) second phase (for $d \in \{-1, 0, +1\}$, $s \in \mathbb{B}$, $A \in \Gamma_\bot$, and $i \in \mathbb{B}$):

1) choose an $\mathcal{M}$ transition $\delta'(p', s_0, s_1) = (q', d_0, d_1)$:

$$\delta\big((p', i), A, (p', s_0, s_1)\big) = \big((q', i, s_i, d_i), A\big)$$

2) register update according to the chosen $\mathcal{M}$ transition:

$$\begin{aligned}
&\delta\big((q', i, s, d), A, a\big) = \big((q', i, d), \#A\big) && \text{(enter update mode)}\\
&\delta\big((q', i, +1), A, a\big) = \big((q', i, 0), \$A\big), && \text{(push \$ to satisfy } d = +1)\\
&\delta\big((q', i, d), A, a\big) = \big((q', i, d), \#A\big) \text{ if } d \neq +1 && \text{(push \# otherwise)}\\
&\delta\big((q', i, d), \#, b\big) = \big((q', i, d), \varepsilon\big), && \text{(ignore \#)}\\
&\delta\big((q', i, -1), \$, b\big) = \big((q', i, 0), \varepsilon\big), && \text{(consume \$ to satisfy } d = -1)\\
&\delta\big((q', i, 0), A, c\big) = \big((q', i), A\big), && \text{(end update only if } d = 0)
\end{aligned}$$

3) verify that the chosen $\mathcal{M}$ transition can be performed:

$$\begin{aligned}
&\delta\big((q', i, s, d), A, c\big) = (\mathit{check}_s, A) && \text{(enter verification mode)}\\
&\delta\big(\mathit{check}_s, \#, b\big) = (\mathit{check}_s, \varepsilon), && \text{(ignore \#)}\\
&\delta\big(\mathit{check}_0, \$, b\big) = (q_\bot, \varepsilon), && \text{(counterexample \$)}\\
&\delta\big(\mathit{check}_1, \bot, b\big) = (q_\bot, \varepsilon), && \text{(counterexample } \bot)
\end{aligned}$$

iii) all other cases:

$$\delta(p, A, x) = (q, W)$$

$$\text{where } q = \begin{cases} q_\bot & \text{if } p \in Q_0, \\ q_\top & \text{if } p \in Q_1, \end{cases} \quad \text{and } W = \begin{cases} \varepsilon & \text{if } d_i = -1, \\ A & \text{if } d_i = 0, \\ \#A & \text{if } d_i = +1. \end{cases}$$

The game is again supposed to start at the vertex $q_0\bot$. In the first phase, Player 1 can play actions $a^n b^m c$ with $n \geq m$. This leads to the vertex $(q_0', i, 0)\#^{n-m}\bot$ with $i = 0$ if $n = m$ or $i = 1$ otherwise. At this state, technically phase 2 starts where Player 1 can first add or remove dummy symbols by playing a second sequence $a^{n'} b^{m'} c$ which eventually hands the control over to Player 0 at the vertex $(q_0', i)\#^{n-m+n'-m'}\bot$. After each $\mathcal{M}$ transition chosen by Player 0, Player 1 has to either update the register accordingly or he can claim a simulation error of his opponent. For the update mode, Player 1 has again to play a sequence $a^{n''} b^{m''} c$ with arbitrary $n'' > 0$ such that exactly one \$ is added or removed according to the desired update. Similar to the first phase, an FSS of Player 0 cannot see whether the stack height was increased. If Player 1 otherwise claims an error, the game continues at state check_s which is to verify that the sign of the simulated register value is $s \in \mathbb{B}$. The stack is popped and Player 1 wins iff a counterexample is found. By this construction, the arguments of the existence of a winning FSS for Player 0 are as in the proof of Theorem 3.1.7. □

In Example 3.1.2, we reduced a problem of finite state XML verification to the regularity problem for safety PDGs. Although we just showed the latter problem to be undecidable, this does not necessarily transfer back to the problem of finite state XML verification considered in Example 3.1.2 due to the direction of the reduction.

3.2 Connecting Games and Automata: Classification Game

In this section, we transfer the regularity problem from pushdown games to pushdown automata. Instead of finite state strategies, we ask for the existence of finite state automata in the automata setting. As a useful connection between games and automata, we introduce a game called 'classification game' which mimics the behavior of a given pushdown ω-automaton on possible input ω-words. The idea of using this game based approach came from Christof Löding and appeared in [LR12, Definition 2] in a variant with weak acceptance that we will introduce later in Definition 3.2.2.

More formally, the game is defined for a given ω-DPDA $\mathcal{A}$ and a set C of colors which can be different from the colors of the ω-automaton. Player 1 gives an infinite input word, whereas Player 0 has to answer after each input letter by a color. In order to win, the color sequence given by Player 0 must be accepting (with respect to the parity acceptance) if, and only if, the input word given by Player 1 is in $L_\omega(\mathcal{A})$.

One can see from the above description that the winning condition has to express the logical biconditional of two parity conditions (both conditions must be fulfilled or both must not be fulfilled) similar to generalized parity conditions (cf. [CHP07]). One condition results from the simulation of $\mathcal{A}$ and the other is for the colors given by Player 0. This could be stated by a Muller condition over the product of C and the colors used by $\mathcal{A}$. Here, we rather stick with parity acceptance and use a standard construction that converts Muller acceptance into parity. This is realized by a memory structure called **latest appearance record (LAR)** [GH82, DJW97]. For a finite set M, let $\mathrm{LAR}_M = M! \times [m]$ consist of all permutations on M such that one of the $m = |M|$ indices is marked. We define functions for the update $\mathrm{UpLAR}_M : \mathrm{LAR}_M \times M \to \mathrm{LAR}_M$ and the prefix $\mathrm{PreLAR}_M : \mathrm{LAR}_M$ as follows:

$$\mathrm{UpLAR}_M\Big((l_0, \ldots, l_{n-1}, i), x\Big) = (l_j, l_0, \ldots, l_{j-1}, l_{j+1}, \ldots, l_{n-1}, j) \qquad \text{where } l_j = x,$$

$$\mathrm{PreLAR}_M\Big((l_0, \ldots, l_{n-1}, i)\Big) = (l_0, \ldots, l_i).$$

The update UpLAR_M looks for the given element in the permutation, moves it to the front, and stores the index of its old position. The prefix PreLAR just returns the first elements of the permutation up to the marked index.

Definition 3.2.1. For a finite set $C \subseteq \mathbb{N}$ of colors and an ω-DPDA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot, \Omega)$ using colors $D = \Omega(Q)$, we define the **classification game** $\mathcal{G}_{\mathcal{A},C} = (Q', \Sigma', \Gamma', \delta', q_0', \bot', \Omega', Q_0')$ as follows:

a) states $Q' = \{q_\bot', q_\top'\} \cup \big(Q \times \mathrm{LAR}_{C\times D} \times \mathbb{B}\big)$ with $Q_0' = \{q_\bot'\} \cup \big(Q \times \mathrm{LAR}_{C\times D} \times \{0\}\big)$ and initial state $q_0' = \big(q_0, (0, \ldots, |C \times D| - 1, 0), 0\big)$,

b) alphabets $\Sigma' = \Sigma \uplus C$ and $\Gamma' = \Gamma$,

c) transitions (for $a \in \Sigma$ and $c \in C$):

$$\begin{aligned}
&\delta'\big((q,\ell,0),A,c\big) = \big((q,\ell',1),A\big) && \text{where } \ell' = \mathrm{UpLAR}_{C\times D}\Big(\ell,\big(c,\Omega(q)\big)\Big),\\
&\delta'\big((q,\ell,1),A,\varepsilon\big) = \big((p,\ell,1),W\big) && \text{where } \delta(q,A,\varepsilon) = (p,W),\\
&\delta'\big((q,\ell,1),A,a\big) = \big((p,\ell,0),W\big) && \text{where } \delta(q,A,a) = (p,W),\\
&\delta'(p',A,x) = (q',A) && \text{otherwise, where } q' = \begin{cases} q_\bot' & \text{if } p' \in Q_0',\\ q_\top' & \text{if } p' \in Q_1', \end{cases}
\end{aligned}$$

d) coloring function (for $q \in Q$, $\ell \in \mathrm{LAR}_{C\times D}$, and $i \in \mathbb{B}$):

$$\begin{aligned}
\Omega'\big((q,\ell,i)\big) \;=\; & 2 * |C \times D| - 2 * |\,\mathrm{PreLAR}_{C\times D}(\ell)| - \big((c_{\min} + d_{\min}) \mod 2\big)\\
&\text{where } c_{\min} = \min\Big\{c \;\Big|\; (c,d) \in \mathrm{PreLAR}_{C\times D}(\ell)\Big\}\\
&\text{and } d_{\min} = \min\Big\{d \;\Big|\; (c,d) \in \mathrm{PreLAR}_{C\times D}(\ell)\Big\},\\
\Omega'(q_\bot') \;=\; & 1,\\
\Omega'(q_\top') \;=\; & 0.
\end{aligned}$$

W.l.o.g., we assume $\mathcal{A}$ to be ε-weak. ◁

To see that the constructed parity condition works as desired (namely, expressing the logical biconditional of the parity conditions of the two players), note that the relevant color (the lowest one occurring infinitely often) is given by the longest prefixes that occur infinitely often. The colors in the tuples of these prefixes are exactly the ones that occur infinitely often. Player 0 wins iff for all such prefixes, $c_{\min} + d_{\min}$ is even, i.e., either both are even or both are odd. Hence, the acceptance status of the color sequence played by Player 0 has to correspond to the acceptance status of the played input word. The ε-weakness is required to prevent that small colors are overseen during ε-transitions.

We further define a simpler version of the classification game that expresses weak parity acceptance. The memory structure becomes much simpler then since we only have to store the lowest color played by Player 0.

Definition 3.2.2. For a finite set $C \subseteq \mathbb{N}$ of colors and an ω-DPDA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot, \Omega)$, we define the **weak classification game** $\mathcal{G}'_{\mathcal{A},C} = (Q', \Sigma', \Gamma', \delta', q'_0, \bot', \Omega', Q'_0)$ as follows:

a) states $Q' = \{q'_\bot, q'_\top\} \cup \big(Q \times C \times \mathbb{B}\big)$ with $Q'_0 = \{q'_\bot\} \cup \big(Q \times C \times \{0\}\big)$ and initial state $q'_0 = \big(q_0, \max(C), 0\big)$,

b) alphabets $\Sigma' = \Sigma \uplus C$ and $\Gamma' = \Gamma$,

c) transitions (for $a \in \Sigma$ and $c \in C$):

$$\begin{aligned}
&\delta'\big((q,c,0),A,d\big) = \big((q,d,1),A\big) && \text{where } c \geq d,\\
&\delta'\big((q,c,1),A,\varepsilon\big) = \big((p,c,1),W\big) && \text{where } \delta(q,A,\varepsilon) = (p,W),\\
&\delta'\big((q,c,1),A,a\big) = \big((p,c,0),W\big) && \text{where } \delta(q,A,a) = (p,W),\\
&\delta'(p',A,x) = (q',A) && \text{otherwise, where } q' = \begin{cases} q'_\bot & \text{if } p' \in Q'_0,\\ q'_\top & \text{if } p' \in Q'_1, \end{cases}
\end{aligned}$$

d) coloring function (for $q \in Q$, $c \in C$, and $i \in \mathbb{B}$):

$$\Omega'\big((q,c,i)\big) = \Omega(q) + c + 2, \qquad \Omega'(q'_\bot) = 1, \qquad \Omega'(q'_\top) = 0.$$

W.l.o.g., we assume $\mathcal{A}$ to be ε-weak. ◁

Note that the winning condition is a parity condition in general, although the colors played by Player 0 correspond to weak acceptance over colors C. The game becomes weak if $\mathcal{A}$ is weak. This is the reason why we have to shift the colors of non-sink states by 2. To see that the parity condition indeed expresses the logical biconditional of the parity conditions of the two players, note that the color sequence chosen by Player 0 eventually stabilizes to some color $c \in C$ which allows us to shift the original parity condition by c (plus 2). The sum is thus even if, and only if, the acceptance status of the color sequence played by Player 0 corresponds to the acceptance status of the played input word.

The main purpose of the (strong and weak) classification games is that a winning strategy of Player 0 that is implemented by some kind of machine (pushdown, finite state, etc.) can be transformed directly into an ω-automaton (based on the same machine model) for $L_\omega(\mathcal{A})$ by using the color output of the strategy as colors for the acceptance. We proceed with a technical observation before we show the connection between ω-automata and winning strategies.

Remark 3.2.3. *Let $\mathcal{A}$ and $\mathcal{A}'$ be two ω-DPDAs with $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$ and let C be a finite set of colors. Then,*

a) the same strategies are winning in $\mathcal{G}_{\mathcal{A},C}$ and $\mathcal{G}_{\mathcal{A}',C}$ for Player 0, and

b) the same strategies are winning in $\mathcal{G}'_{\mathcal{A},C}$ and $\mathcal{G}'_{\mathcal{A}',C}$ for Player 0.

The claims hold true for Player 1 due to Proposition 2.2.1.

The claim follows easily from the two facts that the definition of a strategy only depends on the alphabet of the game (i.e., the input alphabet and the color set) and that the winning condition of the game only depends on the language. Both are independent of the automaton that the classification game is based on.

This insight is helpful to connect simplifications between the winning strategies in the game and the language that it classifies.

Connecting Regularity

Theorem 3.2.4. *Let $\mathcal{A}$ be an ω-DPDA that uses the color set C. $L_\omega(\mathcal{A})$ is regular iff Player 0 can win $\mathcal{G}_{\mathcal{A},C}$ with a finite state strategy.*

Proof. For the left to right implication, suppose $L_\omega(\mathcal{A})$ is regular which means, due to Proposition 2.1.5, that there is an ω-DFSA $\mathcal{A}'$ with $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$. Player 0 can clearly win the FSG $\mathcal{G}_{\mathcal{A}',C}$ by mimicking the transitions and colors of $\mathcal{A}$. By Proposition 2.2.3 (a), Player 0 also has a winning FSS in $\mathcal{G}_{\mathcal{A}',C}$ which is also winning in $\mathcal{G}_{\mathcal{A},C}$ due to Remark 3.2.3 (a).

For the reverse direction, we naturally transform a winning FSS $\mathcal{S} = (Q', \Sigma \uplus C, \delta', q'_0, \sigma)$ of Player 0 into an ω-DFSA $\mathcal{A}' = (Q', \Sigma, \delta'', q'_0, \Omega')$ that recognizes the language. For that purpose, we define the coloring $\Omega'(q') = \sigma(q')$ and transitions $\delta''(q', a) = \delta'\big(\delta'(q', \sigma(q')), a\big)$ for $q' \in Q'$ and $a \in \Sigma$. $\mathcal{A}$ and $\mathcal{A}'$ accept the same (regular) language since $\mathcal{S}$ is winning in the game $\mathcal{G}_{\mathcal{A},C}$ that simulates the acceptance of this language. □

Theorem 3.2.5. *Let $\mathcal{A}$ be a weak ω-DPDA that uses the color set C. $L_\omega(\mathcal{A})$ is regular iff Player 0 can win $\mathcal{G}'_{\mathcal{A},C}$ with a finite state strategy.*

Proof. The proof is similar to the one of Theorem 3.2.4. The reverse direction of the claim is in fact a special case of it since a winning strategy for Player 0 in $\mathcal{G}'_{\mathcal{A},C}$ is especially winning in $\mathcal{G}_{\mathcal{A},C}$ where Player 0 is less restricted in choosing the colors. For the left to right implication, we additionally have to show that there is a desired strategy which produces a weak sequence of colors.

Suppose that $L_\omega(\mathcal{A})$ is regular which means, due to Proposition 2.1.5, that there is a (possibly non-weak) ω-DFSA $\mathcal{A}'$ with $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$. According to Remark 3.2.3 (b), Player 0 can win $\mathcal{G}'_{\mathcal{A}',C}$ by the pushdown strategy that simulates the weak acceptance of $\mathcal{A}$. Due to Proposition 2.2.3 (a), Player 0 has a winning FSS in the FSG $\mathcal{G}_{\mathcal{A}',C}$ which is also winning in $\mathcal{G}_{\mathcal{A},C}$ due to Remark 3.2.3 (b). □

An observation in the latter proof is that the computational power of the automaton model (i.e., pushdown or finite state) is in some sense orthogonal to the expressiveness of the acceptance condition, meaning that the number of colors needed to accept a regular language cannot be reduced by introducing a stack. This statement can also be derived from the proof of Theorem 24 in [Sta83]. Here, we obtain it from the proof of the left to right implication by using an FSS of Player 0 as an equivalent weak ω-DFSA.

Remark 3.2.6. *If the language of a weak ω-DPDA is regular, then there exists a weak ω-DFSA that uses the same colors and accepts the same language.*

Simplifying ω-Acceptance

Besides the connections concerning regularity, we can use the classification games to express simplifications in the acceptance conditions. The following result is similar to Theorems 3.2.4 and 3.2.5 but focuses on the colors instead of regularity.

Theorem 3.2.7. *Let $\mathcal{A}$ be an ω-DPDA. Then,*

a) $L_\omega(\mathcal{A})$ can be recognized by an ω-DPDA with colors C iff Player 0 can win $\mathcal{G}_{\mathcal{A},C}$, and

b) $L_\omega(\mathcal{A})$ can be recognized by a weak ω-DPDA with colors C iff Player 0 can win $\mathcal{G}'_{\mathcal{A},C}$.

Proof. The claim is proven analogously to Theorems 3.2.4 and 3.2.5. From left to right, an (respectively weak) ω-DPDA with colors C can be used as a PDS that answers with colors according to the coloring function of $\mathcal{A}$. That strategy is winning as it colors input words according to their membership in $L_\omega(\mathcal{A})$.

For the reverse direction, the winning player has a winning PDS according to Proposition 2.2.3 (b). This can be used naturally as an (respectively weak) ω-DPDA with colors C that recognizes $L_\omega(\mathcal{A})$. □

Since PDGs are effectively determined (cf. Proposition 2.2.3 (b)), Theorem 3.2.7 allows us to decide whether the acceptance can be simplified. The decidability of these problems were certainly known before. E.g., it was shown how to decide whether an ω-DPDL can be

recognized with reachability or safety acceptance in [CG78, Theorem 6.2.4], respectively, with Büchi or co-Büchi acceptance in [Lin77, Remark 5.1]. The innovation of our approach is rather to describe the problem in terms of parity games, and that synthesized winning strategies almost coincide with ω-DPDAs with the desired acceptance.

Corollary 3.2.8. *It is decidable for any set C of colors whether an ω-DPDL can be recognized by an ω-DPDA (respectively, by a weak ω-DPDA) with colors C. In particular, this covers Büchi, co-Büchi, reachability, and safety acceptance.*

These problems are known to be undecidable when determinism is omitted [CG78, Proposition 6.2.1].

3.3 Regularity Test for Weak ω-DPDAs

In this section, we present a procedure to solve the regularity problem for weak ω-DPDAs. That means to decide, whether the ω-language of a given weak ω-DPDA is regular, i.e., it can also be recognized by an ω-FSA.

By combining Theorem 3.2.5 with the results from Section 3.1, we already obtain the decidability of the regularity problem for ω-DPDLs with reachability acceptance due to Lemma 3.1.3. This approach fails for other acceptance conditions like safety or weak acceptance in general due to Theorems 3.1.7 and 3.1.8. Despite to the negative results on games, we give a decision procedure in this section that works for ω-DPDAs with weak acceptance.

Our work is inspired by a normal form for weak ω-DFSAs which assigns minimal colors to each state (without changing the recognized ω-language) [Löd01]. Then, certain decision problems for weak ω-DFSAs become trivially related to the respective problem for DFSAs on finite words. In our case of pushdown automata, we need some more effort to establish the normal form, as we have to ensure that colors are minimal for each configuration instead of just states. The normal form finally allows us to apply known algorithms for finite words on normalized weak ω-DPDAs.

This section is subdivided in three parts: First, we define the normal form and prove its connection to languages of finite words. Then, we show how a weak ω-DPDA can be converted into the normal form. And finally, we combine both results to obtain our algorithmic results.

For the remainder of this section, let $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot, \Omega)$ be a weak ω-DPDA with largest color $k = \max\big(\Omega(Q)\big)$. W.l.o.g., we assume $k \geq 1$.

As a first important step, we define a language of finite words for an ω-automaton $\mathcal{A}$ by simply setting states with an even color as accepting. Since $\mathcal{A}$ is weak, this language characterizes those points where the acceptance changes between accepting and rejecting.

Definition 3.3.1. The **finitary language** $L_{\circledast}(\mathcal{A}) \subseteq \Sigma^*$ of $\mathcal{A}$ (as above) is the language $L_{\circledast}(\mathcal{A}) = L_*(\mathcal{A}')$ of finite words accepted by the DPDA $\mathcal{A}' = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ with $F = \{q \in Q \mid \Omega(q) \text{ is even}\}$. ◁

By $L_{\circledast}(\mathcal{A}_{qW})$ and $L_\omega(\mathcal{A}_{qW})$, we denote the respective languages recognized by $\mathcal{A}$ starting from the configuration qW instead of the initial configuration. The following observation is a direct consequence of the definitions.

Remark 3.3.2. *For all configurations qW and pV, if $L_{\circledast}(\mathcal{A}_{qW}) = L_{\circledast}(\mathcal{A}_{pV})$, then $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$.*

If the inverse implication would also be true, then we would have established a strong relation between the ω-language of a weak ω-DPDA and its language of finite words. The following example illustrates that this is unfortunately not true in general.

Example 3.3.3. For a number $n \in \mathbb{N}$ and alphabet $\Sigma = \{a, b, c\}$, consider the language defined as follows:

$$L_n = \bigcup_{x \in \{a,b\}} \left(x\{a,b\}^* x \{a,b\}^n c \Sigma^\omega \right).$$

The language depends on a number n to show a lower bound later on.

By its definition via a regular expression, L_n is obviously a regular ω-language. It is further easy to sea that for every ω-DFSA recognizing L_n, the number of states is at least exponential in n because before reading the first c, it has to remember the last $n+1$ symbols. Nevertheless, L_n can be recognized by a weak ω-DPDA with linearly (in n) many states and a constant stack alphabet. The idea is to write the string $w \in \{a, b\}$ that occurs before the first c onto the stack (using the initial state q_0). After the first c, the stack is popped and one can check the property with ease by reading the reversal w^{R} of w from the stack.

We define a weak ω-DPDA $\mathcal{A}_n = (Q, \Sigma, \Gamma, \delta, q_0, \bot, \Omega)$ with $L_\omega(\mathcal{A}_n) = L_n$ that consists of states $Q = \{q_0, \ldots, q_{n+1}, q_a, q_b, q_\top, q_\bot\}$, alphabets $\Sigma = \{a, b, c\}$, $\Gamma = \{\$_a, \$_b, \#_a, \#_b\}$, and the following transitions (where $A \in \Gamma_\bot$, $x, y \in \{a, b\}$, $z \in \Sigma$, $i \in \{1, \ldots, n\}$):

- $\delta(q_0, A, x) = \begin{cases} (q_0, \$_x A) & \text{if } A = \bot, \\ (q_0, \#_x A) & \text{if } A \neq \bot, \end{cases}$
- $\delta(q_0, A, c) = (q_1, A)$,
- $\delta(q_i, \#_x, z) = (q_{i+1}, \varepsilon)$,

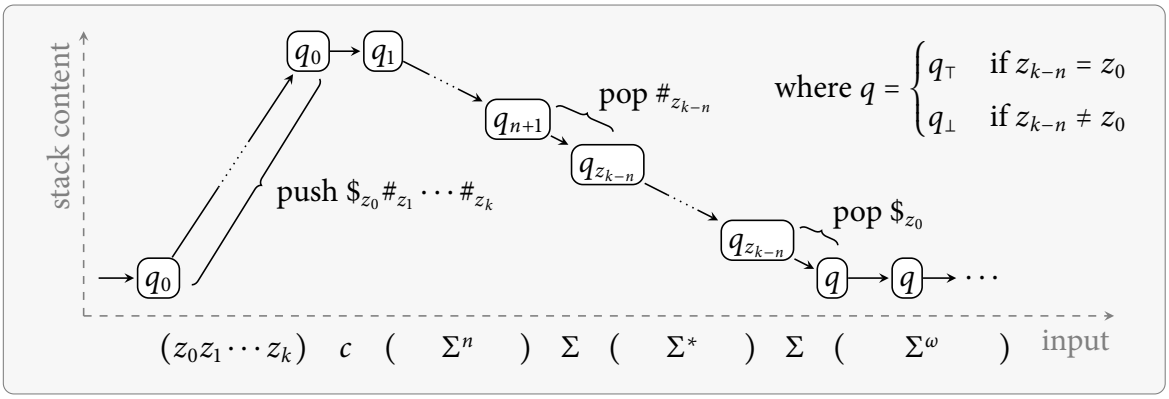


Figure 3.4: Weak ω-DPDA $\mathcal{A}_n$ from Example 3.3.3 that recognizes L_n with $\mathcal{O}(n)$ states

- $\delta(q_{n+1}, \#_x, z) = (q_x, \varepsilon)$,
- $\delta(q_x, \#_y, z) = (q_x, \varepsilon)$,
- $\delta(q_x, \$_y, z) = \begin{cases} (q_\top, \varepsilon) & \text{if } x = y, \\ (q_\bot, \varepsilon) & \text{if } x \neq y, \end{cases}$
- $\delta(q_\top, A, z) = (q_\top, A)$, and for all other transitions: $\delta(q, A, z) = (q_\bot, A)$,

and reachability coloring $\Omega : Q \to \{0,1\}$ where $\Omega(q) = 0$ iff $q = q_\top$. A run of $\mathcal{A}_n$ on an input word $\{a,b\}^{k+1}c\Sigma^\omega$ is sketched in Figure 3.4.

Let us further consider the inverse of Remark 3.3.2. For all configurations with a state from $\{q_1, \ldots, q_{n+1}, q_a, q_b\}$, it is already determined whether infinite words are accepted from there or not as this only depends on the stack content. For example, from configurations $q_a W \$_a \bot$ and $q_b V \$_b \bot$, all infinite words are accepted whereas the acceptance of finite words (in the finitary language) depends on the stack height and thus differs if W and V are of different lengths. Because of this, $L_\otimes(\mathcal{A}_n)$ is especially not regular:

$$L_\otimes(\mathcal{A}_n) = \bigcup_{x \in \{a,b\}} \bigcup_{i \in \mathbb{N}} \Big(x\{a,b\}^i x \{a,b\}^n c \Sigma^{1+i+1+n} \Sigma^* \Big). \qquad \triangleleft$$

3.3.1 Normal Form

The key ingredient of our proof is to establish a normal form for weak ω-DPDAs such that the inverse of Remark 3.3.2 holds true. For that purpose, we refine the coloring from only states to configurations by defining sets K_i (see below). The intuition is that each configuration should be assigned the lowest color possible without changing the accepted language.

Definition 3.3.4. We partition the configurations of $\mathcal{A}$ into **classes** K_i for $i \in \mathbb{N}$, where K_i is the biggest subset of $(Q\Gamma^*\bot) \setminus (\bigcup_{j<i} K_j)$ such that:

a) each run that stays in K_i forever is accepting iff i is even, and

b) each run that leaves K_i goes to $\bigcup_{j<i} K_j$. ◁

From the conditions (a) and (b), one can already see that the class indices form a weak parity condition as well. However, it is a nontrivial insight that the classes indeed form a well-defined partition of the set of configurations. On the one hand, it is easy to see that the maximum is unique because the union of sets fulfilling the two properties still fulfills them. To show that each configuration is contained in some class, we use the following stronger statement saying that each configuration is contained a class the index of which does not exceed its color.

Remark 3.3.5. *For each configuration $qW \in Q\Gamma^*\bot$ holds that $qW \in \bigcup_{j\leq\Omega(q)} K_j$.*

Proof. We use an induction over the colors.

For the initial case of $i = 0$, let $K'_0 = \{qW \mid \Omega(q) = 0\}$ be the set of all configurations with color 0. It is easy to see that $K'_0 \subseteq K_0$ because K'_0 fulfills the properties (a) and (b) of Definition 3.3.4 already.

For the inductive case of $i > 0$, assume that $qW \in \bigcup_{j\leq\Omega(q)} K_j$ holds for each configuration qW with $\Omega(q) < i$. Let all classes K_j with lower index $j < i$ already be determined and further, let $K' = (Q\Gamma^*\bot) \setminus (\bigcup_{j<i} K_j)$ be the nonempty set of configurations not being in any class yet and let $K'_i \subseteq K'$ contain only those configurations of K' that have the lowest color:

$$K'_i = \{qW \in K' \mid \Omega(q) = c\} \quad \text{where} \quad c = \min\{\Omega(q) \mid qW \in K'\}.$$

By showing the inclusion $K'_i \subseteq K_i$, it follows that $qW \in \bigcup_{j\leq\Omega(q)} K_j$ holds for each configuration qW with $\Omega(q) \leq i$ because $i \leq c$. To show the inclusion, we prove that K'_i fulfills the properties (a) and (b) of the definition. The property (b) is obviously fulfilled since we chose all configurations with the minimal color. For the property (a), we have to show that if there is a run staying in K'_i forever, then c and i have the same parity (i.e., c is even iff i is even). In fact, from each configuration in K'_i starts some run staying in K'_i forever because otherwise, the color of such a configuration has no impact on the acceptance and it must belong to some lower class due to the maximality criterion. However, c and i have the same parity because otherwise, c and $i - 1$ would have the same parity and then, the configurations of K'_i must belong to K_{i-1} already or even to some lower class with a parity different from c. □

A canonical representation of a weak ω-DPDA is, when the minimal coloring of each configuration corresponds to the coloring of its state.

Definition 3.3.6. $\mathcal{A}$ is in **normal form** if colors correspond to classes, i.e., $qW \in K_{\Omega(q)}$ for all $qW \in Q\Gamma^*\bot$ that are reachable from the initial configuration. ◁

Example 3.3.7. The classes K_i for $\mathcal{A}_n$ from Example 3.3.3 are (where $\# = \{\#_a, \#_b\}$):

$$K_0 = (q_\top\Gamma^*\bot) \cup \bigcup_{x\in\{a,b\}} \left(q_x\#^*\$_x\Gamma^*\bot\right) \cup \bigcup_{\substack{x\in\{a,b\}\\ i\in\{0,\dots,n\}}} \left(q_{n+1-i}\#^i\#_x\#^*\$_x\Gamma^*\bot\right),$$
$$K_1 = (Q\Gamma^*\bot) \setminus K_0,$$

In general, K_0 are exactly those configurations from where every word is accepted. All other configurations belong to K_1 in our example, like the ones with the bottom state ($q_\bot\Gamma^*\bot \subseteq K_1$) but also the initial state ($q_0\Gamma^*\bot \subseteq K_1$). Note that the classes also contain configurations that are not reachable in the automaton, like $q_\top\Gamma^+\bot \subseteq K_0$. Further, some states occur in different classes for different configurations, like $q_a\$_a\bot \in K_0$ but $q_a\$_b\bot \in K_1$, or $q_1\#_a\#^n\$_a\bot \in K_0$ but $q_1\#_a\#^n\$_b\bot \in K_1$. ◁

Example 3.3.7 shows that $\mathcal{A}_n$ is not in normal form. However, the classes K_0 and K_1 are regular sets of words which is true in general and can be used to transform $\mathcal{A}$ into normal form (see the next subsection). In the remainder of this subsection, we want to develop Lemma 3.3.10 which shows that the inverse direction of Remark 3.3.2 holds for our normal form. We start by exploring some important properties of the classes K_i that reflect the idea of representing the minimal color of each configuration.

Lemma 3.3.8. *Properties of K_i:*

a) For each configuration in K_i, there is a run which stays in K_i forever.

b) For each configuration in K_i with $i \geq 2$, there is a run leading to K_{i-1}.

Proof. a) Assume contrary that there is a configuration $qW \in K_i$ such that each run is eventually leaving it. Then, qW and its subsequent configurations in K_i are in K_j where $j < i$ is the biggest index of a class K_j a path from qW is leading to. This contradicts that qW is in at most one class.

b) Assume contrary that there is a configuration $qW \in K_i$ with $i \geq 2$ being minimal such that no run is leading to K_{i-1}. Then, qW and its subsequent configurations in K_i belong

to K_{i-2} as Definition 3.3.4 (a) and (b) are still fulfilled, which again contradicts that qW is in at most one class. Hence, no run from qW is leaving K_i. In this case, qW and its subsequent configurations belong to K_j where $j = i \bmod 2 < 2$ contrary to the assumption. □

The proof of the inverse direction of Remark 3.3.2 is a simple consequence of the following lemma which generalizes [Löd01, Lemma 7] to pushdown automata.

Lemma 3.3.9. *Let $\mathcal{A}$ be in normal form. If $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$, then $\Omega(q) = \Omega(p)$.*

Proof. Assume contrary that for some configurations $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$ but $\Omega(q) < \Omega(p)$ and choose the configurations such that $\Omega(q) + \Omega(p)$ is minimal. We consider two cases that lead to a contradiction. If $\Omega(q) + \Omega(p)$ is odd, then Lemma 3.3.8 (a) yields an ω-word α such that from pV only color $\Omega(p)$ is visited. The run on α from qW has the same acceptance and hence, due to the weakness it stabilizes at a color smaller than $\Omega(q)$. Let $q'W'$ and $p'V'$ be two configurations in the run on a prefix w of α from qW and pV, respectively, such that $\Omega(q') < \Omega(q)$ and $\Omega(p') = \Omega(p)$. Since $L_\omega(\mathcal{A}_{q'W'}) = L_\omega(\mathcal{A}_{p'V'})$, we have a contradiction to the minimality of $\Omega(q) + \Omega(p)$.

Otherwise, $\Omega(q)+\Omega(p)$ is even which implies $\Omega(q) < \Omega(p)-1$ and $\Omega(p) \geq 2$. Lemma 3.3.8 (b) guarantees that there is a word w leading from pV to $p'V'$ with color $\Omega(p') = \Omega(p)-1$. Let $q'W'$ be the configuration reached by w from qW, then $\Omega(q') \leq \Omega(q) < \Omega(p) = \Omega(p')$. Again $L_\omega(\mathcal{A}_{q'W'}) = L_\omega(\mathcal{A}_{p'V'})$, and we have a contradiction to the minimality of $\Omega(q) + \Omega(p)$. □

Based on this, the desired equivalence is straight forward.

Lemma 3.3.10. *Let $\mathcal{A}$ be in normal form. For all configurations qW and pV, if $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$, then $L_\circledast(\mathcal{A}_{qW}) = L_\circledast(\mathcal{A}_{pV})$.*

Proof. Assume contrary that there exists w.l.o.g. a word $w \in L_\circledast(\mathcal{A}_{qW}) \setminus L_\circledast(\mathcal{A}_{pV})$. Hence, $\Omega(q') \neq \Omega(p')$ for the configurations $q'W'$ and $p'V'$, reached by w from qW and pV, respectively. Lemma 3.3.9 yields $L_\omega(\mathcal{A}_{q'W'}) \neq L_\omega(\mathcal{A}_{p'V'})$ which implies $L_\omega(\mathcal{A}_{qW}) \neq L_\omega(\mathcal{A}_{pV})$. □

3.3.2 Normalization

This subsection is dedicated to Lemma 3.3.12 which proves that each weak ω-DPDA can be effectively transformed into one in normal form. As observed in Example 3.3.7, each class K_i forms a regular set. By proving this fact, we can extend a weak ω-DPDA by annotating the stack content with the run of a deterministic finite state machine (DFSM) that determines the

class. We can finally assign to each (annotated) state the minimal color of its configuration by using the information about the class index of the DFSM.

The following lemma is the first half of this normalization where we prove the regularity of the classes and construct a DFSM that reads a reversed configuration combined with an output function that yields the index i of the corresponding class K_i.

Lemma 3.3.11. *For $\mathcal{A}$, one can generate a DFSM $\mathcal{M}_K = (Q_K, \Sigma_K, \delta_K, q_{0,K})$ over the alphabet $\Sigma_K = \Gamma_\perp$, and a function $f_K : Q \times Q_K \to \mathbb{N}$ that assigns the classes: for each configuration, $qW \in K_i$ iff $f_K\big(q, \delta_K^*(W^{\mathrm{R}})\big) = i$. This computation takes time $2^{\mathcal{O}(|Q| \cdot k)}$ and yields size $|Q_K| \in 2^{\mathcal{O}(|Q| \cdot k)}$ where k is the largest color.*

Proof. We show this lemma by using the weak classification game $\mathcal{G}_{\mathcal{A},C}$ (cf. Definition 3.2.2). The colors chosen by Player 0 naturally represent Definition 3.3.4 meaning that $qW \in K_i$ iff $i \in C$ is the minimal color such that Player 0 can win from the vertex $(q, i, 0)W$. We justify this claim in the following. Further, according to Remark 3.3.5, it suffices to use the colors $C = \{0, \ldots, k\}$ in the classification game.

An obvious winning strategy for Player 0 from such a vertex (or vertices with larger colors) is to just play the corresponding index j such that $pV \in K_j$ for each game configuration $(p, i, 0)V$.

For showing the other direction, assume for a contradiction that Player 0 wins from a configuration $(q, i, 0)W$ with $qW \in K_j$ such that $i < j$ for minimal $i + j$. If $i + j$ is odd, then Player 1 can force a play that stays in W_j due to Lemma 3.3.8 (a). To win, Player 0 has to stabilize at a color $i' < i$ that is even iff j is even, i.e., $i' + j$ is even. Hence, $i' < j$ which contradicts the minimality. If $i + j$ is even, then $i < j - 1$ and $j \geq 2$. Player 1 can force the play to a configuration $(q', i', 0)W'$ with $q'W' \in K_{j-1}$ due to Lemma 3.3.8 (b). By assumption, Player 0 can still win from there which contradicts the minimality since $i' + j - 1 \leq i + j - 1 < i + j$.

From Lemma 2.2.4, we know that the winning region of Player 0 forms a regular language when considering those configurations as words. Let $\mathcal{A}'' = (Q'', \Sigma'', \delta'', q_0'', F'')$ be the according DFSA such that for each configuration $(q, i, 0)W$ of $\mathcal{G}_{\mathcal{A},C}$,

$$(q, i, 0)W \in L_*(\mathcal{A}'')^{\mathrm{R}} \;\Leftrightarrow\; \text{Player 0 wins } \mathcal{G}_{\mathcal{A},C} \text{ from } (q, i, 0)W.$$

As DFSM $\mathcal{M}_K = (Q'', \Sigma'', \delta'', q_0'')$ we take $\mathcal{A}''$ with out the acceptance. It remains to define the function $f_K : Q \times Q'' \to \mathbb{N}$:

$$f(q, q'') \;=\; \min\left\{ i \;\middle|\; \delta''\big(q'', (q, i, 0)\big) \in F'' \right\}$$

Then, we indeed have that $qW \in K_i$ iff $i = f(q, q'')$ with $q'' = \delta''^*(W^{\mathrm{R}})$ because:

$$\begin{aligned}\delta''\Big(q'', (q, i, 0)\Big) \in F'' &\Leftrightarrow \Big(W^{\mathrm{R}}(q, i, 0)\Big) = \Big((q, i, 0)W\Big)^{\mathrm{R}} \in L_*(\mathcal{A}'') \\ &\Leftrightarrow (q, i, 0)W \in L_*(\mathcal{A}'')^{\mathrm{R}} \\ &\Leftrightarrow \text{Player 0 wins } \mathcal{G}_{\mathcal{A},C} \text{ from } (q, i, 0)W.\end{aligned}$$

This computation takes time $2^{\mathcal{O}}(|Q| \cdot k)$ and yields size $|Q_K| \in 2^{\mathcal{O}(|Q| \cdot k)}$.

Lemma 2.2.4 yields that $|Q_K| \in 2^{\mathcal{O}(|Q| \cdot k)}$ since we applied it to $\mathcal{G}_{\mathcal{A},C}$ which has $\mathcal{O}(|Q| \cdot k)$ many states. To obtain the claimed time complexity, note that the winning condition of $\mathcal{G}_{\mathcal{A},C}$ is a weak PDG since $\mathcal{A}$ is weak. This can be rewritten as a Büchi condition by redefining odd colors as 1 and even colors as 0. For the resulting Büchi game, the running time of Lemma 2.2.4 becomes exponential in $\mathcal{O}(|Q| \cdot k)$. □

Example 3.3.7 illustrates that a DFSM that reads the configurations in reverse indeed has to be of size exponential in the size of the weak ω-DPDA.

The second half of the construction is to simulate the DFSM $\mathcal{M}_K$ on the configurations along a run of $\mathcal{A}$ by storing states of $\mathcal{M}_K$ on the stack in parallel to the actual stack symbols. This annotation tells in which state $\mathcal{M}_K$ would be after reading the configuration of $\mathcal{A}$ from the bottom up to the respective position. When additionally assigning the colors according to $\mathcal{M}_K$, we end up with the normalized weak ω-DPDA $\mathcal{A}'$. In constructions following later on, we are only interested in the finitary language which is the reason why we also construct a DPDA $\mathcal{A}''$ in the previous lemma.

Lemma 3.3.12. *For $\mathcal{A}$, one can compute in exponential time a weak ω-DPDA $\mathcal{A}'$ in normal form such that $L_\omega(\mathcal{A}') = L_\omega(\mathcal{A})$, and a DPDA $\mathcal{A}''$ such that $L_*(\mathcal{A}'') = L_\otimes(\mathcal{A}')$, where $\mathcal{A}''$ has $\mathcal{O}(|Q|)$ states and $|\Gamma| \cdot 2^{\mathcal{O}(|Q| \cdot k)}$ stack symbols.*

Proof. Let $\mathcal{M}_K = (Q_K, \Sigma_K = \Gamma_\perp, \delta_K, q_{0,K})$ be a DFSM with function f_K as in Lemma 3.3.11. We define a weak ω-DPDA $\mathcal{A}' = (Q', \Sigma, \Gamma', \delta', q_0', \perp', \Omega')$ with coloring $\Omega' : Q' \to \mathbb{N}$ in such a way that it runs like $\mathcal{A}$ and annotates the stack with a simulation of $\mathcal{M}_K$. Technically, $\mathcal{A}'$ consists of:

a) states $Q' = Q \times \{0, \ldots, k\}$ with $q_0' = (q_0, c_0)$ where $c_0 = f_K\Big(q_0, \delta_K(q_{0,K}, \perp)\Big)$, i.e., $q_0\perp \in K_{c_0}$,

b) stack symbols $\Gamma' = \Gamma \times Q_K$ with $\perp' = (\perp, q_{0,K})$,

c) transitions $\delta'\Big((q, c), (A, p_{0,K}), a\Big) = \Big((p, d), A_n' \cdots A_1'\Big)$ where

i) $\delta(q, A, a) = (p, A_n \cdots A_1)$,

ii) $A'_i = (A_i, p_{i-1,K})$ and $p_{i,K} = \delta_K(p_{i-1,K}, A_i)$ for $i \in \{1, \ldots, n\}$,

iii) $d = f_K(p, p_{n,K})$,

d) coloring $\Omega'\big((q, i)\big) = i$.

This construction converts a configuration $qA_n \cdots A_1$ of $\mathcal{A}$ to an annotated configuration $(q, i)A'_n \cdots A'_1$ of $\mathcal{A}'$ where

a) for each $\ell \in \{1, \ldots, n\}$, the stack symbol $A'_\ell = (A_\ell, q_{\ell,K})$ is annotated by the state that $\mathcal{M}_K$ is in after reading the configuration up to this point, i.e., $q_{\ell,K} = \delta^*_K(q_{0,K}, A_1 \cdots A_{\ell-1})$, and

b) the state is annotated by the minimal color $i = f_K\big(q, \delta^*_K(q_{0,K}, A_1 \cdots A_n)\big)$ of the configuration, i.e., $qA_n \cdots A_1 \in K_i$.

Hence, a bijection between the configurations of $\mathcal{A}$ and properly annotated configurations of $\mathcal{A}'$ is established. From Definition 3.3.4 (a), we get $L_\omega(\mathcal{A}') = L_\omega(\mathcal{A})$ because the colors in $\mathcal{A}'$ are set according to the classes K_i in $\mathcal{A}$. From Definition 3.3.4 (b) follows that $\mathcal{A}'$ is weak and the claimed complexity follows from the construction and Lemma 3.3.11.

We have already stated that the coloring of $\mathcal{A}'$ corresponds to the minimal coloring of $\mathcal{A}$. It remains to show that $\mathcal{A}'$ is in normal form, i.e., that this coloring also minimal for $\mathcal{A}'$. Let K'_i be the classes of $\mathcal{A}'$. We claim that $(q, i)A'_n \cdots A'_1 \in K'_i$ in $\mathcal{A}'$ iff $qA_n \cdots A_1 \in K_i$ in $\mathcal{A}$, i.e., that the classes in $\mathcal{A}'$ did not change in comparison to $\mathcal{A}$. For an inductive proof, assume that the correspondence of the classes is already established for indices 0 up to $i - 1$. Now consider K_i and the case that i is even (the other case is similar). K_i contains precisely those configurations from which all infinite runs are either accepting or are leading to K_j with $j < i$. Since the correspondence of the classes has been established up to $i - 1$, we know that for every run of $\mathcal{A}'$ from $(q, i)A'_n \cdots A'_1$ that leads to K'_j for $j < i$, the corresponding run of $\mathcal{A}$ from $qA_n \cdots A_1$ also leads to K_j, and vice versa. Furthermore, if all infinite runs from $(q, i)A'_n \cdots A'_1$ that do not lead to a smaller class are accepting, the same must be true for $qA_n \cdots A_1$ (and vice versa) because the same language is accepted from both configurations. Hence, the classes K_i and K'_i coincide (under the bijection) and $\mathcal{A}'$ is in normal form.

To obtain the DPDA $\mathcal{A}''$, we relax the additional information attached to the states of $\mathcal{A}'$. This information only indicates the class K_i the configuration is in, whereas it has no influence on the transitions of $\mathcal{A}'$. For the finitary language, it is enough to know whether the color is even or odd, which is where we can reduce the information to. It results in only $2 \cdot |Q|$ many

states instead of $k \cdot |Q|$. Each step of the computation can be done in at most exponential time. □

3.3.3 Decidability Results

An interesting special case of the combination of Remark 3.3.2 and Lemma 3.3.10 arises when we consider the initial configurations of two weak ω-DPDAs.

Remark 3.3.13. *Let $\mathcal{A}$ and $\mathcal{A}'$ be weak ω-DPDAs in normal form. Then, $L_\otimes(\mathcal{A}) = L_\otimes(\mathcal{A}')$ iff $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$.*

The following result is an immediate consequence since the **equivalence problem** for DPDAs was shown to be decidable in [Sén01, Sén02]. Although we did not intend to find it, it is a nice result as it concerns a fundamental decision problem in automata theory.

Corollary 3.3.14. *The equivalence problem is decidable for weak ω-DPDAs.*

Remark 3.3.13 further helps us to also reduce the regularity problem from the case of infinite words to finite words.

Theorem 3.3.15. *Let $\mathcal{A}$ be a weak ω-DPDA in normal form. Then, $L_\otimes(\mathcal{A})$ is regular iff $L_\omega(\mathcal{A})$ is regular.*

Proof. For the implication from finite to infinite words, suppose $L_\otimes(\mathcal{A})$ is regular, i.e., $L_\otimes(\mathcal{A}) = L_*(\mathcal{A}')$ for some DFSA $\mathcal{A}'$. We can view $\mathcal{A}'$ as a Büchi automaton $\mathcal{A}''$ by assigning color 0 to accepting states and color 1 to rejecting states. Obviously, for each input word, the state reached in $\mathcal{A}$ has even color iff the state reached in $\mathcal{A}''$ has color 0. Since the acceptance can only change finitely often, it is easy to see that $\mathcal{A}''$ can be converted to a weak ω-DFSA for an appropriate coloring. In each run of $\mathcal{A}$, the colors stabilize and hence, $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}'')$ which is a regular ω-language.

If $L_\omega(\mathcal{A})$ is regular, then it is recognizable by a weak ω-DFSA $\mathcal{A}'$ according to Remark 3.2.6. We assume $\mathcal{A}'$ to be in normal form (using our construction or the results in [Löd01]). By applying Remark 3.3.13, we obtain that $L_\otimes(\mathcal{A}) = L_\otimes(\mathcal{A}')$ is regular which shows the inverse implication. □

To obtain our desired decidability result, we use the fact that regularity is decidable for DPDAs [Ste67, Val75].

Corollary 3.3.16. *The regularity problem is decidable for weak ω-DPDAs.*

Complexity. Our method to decide the regularity problem for a weak ω-DPDA $\mathcal{A}$ is the composition of the following two subroutines:

a) synthesizing a DPDA $\mathcal{A}''$ that accepts the finitary language of the normal form of $\mathcal{A}$ (see Lemma 3.3.12), and

b) applying the known regularity test for DPDAs on $\mathcal{A}''$ (see Theorem 3.3.15 and [Val75]).

The first step runs in exponential time whereas the second step has a running time that is doubly exponential in the size of its input automaton. In total, this yields a triply exponential upper bound for the running time of Corollary 3.3.16.

Another interesting aspect in terms of complexity is the size of an equivalent weak ω-DFSA (if one exists). For the regularity test in the second step, we refer to [Val75] rather than [Ste67] as it gives better complexity bounds. Accordingly, the DPDA $\mathcal{A}''$ (recognizing a regular language) can be transformed into an equivalent DFSA with $E^2(n^2 \log n + \log t + \log h)$ states, where $E^i(f) = \exp^i\big(\mathcal{O}(f)\big)$ denotes an exponentiation tower of height i and $\mathcal{A}''$ has n states, t stack symbols, and words of at most length h in its transitions. According to Lemma 3.3.12, $\mathcal{A}''$ has $\mathcal{O}(|Q|)$ states and $|\Gamma| \cdot 2^{\mathcal{O}(|Q|^2)}$ stack symbols (we assume $k \in \mathcal{O}(|Q|)$). The composition yields a DFSA for the finitary language where the number of states is bounded by $E^2\big(|Q|^2 \log |Q| + \log |\Gamma| + \log h\big)$. This DFSA can be colored appropriately to become a weak ω-DFSA that is equivalent to $\mathcal{A}$.

Hence, in terms of computation time, our regularity test for ω-languages is exponentially more expensive than for languages of finite words. This is due to the exponential blowup in the normalization step. Nevertheless, this blowup has no impact on the size of the resulting weak ω-DFSA (if an equivalent one exists) since the exact same bound on the number of states applies as in the case of finite words. It is unclear, whether this bound is tight. From Example 3.3.7, we see that at least a single exponential blowup is unavoidable for the resulting weak ω-DFSA.

3.4 Congruences for Strong ω-DPDAs

In the previous section was shown how one can decide the regularity problem for ω-DPDAs with weak acceptance. Now, we consider regularity for general ω-DPDAs (with 'strong' acceptance) which was posed as an open problem decades ago in [CG78]. Unfortunately, we were not able to solve it either. We nevertheless want to present a congruence relation on finite words that characterizes regularity, meaning that the language of an ω-DPDA is regular iff the congruence has a finite index, i.e., finitely many congruence classes. The decidability of

these properties remains open. These results were obtained in collaboration with Wladimir Fridman.

For languages of finite words, congruences were used by Myhill and Nerode in the late 1980's to characterize regularity. In [AKMV05], a congruence was developed to characterize whether a language can be recognized by a VPDA. The domain of our congruence are also finite words, but it is different in the sense that its characterization property holds for (a restricted class of) ω-languages. The idea is based on a technique called Ramsey decomposition that was used in [Büc62] and involves Ramsey's theorem [Ram30]. First, we introduce the basic notations for congruences.

An **equivalence** is a binary relation that is reflexive, symmetric, and transitive. In this section, we consider equivalences over nonempty finite words. For an equivalence $\sim \subseteq (\Sigma^+)^2$ and a word $u \in \Sigma^+$, let the **class** $[u]_\sim = \{v \mid u \sim v\}$ of u be the set of all equivalent words, and let **index** $\mathrm{Ind}_\sim = \left|\{[v]_\sim \mid v \in \Sigma^*\}\right|$ be the number of classes. We call an equivalence on Σ^+ a **right congruence** (just called congruence in the following) if it is compliant with appending words to the right, i.e., if $u \sim v$, then $uw \sim vw$ for all $u, v, w \in \Sigma^+$.

Ramsey Decomposition

Our aim is to find a congruence enjoying a certain property that is about periodicity and depends on the ω-language L: for all $u, v \in \Sigma^+$ with $u \sim uv$ and $v \sim vv$, the ω-language $[u]_\sim[v]_\sim^\omega$ is either entirely contained in L or entirely outside of L. If the index of such a congruence happens to be finite, it is possible to represent L regularly by periodically repeating classes as in the above property. In [Büc62], this approach was used to show that Büchi automata are closed under complementation (which also proved the decidability of the theory of MSO logic).

The periodic decomposition is based on a combinatorial argument known as Ramsey's theorem [Ram30]. It considers a coloring function that assigns finitely many colors to the subsets of $\mathbb{N}$ that are of some fixed size k. The theorem predicts the existence of an infinite subset $M \subseteq \mathbb{N}$ that is monochromatic, i.e., all its k-size subsets have the same color. For a set S, let $\binom{S}{k}$ be all its subsets of size k.

Proposition 3.4.1 (Infinite Ramsey Theorem [Ram30]). *Let $c : \binom{\mathbb{N}}{k} \to C$ for some finite C. Then, there exists an infinite subset $M \subseteq \mathbb{N}$ such that $c(u) = c(v)$ for each $u, v \in \binom{M}{k}$.*

We proceed by giving the details of how this result can be used to obtain a regular representation.

Remark 3.4.2. *Let $L \subseteq \Sigma^\omega$ be an ω-language and $\sim$ be a right congruence on Σ^+ with finite index.*

a) Each class $[u]_\sim \subseteq \Sigma^+$ is regular.

b) Infinite words can be decomposed as follows:

$$\Sigma^\omega = \bigcup_{\substack{v \sim vv \\ u \sim uv}} [u]_\sim [v]_\sim^\omega.$$

c) If for all $u, v \in \Sigma^+$ with $u \sim uv$ and $v \sim vv$, either $[u]_\sim [v]_\sim^\omega \subseteq L$ or $[u]_\sim [v]_\sim^\omega \subseteq \overline{L}$ holds, then L is regular (and so is its complement) :

$$L = \bigcup_{\substack{v \sim vv \\ u \sim uv \\ uv^\omega \in L}} [u]_\sim [v]_\sim^\omega \quad \text{and} \quad \overline{L} = \bigcup_{\substack{v \sim vv \\ u \sim uv \\ uv^\omega \notin L}} [u]_\sim [v]_\sim^\omega.$$

Proof. a) For a right congruence, it holds that $[uw]_\sim = [vw]_\sim$ if $[u]_\sim = [v]_\sim$ for each $u, v, w \in \Sigma^+$. One can hence construct a canonical ε-free DFSM $\mathcal{M} = (Q, \Sigma, \delta, q_0)$ with finite state set $Q = \{q_0\} \uplus \left\{ [u]_\sim \;\middle|\; u \in \Sigma^+ \right\}$ and transitions $\delta(q_0, a) = [a]_\sim$ and $\delta\big([u]_\sim, a\big) = [ua]_\sim$. Then, $[u]_\sim = L_*(\mathcal{A}^u)$ where $\mathcal{A}^u = (\mathcal{M}, \{[u]_\sim\})$ has $[u]_\sim$ as the only accepting state.

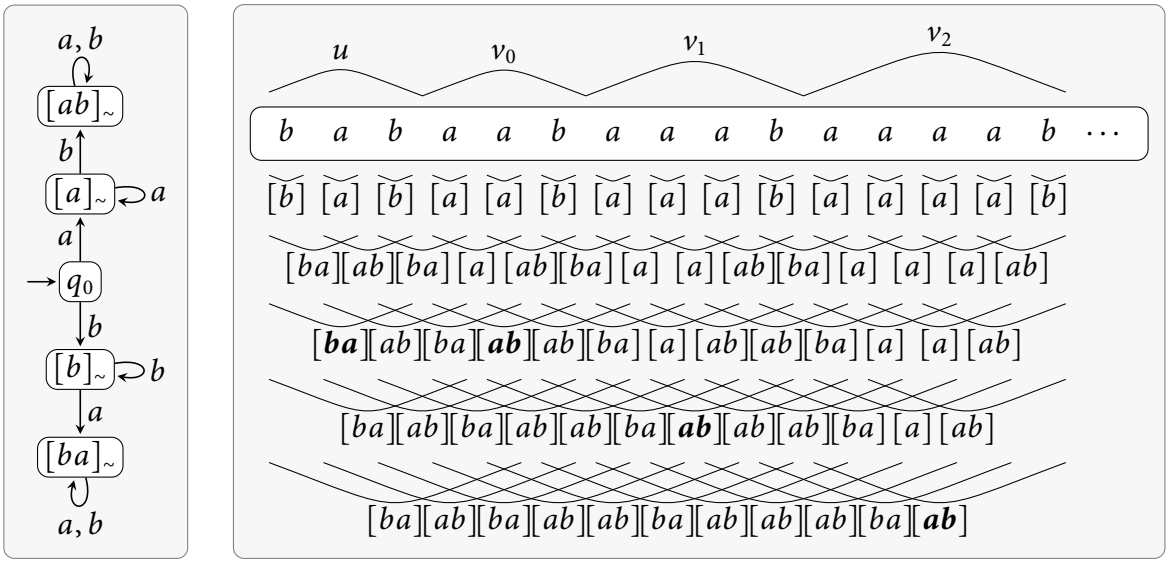
b) We have to show that for all $\alpha \in \Sigma^\omega$, there exists a decomposition $\alpha = uv_0v_1\cdots$ with $v_i \sim v_{i+1} \sim v_iv_{i+1}$ and $u \sim uv_0$ (and hence $u \sim uv_0 \cdots v_i$) for all $i \in \mathbb{N}$.

For that, we use Proposition 3.4.1 with k=2 and the coloring function $c : \binom{\mathbb{N}}{2} \to \left\{ [u]_\sim \;\middle|\; u \in \Sigma^+ \right\}$ which maps infixes of α to their respective equivalence classes: $c\big(\{i, j\}\big) = [a_i \cdots a_{j-1}]_\sim$ where $i < j$ and a_ℓ denotes the letter of α at position $\ell \in \mathbb{N}$. The Ramsey Theorem yields a set $M \subseteq \mathbb{N}$ of infinitely many word positions $m_0 < m_1 < \ldots$ such that all infixes in between two such positions are equivalent. Formally, the decomposition $\alpha = u'v_0'v_1' \cdots$ with $u' = a_0 \cdots a_{m_0-1}$ and $v_i' = a_{m_i} \cdots a_{m_{i+1}-1}$ meets the first property due to the choice of c: $v_i' \sim v_{i+1}' \sim v_i'v_{i+1}'$ for all $i \in \mathbb{N}$.

It remains to additionally establish equivalent prefixes. Based on the above decomposition, we partition the prefixes of α depending on the classes they belong to:

$$I_{[w]_\sim} = \left\{ i \in \mathbb{N} \;\middle|\; w \sim u'v_0' \cdots v_{i-1}' \right\} \quad \text{for all classes } [w]_\sim.$$

By the pidgin hole principle, there must be some class $[w]_\sim$ such that $I_{[w]_\sim}$ contains infinitely many indices $i_0 < i_1 < \ldots$. We can finally define the coarser decomposition $u = u'v_0' \cdots v_{i_0-1}'$ and $v_j = v_{i_j}' \cdots v_{i_{j+1}-1}'$ which still fulfills the first property and additionally the second one: $u \sim uv_0$.



(a) DFSM to describe congruence classes

(b) Decomposition of an ω-word (the classes of words in the decomposition are highlighted; index ~ is omitted for readability)

Figure 3.5: Illustrations for Remark 3.4.2 (a) and (b) as explained in Example 3.4.3

c) Direct consequence of (a) and (b). □

Example 3.4.3. Consider a right congruence $\sim \subseteq (\Sigma^+)^2$ over the alphabet $\Sigma = \{a, b\}$ consisting of the following classes:

$$[a]_\sim = a^+, \qquad [b]_\sim = b^+, \qquad [ab]_\sim = a^+b\Sigma^*, \qquad [ba]_\sim = b^+a\Sigma^*.$$

The ε-free DFSM $\mathcal{M}$ according to Remark 3.4.2 (a) that can be used to recognize the classes of $\sim$ is depicted in Figure 3.5a.

In Figure 3.5b, we give an example of a decomposition of the ω-word $\alpha = ba^1ba^2ba^3 \cdots \in \Sigma^\omega$ in the spirit of Remark 3.4.2 (b). All possible decompositions must pick $u \sim ba$ because $[ba]_\sim$ is the only class that contains infinitely many prefixes of α. For our example, we choose $u = bab \sim ba$ and further $v_i = a^{i+2}b \sim ab$ for all $i \in \mathbb{N}$. This fulfills the desired conditions $v_i \sim v_{i+1} \sim v_iv_{i+1}$ and $u \sim uv_i$ for all $i \in \mathbb{N}$.

Finally, for Remark 3.4.2 (c), let ω-language $L = a(b^*a)^\omega + b(a^*b)^\omega \subseteq \Sigma^\omega$. It contains exactly those ω-words where the first letter occurs infinitely often. The following table summarizes the relation of all ω-languages $[u]_\sim[v]_\sim^\omega$ with respect to L (where '+' means $[u]_\sim[v]_\sim^\omega \subseteq L$ and '–' means $[u]_\sim[v]_\sim^\omega \subseteq \overline{L}$):

$u \setminus v$	a	b	ab	ba
a	+	−	+	+
b	−	+	+	+
ab	+	−	+	+
ba	−	+	+	+

Consequently, we obtain another representation of L (column-wise from the table):

$$L = \Big([a]_\sim + [ab]_\sim\Big)[a]_\sim^\omega + \Big([b]_\sim + [ba]_\sim\Big)[b]_\sim^\omega + \Sigma^*[ab]_\sim^\omega + \Sigma^*[ba]_\sim^\omega. \qquad \triangleleft$$

Regularity

In order to characterize regularity by using the above techniques, it remains to define a congruence being compliant with the preconditions of Remark 3.4.2 (c). The congruence that we consider here is composed of two sub-congruences: one that compares the prefix behavior of a word, and one that relates the behavior when occurring in a period.

Definition 3.4.4. For a language $L \subseteq \Sigma^\omega$ over some alphabet Σ, we define the following right congruences over nonempty words (where $u, v \in \Sigma^+$):

$$\begin{aligned}
u \sim v \quad &\text{iff} \quad u \approx v \ \wedge\ u \approxeq v,\\
u \approx v \quad &\text{iff} \quad \forall y \in \Sigma^\omega : \Big(uy \in L \ \Leftrightarrow\ vy \in L\Big),\\
u \approxeq v \quad &\text{iff} \quad \forall x, y \in \Sigma^* : \Big(x(uy)^\omega \in L \ \Leftrightarrow\ x(vy)^\omega \in L\Big).
\end{aligned} \qquad \triangleleft$$

Note that these three congruences depend on L. Nevertheless, we omit this in our notation as it will be clear which language we are working with. It follows easily from the definition that $\approx$ and $\approxeq$ are reflexive, symmetric, transitive, and compliant with appending words to the right. Finally, $\sim$ derives these properties.

Since $\sim$ is a conjunction, it holds that $\mathrm{Ind}_\sim$ is finite iff $\mathrm{Ind}_\approx$ and $\mathrm{Ind}_\approxeq$ are both finite. Further, if $L \subseteq \Sigma^\omega$ is regular, then $\mathrm{Ind}_\sim$, $\mathrm{Ind}_\approx$, and $\mathrm{Ind}_\approxeq$ are each finite where the following upper bounds can be obtained easily from an ε-free ω-DFSA that recognizes L with n states:

a) $\mathrm{Ind}_\sim \leq \mathrm{Ind}_\approx \cdot \mathrm{Ind}_\approxeq$ holds since each $\sim$-class is characterized by the intersection of a $\approx$-class with a $\approxeq$-class. Some intersections might happen to be empty.

b) $\mathrm{Ind}_\approx \leq n$ holds because $u \approx v$ follows for two words $u, v \in \Sigma^+$ if they lead to the same state, i.e., $\delta^*(u) = \delta^*(v)$.

c) $\mathrm{Ind}_{\approxeq} \leq (n^2)^n = n^{2n}$ holds because $u \approxeq v$ follows for two words $u, v \in \Sigma^+$ if they induce the same state transformation with the same lowest color, i.e., for all states $p, q \in Q$ and colors $c \in C$ (w.l.o.g. $|C| \leq |Q|$), the run $p \xrightarrow{u} q$ has lowest color c iff the run $p \xrightarrow{v} q$ exists and has the same lowest color.

Example 3.4.5. Reconsider the ω-language $L \subseteq \Sigma^\omega$ over alphabet $\Sigma = \{a, b\}$ from Example 3.4.3. The following table gives the two $\approx$-classes and three $\approxeq$-classes of L, and also four (nonempty) $\sim$-classes that result from the intersection:

$\cap$	$[a]_{\approxeq} = a^+$	$[b]_{\approxeq} = b^+$	$[ab]_{\approxeq} = \Sigma^*(ab + ba)\Sigma^*$
$[a]_{\approx} = a\Sigma^*$	a^+	$\varnothing$	$a^+b\Sigma^*$
$[b]_{\approx} = b\Sigma^*$	$\varnothing$	b^+	$b^+a\Sigma^*$

The resulting $\sim$-classes happen to coincide with those from Example 3.4.3. ◁

Before the example, we have seen that the index of $\sim$ is finite when the considered language is regular. We now show the inverse direction of that implication.

Theorem 3.4.6. *Let $L \subseteq \Sigma^\omega$ be an ω-DPDL. The index of $\sim$ is finite iff L is regular.*

Proof. We just have observed that the index of $\sim$ is finite if L is regular. For the other direction, we use Remark 3.4.2 (c), where two cases arise:

a) Assume there is an ω-word $\alpha = uv^\omega \in L$ for some $u, v \in \Sigma^+$ such that there exists another ω-word $\alpha' \in [u]_\sim [v]_\sim^\omega$ with $\alpha' \notin L$. Fix some decomposition $\alpha' = u'v'_0v'_1 \cdots$ with $u \sim u'$ and $v \sim v'_i$ for all $i \in \mathbb{N}$. Let further $L = L_\omega(\mathcal{A})$ be accepted by an ω-DPDA $\mathcal{A}$ and let the class $[v]_\sim = L_*(\mathcal{A}_v) \subseteq \Sigma^+$ be recognized by an ε-free DFSA $\mathcal{A}_v$ according to Remark 3.4.2 (a).

   For the proof, we decompose the rejecting run of $\mathcal{A}$ on α'. By infinitely pumping a certain infix of α', we construct an ultimately periodic ω-word $\alpha'' = u''v''^\omega$ that $\mathcal{A}$ still has a rejecting run on. A contradiction with the definition of $\sim$ will occur since $u \sim u''$ and $v \sim v''$ holds.

   For the decomposition of the run of $\mathcal{A}$ on α', consider the **stairs** of the run, i.e., all configurations such that in the future, the stack height does not drop below the current value. There can be multiple stairs at the same height if the stack reaches that height several times without dropping below later on. The infinite run induces infinitely many stairs. We refine this sequence as follows:

i) By the infinite pidgin hole principle, there must be an infinite subsequence of stairs that share the same state and topmost stack symbol. This allows us to arbitrarily pump the run in between two such stairs. Some prefix of the run leads to the earlier stair qAW for $q \in Q$, $A \in \Gamma_\perp$, and $W \in \Gamma_\perp^*$. From there, some input $w \in \Sigma^*$ leads to the later stair $qAVW$ for $V \in \Gamma_\perp^*$. Since the state and the stack top are the same and the stack content W below the level of the stair is never used again, it is possible to continue the run in an infinite loop by continuing with the input w^ω. Figure 3.6 depicts the runs and the two stairs as just described.

ii) To retain the acceptance behavior during the loop, we consider a subsequence of stairs such that the lowest color occurring in between two of them is odd. This sequence is still infinite since the run is rejecting. When looping between two such stairs, the run is rejecting.

iii) The word we want to pump has to be in the part where $v'_0 v'_1 \cdots \in [v]_\sim^\omega$ is processed. We hence pick the start of our stair sequence to be somewhere after the prefix u' has been read.

iv) To understand the final step of the refinement, let for a word $a_1 \cdots a_n \in \Sigma^n$ of n letters and $k \leq n$

$$[a_1 \cdots a_n \mid {<}k] = a_1 \cdots a_{k-1} \quad \text{and}$$
$$[a_1 \cdots a_n \mid {\geq}k] = a_k \cdots a_n$$

denote its prefix up to position k (exclusively) and its suffix from position k (inclusively) onwards. For each stair, $\mathcal{A}$ has processed a prefix $u' v'_0 \cdots v'_{i-1} [v'_i \mid {<}k]$ of α' for some $i \in \mathbb{N}$ and $k \in \{0, \ldots, |v'_i|\}$. We now additionally consider the run of $\mathcal{A}_v$ on v'_i and especially its state after the first k letters were processed: $\delta_v^*([v'_i \mid {<}k])$. Again, by the infinite pidgin hole principle, there must be an infinite subsequence of stairs that share the same state of $\mathcal{A}_v$ at the respective position of the stair.

For the remainder of the proof, we fix two of these infinitely many stairs. Let $i < j$ be the indices of the words v'_i and v'_j currently processed at the two stair positions, and let k, l be the respective positions within v'_i and v'_j, i.e., the prefixes $u' v'_0 \cdots v'_{i-1} [v'_i \mid {<}k]$ and $u' v'_0 \cdots v'_i \cdots v'_{j-1} [v'_j \mid {<}l]$ of α' have been processed at the two stair positions, respectively. The runs of $\mathcal{A}_v$ and the decompositions of v'_i and v'_j are also depicted in Figure 3.6.

We construct another ω-word $\alpha'' \in \Sigma^\omega$ which starts with the prefix $u' v'_0 \cdots v'_{i-1} [v'_i \mid {<}k]$ up to the first stair, and is followed by the extension $[v'_i \mid {\geq}k]\, v'_{i+1} \cdots v'_{j-1} [v'_j \mid {<}l]$ to the prefix at the second stair. The latter word is repeated ad infinitum. This ultimately

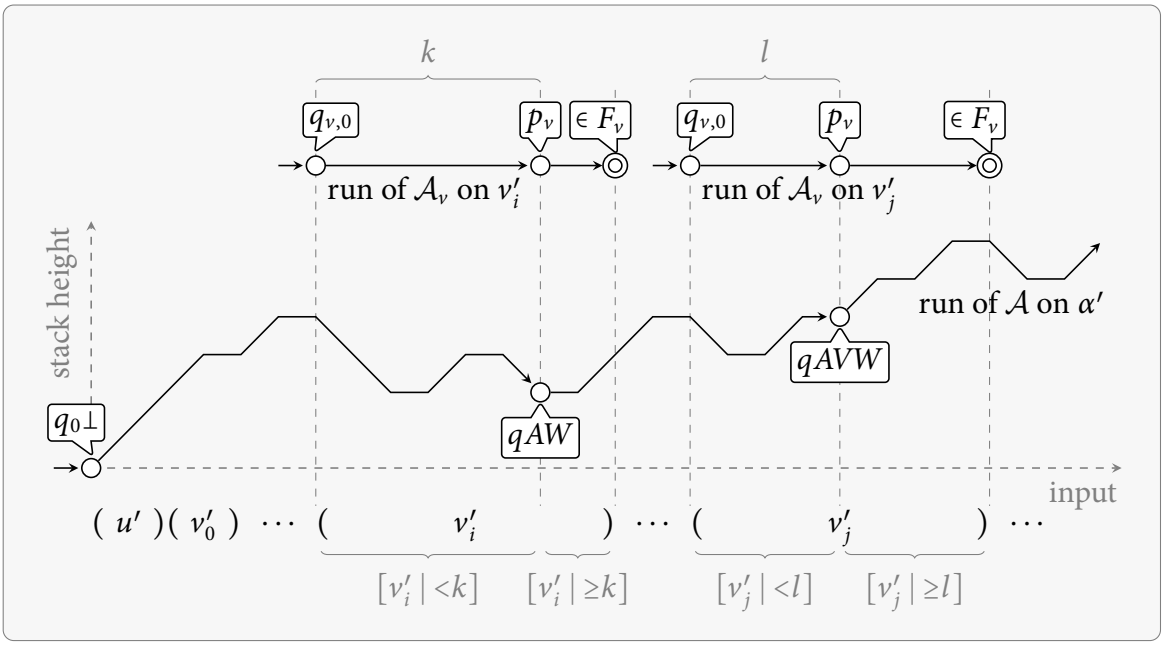


Figure 3.6: Selecting stairs for the proof of Theorem 3.4.6

periodic ω-word α'' is again not contained in the language by the choice of i, j, k, l. We obtain then representation $\alpha'' = u''(v'')^\omega$ when rearranging α'' as follows:

$$\begin{aligned}
\alpha'' &= u'\, v'_0 \cdots v'_{i-1} [v'_i \mid <k] \Big([v'_i \mid \geq k]\, v'_{i+1} \cdots v'_{j-1} [v'_j \mid <l] \Big)^\omega \\
&= u'\, v'_0 \cdots v'_{i-1} \underbrace{[v'_i \mid <k]\, [v'_i \mid \geq k]} \Big(v'_{i+1} \cdots v'_{j-1} [v'_j \mid <l]\, [v'_i \mid \geq k] \Big)^\omega \\
&= \underbrace{u'\, v'_0 \cdots v'_{i-1} \qquad v'_i} \quad \underbrace{\Big(v'_{i+1} \cdots v'_{j-1} [v'_j \mid <l]\, [v'_i \mid \geq k] \Big)^\omega} \\
&= \qquad\qquad u'' \qquad\qquad \Big(\qquad\qquad v'' \qquad\qquad \Big)^\omega \notin L
\end{aligned}$$

It follows that $u \sim u''$ since $u \sim uv \sim uv^{i+1} \sim u'v'_0 \cdots v'_i = u''$. By the choice of k, l, we further conclude that $[v'_j \mid <l][v'_i \mid \geq k] \in L_*(\mathcal{A}_v) = [v]_\sim$ which implies $v \sim v''$ since $v \sim vv$. We finally see from the definition of $\sim$ that $u''(v'')^\omega \notin L$ contradicts $uv^\omega \in L$ by Definition 3.4.4 since $u \approx u''$ and $v \approx v''$. Hence, $[u]_\sim [v]_\sim^\omega \subseteq L$.

b) If $uv^\omega \notin L$, then $[u]_\sim [v]_\sim^\omega \subseteq \overline{L}$ follows from the first case for the complement language since ω-DPDLs are closed under complement (see [HU79, Chapter 10.2]). □

Limitations

Note that ∼ can only be used for ω-DPDLs to characterize regularity. There are more complex languages that also have finite index.

Example 3.4.7. Consider the ω-language $L_{\mathrm{u}} \subseteq \{a, b\}^\omega$ that contains all words with a-blocks of unbounded length:

$$L_{\mathrm{u}} = \left\{ a^{n_0} b a^{n_1} b a^{n_2} b \cdots \,\middle|\, \text{the exponents } n_0, n_1, \ldots \in \mathbb{N} \text{ are unbounded} \right\} \subseteq \{a, b\}^\omega.$$

Then, $\mathrm{Ind}_{\sim} = 1$, i.e., all words are congruent with respect to ∼, ≈, and ≋, respectively. This is because changing the prefix of a word does not change its membership in L_{u} and further, periodic words are excluded from L_{u} by definition.

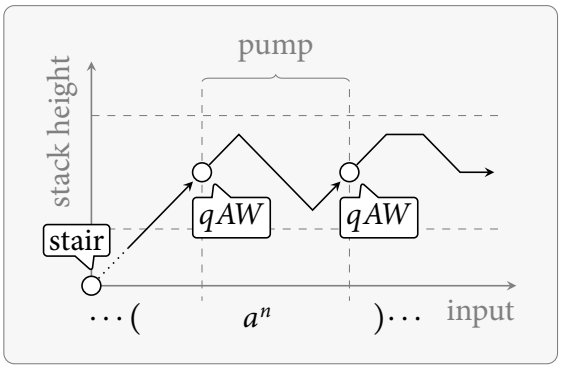
From Theorem 3.4.6 follows that L_{u} cannot be an ω-DPDL. In fact, L_{u} as well as its complement $\overline{L_{\mathrm{u}}}$ are not even recognizable by ω-PDAs:

a) The first claim follows since every nonempty ω-PDL contains an ultimately periodic word, which is not the case for L_{u}. Assume $L_{\mathrm{u}} = L_\omega(\mathcal{A})$ to be recognized by some ω-PDA $\mathcal{A}$ and consider the accepting run of $\mathcal{A}$ on some word. As in the proof of Theorem 3.4.6, there are two stairs in the run that share the same state and stack top and further, the lowest color occurring in between is even. A repetition of the segment between the two stairs yields an ultimately periodic word in L_{u}.

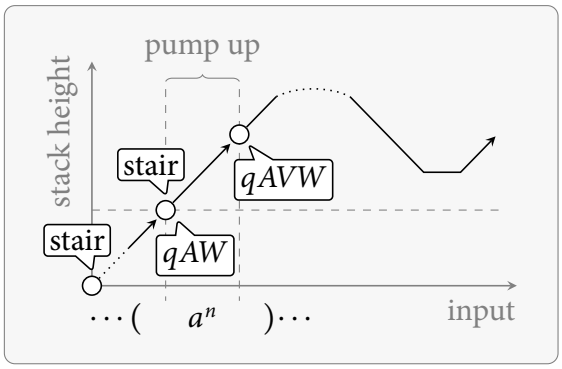
b) For the other claim, assume $\overline{L_{\mathrm{u}}}$ to be context-free and recognized by some ω-PDA $\mathcal{A}$. For each ω-word $\alpha \notin L_{\mathrm{u}}$, there exists an integer $i \in \mathbb{N}$ such that α does not contain the infix a^i. Consider a word $\beta = (a^n b)^\omega \notin L_{\mathrm{u}}$ for some sufficiently large integer $n \in \mathbb{N}$ together with an accepting run of β on $\mathcal{A}$. We show that infinitely many a-blocks in β can be pumped arbitrarily often. This results in an ω-word $\beta' \in L_\omega(\mathcal{A}) \cap L_{\mathrm{u}}$ with a-blocks of unbounded length when we pump the i-th block i times.

   A pumpable a-block can be found as follows. From an arbitrary position in the run on β, we first go to the next stair and then, we consider the first complete a^n-block that is read afterwards. Four cases arise when the a^n-block is processed by $\mathcal{A}$ (depicted in Figure 3.7):

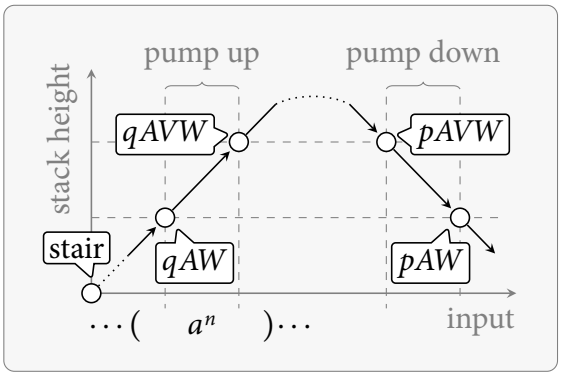
   i) The increase and decrease of the stack height is small, i.e., the difference between the minimal and maximal stack height during the processing of a^n is bounded. Then, two configurations are the same. The part in between these configurations can be pumped.



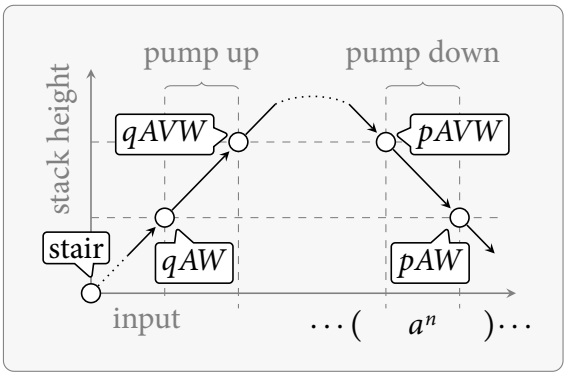
(a) Insufficient increase of the stack height



(b) Sufficient increase of the stack height with stair



(c) Sufficient increase of the stack height without stair



(d) Sufficient decrease of the stack height

Figure 3.7: Distinction of cases while pumping the run of an ω-PDA for $\overline{L_u}$ in Example 3.4.7

ii) There is a stair and another configuration (with higher stack) later on that share the same state and stack top. Again, the part in between these configurations can be pumped.

iii) There are two configurations of increasing stack height that are not stairs that share the same state and stack top, and later on, when the stack height is first decreased to the respective levels, these respective popping configurations also share the same state. The part in between the two pushing configurations can be pumped to increase the length of the a-block. The part in between the two popping configurations has to be pumped in the same way to restore the stack height afterwards.

iv) Opposite of the previous case: There are two configurations of decreasing stack height that share the same state, and before, when the stack height is last increased

to the respective levels (since we started our consideration at a stair), these respective pushing configurations also share the same state and stack top. The part in between the two popping configurations can be pumped to increase the length of the a-block. The part in between the two pushing configurations has to be pumped in the same way to prepare the stack height before. ◁

It might not be surprising that Theorem 3.4.6 holds not true for an ω-language as complex as L_{u}. But, there is also an example of a much simpler ω-language that admits finitely many congruence classes.

Example 3.4.8. Consider the ω-language L_{e} over $\Sigma = \{a, b\}$ containing all words that either have only finitely many letters b or have infinitely many a-blocks such that an a-block of the same length occurs somewhere later on:

$$L_{\mathrm{e}} = \left(\Sigma^* a^\omega\right) \cup \bigcap_{i\in\mathbb{N}} \bigcup_{j\in\mathbb{N}} \left((\Sigma^i \Sigma^* b) \cdot a^j \cdot (b + b\Sigma^* b) \cdot a^j \cdot (b\Sigma^\omega)\right) \subseteq \Sigma^\omega.$$

Then, $\mathrm{Ind}_{\sim} = 1$ again. On the one hand, the prefix of an ω-word is not relevant for the membership in L_{e}. Further, each ultimately periodic ω-word is contained in L_{e} because if the (nonempty) period contains no letter b, then the first case of the definition of L_{e} applies and otherwise, there is at least one b in the period that induces infinitely many (possibly empty) a-blocks all having the same length.

It is easy to see that L_{e} can be recognized by an ε-free ω-OCA that guesses which case applies, and in the second case, it guesses the a-blocks and where they are repeated while their lengths are compared with the counter. Such an automaton is depicted in Figure 3.8. ◁

To summarize, we have introduced a congruence (for a given ω-language) that characterizes regularity, i.e., the index of which is finite if the ω-language is regular. The latter example shows that this characterization only holds for ω-PDLs that are deterministic. The decidability of the regularity problem for ω-DPDLs remains open.

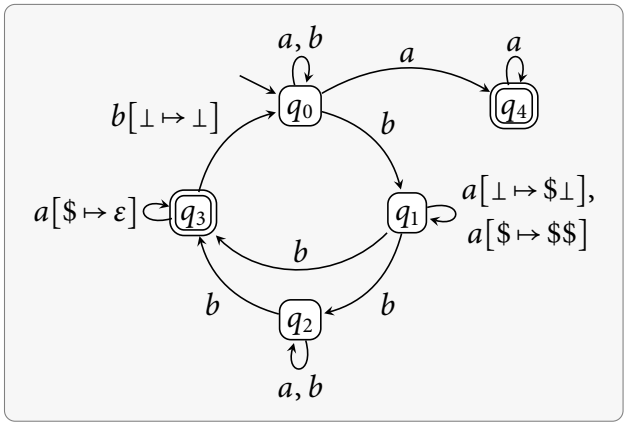


Figure 3.8: An ε-free ω-OCA that recognizes the language L_e from Example 3.4.8 (doubly circled states have color 0, otherwise color 1, and the stack is ignored if no operation is given)

Chapter 4

Lookahead Delegation for Nondeterministic Automata

This chapter is dedicated to delegation problems for nondeterministic automata. A lookahead delegator is a function similar to a transition function in the sense that in each step of a run, the delegator shall deterministically choose the next transition. But, a lookahead delegator may additionally use some bounded lookahead on the input word to make a decision. If such a function leads to an accepting state for some input word, then the word is recognized by the automaton, too, since only transitions of the automaton are chosen. To make such a function a delegator, we further require it to fulfill the inverse direction, i.e., the function has to lead to an accepting state for each word accepted by the automaton.

Delegators for nondeterministic automata where first introduced and studied in [RS07]. However, the motivation is based on a different formulation that involves only deterministic automata and that occurs during the composition problem for so-called e-services [BCG+03]. The question is to decide whether a set of available services can be composed to fulfill a certain specification. Each service and the target specification are given by deterministic automata. The composition is made by a function that maps each letter of the input word to one of the sub-automata. It is demanded that for each word recognized by the specification and for each sub-automaton, the sub-automaton accepts the letter sequence that is mapped to it. For the general composition, this function knows the entire input word. A restricted version of the function is considered in [GHIS04], namely one with bounded lookahead on the input word suffices during the mapping. Such a function is called a lookahead delegator.

Besides defining lookahead delegators, it was further shown in [GHIS04] that for DFSAs, the existence of a k-lookahead delegator can be decided in time polynomial in the alphabet and size of all DFSAs and exponential in k and the number of DFSAs. The EXPTIME-hardness of this problem was shown in [MW08]. For so-called reversal-bounded one-counter automata, this

problem was shown to be decidable in [DIS05]. Finally, deciding the existence of a bounded delegator was posed as an open problem in [GHIS04].

This delegation problem for a tuple of (deterministic) automata can be reduced by a product construction to the delegation problem that we consider, i.e., for a single nondeterministic automaton. Let $\times$ denote the synchronous product and $\otimes$ denote the fully asynchronous product. Then, a specification $\mathcal{A}$ is composable with lookahead k from some services $\mathcal{A}_1, \ldots, \mathcal{A}_n$ if, and only if, there exists a k-lookahead delegator for $\mathcal{A} \times (\mathcal{A}_1 \otimes \ldots \otimes \mathcal{A}_n)$. Note that the asynchronous products introduce nondeterminism. For a single nondeterministic automaton, the delegation problem is more general. Hence, only lower bounds can be transfered to this setting. Then, one must also note that the product automaton might be exponentially larger then the original tuple of automata. However, the composition of e-services motivates delegators, i.e., why we want to 'determinize' the automaton without changing its states or transitions.

In this chapter, we study the complexity of delegator synthesis for nondeterministic automata. This means in detail that we reconsider the three versions of the decision problem as they were defined in [RS07, Section 4]:

a) ***k*-DELEGATOR** for a fixed number $k \in \mathbb{N}$: decide for a given automaton $\mathcal{A}$ whether $\mathcal{A}$ has a k-lookahead delegator.

b) **DELEGATOR**: decide for a given automaton $\mathcal{A}$ and $k \in \mathbb{N}$ whether $\mathcal{A}$ has a k-lookahead delegator.

c) **BOUNDED-DELEGATOR**: decide for a given automaton $\mathcal{A}$ whether $\mathcal{A}$ has a bounded lookahead delegator.

We study these problems for finite state automata (Section 4.1) and for pushdown automata (Section 4.2). Our results were partially presented in [LR13].

4.1 Delegation for Finite State Automata

In [RS07], complexity upper bounds were given only for a restricted subclass of FSAs. We extend this research to FSAs in general and show the decidability of all three problems listed above, each of them being handled in a separate subsection. Before, we start with some preliminaries concerning delegators for FSAs.

We restrict our studies to FSAs that are total and ε-free. The first property avoids unnecessary technical difficulties. Regarding ε-transitions, it was already mentioned in [RS07, Lemma

1] that an FSA with ε-transitions has a k-delegator if, and only if, its ε-free pendant has a k-delegator where we assume the standard procedure for ε-removal.

We proceed with the formal definition of a lookahead delegator. For the sake of readability, we only consider FSAs and give an extended version later in Definition 4.2.1. For $k \in \mathbb{N}$, let $\Sigma^{\leq k} = \bigcup_{i=0}^{k} \Sigma^i$.

Definition 4.1.1. For an ε-free FSA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ and a number $k \in \mathbb{N}$, a ***k*-lookahead delegator** (or ***k*-delegator** for short) is a function $f : Q \times \Sigma\Sigma^{\leq k} \to Q$ such that

a) $f(q, aw) = p$ implies $(q, a, p) \in \Delta$ for each $q \in Q$, $a \in \Sigma$, $w \in \Sigma^{\leq k}$, and

b) $f^*(q_0, w) \in F$ for each $w \in L_*(\mathcal{A})$, where $f^* : Q \times \Sigma^* \to Q$ extends f to words and is defined inductively as follows: let $f^*(q, \varepsilon) = q$ for each state q, and let $f^*(q, a_1 \cdots a_n) = f^*\big(f(q, a_1 \cdots a_{\min(n,k+1)}), a_2 \cdots a_n\big)$ for each state q and nonempty word $a_1 \cdots a_n$ of length n. ◁

In a sentence, the two conditions express that only transitions are chosen by the delegator and that it leads to an accepting state for each word accepted by the automaton. We say that $\mathcal{A}$ has a **bounded lookahead delegator** if it has a k-lookahead delegator for some $k \in \mathbb{N}$.

Note that in our notion of k-lookahead, we follow [GHIS04, DIS05] by counting the additional lookahead, whereas in [RS07], the current input symbol counts as a letter of the lookahead. In our setting, a 0-lookahead delegator can hence be identified with a deterministic subset of the transitions such that the same language is accepted.

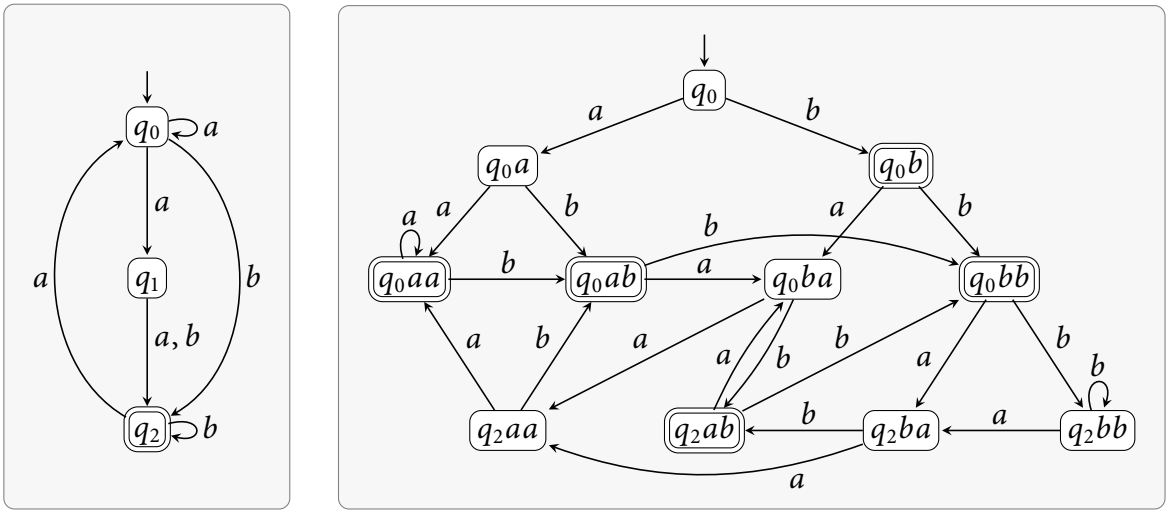
Example 4.1.2. Consider the FSA $\mathcal{A}$ over the alphabet $\Sigma = \{a, b\}$ as depicted in Figure 4.1a which accepts the language $L_*(\mathcal{A}) = \{aa\} \cup \Sigma^*\{b, aaa\}$. The only nondeterministic choice of a transition occurs at state q_0 for symbol a. It is possible to delegate this transition with lookahead 2; e.g. by the function $f : Q \times \Sigma\Sigma^{\leq 2} \to Q$ with $f(q_0, aa) = 1$ and $f(q, aw) = 0$ for all $w \in \Sigma^{\leq 2} \setminus \{a\}$.

A very easy way of showing that f is a delegator is to consider the 'delegated' automaton $\mathcal{A}_f$ which simulates f on $\mathcal{A}$. Formally, for any ε-free FSA $\mathcal{A}$, number $k \in \mathbb{N}$, and function $f : Q \times \Sigma\Sigma^{\leq k} \to Q$ with $\big(q, a, f(q, aw)\big) \in \Delta$ for all $(q, aw) \in \mathrm{dom}(f)$, we can construct the ε-free DFSA $\mathcal{A}_f = (Q', \Sigma, \delta', q_0', F')$ consisting of

a) states $Q' = Q \times \Sigma^{\leq k}$, $q_0' = (q_0, \varepsilon)$, and $F' = \Big\{(q, w) \in Q' \;\Big|\; f^*(q, w) \in F\Big\}$, and

b) transitions as follows:

$$\begin{aligned} &\delta'\big((q, w), a\big) = (q, wa) && \text{if } w \in \Sigma^{\leq k-1}, \\ &\delta'\big((q, a_0 \cdots a_{k-1}), a_k\big) = \big(f(q, a_0 \cdots a_k), a_1 \cdots a_k\big) && \text{where } a_i \in \Sigma \text{ for all } i \in [k]. \end{aligned}$$



(a) FSA $\mathcal{A}$

(b) DFSA $\mathcal{A}_f$ resulting from $\mathcal{A}$ and f

Figure 4.1: FSA and its 'delegated' DFSA (doubly bordered states are accepting)

By construction, we have that $L_*(\mathcal{A}) = L_*(\mathcal{A}_f)$ iff f is a k-delegator. The delegated automaton of this example is depicted in Figure 4.1b. ◁

The above construction yields a naïve method to decide the existence of a k-delegator. However, the complexity is far from optimal: it shows Delegator to be in 2EXPSPACE, and k-Delegator to be in NP. We improve these bounds by using some theoretical insight that is based on left quotients.

Left Quotients. For $w \in \Sigma^*$ and $L \subseteq \Sigma^*$, let the **left quotient** $w^{-1}L = \{v \in \Sigma^* \mid wv \in L\}$ of w with L be the language containing each word that completes w to a word in L. Note that the composition of left quotients can be written as concatenation: $v^{-1}(u^{-1}L) = (uv)^{-1}L$. Further, we write $L_*(\mathcal{A}_q)$ for the language of $\mathcal{A}$ where q is taken as the initial state.

We now present a technical lemma that occurs as a key ingredient in the proofs and algorithms of the following three sections. A similar statement can be found in [RS07, Lemma 7] using a notion called blindness. It gives a language-theoretical characterization of the main property a k-delegator has to fulfill when it selects a transition. That is, whenever a transition $(q, a, p) \in \Delta$ has to be chosen for some lookahead w, then $(aw)^{-1}L_*(\mathcal{A}_q) = w^{-1}L_*(\mathcal{A}_p)$, i.e., the accepted words are the same before and after the transition is taken. If this is not the case, one can easily show that the definition of a delegator is not fulfilled by picking a word from the difference. On the other hand, if this property holds for each transition, then it follows in-

ductively that the language is preserved. Note that the inclusion $(aw)^{-1}L_*(\mathcal{A}_q) \supseteq w^{-1}L_*(\mathcal{A}_p)$ holds generally, since $(q, a, p) \in \Delta$.

Lemma 4.1.3. *An ε-free FSA $\mathcal{A}$ has a k-delegator iff there exists a set $Q' \subseteq Q$ such that $q_0 \in Q'$ and for each $q \in Q'$, $a \in \Sigma$, $w \in \Sigma^k$, there exists $p \in Q'$ such that $(q, a, p) \in \Delta$ and $(aw)^{-1}L_*(\mathcal{A}_q) = w^{-1}L_*(\mathcal{A}_p)$ hold.*

Proof. For the direction from left to right, let f be a k-delegator and Q' be the set of states reachable by f from q_0 with full lookahead, i.e., the smallest set such that $q_0 \in Q'$ and $f(q, aw) \in Q'$ for each $q \in Q'$, $a \in \Sigma$, and $w \in \Sigma^k$. For a contradiction, assume there are $q \in Q'$, $a \in \Sigma$, and $w \in \Sigma^k$ such that $(aw)^{-1}L_*(\mathcal{A}_q) \neq w^{-1}L_*(\mathcal{A}_p)$ for all $p \in Q'$ with $(q, a, p) \in \Delta$. Since q is reachable from q_0, fix a word $u \in \Sigma^*$ such that f leads to q and assume w.l.o.g. that q is the first state on that run with the above property. Let $p = f(q, aw)$. Then, there is a word $v \in (aw)^{-1}L_*(\mathcal{A}_q) \setminus w^{-1}L_*(\mathcal{A}_p)$, i.e., $awv \in L_*(\mathcal{A}_q)$ but $wv \notin L_*(\mathcal{A}_p)$. Hence, we have that $f^*(q_0, uawv) = f^*(q, awv) = f^*(p, wv) \notin F$ whereas $uawv \in L_*(\mathcal{A})$ in contradiction to the definition of a k-delegator.

For the other direction, we construct a k-delegator f from a set Q' with the above properties. For each $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^{\leq k}$, we set $f(q, aw) = p$ as follows.

a) If $|w| = k$ and $q \in Q'$, then $p \in Q'$ becomes some state such that $(q, a, p) \in \Delta$ and $(aw)^{-1}L_*(\mathcal{A}_q) = w^{-1}L_*(\mathcal{A}_p)$ as directly guaranteed by the property.

b) If $|w| < k$ and $aw \in L_*(\mathcal{A}_q)$, then there is an a-successor $p \in Q$ of q such that $w \in L_*(\mathcal{A}_p)$.

c) If $|w| < k$ and $aw \notin L_*(\mathcal{A}_q)$, then fix some arbitrary a-successor $p \in Q$ of q.

Note that the case $|w| = k$ and $q \notin Q'$ cannot occur due to the above property. It easily follows by the definition of f^* that $f^*(q_0, v) \in F$ if $v \in L_*(\mathcal{A})$. Hence, f is a k-delegator for $\mathcal{A}$. □

Example 4.1.4. From the perspective of Lemma 4.1.3, let us reconsider the FSA $\mathcal{A}$ given in Example 4.1.2. Since q_0 is the initial state, a set Q' satisfying the property of Lemma 4.1.3 must contain q_0. The only nondeterministic choice happens at q_0 with letter a. The two a-successors of q_0 are q_0 itself and q_1. The left quotients that are relevant for Lemma 4.1.3 are listed in the following table for some relevant words w where $L_i = L_*(\mathcal{A}_{q_i})$:

w	$(aw)^{-1}L_0$	$w^{-1}L_0$	$w^{-1}L_1$
ε	$L_0 \cup L_1$	L_0	L_1
a	$L_0 \cup L_1 \cup L_2$	$L_0 \cup L_1$	L_2
aa	$L_0 \cup L_1 \cup L_2$	$\underline{L_0 \cup L_1 \cup L_2}$	L_0
ab	L_2	$\underline{L_2}$	$\underline{L_2}$
b	L_2	$\underline{L_2}$	$\underline{L_2}$

The underlined languages indicate correct choices according to Lemma 4.1.3. In detail, this means that for $w = \varepsilon$, one can see that neither the language $w^{-1}L_0$ nor $w^{-1}L_1$ is equivalent to $(aw)^{-1}L_0$ which shows that there is no 0-delegator for $\mathcal{A}$. The row for $w = a$ shows analogously that there is no 1-delegator, either. When the lookahead is increased to 2, then the condition of Lemma 4.1.3 is finally fulfilled. For $w = aa$, the left quotients are the same if and only if a delegator chooses q_0 as the a-successor of q_0. The remaining two lines show that the choice does not matter for $w \in \{ab, b, ba, bb\}$.

Note that the choice does also matter for the case $w = a$. This is not a contradiction, as the condition in Lemma 4.1.3 only considers a lookahead that is completely filled. For shorter lookahead at the end of the input, it follows that one can always reach an accepting state if a word is accepted. In this example, in order to obtain a delegator, one further to go to q_1 in case of $w = a$. ◁

4.1.1 Fixed Lookahead

In the following, we present for an arbitrary fixed number $k \in \mathbb{N}$ an algorithm that solves the problem k-Delegator, i.e., the algorithm decides the existence of a k-delegator for a given FSA $\mathcal{A}$ and computes one if it exists. The special case 0-Delegator corresponds to deciding whether a given FSA $\mathcal{A}$ can be turned into an equivalent DFSA just by removing transitions. The polynomial time decidability of this latter special case was shown in [AKL10, Theorem 4.1] and independently, it has been mentioned in the survey article [Col12, Theorem 15] without a proof. We generalize this result to an arbitrary fixed amount of lookahead.

The rough idea behind our approach is to construct a safety game $\mathcal{G}_{\mathcal{A},k}$ that simulates the delegation. The two players play a sequence of actions in alternation. Player 1 is in charge of the input, i.e., he has to choose the letters of an input word. Player 0 is in charge of transitions. After each letter played by his opponent, he has to choose an appropriate transition. The goal of Player 0 is to play an accepting run if a word contained in the language $L_*(\mathcal{A})$ is formed by Player 1.

The key property we want to achieve for this game is to show (later, in Lemma 4.1.7) that Player 0 has a winning strategy if, and only if, $\mathcal{A}$ has a k-delegator. By using a game-based approach, we obtain an intuitive formulation of the problem on the one hand, while we have many game-theoretic results available on the other. Here, we especially use the facts that such games are determined, that winning strategies are positional, and that such a winning strategy can be synthesized in time linear in the size of the game. Ideally, the positions of $\mathcal{G}_{\mathcal{A},k}$ store only the following information: the current state of $\mathcal{A}$ reached by Player 0, and a lookahead of k letters (or less if the play goes towards the end of the input word) provided by Player 1. If this

is the case, then a positional winning strategy for Player 0 directly corresponds to a delegator.

The main challenge is now to create a safety game where the states only store the information mentioned above (which is also polynomial). The winning condition should express that the state of Player 0 is accepting if the input word played by Player 1 is in the language. The latter property does not depend on the current position alone but on the whole play instead. Naïvely, one can implement this as a safety condition by additionally keeping track of the set of reachable states by the input played by Player 1. However, the game vertices contain too much information then, and it leads to a blowup that is exponential in the size of the automaton.

To solve this problem, we modify our game in such a way that Player 1 also has to choose a transition for each input, but after Player 0 has chosen one. We show that, since Player 0 has to make the choice of the transition first, the additional information on the transition chosen by Player 1 does not help Player 0 (because basically, Player 0 has to choose a transition according to Lemma 4.1.3, which only depends on the current state of Player 0). This means that in this modified game, a winning strategy for Player 0 still corresponds to a k-delegator.

To summarize, the game $\mathcal{G}_{\mathcal{A},k}$ goes as follows. First, Player 1 gives the initial content of the lookahead. Then, both players play in alternation. Player 0 chooses a transition for the next input letter. Afterwards, Player 1 also chooses a transition for the same letter. Simultaneously, he removes this letter from the lookahead (as both players have just processed it) and appends a new letter, or he does not refill the lookahead if the input word should end. Consequently, a game position encodes the content of the lookahead as well as one state for each player. The safety condition for Player 0 now simply states that such vertices have to be avoided, where the state of Player 1 is accepting and the state of Player 0 is non-accepting although the lookahead is empty.

Definition 4.1.5. Given an ε-free FSA $\mathcal{A}$ and $k \in \mathbb{N}$, we define the ***k*-delegator game** to be the safety game $\mathcal{G}_{\mathcal{A},k} = (V, V_0, E, \Omega)$ where:

a) $V = \{\top\} \cup \Big(\{0,1\} \times \Sigma^{\leq k+1} \times Q \times Q\Big)$, (initial vertex and simulation vertices)

b) $V_0 = \Big(\{0\} \times \Sigma^{\leq k+1} \times Q \times Q\Big)$,

c) $E \subseteq V \times V$ containing the following edges:

i) $\Big(\top, (0, w, q_0, q_0)\Big)$ for $w \in \Sigma^{\leq k+1}$, (initiate buffer)

ii) $\Big((0, aw, q, p), (1, aw, q', p)\Big)$ for $(q, a, q') \in \Delta$ and $w \in \Sigma^{\leq k}$, (Player 0 applying transition)

iii) $\Big((1, aw, q', p), (0, wb, q', p')\Big)$ for $(p, a, p') \in \Delta$, $w \in \Sigma^k$, and $a, b \in \Sigma$,
(Player 1 applying transition, removing leftmost symbol, and refilling lookahead)

iv) $\Big((1, aw, q', p), (0, w, q', p')\Big)$ for $(p, a, p') \in \Delta$, $w \in \Sigma^{\leq k}$, and $a \in \Sigma$,
(Player 1 applying transition and removing leftmost symbol without refilling)

v) $\Big((0, \varepsilon, q, p), (0, \varepsilon, q, p)\Big)$ for $q, p \in Q$, (make vertices non-terminal)

d) $\Omega(v) = \begin{cases} 1 & \text{if } \Big\{(0, \varepsilon, q, p) \;\Big|\; q \notin F \wedge p \in F\Big\} \\ 2 & \text{otherwise.} \end{cases}$ ◁

The number of vertices of $\mathcal{G}_{\mathcal{A},k}$ is in $\mathcal{O}\big((k+1) \cdot |\Sigma|^{k+1} \cdot |Q|^2\big)$ which is polynomial in $|Q|$ and $|\Sigma|$ for a fixed k. The game can be constructed in time polynomial in the number of vertices.

Example 4.1.6. Let us reconsider the FSA $\mathcal{A}$ of Example 4.1.2. In Figure 4.2, a part of the 1-delegator game $\mathcal{G}_{\mathcal{A},1}$ is depicted in such a way that for Player 1, only one edge is enabled from each vertex whereas for Player 0, all successors are considered. This deterministic choice corresponds to a positional strategy for Player 1. The strategy always forces the play to an unsafe vertex, no matter how Player 0 reacts. It hence is a positional winning strategy for Player 1 in $\mathcal{G}_{\mathcal{A},1}$ from $\top$. ◁

We show next that this is because there exists no 1-delegator for $\mathcal{A}$ (cf. Example 4.1.4).

Lemma 4.1.7. *An ε-free FSA $\mathcal{A}$ has a k-lookahead delegator iff Player 0 has a positional winning strategy in $\mathcal{G}_{\mathcal{A},k}$ from $\top$.*

Proof. The key observation is that for (a strategy of) Player 0, it is not important to know which states Player 1 chooses since Lemma 4.1.3 states that left quotients are important rather than exact states. As long as the property is fulfilled locally, Player 1 can w.l.o.g. be assumed to just copy the choices of his opponent.

For the direction from right to left, suppose Player 0 has a positional winning strategy s. We show the existence of a k-delegator for $\mathcal{A}$ by proving that the condition from Lemma 4.1.3 is satisfied. For this purpose, let Q' be the set consisting of all states $q \in Q$ such that a vertex of the form $(0, aw, q, q)$, with $a \in \Sigma$ and $w \in \Sigma^k$, can be reached with s for some sequence of moves of Player 1. Since trivially $q_0 \in Q'$, it remains to show that for each such vertex, there is a successor $(1, aw, p, q)$ with $(aw)^{-1}L_*(\mathcal{A}_q) = w^{-1}L_*(\mathcal{A}_p)$. To the contrary, assume that there exists a state $q \in Q'$ such that $(aw)^{-1}L_*(\mathcal{A}_q) \neq w^{-1}L_*(\mathcal{A}_p)$ for each $p \in Q$ with $(q, a, p) \in \Delta$. Then, no matter which p is chosen by Player 0, there is a word $v \in (aw)^{-1}L_*(\mathcal{A}_q) \smallsetminus w^{-1}L_*(\mathcal{A}_p)$,

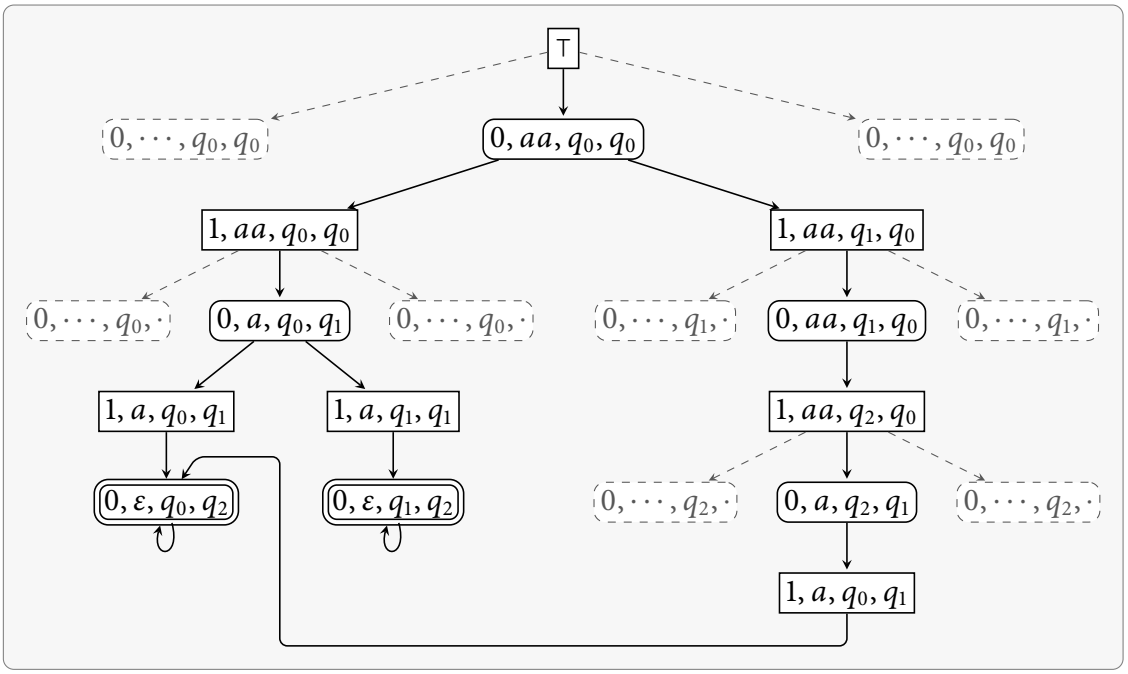


Figure 4.2: The (partial) 1-delegator game $\mathcal{G}_{\mathcal{A},1}$ showing a positional winning strategy for Player 1 (rounded vertices belong to Player 0, boxed ones to Player 1, and doubly circled vertices are unsafe)

i.e., $awv \in L_*(\mathcal{A}_q)$ but $wv \notin L_*(\mathcal{A}_p)$. Player 1 can win from $(1, aw, p, q)$ by continuing to play the word v since there is a sequence of states that accepts awv from q but none that accepts wv from p. This contradicts the property that s is a winning strategy for Player 0.

For the other direction, suppose $\mathcal{A}$ has a k-delegator f. We can naturally use it to define a positional winning strategy $s : V_0' \to V$ for Player 0 where $s(0, aw, q, p) = \big(1, aw, f(q, aw), p\big)$. One can easily see by the construction of $\mathcal{G}_{\mathcal{A},k}$ and s that $q = f^*(q_0, x)$ holds whenever a terminal vertex $(0, \varepsilon, q, p)$ is reached after Player 1 has played a complete word $x \in \Sigma^*$. Player 0 wins because $p \in F$ implies $x \in L_*(\mathcal{A})$ and hence, $q \in F$. □

By combining Lemma 4.1.7 with the linear-time determinacy of safety games from Proposition 2.2.2, we get the main result of this section. Consequently, the existence of a k-lookahead delegator can be decided in time $\mathcal{O}\big((k+1) \cdot |\Sigma|^{k+1} \cdot |Q|^2\big)$ for a given FSA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ and a positive number k. This yields polynomial running time for a fixed k that we consider in this section.

Corollary 4.1.8. *For each $k \in \mathbb{N}$, the problem k-Delegator for ε-free FSAs can be solved in polynomial time.*

This generalizes [RS07, Theorem 2] where polynomial time decidability of k-Delegator for each fixed k is shown for unambiguous FSAs. Further, the false statement of [RS07, Theorem 5] is corrected.

As explained before, we consider the input to be a single FSA whereas in the original motivation, the input consists of several DFSAs. It is shown in [MW08] that the problem 0-Delegator is EXPTIME-hard in the original setting were a tuple of DFSAs is given. This is caused by the fact that the construction of a product of the DFSAs yields an FSA that is exponentially larger.

4.1.2 Given Lookahead

We now consider the complexity of the problem Delegator (where an FSA and a bound k are given). Note that for deciding whether $\mathcal{A}$ has a k-lookahead delegator, the game-based algorithm from the previous section yields a running time that is doubly exponential in the binary representation of k. However, using a different algorithm, we can show that the problem can be solved in polynomial space. The idea of our algorithm running in polynomial space is to check whether the property of Lemma 4.1.3 holds. The main problem in checking this condition with our space restriction is that we cannot enumerate all words $w \in \Sigma^k$ because their length is exponential in the binary representation of k.

For that purpose, we introduce transition profiles, which can be used to circumvent this problem. Intuitively, a transition profile of a word w for a given FSA $\mathcal{A}$ describes the possible state transformations induced by w on $\mathcal{A}$, i.e., it consists of all pairs of states (p, q) such that there is a w-labeled path from p to q.

Definition 4.1.9. For an FSA $\mathcal{A}$ and a word $w \in \Sigma^*$, we define the **transition profile**

$$\Delta_w = \left\{ (q, p) \in Q^2 \;\middle|\; q \xrightarrow{w} p \right\} \subseteq Q^2. \qquad \triangleleft$$

The main idea for checking the condition of Lemma 4.1.3 in polynomial space is to use transition profiles that are induced by words of length k, instead of working directly with the words. This is justified by the following simple observation that words with the same profile have the same left quotient, too.

Lemma 4.1.10. *Let $x, y \in \Sigma^*$ be such that $\Delta_x = \Delta_y$ for an FSA $\mathcal{A}$. Then, $x^{-1}L_*(\mathcal{A}_q) = y^{-1}L_*(\mathcal{A}_q)$ for all $q \in Q$.*

Proof. A trivial consequence of Definition 4.1.9 is that $\Delta_{xw} = \Delta_{yw}$ for all $w \in \Sigma^*$. Then,

$$\begin{aligned} w \in x^{-1}L_*(\mathcal{A}_q) &\Leftrightarrow xw \in L_*(\mathcal{A}_q) \\ &\Leftrightarrow \exists p \in F : (q,p) \in \Delta_{xw} \\ &\Leftrightarrow \exists p \in F : (q,p) \in \Delta_{yw} \\ &\Leftrightarrow yw \in L_*(\mathcal{A}_q) \\ &\Leftrightarrow w \in y^{-1}L_*(\mathcal{A}_q). \end{aligned}$$

□

Theorem 4.1.11. *The problem Delegator for ε-free FSAs is in PSPACE.*

Proof. Let an ε-free FSA $\mathcal{A}$ (with the usual components) and k be given. We show that for each $Q' \subseteq Q$, there is a nondeterministic PSPACE algorithm that checks whether the property of Lemma 4.1.3 is satisfied. Savitch's theorem (see [Pap94, Theorem 7.5]) implies that there is also a deterministic PSPACE algorithm.

So let $Q' \subseteq Q$ with $q_0 \in Q$. The algorithm tests for each $q \in Q$ and each $a \in \Sigma$, whether for each word $w \in \Sigma^k$ there is an a-successor p of q such that $(aw)^{-1}L_*(\mathcal{A}_q) = w^{-1}L_*(\mathcal{A}_p)$. As mentioned above, we cannot enumerate all words $w \in \Sigma^k$ because their length is exponential in the binary representation of k. Instead, we work with the transition profiles induced by the words w. Each such transition profile is of size polynomial in $\mathcal{A}$ and contains sufficient information to test $(aw)^{-1}L_*(\mathcal{A}_q) = w^{-1}L_*(\mathcal{A}_p)$. Lemma 4.1.10 allows us to restrict the test to transition profiles, as words with the same transition profile induce the same left quotient.

We now describe the algorithm. Given Q', $q \in Q'$, and $a \in \Sigma$, the algorithm proceeds as follows. For each transition profile $\tau \in 2^{Q \times Q}$:

a) Check if $\tau = \Delta_w$ for some word w of length k. If it is the case, proceed with the next point. Otherwise, move on to the next transition profile.

b) Let $p_1, \ldots, p_n$ be the a-successors of q in Q'. For $i \in \{1, \ldots, n\}$, let $R_i = \{p \in Q \mid (p_i, p) \in \tau\}$ be the set of states that are reached from p_i in the profile τ. Let $R = \bigcup_{1 \le i \le n} R_i$. Note that $L_{R_i} = w^{-1}L_{p_i}$ and $L_R = (aw)^{-1}L_*(\mathcal{A}_q)$, where for $S \subseteq Q$, we let $L_S = \bigcup_{s \in S} L_s$.

c) Check if there is an index $i \in \{1, \ldots, n\}$ such that $L_R = L_{R_i}$.

If the last test fails (meaning that there is no such index i), then Q' does not satisfy the property of Lemma 4.1.3. If the test passes for all q, all a, and all the relevant transition profiles (those passing the first test), then Q' has the desired property and thus, $\mathcal{A}$ has a k-lookahead delegator.

It remains to verify that the steps of the algorithm can be carried out in polynomial space. The first test uses the idea of checking reachability in a directed graph in logarithmic space.

In our setting, we use a counter for counting up to k (note that the number of bits needed for the counter corresponds to the size of the binary representation of k), and successively guess k steps to reach the transition profile τ. That is, we start with the transition profile Δ_ε of the empty word. In each step, we guess a letter $b \in \Sigma$ and extend the current transition profile Δ_v to Δ_{vb}. After k steps, we check whether the resulting profile Δ_w is equal to τ. At each moment, we only need to store the counter and the intermediate transition profile, which requires polynomial space.

The second step just computes (in LOGSPACE) some sets from the transition profile τ.

The third step requires us to test n equivalences $L_R = L_{R_i}$, where the languages are given by FSAs with the sets R and R_i as initial states, respectively. Since equivalence of FSAs can be tested in polynomial space (see [AHU74]), this step is also in PSPACE. □

Theorem 4.1.12. *The problem* Delegator *for ε-free FSAs is PSPACE-complete.*

Proof. The upper bound follows from Theorem 4.1.11.

For the lower bound, let $\mathcal{M}$ be some polynomially space bounded Turing machine that solves a PSPACE-hard problem. We show that the word problem for $\mathcal{M}$ can be reduced to the problem of the existence of a bounded lookahead delegator. The word problem for $\mathcal{M}$ is to decide for a given word whether $\mathcal{M}$ accepts w, which is clearly PSPACE-hard because $\mathcal{M}$ solves a PSPACE-hard problem.

Let h be the polynomial for the space bound of $\mathcal{M}$. Given a word w, we construct an FSA $\mathcal{A}$ that has a $\big(2h(n)+2\big)$-lookahead delegator iff $\mathcal{M}$ rejects w, where $n = |w|$.

As usual, we encode configurations of $\mathcal{M}$ by words of the form $\kappa = usv$, where uv is the content of the tape of $\mathcal{M}$, and s the current control state. The head of $\mathcal{M}$ in configuration usv is on the first position of v. We can assume that $|uv| = h(n)$. A computation of $\mathcal{M}$ is then encoded by a word of the form $\#\kappa_0\#\kappa_1\#\cdots\kappa_\ell\#$, where κ_0 represents the initial configuration of $\mathcal{M}$ on w, each κ_{i+1} encodes the successor configuration of κ_i, and κ_ℓ encodes an accepting configuration.

The core of the reduction is an FSA $\mathcal{A}_w$ that accepts a word if it does **not** encode an accepting computation of $\mathcal{M}$ on w (see [AHU74, Lemma 10.2] for such a construction for regular expressions instead of FSAs). For this purpose, $\mathcal{A}_w$ uses a product of automata testing the following properties:

a) The word is not of the required form $\#\kappa_0\#\kappa_1\#\cdots\kappa_\ell\#$ where each κ_i is of the form $u_i s_i v_i$ with $|u_i v_i| = h(n)$.

b) The first configuration is not the initial configuration of $\mathcal{M}$ on w.

c) The last configuration is not an accepting configuration.

d) There is an i such that κ_{i+1} is not the $\mathcal{M}$-successor configuration of κ_i.

The first three properties can be easily checked by DFSAs of size linear in $h(n)$. The last property can be checked by an FSA that guesses at some symbol # that this corresponds to the index i, and then guesses a position j in κ_i and tests whether κ_{i+1} has been updated in a wrong way at position j (to detect this, the three symbols at positions $j-1$, j, and $j+1$ are sufficient). The size of such an FSA is also linear in $h(n)$ (it needs to count up to $h(n)$ for finding the corresponding cell j in κ_{i+1}). All the automata can be constructed in logarithmic space from $\mathcal{M}$ and w. The automaton $\mathcal{A}_w$ is the product of these four automata that accepts if one of its components accepts. Note that $\mathcal{A}_w$ has a $\big(2h(n)+2\big)$-lookahead delegator because it is sufficient to know the next two configurations to decide which transition to take in the FSA for the last property. Further, note that $\mathcal{A}_w$ accepts all words if there is no accepting computation of $\mathcal{M}$ on w. And if there is such an accepting computation, then $\mathcal{A}_w$ does not accept the word encoding it.

We now embed $\mathcal{A}_w$ into an FSA $\mathcal{A}$ to obtain the desired reduction. Let Σ be the alphabet of $\mathcal{A}_w$, and let X, Y, Z be new letters. Define the languages

$$\begin{aligned} L_1 &= X^* \cdot L_*(\mathcal{A}_w) \cdot (Y + Z), \\ L_2 &= X^* \cdot \Sigma^* \cdot Y. \end{aligned}$$

Note that $L_2 \subseteq L_1$ iff $L_*(\mathcal{A}_w) = \Sigma^*$ iff $\mathcal{M}$ rejects w.

We construct $\mathcal{A}$ to accept the language $X \cdot (L_1 \cup L_2)$. For this purpose, $\mathcal{A}$ nondeterministically chooses from its initial state on the first X to either go to an automaton $\mathcal{A}_1$ for L_1 or to an automaton $\mathcal{A}_2$ for L_2. The automaton $\mathcal{A}_1$ is a simple extension of $\mathcal{A}_w$ by an X-loop at the beginning and transitions for processing the last Y or Z. The automaton $\mathcal{A}_2$ just consists of an X-loop, followed by a Σ-loop, followed by a transition for Y into an accepting state.

Now, assume that $\mathcal{M}$ rejects w. Then, $L_2 \subseteq L_1$, as noted above, and a lookahead delegator for $\mathcal{A}$ can always choose the transition going to $\mathcal{A}_1$ from the initial state. We already noted that $\mathcal{A}_w$ has a $\big(2h(n)+2\big)$-lookahead delegator. Overall, we obtain a $\big(2h(n)+2\big)$-lookahead delegator for $\mathcal{A}$ in this case.

Now, assume that $\mathcal{M}$ accepts w and that $\mathcal{A}$ has a k-lookahead delegator f for some k. Consider the decision of f on the input prefix X^{k+1} which is the moment that a f has to choose the first transition. If f moves to $\mathcal{A}_1$, then pick the word v encoding the accepting computation of $\mathcal{M}$ on w, followed by the letter Y. $\mathcal{A}_1$ does not accept this word and therefore, f cannot be a k-lookahead delegator because $X^{k+1}vY \in L_*(\mathcal{A})$.

If f moves to $\mathcal{A}_2$, then consider any word v accepted by $\mathcal{A}_w$ followed by Z. Then, $X^k v Z \in L_1$ but $\mathcal{A}_2$ only accepts words ending with Y. Hence, also in this case, f cannot be a k-lookahead delegator.

This shows that $\mathcal{A}$ has a k-lookahead delegator for some k iff $\mathcal{M}$ rejects w. Furthermore, k can be chosen as $2h(n) + 2$. □

We note that in [RS07, Theorem 3], it is shown that the problem Delegator for unambiguous FSAs is contained in co-NP.

4.1.3 Bounded Lookahead

In this last subsection, we study the problem Bounded-Delegator that is to decide whether a given FSA has a k-delegator for some $k \in \mathbb{N}$. The authors of [RS07] showed that this problem is decidable for unambiguous FSAs and they believed it to be decidable for FSAs in general. We solve this problem by proving an upper bound on the amount of lookahead that is required maximally. The combination of this upper bound and the decision procedures of the previous subsection classifies this problem also to be PSPACE-complete, too. But before that, we provide an exponential lower bound on the amount of lookahead that is required in the worst case.

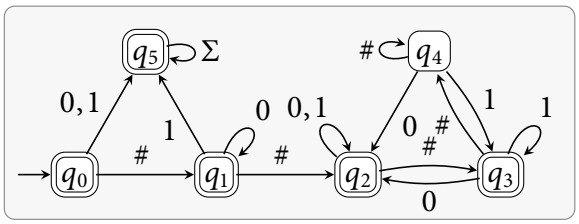
Example 4.1.13. Over a fixed alphabet, we give a family of automata each having a bounded lookahead delegator. But, the required amount of lookahead grows exponentially in the number of states.

The key to this lower bound is to enumerate all binary numbers with n bits for some fixed arbitrary number $n \in \mathbb{N}_+$. Over the ternary alphabet $\Sigma = \mathbb{B} \cup \{\#\}$ we define the word

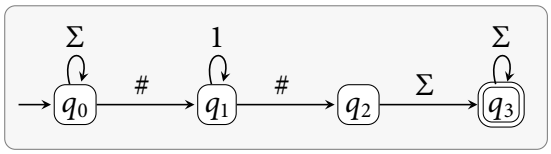
$$w_n = \#\underbrace{0\cdots000}_{\mathrm{bin}_n(0)}\#\underbrace{0\cdots001}_{\mathrm{bin}_n(1)}\#\underbrace{0\cdots010}_{\mathrm{bin}_n(2)}\#\cdots\#\underbrace{1\cdots110}_{\mathrm{bin}_n(2^n-2)}\#\underbrace{1\cdots111}_{\mathrm{bin}_n(2^n-1)}\#$$

where $\mathrm{bin}_n : [2^n] \to \mathbb{B}^n$ represents a number binarily with n bits. It is possible to detect an error in w_n by an FSA with $\mathcal{O}(n)$ states. We hence consider the language $L_n = \Sigma^* \setminus \{w_n\}$ which is accepted by the union $\mathcal{A} = \mathcal{A}_{\mathrm{border}} \cup \mathcal{A}_{\mathrm{block}} \cup \mathcal{A}_{\mathrm{end}} \cup \mathcal{A}_{\mathrm{incr}} \cup \mathcal{A}_{\mathrm{copy}(0)} \cup \mathcal{A}_{\mathrm{copy}(1)}$ of the ε-free FSAs depicted in Figure 4.3.

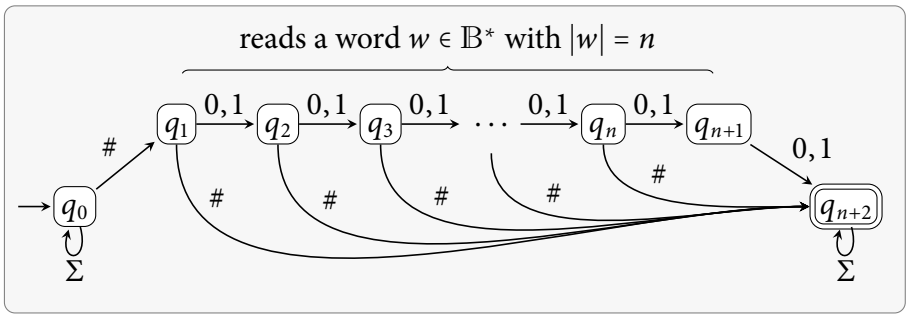
Formally, the FSA $\mathcal{A}$ that recognizes the union $\mathcal{A}_{\mathrm{border}} \cup \mathcal{A}_{\mathrm{block}} \cup \mathcal{A}_{\mathrm{end}} \cup \mathcal{A}_{\mathrm{incr}} \cup \mathcal{A}_{\mathrm{copy}(0)} \cup \mathcal{A}_{\mathrm{copy}(1)}$ has states that are the disjoint union of the state sets of the FSAs plus a new initial state and a sink state. The transition of the FSAs are copied. Further, each transition from an old initial state can now also be used from the new initial state (with the same letter and target) and, there is a transition from each state to the sink state for each letter. The property of a state being accepting is inherited from the FSAs, whereas the new initial state is accepting if one of



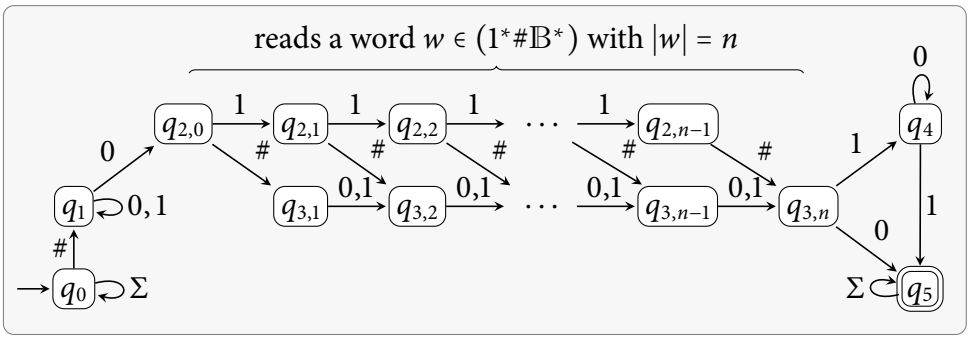
(a) $L_*(\mathcal{A}_{\text{border}}) = \Sigma^* \setminus \left(\#0^*\#\Sigma^*\#1^*\#\right)$



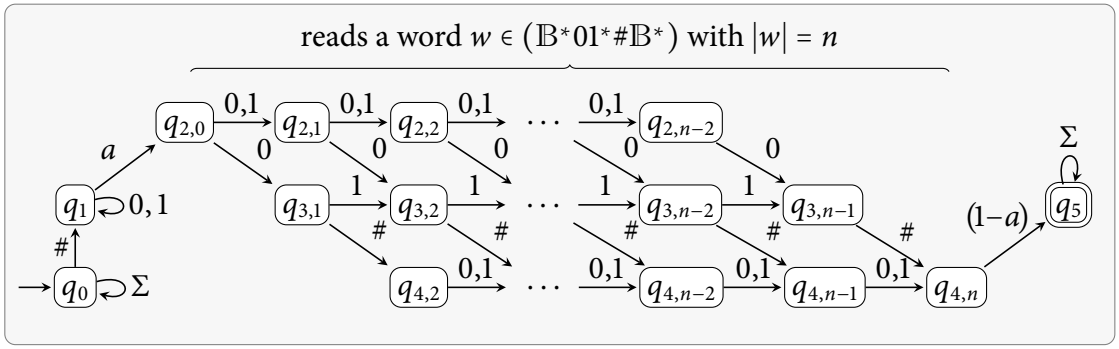
(b) $L_*(\mathcal{A}_{\text{end}}) = \Sigma^*\#1^*\#\Sigma\Sigma^*$



(c) $L_*(\mathcal{A}_{\text{block}}) = \Sigma^*\left(\#\mathbb{B}^{\leq n-1}\# + \#\mathbb{B}^{n+1}\right)\Sigma^*$



(d) $L_*(\mathcal{A}_{\text{incr}}) = \Sigma^*\left\{\#\mathbb{B}^*u\#v w \;\middle|\; u \in 01^*, v \in \mathbb{B}^* \text{ with } |uv| = n, w \in (0 + 10^*1)\right\}\Sigma^*$



(e) $L_*(\mathcal{A}_{\text{copy}(a)}) = \Sigma^*\left\{\#uav\#u'(1-a) \;\middle|\; u, u' \in \mathbb{B}^*, v \in \mathbb{B}^*01^* \text{ with } |u| = |u'| \text{ and } |uav| = n\right\}\Sigma^*$

Figure 4.3: FSAs accepting L_n

the old initial states is accepting (which is the case here as caused by $\mathcal{A}_{\text{border}}$). Then, $L_*(\mathcal{A})$ is the union of the other FSAs.

It is easy to see from the construction that the number of states of $\mathcal{A}$ is in $\Theta(n)$ and the alphabet is constant. It remains to show that w_n is the only word not accepted by $\mathcal{A}$ and that a bounded delegator for $\mathcal{A}$ needs at least exponential lookahead. Regarding the first point, consider a word $w' \neq w_n$ over Σ that is not accepted by $\mathcal{A}_{\text{border}} \cup \mathcal{A}_{\text{block}}$. We then know that $w' \in \#0^n\#(\mathbb{B}^n\#)^*1^n\#$, i.e., w' consists of binary blocks of length n starting with 0^n and ending with 1^n and that somewhere in between, the increment fails. Incrementing a binary number (which is not of the form 1^*) is simple: the suffix belonging to 01^* is replaced by the suffix of the form 10^* of the same length whereas the prefix remains unchanged. Since we have already fixed the block length, the increment can fail in two ways: Firstly, the suffix is not replaced correctly or secondly, some letter in the prefix changes. The first error is detected by $\mathcal{A}_{\text{incr}}$ whereas $\mathcal{A}_{\text{copy}(0)}$ and $\mathcal{A}_{\text{copy}(1)}$ detect whether in the prefix, a 0 turns into a 1 or the other way around, respectively. $\mathcal{A}_{\text{end}}$ finally detects when the block $\#1^n\#$ is not the end of the sequence (since incrementing does not work in this case as described above). Hence, w' is accepted by $\mathcal{A}_{\text{end}} \cup \mathcal{A}_{\text{incr}} \cup \mathcal{A}_{\text{copy}(0)} \cup \mathcal{A}_{\text{copy}(1)}$.

When $\mathcal{A}$ accepts a word, then it has to guess in the very first transition which error occurs by entering the respective FSA of the union. The smallest lookahead of a delegator for $\mathcal{A}$ is $k = |w_n| = 2^n(n+1) + 1$. In the first transition, a k-delegator for $\mathcal{A}$ can check whether the lookahead is the word w_n itself and chooses to enter $\mathcal{A}_{\text{end}}$ in this case. Otherwise, one of the error cases described above has occurred already and f can enter the respective FSA. For a contradiction, suppose $\mathcal{A}$ has a $(k-1)$-delegator f. Let $w'' \in \Sigma^*$ be the prefix of w_n of length $k-1$, i.e., $w''\# = w_n$. If f chooses to enter $\mathcal{A}_{\text{border}}$ for the first transition of w'' on $\mathcal{A}$, then it fails to accept the word $w_n\#\# \in L$. Otherwise, the delegator cannot accept the word $w'' \in L$ itself. We can summarize that for each $n \in \mathbb{N}_+$, the FSA $\mathcal{A}$ has a delegator but its lookahead is at least exponential in the number of states. ◁

We now show that if an FSA $\mathcal{A}$ has some k-delegator, then $\mathcal{A}$ also has some K-delegator for a number K that is singly exponential in the size of $\mathcal{A}$. To establish a bound K, we use a technique inspired by [HKT12], where two-player games with lookahead for one of the players are considered (by constructing a variant of the delegator game from Definition 4.1.5 where Player 0 is allowed to use a delay).

In our setting, the main idea is the following. If the lookahead K is big enough, then it contains an infix that can be pumped such that the considered lookahead word can be extended to a word of length k (with k and K as explained above where we assume w.l.o.g. $k > K$). On this longer lookahead word of length k, one can query the existing k-delegator and use the

same decisions to obtain a shorter K-delegator.

The required pumping argument that we just mentioned is formalized by an extension of Lemma 4.1.10 where we use transition profiles (cf. Definition 4.1.9) again.

Lemma 4.1.14. *Let $x, y, z \in \Sigma^*$ be such that $\Delta_x = \Delta_{xy}$ for an FSA $\mathcal{A}$. Then, $(xyz)^{-1}L_*(\mathcal{A}_q) = (xy^iz)^{-1}L_*(\mathcal{A}_q)$ for all $q \in Q$ and $i \in \mathbb{N}$.*

Proof. An easy induction shows that $\Delta_{xy} = \Delta_{xy^i}$ for all $i \in \mathbb{N}$. Consequently, $\Delta_{xyz} = \Delta_{xy^iz}$ holds and the claim follows directly by Lemma 4.1.10. □

Using a simple counting argument, we can show that each word of a certain length has a decomposition xyz as in Lemma 4.1.14.

Lemma 4.1.15. *For an FSA $\mathcal{A}$, each word $w \in \Sigma^K$ of length $K = 2^{|Q|^2}$ can be decomposed as $w = xyz$ with $y \neq \varepsilon$ and $\Delta_x = \Delta_{xy}$.*

Proof. A word of length $2^{|Q|^2}$ has $2^{|Q|^2} + 1$ prefixes. Two different prefixes must have the same transition profile since there are at most $2^{|Q|^2}$ transition profiles. This implies the existence of the claimed decomposition. □

We now combine Lemma 4.1.14 and Lemma 4.1.15 to prove that the bound $K = 2^{|Q|^2}$ is the maximal 'useful' lookahead.

Theorem 4.1.16. *An ε-free FSA $\mathcal{A}$ has a K-lookahead delegator if it has a bounded lookahead delegator.*

Proof. Let f be a k-delegator for $\mathcal{A}$ where $k > K$ w.l.o.g. We show that the property on the right hand side of Lemma 4.1.3, which holds for k by assumption, also holds for K for the same set $Q' \subseteq Q$. To this end, we have to show that for every $q \in Q'$, $a \in \Sigma$, and $w \in \Sigma^K$, there is some $p \in Q'$ with $(q, a, p) \in \Delta$ and $(aw)^{-1}L_*(\mathcal{A}_q) = w^{-1}L_*(\mathcal{A}_p)$. We know that there is a decomposition $w = xyz$ with $y \neq \varepsilon$ and $\Delta_x = \Delta_{xy}$. Choose $i \in \mathbb{N}$ and a proper prefix y' of y such that $|xy^iy'| = k$. Let y'' be such that $y = y'y''$. Finally, we pick some $p \in Q'$ with $(q, a, p) \in \Delta$ such that $(axy^iy')^{-1}L_*(\mathcal{A}_q) = (xy^iy')^{-1}L_*(\mathcal{A}_p)$ the existence of which is guaranteed by Lemma 4.1.3 for k-lookahead. With Lemma 4.1.14, we can now show that the

desired property holds for K-lookahead:

$$\begin{aligned}
(axyz)^{-1}L_*(\mathcal{A}_q) &= (axy^{i+1}z)^{-1}L_*(\mathcal{A}_q) && \text{(Lemma 4.1.14)}\\
&= (axy^iy'y''z)^{-1}L_*(\mathcal{A}_q) && (y = y'y'')\\
&= (y''z)^{-1}\big((axy^iy')^{-1}L_*(\mathcal{A}_q)\big) && \text{(composition of left quotients)}\\
&= (y''z)^{-1}\big((xy^iy')^{-1}L_*(\mathcal{A}_p)\big) && \text{(Lemma 4.1.3 for } k\text{-lookahead)}\\
&= (xy^iy'y''z)^{-1}L_*(\mathcal{A}_p) && \text{(composition of left quotients)}\\
&= (xy^{i+1}z)^{-1}L_*(\mathcal{A}_p) && (y = y'y'')\\
&= (xyz)^{-1}L_*(\mathcal{A}_p) && \text{(Lemma 4.1.14)}
\end{aligned}$$

□

Since the bound K is singly exponential in the number of states of $\mathcal{A}$ and therefore has a binary representation that is polynomial in the size of $\mathcal{A}$, Theorem 4.1.11 implies that Bounded-Delegator also is in PSPACE. Furthermore, our reduction showing that Delegator is PSPACE-hard also shows that Bounded-Delegator is PSPACE-hard (see proof of Theorem 4.1.12).

Corollary 4.1.17. *The problem Bounded-Delegator for ε-free FSAs is PSPACE-complete.*

In [RS07, Theorem 4], it is shown that Bounded-Delegator is in PSPACE for unambiguous FSAs. This result is generalized by Corollary 4.1.17. However, the FSA constructed in the proof of Theorem 4.1.12 for the PSPACE lower bound is ambiguous, in general, and therefore, our completeness result does not extend to unambiguous automata.

4.2 Delegation for Pushdown Automata

The concept of lookahead delegation can also be extended to pushdown automata. The delegator has to choose a transition based on the current state, stack top symbol, and the input with lookahead. Note that there is no lookahead on the stack content. This is also the reason why we have to avoid ε-transitions here, although they can be useful for pushdown automata. Our results get not in touch with ε-transitions anyway. We first define delegators for pushdown automata which extends Definition 4.1.1.

Definition 4.2.1. For an ε-free PDA $\mathcal{A}$ and a number $k \in \mathbb{N}$, a ***k*-lookahead delegator** (or ***k*-delegator** for short) is a function $f : Q \times \Gamma_\perp \times \Sigma\Sigma^{\leq k} \to Q \times \Gamma_\perp^*$ such that

a) $f(q, A, aw) = (p, W)$ implies $(q, A, a, p, W) \in \Delta$ for each $q \in Q, A \in \Gamma_\perp, a \in \Sigma, w \in \Sigma^{\leq k}$, and

b) $f^*(q_0\bot, w) \in F\Gamma^*\bot$ for each $w \in L_*(\mathcal{A})$, where $f^* : (Q\Gamma^*\bot) \times \Sigma^* \to Q\Gamma^*\bot$ extends f to words and is defined inductively as follows: let $f^*(qW, \varepsilon) = qW$ for each configuration qW, and let $f^*(qAV, a_1 \cdots a_n) = f^*\big(pUV, a_2 \cdots a_n\big)$ with $f\big(q, A, a_1 \cdots a_{\min(n,k+1)}\big) = (p, U)$ for a configuration qAV and a nonempty word $a_1 \cdots a_n$ of length n. ◁

The two conditions again express that only transitions are chosen by the delegator and that it leads to an accepting state for each word accepted by the automaton.

We first consider the restricted case of visibly pushdown automata. For them, we present a construction which borrows the idea of the 'delegated automaton' from Example 4.1.2 and thereby, we finally obtain a reduction to the emptiness problem for PDAs.

Theorem 4.2.2. *For given $k \in \mathbb{N}$, it is decidable whether a VPDA has a k-lookahead delegator. When k and Σ are fixed, the problem lies in NP.*

Proof. For a given VPDA $\mathcal{A}$ and a delegator candidate f, we want to construct an automaton $\mathcal{A}_f$ that stores the lookahead information in its state space and deterministically simulates the behavior of f. It then remains to check the language inclusion by reducing it to the emptiness problem for VPDAs: $L_*(\mathcal{A}) \subseteq L_*(\mathcal{A}_f)$ iff $L_*(\mathcal{A}) \cap \overline{L_*(\mathcal{A}_f)} = \varnothing$. The complement of $\mathcal{A}_f$ can be computed easily due to the determinism of $\mathcal{A}_f$. The problem is rather that $\mathcal{A}_f$ would not fulfill the visibly property any longer. The emptiness of the intersection with another pushdown language is not decidable in general.

We overcome this difficulty by given an explicit product construction for $\mathcal{A} \times \overline{\mathcal{A}_f}$, i.e., the product of $\mathcal{A}$ and the complement of $\mathcal{A}_f$. The key idea is that both automata take transitions synchronously, i.e., they process input letters synchronously which implies that the stack heights are synchronous, too. The stack of the product can hence be implemented by a stack of symbol tuples. The first component can choose transitions nondeterministically whereas the second component must stick to f. This yields a PDA (with only ε-transitions) which can be checked for emptiness in P (see [HU79]).

Formally, for a VPDA $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \bot, F)$ and f as above, we define the PDA $\mathcal{A}' = (Q', \Sigma, \Gamma', \Delta', q_0', \bot', F')$ consisting of:

a) states $Q' = (Q \times Q \times \Sigma^{\leq k}) \uplus \{q_0'\}$ with $F' = F \times (Q \setminus F) \times \{\varepsilon\}$,

b) stack alphabet $\Gamma' = \Gamma^2$ with $\bot' = (\bot, \bot)$,

c) transitions:

i) $\Delta' \ni \Big(q_0', \bot', \varepsilon, (q_0, q_0, w), \bot'\Big)$ where $w \in \Sigma^{\leq k}$,

ii) $\Delta' \ni \Big((q_1, q_2, w), (A_1, A_2), \varepsilon, (p_1, p_2, w'), (B_{1,1}, B_{2,1}) \cdots (B_{1,n}, B_{2,n})\Big)$ for some letter $a \in \Sigma$ and decomposition $wa = a'w'$ such that $w, w' \in \Sigma^k$, $a' \in \Sigma$, and where $\Delta \ni (q_1, A_1, a', p_1, B_{1,1} \cdots B_{1,n})$ and $f(q_2, A_2, a'w') = (p_2, B_{2,1} \cdots B_{2,n})$,

iii) $\Delta' \ni \Big((q_1, q_2, aw), (A_1, A_2), \varepsilon, (p_1, p_2, w), (B_{1,1}, B_{2,1}) \cdots (B_{1,n}, B_{2,n})\Big)$ where $a \in \Sigma$, $w \in \Sigma^{\leq k-1}$, $\Delta \ni (q_1, A_1, a, p_1, B_{1,1} \cdots B_{1,n})$, and $f(q_2, A_2, aw) = (p_2, B_{2,1} \cdots B_{2,n})$.

The three schemes for transitions are motivated as follows. Transitions of the first scheme are used to initiate the lookahead buffer with a (possibly shorter) word. The second scheme defines transitions for a completely filled lookahead. This is done by guessing some letter a to append to the lookahead w, to guess a transition for the first component, and to use the transition suggested by f in the second component. The transitions of the last scheme are used at the end of the simulated input, which is when the lookahead is not completely filled any more. The transitions for the components are chosen analogous to the previous case. By construction, we obtain that $L_*(\mathcal{A}') = \varnothing$ iff $L_*(\mathcal{A}) \cap \overline{L_*(\mathcal{A}_f)} = \varnothing$.

Complexity: We construct the PDA $\mathcal{A}'$ with $\mathcal{O}\big(k \cdot |\Sigma|^k \cdot |Q|^2\big)$ states over a stack alphabet of size $\mathcal{O}\big(|\Gamma|^2\big)$. The worst case running time of the emptiness test on $\mathcal{A}'$ dominates the whole procedure. □

To prove the decidability of k-Delegator in Theorem 4.2.2, we crucially relied on the decidability of the inclusion problem. However, it is known for decades that language inclusion is undecidable for DOCAs (recently shown in [BG11] by using [Min61, VP75]).

Theorem 4.2.3. *It is undecidable whether an ε-free OCA has a 0-lookahead delegator.*

Proof. We reduce the inclusion problem for ε-free DOCAs (which is undecidable [BG11]) to 0-Delegator for ε-free OCAs. Let $\mathcal{A}_1$ and A_2 be two ε-free DOCAs over some alphabet Σ not containing the symbol #, and let $q_{0,1}$ and $q_{0,2}$ be the initial states, respectively. Figure 4.4 depicts an ε-free OCA $\mathcal{A}$ that accepts the language $L_*(\mathcal{A}) = \#\Sigma^*\# \cup \#L_*(\mathcal{A}_1) \cup \#L_*(\mathcal{A}_2)$ over the alphabet $\Sigma' = \Sigma \uplus \{\#\}$. Note that the only nondeterministic choice occurs with symbol # at the initial state q_0 of $\mathcal{A}$. We complete the proof by showing that $\mathcal{A}$ has a 0-delegator iff $L_*(\mathcal{A}_1) \subseteq L_*(\mathcal{A}_2)$. The idea behind this construction is that in case of $L_*(\mathcal{A}_1) \subseteq L_*(\mathcal{A}_2)$, one can always safely enter $\mathcal{A}_2$ in the first step. But in the other case, nondeterminism is needed to be able to accept both ## and #w for some word $w \in L_*(\mathcal{A}_1) \setminus L_*(\mathcal{A}_2)$.

Suppose $L_*(\mathcal{A}_1) \subseteq L_*(\mathcal{A}_2)$. Since $\#^{-1}L_*(\mathcal{A}) = L_*(\mathcal{A}_{q_{0,2}}) = L_*(\mathcal{A}_2) \cup \Sigma^*\#$, there is no need to enter $\mathcal{A}_1$. Hence, a 0-delegator f would always choose to go to $\mathcal{A}_2$, i.e., $f(q_0, \#) = q_{0,2}$.

For the other direction, suppose $\mathcal{A}$ has a 0-delegator f. Then, it holds that $f(q_0, \#) = q_{0,2}$, because if $f(q_0, \#) \neq q_{0,1}$, then f would not accept the word ## $\in L_*(\mathcal{A})$. From $f(q_0, \#) = q_{0,2}$

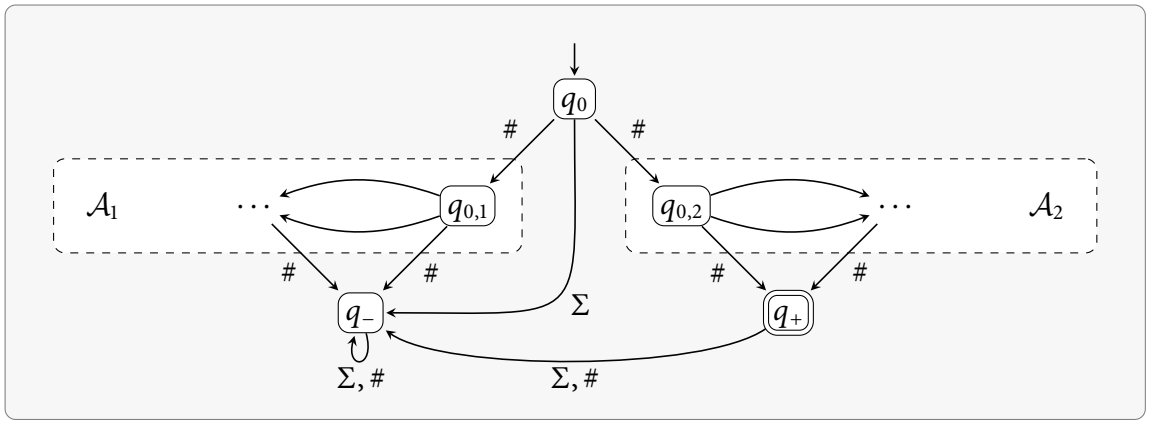


Figure 4.4: Automaton accepting the language $\#\Sigma^*\# \cup \#L_*(\mathcal{A}_1) \cup \#L_*(\mathcal{A}_2)$

and $\#L_*(\mathcal{A}_1) \subseteq L_*(\mathcal{A}_{q_0})$ directly follows that $L_*(\mathcal{A}_1) \subseteq L_*(\mathcal{A}_{q_{0,2}})$ which yields the desired inclusion when restricting to Σ, i.e., $L_*(\mathcal{A}_1) \subseteq L_*(\mathcal{A}_2)$. □

Theorem 4.2.4. *It is undecidable whether an ε-free OCA has a bounded lookahead delegator.*

Proof. The idea is based on the proof of [FLZ11, Theorem 4]. Let $\mathcal{M}$ be a 2-register machine with state set Q. We demand w.l.o.g. that no configuration occurs twice in the unique run of $\mathcal{M}$ (which can be achieved by adding a third register which is incremented every other step and then, simulating this 3-register machine by a 2-register machine). A configuration (q, n_0, n_1) of $\mathcal{M}$ is encoded by the word $q\$_0^{n_0}\$_1^{n_1}$. Over the alphabet $\Sigma = Q \cup \{\$_0, \$_1, \#\}$, we consider the language $L \subseteq \Sigma^*$ consisting of exactly those words $w \in \#0\#(Q\$_0^*\$_1^*\#)^*$ that encode a sequence of configurations $c_0, \ldots, c_n$ starting with the initial configuration c_0 and containing a pair c_j, c_{j+1} of successive configurations where c_{j+1} is not the successor of c_j according to $\mathcal{M}$. Note that the encoding of a configuration and its successor differ in length by at most 1. L is accepted by an ε-free OCA $\mathcal{A}$ that guesses at the beginning of each configuration whether its successor is updated incorrectly. If this is the case, then the update was wrong in one of the three components of a configuration: the state, register 0, or register 1. The respective case can also be guessed and checked by $\mathcal{A}$ using its counter. Since the halting problem for 2-register machines is undecidable, it remains to show that $\mathcal{A}$ has a delegator iff $\mathcal{M}$ halts (cf. Section 2.4).

Suppose $\mathcal{M}$ halts after h steps. Consequently, every run $c_0c_1\cdots c_{h+1}$ with $h+1$ steps has to contain an erroneous update. The encoding of such a run has a length of at most $k = (h+3) + \sum_{i=0}^{h+1}(i+1)$. By using a k-lookahead, one can deterministically detect and verify an error of each type. Hence, $\mathcal{A}$ has a k-delegator.

For the other direction, suppose $\mathcal{M}$ does not halt. Since the run is infinite and we excluded a loop in the configurations of its run, we can conclude that the encoded configurations have unbounded length. Hence, no bounded lookahead can detect an erroneous update of a sufficiently long configuration. □

Although we achieved decidability results for k-DELEGATOR and DELEGATOR in Theorem 4.2.2, the latter two results prove the undecidability of delegation for pushdown automata in general.

Chapter 5

Conclusion

In this thesis, we have studied various simplification problems for automata and games. First, we have considered the regularity problem for pushdown automata. Motivated by its decidability for finite words, we have tackled the case for infinite words which was an open problem. We have further generalized the regularity problem to pushdown games as those games are specified by pushdown ω-automata. As a second aspect of simplification, we have considered lookahead delegation for nondeterministic automata.

Regularity Problems

In Chapter 3, our first contributions are about regular winning strategies for pushdown games. We have shown that it is impossible in general to decide the existence of a finite state winning strategy. In fact, the only decidability result we have obtained concerns the very limited case of games with reachability conditions. We have proven this case to trivially omit very simple strategies. For the dual case of games with safety conditions, we have proven the regularity problem to be undecidable. Although the safety condition itself is simple already, we showed the undecidability even for two different restrictions of the pushdown game specification, namely for one-counter games and for visibly pushdown games. It seems that the hardness of the problem is more connected to the winning condition than to the pushdown specification of a game.

We have proceeded by contributing the classification game. It is designed to express acceptance of a pushdown ω-automaton by a pushdown game. We have proven that various aspects of simplification of the ω-automaton are directly connected to the corresponding simplifications in the game. We have shown that the classification game can be used to decide whether an ω-language can be recognized by ω-automata with certain restricted acceptance condition; like Büchi, co-Büchi, weak, safety, or reachability. Although these results were known before, the novelty of our game-based approach is an intuitive understanding and

that it leads to less technical proofs. Regarding regularity, we have shown that the automaton recognizes a regular language iff the classification game omits a regular winning strategy. This reveals the regularity problem for pushdown ω-languages to be a special case of the one for pushdown games. However, as we have previously shown the game-based approach to be undecidable, this tells us that we require specific solutions for ω-languages.

As a first contribution regarding regularity of pushdown ω-languages, we have presented a normal form for weak pushdown ω-automata. We have proven that it allows us to simply lift some known decision procedures from finite words to ω-words. This way, we have shown both, the equivalence problem and the regularity problem to be decidable for weak pushdown ω-automata. Regarding the complexity, we have proven our regularity test to run in triply exponential time and to produce an equivalent finite state ω-automaton having a worst case size between singly and doubly exponential. Surprisingly, our space complexity corresponds to the one for finite words, although, our normal form induces an exponential blowup of the automaton.

By a last contribution, we have tackled the case of regularity for pushdown ω-automata with non-weak acceptance condition. We have established a congruence relation and have proven it to characterize regularity for this class of ω-languages. In an example, we have shown that the characterization property is only obtained when the congruence is applied to this class of ω-languages. Further, the decidability of the regularity problem remains open.

To summarize the regularity problems for pushdown automata and games, the following table orders the problems by increasing complexity beginning from pushdown automata on finite words up to games on pushdown ω-automata and it gives an overview of our main contributions:

	Known results:	Our contributions:		
Problem	DPDA	weak ω-DPDA	ω-DPDA	PDG
Regularity	3EXPTIME [Val75]	3EXPTIME	–	undecidable

For each of the three variants of regularity that we have considered, questions for future work arise. The most interesting one is surely the old open problem from [CG78] whether the regularity problem is decidable for pushdown ω-automata. For pushdown games, it might be possible to obtain more decidability results for restricted cases like visibly one-counter games. For weak pushdown ω-automata, there remains a small gap in the size complexity of our construction. It is further interesting whether the complexity can be improved for restricted cases; especially for weak one-counter ω-automata. This is motivated by the existence of faster

algorithms regarding one-counter automata on finite words for the regularity problem [VP75] and the equivalence problem [BG11, BGJ13].

Lookahead Delegation

In Chapter 4, we have considered lookahead delegators for nondeterministic automata. Our research was motivated by [RS07] where the delegator problem was introduced for finite state automata, subdivided into three different formulations, and upper bounds were given for each formulation but only for the restricted subclass of unambiguous finite state automata.

We continued these studies and have contributed respective complexity results for finite automata in general. First, we have given an algorithm based on safety games that shows how the existence of a delegator can be computed in polynomial time when the amount of lookahead is fixed. This results further corrects a wrong result from the literature. However, our algorithms has a rather poor complexity if the amount of lookahead becomes a part of the input. As a second contribution, we have developed different decision procedure for that case and have used it to prove that the problem is PSPACE-complete. In our third contribution, we have provided an upper bound on the maximally required amount of lookahead. Combined with the previous result, we have shown that deciding the existence of a lookahead delegator for some bounded lookahead is PSPACE-complete, too. In total, our results have given a complete picture for the complexities of delegator problems for finite state automata.

We have further generalized the delegator problems to pushdown automata. We have shown that the naïve approach can be applied to visibly pushdown automata which has allowed us the decide the existence of a delegator for a given amount of lookahead. In fact, this problem is contained in NP if the lookahead is fixed. Besides this positive result, we have further proven that for one-counter automata, even the simplest problem is undecidable; namely deciding the existence of a delegator without lookahead. The undecidability for this very restricted case is passed on to other less restricted cases.

The following table gives an overview of results about delegation for various classes of automata (ordered by expressiveness):

	Known results:	Our contributions:		
Problem	unamb. FSA	FSA	VPDA	OCA & PDA
Fixed	P [RS07]	P	NP	undecidable
Given	co-NP [RS07]	PSPACE-complete	decidable	undecidable
Bounded	PSPACE [RS07]	PSPACE-complete	–	undecidable

The above pictures does not lead to many questions for future work. Although, there are some problems open that concern restricted cases. We have obtained our PSPACE-hardness results with finite state automata that are ambiguous. Hence, the upper bounds for unambiguous automata might be improved. Another open problem remains for a restricted pushdown case: is it decidable whether a visibly pushdown automaton has a delegator for some bounded lookahead? It would further be interesting to extend lookahead delegation to ω-automata.

Bibliography

[ACKM04] G. Alonso, F. Casati, H. A. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer, 2004.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[AKL10] B. Aminof, O. Kupferman, and R. Lampert. Reasoning about Online Algorithms with Weighted Automata. *ACM Transactions on Algorithms*, 6(2), 2010.

[AKMV05] R. Alur, V. Kumar, P. Madhusudan, and M. Viswanathan. Congruences for Visibly Pushdown Languages. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 1102–1114. Springer, 2005.

[AM04] R. Alur and P. Madhusudan. Visibly Pushdown Languages. In L. Babai, editor, *STOC*, pages 202–211. ACM, 2004.

[BCG+03] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic Composition of E-services That Export Their Behavior. In M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, editors, *ICSOC*, volume 2910 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2003.

[BG11] S. Böhm and S. Göller. Language Equivalence of Deterministic Real-Time One-Counter Automata Is NL-Complete. In F. Murlak and P. Sankowski, editors, *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 2011.

[BGJ13] S. Böhm, S. Göller, and P. Jancar. Equivalence of Deterministic One-Counter Automata is NL-complete. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *STOC*, pages 131–140. ACM, 2013.

[BK08] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[BL69] J. R. Büchi and L. H. Landweber. Solving Sequential Conditions by Finite-State Strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

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

[Cac02] T. Cachat. Symbolic Strategy Synthesis for Games on Pushdown Graphs. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 704–715. Springer, 2002.

[CG77a] R. S. Cohen and A. Y. Gold. Theory of ω-Languages. I. Characterizations of ω-Context-Free Languages. *Journal of Computer and System Sciences*, 15(2):169–184, 1977.

[CG77b] R. S. Cohen and A. Y. Gold. Theory of ω-Languages. II. A Study of Various Models of ω-Type Generation and Recognition. *Journal of Computer and System Sciences*, 15(2):185–208, 1977.

[CG78] R. S. Cohen and A. Y. Gold. ω-Computations on Deterministic Pushdown Machines. *Journal of Computer and System Sciences*, 16(3):275–300, 1978.

[CHP07] K. Chatterjee, T. A. Henzinger, and N. Piterman. Generalized Parity Games. In H. Seidl, editor, *FoSSaCS*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007.

[Chu57] A. Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume 1, pages 3–50. Cornell Univ., Ithaca, 1957.

[Chu63] A. Church. Logic, Arithmetic, and Automata. In *Proc. International Congress of Mathematicians, Inst. Mittag-Leffler, Djursholm, Sweden*, pages 23–35. Almqvist and Wiksells, Uppsala, 1963.

[CKS81] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[CM99] O. Carton and R. Maceiras. Computing the Rabin Index of a Parity Automaton. *Informatique Théorique et Applications*, 33(6):495–506, 1999.

[Col12] T. Colcombet. Forms of Determinism for Automata (Invited Talk). In C. Dürr and T. Wilke, editors, *STACS*, volume 14 of *LIPIcs*, pages 1–23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[COT12] N. Chaturvedi, J. Olschewski, and W. Thomas. Languages versus ω-Languages in Regular Infinite Games. *International Journal of Foundations of Computer Science*, 23(5):985–1000, 2012.

[DIS05] Z. Dang, O. H. Ibarra, and J. Su. On composition and lookahead delegation of *e*-services modeled by automata. *Theoretical Computer Science*, 341(1-3):344–363, 2005.

[DJW97] S. Dziembowski, M. Jurdzinski, and I. Walukiewicz. How Much Memory is Needed to Win Infinite Games? In *LICS*, pages 99–110. IEEE Computer Society, 1997.

[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, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000.

[EJ91] E. A. Emerson and C. S. Jutla. Tree Automata, Mu-Calculus and Determinacy (Extended Abstract). In *FOCS*, pages 368–377. IEEE Computer Society, 1991.

[FLZ11] W. Fridman, C. Löding, and M. Zimmermann. Degrees of Lookahead in Context-free Infinite Games. In M. Bezem, editor, *CSL*, volume 12 of *LIPIcs*, pages 264–276. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[Fri10] W. Fridman. Formats of Winning Strategies for Six Types of Pushdown Games. In A. Montanari, M. Napoli, and M. Parente, editors, *GANDALF*, volume 25 of *EPTCS*, pages 132–145, 2010.

[GH82] Y. Gurevich and L. Harrington. Trees, Automata, and Games. In H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber, editors, *STOC*, pages 60–65. ACM, 1982.

[GHIS04] C. E. Gerede, R. Hull, O. H. Ibarra, and J. Su. Automated Composition of E-services: Lookaheads. In M. Aiello, M. Aoyama, F. Curbera, and M. P. Papazoglou, editors, *ICSOC*, pages 252–262. ACM, 2004.

[Grä11] E. Grädel. Back and Forth Between Logics and Games. In *Lectures in Game Theory for Computer Scientists*, pages 99–145. Springer, 2011.

[HKT12] M. Holtmann, L. Kaiser, and W. Thomas. Degrees of Lookahead in Regular Infinite Games. *Logical Methods in Computer Science*, 8(3), 2012.

[HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (Second Edition)*. Addison-Wesley series in computer science. Addison-Wesley-Longman, 2001.

[HO09] M. Hague and C.-H. L. Ong. Winning Regions of Pushdown Parity Games: A Saturation Method. In M. Bravetti and G. Zavattaro, editors, *CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 384–398. Springer, 2009.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Lan69] L. H. Landweber. Decision Problems for ω-Automata. *Mathematical Systems Theory*, 3(4):376–384, 1969.

[Lin77] M. Linna. A Decidability Result for Deterministic ω-Context-Free Languages. *Theoretical Computer Science*, 4(1):83–98, 1977.

[Löd98] C. Löding. Methods for the Transformation of ω-Automata: Complexity and Connection to Second order Logic. Diplomarbeit, Christian-Albrechts-Universität of Kiel, 1998.

[Löd01] C. Löding. Efficient minimization of deterministic weak ω-automata. *Information Processing Letters*, 79(3):105–109, 2001.

[LR12] C. Löding and S. Repke. Regularity Problems for Weak Pushdown ω-Automata and Games. In B. Rovan, V. Sassone, and P. Widmayer, editors, *MFCS*, volume 7464 of *Lecture Notes in Computer Science*, pages 764–776. Springer, 2012.

[LR13] C. Löding and S. Repke. Decidability Results on the Existence of Lookahead Delegators for NFA. In A. Seth and N. K. Vishnoi, editors, *FSTTCS*, volume 24

of *LIPIcs*, pages 327–338. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[McN66] R. McNaughton. Testing and Generating Infinite Sequences by a Finite Automaton. *Information and Control*, 9(5):521–530, 1966.

[Min61] M. L. Minsky. Recursive Unsolvability of Post's Problem of "Tag" and other Topics in Theory of Turing Machines. *The Annals of Mathematics*, 74(3):437–455, November 1961.

[Mos91] A. W. Mostowski. Games with Forbidden Positions. Technical Report 78, Uniwersytet Gdański, Instytut Matematyki, 1991.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[MW08] A. Muscholl and I. Walukiewicz. A Lower Bound on Web Services Composition. *Logical Methods in Computer Science*, 4(2), 2008.

[Pap94] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[PP04] D. Perrin and J.-É. Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

[Ram30] F. P. Ramsey. On a problem in formal logic. *Proc. London Mathematical Society*, 30(3):264–286, 1930.

[RS07] B. Ravikumar and N. Santean. On the Existence of Lookahead Delegators for NFA. *International Journal of Foundations of Computer Science*, 18(5):949–973, 2007.

[RT07] A. Rabinovich and W. Thomas. Logical Refinements of Church's Problem. In J. Duparc and T. A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2007.

[Saf88] S. Safra. On the Complexity of ω-Automata. In *FOCS*, pages 319–327. IEEE Computer Society, 1988.

[Sén01] G. Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theoretical Computer Science*, 251(1-2):1–166, 2001.

[Sén02] G. Sénizergues. L(A)=L(B)? A simplified decidability proof. *Theoretical Computer Science*, 281(1-2):555–608, 2002.

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

[SS07] L. Segoufin and C. Sirangelo. Constant-Memory Validation of Streaming XML Documents Against DTDs. In T. Schwentick and D. Suciu, editors, *ICDT*, volume 4353 of *Lecture Notes in Computer Science*, pages 299–313. Springer, 2007.

[Sta83] L. Staiger. Finite-State ω-Languages. *Journal of Computer and System Sciences*, 27(3):434–448, 1983.

[Ste67] R. E. Stearns. A Regularity Test for Pushdown Machines. *Information and Control*, 11(3):323–340, 1967.

[Str94] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Basel, Switzerland, 1994.

[SV02] L. Segoufin and V. Vianu. Validating Streaming XML Documents. In L. Popa, S. Abiteboul, and P. G. Kolaitis, editors, *PODS*, pages 53–64. ACM, 2002.

[Val75] L. G. Valiant. Regularity and Related Problems for Deterministic Pushdown Automata. *Journal of the ACM*, 22(1):1–10, 1975.

[VP75] L. G. Valiant and M. Paterson. Deterministic One-Counter Automata. *Journal of Computer and System Sciences*, 10(3):340–350, 1975.

[Wal01] I. Walukiewicz. Pushdown Processes: Games and Model-Checking. *Information and Computation*, 164(2):234–263, 2001.

[Zie98] W. Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

Index