International Journal of Foundations of Computer Science
Vol. 26, No. 8 (2015) 1041–1068

DOI: 10.1142/S0129054115400122

Simplification Problems for Deterministic Pushdown Automata on Infinite Words

Christof Löding

Lehrstuhl Informatik 7, RWTH Aachen University, 52056 Aachen, Germany
loeding@cs.rwth-aachen.de

Received 27 October 2014
Accepted 12 July 2015
Communicated by Zoltán Ésik

The article surveys some decidability results concerning simplification problems for DPDAs on infinite words (ω-DPDAs). We summarize some recent results on the regularity problem, which asks for a given ω-DPDA, whether its language can also be accepted by a finite automaton. The results also give some insights on the equivalence problem for a subclass of ω-DPDA. Furthermore, we present some new results on the parity index problem for ω-DPDAs. For the specification of a parity condition, the states of the ω-DPDA are assigned priorities (natural numbers), and a run is accepting if the highest priority that appears infinitely often during a run is even. The basic simplification question asks whether one can determine the minimal number of priorities that are needed to accept the language of a given ω-DPDA. We provide some decidability results on variations of this question for some classes of ω-DPDAs.

Keywords: Automata theory; pushdown automata; ω-automata; parity index.

1. Introduction

Finite automata, which are used as a tool in many areas of computer science, have good closure and algorithmic properties. For example, language equivalence and inclusion are decidable (see [10]), and for many subclasses of the regular languages it is decidable whether a given automaton accepts a language inside this subclass (see [21] for some results of this kind). In contrast to that, the situation for pushdown automata is much more difficult. For nondeterministic pushdown automata, many problems like language equivalence and inclusion are undecidable (see [10]), and it is undecidable whether a given nondeterministic pushdown automaton accepts a regular language. The class of languages accepted by deterministic pushdown automata forms a strict subclass of the context-free languages. While inclusion remains undecidable for this subclass, a deep result from [17] shows the decidability of the equivalence problem. Furthermore, the regularity problem for deterministic pushdown automata is also decidable [19, 23].

While automata on finite words are a very useful model, some applications, in particular in verification by model checking (see [3]), require extensions of these

models to infinite words. Although the theory of finite automata on infinite words (called ω-automata in the following) usually requires more complex constructions because of the more complex acceptance conditions, many of the good properties of finite automata on finite words are preserved (see [15] for an overview). Pushdown automata on infinite words (pushdown ω-automata) have been studied because of their ability to model executions of non-terminating recursive programs. In [7] efficient algorithms for checking emptiness of Büchi pushdown automata are developed (a Büchi automaton accepts an infinite input word if it visits an accepting state infinitely often during its run). Besides these results, the algorithmic theory of pushdown ω-automata has not been investigated very much. For example, in [6] the decidability of the regularity problem for deterministic pushdown ω-automata has been posed as an open question and to our knowledge no answer to this question is known. Furthermore, it is unknown whether the equivalence of deterministic pushdown ω-automata is decidable.

In this article, we analyze several decision problems for deterministic ω-DPDAs that are mainly concerned with simplification questions. The first part of this article summarizes some recent partial results on the regularity problem for ω-DPDAs from [14], thus a simplification problem concerning the memory structure of the automaton.

In the second part we consider simplification questions for the acceptance condition of the automata. One of the standard acceptance conditions of ω-automata is the parity condition (see [9] for an overview of possible acceptance conditions). Such a condition is specified by assigning priorities (natural numbers) to the states of the automaton, using even priorities for "good" states and odd priorities for the "bad" states. A run is accepting if among the states that occur infinitely often the highest priority is even. For deterministic automata (independent of the precise automaton model), one can show that more languages can be accepted if more priorities are used. So the number of priorities required for accepting a language is a measure for the complexity of the language. A natural decision problem arising from that, is the question of determining for a given deterministic parity automaton the smallest number of priorities that are needed for accepting the language of the automaton. This referred to as the parity index problem.

For finite deterministic parity automata, the minimal number of priorities required for accepting the language can be computed in polynomial time, and a corresponding automaton can be constructed by simply reassigning priorities in the allowed range to the states of the given automaton [5]. For deterministic pushdown parity automata it was shown in [11] that it is decidable whether a given automaton is equivalent to a deterministic pushdown Büchi automaton. We present here the general result that the parity index problem for deterministic pushdown parity automata is decidable. The method is based on parity games on pushdown graphs and has already been described in the PhD thesis [16].

We further consider a model of deterministic pushdown automata in which the types of the action on the pushdown store are determined by the input symbols,

called visibly pushdown automata (VPA) [2]. In these automata, the input alphabet is partitioned into three sets of symbols, referred to as call, return, and internal symbols. On reading a call, the pushdown automaton has to add a symbol to the stack, on reading a return, it has to remove a symbol from the stack, and on reading an internal, it does not alter the stack. It turns out that, for a fixed partition of the input alphabet, this class of automata has good closure and algorithmic properties [2]. On finite words it is even possible to determinize such VPAs. However, it turns out that Büchi VPAs cannot, in general, be transformed into equivalent deterministic Muller or parity VPAs [2]. To resolve this problem, in [13] a variation of the parity condition has been proposed, referred to as stair parity condition. It is defined as a standard parity condition, however, it is not evaluated on the sequence of all states but only on the sequence of states that occur on steps of the run. A step is a configuration in the run such that no later configuration has a smaller stack height. In [13] it is shown that each nondeterministic Büchi VPA can be transformed into an equivalent deterministic stair parity VPA. We prove here that the stair parity index problem for deterministic VPAs can be solved in polynomial time. We also consider the question whether a given stair parity VPA is equivalent to a parity VPA (with a standard parity condition instead of a stair condition). For the particular case of stair Büchi VPAs we show that this problem is decidable.

The remainder of this paper is structured as follows. In Sec. 2 we introduce some basic terminology and definitions. In Sec. 3 we consider the regularity problem for ω-DPDAs, and mention a consequence for the equivalence problem. Section 4 is about simplifying the acceptance condition of parity DPDAs and stair parity DVPAs: in Secs. 4.1 and 4.2 we show how to reduce the number of priorities required in a given parity DPDA or stair parity DVPA, respectively, and in Sec. 4.3 we consider the problem of deciding whether a given stair parity DVPA can be turned into an equivalent parity DVPA (without stair condition). Finally, in Sec. 5 we give a short conclusion.

2. Preliminaries

We denote the set of natural numbers (including 0) by $\mathbb{N}$. For a set S we denote its cardinality by $|S|$. Let A be an alphabet, i.e., a finite set of symbols, then A^* is the set of finite words over A, and A^ω the set of ω-words over A, i.e., infinite sequences of A symbols indexed by the natural numbers. The subsets of A^* are called languages, and subsets of A^ω are called ω-languages. The length of a finite word $w \in A^*$ is denoted by $|w|$, and the empty word is ϵ. We assume the reader to be familiar with regular languages, i.e., the languages specified by regular expressions or equivalently by finite state automata (see, for example, [10] for basics on regular languages).

We are mainly concerned with deterministic pushdown automata in this work. We first define pushdown machines, which are pushdown automata without

acceptance condition. We then obtain pushdown automata by adding an acceptance condition.

A *deterministic pushdown machine* $\mathcal{M} = (Q, A, \Gamma, \delta, q_0, \bot)$ consists of

- a finite state set Q and initial state $q_0 \in Q$,
- a finite input alphabet A (we abbreviate $A_\epsilon = A \cup \{\epsilon\}$),
- a finite stack alphabet Γ and initial stack symbol $\bot \notin \Gamma$ (let $\Gamma_\bot = \Gamma \cup \{\bot\}$),
- a partial transition function $\delta : Q \times \Gamma_\bot \times A_\epsilon \to Q \times \Gamma_\bot^*$ such that for each $p \in Q$ and $A \in \Gamma_\bot$:
  - $\delta(p, Z, a)$ is defined for all $a \in A$ and $\delta(p, Z, \epsilon)$ is undefined, or the other way round.
  - For each transition $\delta(p, Z, a) = (q, W)$ with $a \in A_\epsilon$ the bottom symbol $\bot$ stays at the bottom of the stack and only there, i.e., $W \in \Gamma^*\bot$ if $Z = \bot$ and $W \in \Gamma^*$ if $Z \neq \bot$.

The set of configurations of $\mathcal{M}$ is $Q\Gamma^*\bot$ where $q_0\bot$ is the initial configuration. The stack consisting only of $\bot$ is called the empty stack. A configuration $q\sigma$ is also written (q, σ). For a given input word $w \in A^*$ or $w \in A^\omega$, a finite resp. infinite sequence $q_0\sigma_0, q_1\sigma_1, \ldots$ of configurations with $q_0\sigma_0 = q_0\bot$ is a run of w on $\mathcal{M}$ if there are $a_i \in A_\epsilon$ with $w = a_1a_2\cdots$ and $\delta(q_i, Z, a_{i+1}) = (q_{i+1}, U)$ is such that $\sigma_i = ZV$ and $\sigma_{i+1} = UV$ for some stack suffix $V \in \Gamma_\bot^*$.

For finite words, we consider the model of a deterministic pushdown automaton (DPDA) $\mathcal{A} = (\mathcal{M}, F)$ consisting of a deterministic pushdown machine $\mathcal{M} = (Q, A, \Gamma, \delta, q_0, \bot)$ and a set of final states $F \subseteq Q$. It accepts a word $w \in A^*$ if w induces a run ending in a final state. These words form the language $L_*(\mathcal{A}) \subseteq A^*$. For ω-words, we consider two types of acceptance conditions, namely Büchi and parity conditions. A Büchi DPDA $\mathcal{A} = (\mathcal{M}, F)$ is specified in the same way as a DPDA on finite words. The ω-language $L_\omega(\mathcal{A})$ defined by $\mathcal{A}$ is the set of all ω-words w for which the run of $\mathcal{A}$ on w contains a state from F at infinitely many positions.

For a parity DPDA, the acceptance condition is specified by a function $\Omega : Q \to \mathbb{N}$, which assigns a number to each state, referred to as its priority. A sequence of priorities satisfies the parity condition if the highest priority that occurs infinitely often is even. A run is accepting if the corresponding sequence of priorities satisfies the parity condition. Note that Büchi conditions can be specified as parity conditions by assigning priority 2 to states in F and priority 1 to states outside F.

In Sec. 3 we consider the class of weak DPDAs. These are parity DPDAs, in which the transitions can never lead from one state q to another state q' with a smaller priority. Hence, in a run of a weak DPDA the sequence of priorities is monotonically increasing, which implies that the sequence is ultimately constant. It follows that each weak DPDA is equivalent to the Büchi DPDA that uses the set of states with even priority as set of final states. We therefore also use term weak Büchi DPDAs to emphasize that it is a subclass of Büchi DPDAs.

In general, we refer to DPDAs on infinite words as ω-DPDAs if we do not explicitly specify the type of acceptance. For simplicity, we assume that infinite sequences

of ϵ-transitions are not possible in ω-DPDAs. For eliminating such sequences, it is sufficient to compute the pairs (q, Z) of states q and top stack symbols Z such that there is a run of ϵ-transitions leading from $qZ\bot$ to some configuration of the form $qZWZ\bot$, such that the Z at the bottom of the stack is never removed during this run. These pairs can be computed efficiently (see [7]). Redirecting the ϵ-transitions from these pairs (q, Z) into a sink state is sufficient for eliminating all infinite ϵ-sequences. The acceptance status of such a sink state would depend on the exact semantics one uses for runs that end in an infinite ϵ-sequence.

We also consider the model of deterministic visibly pushdown automata (DVPA) [2]. These automata are defined with respect to a partitioned alphabet $A = A_c \cup A_i \cup A_r$, where transitions on A_c always push some symbol onto the stack, transitions on A_r pop a symbol from the stack, and transitions on A_i leaving the stack unchanged. We refer to the letters in A_c, A_i, and A_r as call symbols, internal symbols, and return symbols, respectively. Furthermore, DVPAs do not have ϵ-transitions. We also adopt the general convention that transitions of VPAs on call or internal symbols only depend on the current state (and not on the top-most stack symbol). This simplifies several arguments. We can make this assumption without loss of generality, because by enriching the state space and the stack alphabet, it is possible to always keep track of the current top-most stack symbol (of the original automaton).

Formally, a deterministic visibly pushdown machine over the partitioned alphabet $A = A_c \cup A_i \cup A_r$ is of the form $\mathcal{M} = (Q, A, \Gamma, \delta, q_0, \bot)$, where δ consists of three transition functions

$$\begin{aligned} \delta_c &: Q \times A_c \to Q \times \Gamma \\ \delta_r &: Q \times \Gamma \times A_r \to Q \\ \delta_i &: Q \times A_i \to Q\,. \end{aligned}$$

Instead of defining the semantics of these transitions directly, we simply describe how the corresponding transitions in a standard DPDA would look like. A call transition $\delta_c(q, c) = (p, Z)$ corresponds to a set of transitions $\delta(q, Y, c) = (p, ZY)$ for each $Y \in \Gamma_\bot$. A return transition $\delta_r(q, Z, r) = p$ corresponds to the transition $\delta_r(q, Z, r) = (p, \epsilon)$, and an internal transition $\delta_i(q, i) = p$ to a set of transitions $\delta(q, Y, i) = (p, Y)$ for each $Y \in \Gamma_\bot$. Note that this definition does not admit transitions for return symbols on the empty stack. In [2] such transitions are possible, but we prefer to use the simpler model here to ease the presentation.

By adding an acceptance condition, we obtain DVPAs. As for ω-DPDAs, we are interested in ω-DVPAs with Büchi or parity condition. However, we also consider a variant of the parity condition referred to as stair parity condition [13]. The condition is specified in the same way as before, however, it is evaluated only on a subsequence of the run, namely on the sequence of steps, as defined below.

A configuration $q\sigma$ in a run of a DVPA $\mathcal{A}$ is called a step if the stack height of all configurations $q'\sigma'$ that come later in the run, is bigger than the stack height of $q\sigma$, i.e., $|\sigma| \leq |\sigma'|$. Note that the positions of the steps do not depend on the automaton, but only on the input word, because the type of the stack operation

is determined for each input symbol. We can now define stair visibly pushdown automata. The only difference to visibly pushdown automata is that they evaluate the acceptance condition only for the subsequence of the run consisting of the steps.

In other words, a stair parity DVPA has the same components as a parity DVPA. An input is accepted if in the run on this input the maximal priority that occurs infinitely often on a step is even. In the same way, we obtain stair Büchi DVPAs, which accept if an accepting state occurs on infinitely many steps.

We end this section by introducing some more terminology for visibly pushdown automata that is used in Sec. 4.

The set of well-matched words over $A = A_c \cup A_i \cup A_r$ is, intuitively speaking, the set of well-balanced words in which for each position with a call symbol, there is a later position at which this call is "closed" by some return symbol (and vice versa, each return position has a corresponding previous call position). Formally, the set is defined inductively by the following rules:

(1) Each $a \in A_i$ is a well-matched word.
(2) If u and v are well-matched words, then uv is a well-matched word.
(3) If w is a well-matched word, then cwr is a well-matched word for each $c \in A_c$ and each $r \in A_r$.

The words that are created by the last rule are referred to as minimally well-matched words (because they cannot be decomposed into well-matched factors). Let L_{mwm} denote this set, i.e., the words of the form cwr with a call c, a return r, and a well-matched word w.

The canonical language that can be accepted by a stair Büchi DVPA but by no parity DVPA is the language L_{su} of strictly unbounded words, containing all words over $\langle\{c\}, \emptyset, \{r\}\rangle$ with an infinite number of unmatched calls. More formally, an infinite word is in L_{su} if it is of the form $w_1 c w_2 c w_3 c \cdots$ for well-matched words w_i. In [2] it is shown that L_{su} cannot be accepted by a parity DVPA. But it is easy to construct a stair Büchi DVPA $\mathcal{A}$ for L_{su} using only a single stack symbol and one accepting and one non-accepting state (see [13]), where $\mathcal{A}$ moves into the accepting state for each c, and into the non-accepting state for each r. Note that the position after reading a c is a step in the run iff this c does not have a matching return. Thus, there are infinitely many unmatched calls iff there are infinitely many accepting states on steps.

3. The Regularity Problem: Simplifying the Memory Structure

In this section, we summarize results from [14] that show how to solve the regularity problem for weak ω-DPDAs. The proof uses a reduction to the corresponding problems for DPDAs on finite words, and thus also yields a result for the equivalence problem for weak ω-DPDAs. More details on these results can be found in [14] and in [16].

The regularity problem for DPDAs is the problem of deciding for a given DPDA whether it accepts a regular language. It has been shown to be decidable in [19] and the complexity has been improved in [23].

Theorem 1 ([19]). *The regularity problem for DPDAs is decidable.*

The rough idea of the proof is as follows. Assuming that the language of the given DPDA is regular, one shows that for each configuration above a certain height (depending on the size of the DPDA), there is an equivalent configuration of smaller height. A finite state machine can then be constructed by redirecting the transitions into higher configurations to their equivalent smaller counterparts. Here, two configurations are considered to be equivalent if they define the same language when used as initial configuration of the DPDA. The decision method for the regularity problem is then based on the characterization of the regular languages in terms of the Myhill/Nerode equivalence. For a language $L \subseteq A^*$, the Myhill/Nerode equivalence is defined as follows for words $u, v \in A^*$:

$$u \sim_L v \text{ iff } \forall w \in A^* \ : \ uw \in L \Leftrightarrow vw \in L.$$

A language of finite words is regular if, and only if, it has finitely many Myhill/Nerode equivalence classes, and these classes can be used as states for a canonical finite automaton for the language.

Unfortunately, a corresponding result is not true for ω-regular languages, in general. However, the subclass of weak ω-regular languages possesses a similar characterization in terms of an equivalence [18]. This similarity raises the question whether the decidability results for DPDAs on finite words can be lifted to weak DPDAs on infinite words.

In [14] it is shown that this is indeed possible. In fact, it is even possible to reduce questions for weak ω-DPDAs to DPDAs on finite words. To establish such a connection, we associate a language $L_*(\mathcal{A})$ of finite words to a weak ω-DPDA $\mathcal{A}$, which is obtained by viewing $\mathcal{A}$ as a DPDA on finite words and taking the set of states with an even priority as the set of final states.

The first attempt for reducing the regularity problem for weak ω-DPDAs to the regularity problem for DPDAs would be to test $L_*(\mathcal{A})$ for regularity, where $\mathcal{A}$ is the given weak ω-DPDA. This approach is sound because regularity of $L_*(\mathcal{A})$ implies ω-regularity of $L_\omega(\mathcal{A})$: a finite deterministic automaton for $L_*(\mathcal{A})$ viewed as a Büchi automaton defines $L_\omega(\mathcal{A})$ because it visits final states at the same positions as $\mathcal{A}$.

That the approach is not complete is illustrated by the following simple example.

Example 2. *Consider the alphabet $\{a, b\}$ and the ω-language a^*b^ω of words starting with a finite sequence of a followed by an infinite sequence of b. Obviously, this language is regular. A weak ω-DPDA $\mathcal{A}$ could proceed as follows to accept this language. It starts by pushing a symbol onto the stack for each a. When the first b comes in the input, it changes its state and starts popping the stack symbols again. Once the bottom of the stack is reached, it changes to an accepting state and remains*

there as long as it reads further b (if another a comes, then the input is rejected). Since the finite a-sequence is followed by infinitely many b, it is guaranteed that $\mathcal{A}$ reaches the accepting state if the input is from a^*b^ω. Note that this is a weak ω-DPDA because it can change once from non-accepting to accepting states, and once more back to non-accepting states. The language $L_*(\mathcal{A})$ of this weak ω-DPDA is the set of all finite words of the form $a^m b^n$ with $n \geq m$ because $\mathcal{A}$ reaches the accepting state only after it has read as many b as a. Thus, $L_*(\mathcal{A})$ is non-regular although $L_\omega(\mathcal{A})$ is.

For this example, the problem would be solved if $\mathcal{A}$ switches to an accepting state as soon as the first b is read (instead of deferring this change to the stack bottom). In general, one can show that each weak ω-DPDA $\mathcal{A}$ can be transformed in such a way, that the regularity test for $L_\omega(\mathcal{A})$ reduces to the regularity test for $L_*(\mathcal{A})$, as stated in the following theorem.

Theorem 3 ([14]). *There is a normal form for weak ω-DPDAs with the following properties:*

(1) *For a weak ω-DPDA $\mathcal{A}$ in normal form, the language $L_\omega(\mathcal{A})$ is ω-regular if, and only if, $L_*(\mathcal{A})$ is regular.*
(2) *Given two weak ω-DPDAs $\mathcal{A}$ and $\mathcal{B}$ in normal form, $L_\omega(\mathcal{A}) = L_\omega(\mathcal{B})$ if, and only if, $L_*(\mathcal{A}) = L_*(\mathcal{B})$.*

Furthermore, every weak ω-DPDA can be effectively transformed into an equivalent weak ω-DPDA in normal form.

Combining the first part of Theorem 3 with Theorem 1, we get the decidability of the regularity problem for weak ω-DPDAs.

Corollary 4 ([14]). *The regularity problem for weak ω-DPDAs is decidable.*

The second part of the theorem can be used to show the decidability of the equivalence problem for weak ω-DPDAs, based on the corresponding deep result for DPDAs.

Theorem 5 ([17]). *The equivalence problem for DPDAs is decidable.*

Corollary 6 ([14]). *The equivalence problem for weak ω-DPDAs is decidable.*

The two problems for the full class of ω-DPDAs remain open. In [16] a congruence for ω-languages is identified that characterizes regularity within the class of ω-DPDA recognizable languages (a language accepted by an ω-DPDA is regular if, and only if, this congruence has finitely many equivalence classes). This might be a step towards a solution for the regularity problem. However, the decidability of this characterizing criterion from [16] remains open.

4. Simplifying the Acceptance Condition

This section contains the main contribution of this paper. We consider the problem of simplifying the acceptance condition for different variants of ω-DPDAs with a parity condition. We now formally introduce the three decision problems that are analyzed in the subsequent three parts of this section.

Let $P \subset \mathbb{N}$ be a finite set of priorities. A parity DPDA using only priorities from P is referred to as a P-parity DPDA. Note that one can always assume that P contains a segment of the natural numbers (without gaps) starting in 0 or 1. However, in the statements below we do not impose this restriction.

The parity index problem for parity DPDA is the following decision problem:

Given: A parity DPDA $\mathcal{A}$ and a finite set $P \subset \mathbb{N}$.
Question: Does there exist a P-parity DPDA $\mathcal{A}'$ with $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$?

Similarly, a stair P-parity DVPA is a stair parity DVPA that uses only priorities from P. The stair parity index problem for DVPA is then stated as:

Given: A stair parity DVPA $\mathcal{A}$ and a finite set $P \subset \mathbb{N}$.
Question: Does there exist a stair P-parity DVPA $\mathcal{A}'$ with $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$?

The third decision problem is about converting a stair parity DVPA into a standard parity DVPA:

Given: A stair parity DVPA $\mathcal{A}$.
Question: Does there exist a parity DVPA $\mathcal{A}'$ with $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$?

We show the first two problems to be decidable in Secs. 4.1 and 4.2, respectively, and the special case for a given stair Büchi DVPA (instead of stair parity) of the third problem in Sec. 4.3.

4.1. *The parity index problem*

For finite parity automata, it suffices to change the priority assignment, in order to obtain an equivalent automaton with a smallest set P of priorities, and such a modified priority function can be computed in polynomial time [5].

For parity DPDAs the situation is different, as illustrated by the following example.

Example 7. *Figure* 1 *shows a DVPA* (*taken from* [20]), *where* c_1, c_2 *are calls,* r_1, r_2 *are returns,* i_1, i_2 *are internals, and* Z_1, Z_2 *are stack symbols. The transitions on call symbols are annotated with the stack symbol to be pushed, and for the return symbols with the stack symbol to be popped. The priority function of the DVPA in Fig.* 1 (*indicated as labels of the states*) *is minimal for the state set and the transition structure. The problem is caused by the state* q_1*, which is part of the loop in the upper and the lower branch. However, there is no run of the automaton that traverses both the upper and the lower branch. If the first symbol in the input is*

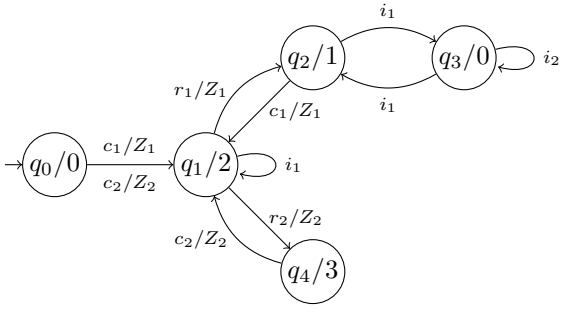


Fig. 1. DVPA with minimal number of priorities for the given transition structure (Example 7).

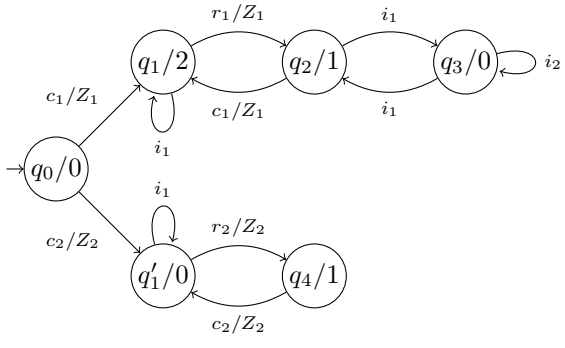


Fig. 2. DVPA equivalent to the one from Fig. 1 using less priorities.

c_1, then the automaton stores Z_1 on the stack. Whenever the automaton reaches q_1 in the future, Z_1 will be on top of the stack and the automaton can only use the top branch. For the lower branch and c_2 as the first input symbol the situation is similar.

Splitting q_1 into two copies as done in the DVPA in Fig. 2*, makes it possible to reassign priorities without using priority* 3*.*

The example illustrates that we need to take a different approach for computing the parity index of pushdown automata. The game based approach that we present below is also described in [16].

To decide whether a given parity DPDA $\mathcal{A}$ has an equivalent P-parity DPDA, consider the following game (for the background on infinite duration games we refer the reader to [9,12]). There are two players, referred to as Automaton and Classifier. The role of Automaton is to play a sequence of input letters and thus to simulate a run of $\mathcal{A}$. Classifier in each rounds chooses one priority from P. The idea is that the classifier wants to prove that there is a P-parity DPDA that accepts $L_\omega(\mathcal{A})$. If Classifier chooses priority k in a move, this can be interpreted as "the parity DPDA that I have in mind would now be in a state with priority k". The winning condition

for Classifier is as follows: the input word played by Automaton is in $L_\omega(\mathcal{A})$ if, and only if, the sequence of priorities played by Classifier satisfies the parity condition.

This game can be formalized as a game $\mathcal{G}_{\mathcal{A},P}$ over a pushdown graph (basically, the configuration graph of $\mathcal{A}$ enriched by the bounded number of choices for Classifier). If $C(\mathcal{A})$ denotes the set of configurations of $\mathcal{A}$, then the set vertices of $\mathcal{G}_{\mathcal{A},P}$ is $C(\mathcal{A}) \cup (C(\mathcal{A}) \times P)$. The edges (or moves) of the game are as follows:

- If a play is in a node $v \in C(\mathcal{A})$, then Classifier can move to the nodes (v, p) for all $p \in P$.
- If a play is in a node $(v, p) \in C(\mathcal{A}) \times P$, then Automaton can move to a node v' such that v' is a successor configuration of v (thus, choosing an input letter determines v' because $\mathcal{A}$ is deterministic).

The winning condition states that an infinite play is won by Classifier if, and only if, the two priority sequences, one induced by the configurations chosen by Automaton, the other given by the choices of Classifier, are either both accepting or both rejecting. We refer to $\mathcal{G}_{\mathcal{A},P}$ as the classification game for $\mathcal{A}$ and P.

Lemma 8. *Classifier has a winning strategy in the classification game for $\mathcal{A}$ and P if, and only if, there is P-parity DPDA accepting $L_\omega(\mathcal{A})$.*

Proof. It suffices to observe the following things. If there is a P-parity DPDA $\mathcal{B}$ accepting $L_\omega(\mathcal{A})$, then Classifier can simulate the run of $\mathcal{B}$ on the inputs played by Automaton, and always choose the priority of the current state of $\mathcal{B}$. This obviously defines a winning strategy because $\mathcal{A}$ and $\mathcal{B}$ accept the same language.

For the other direction, one uses the fact that a winning strategy for Classifier can be implemented by a pushdown automaton that reads the moves of Automaton and outputs the moves of Classifier [25] (see also [8]). Since input letters correspond to moves of Automaton, and moves of Classifier correspond to priorities, this pushdown automaton for the strategy can easily be converted into a P-parity DPDA for $L_\omega(\mathcal{A})$, by turning the function that outputs the moves of Classifier into the priorities of the states. □

Since it is decidable in exponential time which player has a winning strategy in the classification game [25], we obtain an algorithm for solving the parity index problem (the special case of deciding whether an ω-DPDA is equivalent to a Büchi DPDA has been shown to be decidable in [11]).

Theorem 9. *The parity index problem for parity DPDA can be solved in exponential time.*

We leave open the question of whether the problem is hard for exponential time.

4.2. *The stair parity index problem*

We now turn to the stair parity index problem for stair parity DVPAs. In fact, it is possible to use the same game-based approach as in Sec. 4.1 because pushdown games with stair conditions can also be solved algorithmically [13]. However, for stair parity VPAs one can also adapt the much simpler solution for computing the parity index of finite parity automata. Note that in the example from Fig. 1 the "critical" state q_1 can never occur on a step (moving out of q_1 requires to read a return and thus to pop a symbol). Thus, the priority of q_1 is not important in a stair parity acceptance condition. It turns out that this is not a coincidence. The result presented below has been obtained in collaboration with Philipp Stephan, see [20].

Let $\mathcal{A} = (\mathcal{M}, \Omega)$ with $\mathcal{M} = (Q, A, \Gamma, \delta, q_0, \bot)$ and $\Omega : Q \to \mathbb{N}$ be a stair parity DVPA. We define the transformation graph $\mathrm{TG}_{\mathcal{M}}$ of the underlying pushdown machine $\mathcal{M}$ as follows. The vertices are the states of $\mathcal{M}$. There is an edge from q_1 to q_2 if q_1 and q_2 can occur on successive steps in a run of $\mathcal{M}$. The latter condition is equivalent to the existence of a minimally well-matched word w such that there is a finite run of $\mathcal{M}$ from $q_1\bot$ to $q_2\bot$ labelled w.

Lemma 10. *The transformation graph* $\mathrm{TG}_{\mathcal{M}}$ *of* $\mathcal{M}$ *can be computed in polynomial time.*

Proof. An input connecting two successive steps of a run is either an internal symbol or a minimally well-matched word. Therefore, $\mathrm{TG}_{\mathcal{M}}$ can be computed inductively based on the definition of well-matched words from Sec. 2. One starts with the graph containing only the edges for the internal symbols (Condition (1) in the definition of well-matched words). Denote this initial graph by G_0.

Given graph G_i after iteration i, one computes its transitive closure (Condition (2) in the definition of well-matched words), denoted by G_i^+. This can be done efficiently (see [1]). Then one checks whether there are transitions $\delta(q, c) = (q', Z)$ and $\delta(p', r, Z) = p$ for a call c, a return r, and a stack symbol Z, such that $(q', p') \in G_i^+$. In this case we add the edge (q, p) to the graph (Condition (3) in the definition of well-matched words), obtaining graph G_{i+1}. We repeat this procedure until no more edges are added.

Each iteration can be computed in polynomial time and obviously only polynomially many edges can be added. □

We aim at showing that reducing the number of priorities can be done on the graph $\mathrm{TG}_{\mathcal{M}}$. For the formal statements we need the following definitions. We say that $\Omega' : Q \to \mathbb{N}$ is $\mathcal{M}$-equivalent to Ω, if $L_\omega(\mathcal{M}, \Omega) = L_\omega(\mathcal{M}, \Omega')$. Furthermore, Ω and Ω' are called equivalent on $\mathrm{TG}_{\mathcal{M}}$ if for all infinite paths $q_0q_1q_2\cdots$ through $\mathrm{TG}_{\mathcal{M}}$ that start in the initial state q_0, the two priority sequences $\Omega(q_0)\Omega(q_1)\cdots$ and $\Omega'(q_0)\Omega'(q_1)\cdots$ either both satisfy the parity condition, or both do not satisfy the parity condition.

Since the paths through the transformation graph correspond to the possible sequences of states on steps in runs of $\mathcal{M}$, we directly obtain the following remark.

Remark 11. *The two mappings Ω and Ω' are $\mathcal{M}$-equivalent if, and only if, they are equivalent on* $\mathrm{TG}_{\mathcal{M}}$.

The next lemma shows that solving the stair parity index problem for $\mathcal{A}$ boils down to solving the corresponding problem on $\mathrm{TG}_{\mathcal{M}}$. In other words, if there is a stair P-parity DVPA equivalent to $\mathcal{A}$, then there is one using the same pushdown machine $\mathcal{M}$ as $\mathcal{A}$.

Lemma 12. *For a finite set P of priorities, there is a stair P-parity DVPA $\mathcal{B}$ with $L_\omega(\mathcal{B}) = L_\omega(\mathcal{A})$ if, and only if, there is $\Omega' : Q \to P$ that is equivalent to Ω on* $\mathrm{TG}_{\mathcal{M}}$.

Proof. We start with the easy direction of the claim. If there is an $\Omega' : Q \to P$ that is equivalent to Ω on $\mathrm{TG}_{\mathcal{M}}$, then by Remark 11 the stair P-parity DVPA $\mathcal{B} = (\mathcal{M}, P)$ is equivalent to $\mathcal{A}$.

For the other direction, let $\mathcal{B} = (\mathcal{M}_{\mathcal{B}}, \Omega_{\mathcal{B}})$ be a stair P-parity DVPA with $L_\omega(\mathcal{B}) = L_\omega(\mathcal{A})$. We now want to view $\mathrm{TG}_{\mathcal{M}}$ and $\mathrm{TG}_{\mathcal{M}_{\mathcal{B}}}$ as finite deterministic parity automata that accept the same language. For this purpose, we need a suitable input alphabet and a deterministic labelling of the transitions.

For technical reasons, assume that the state sets of $\mathcal{A}$ and $\mathcal{B}$ are disjoint. Note that each edge in a transformation graph correspond to a set of well-matched words: If the edge leads from q_1 to q_2, then this set consists of all words that induce a run from $q_1\bot$ to $q_2\bot$ in the corresponding pushdown machine. Denote this set by $W_{(q_1,q_2)}$. Since the pushdown machines are deterministic, two sets $W_{(q_1,q_2)}$ and $W_{(q_1,q_3)}$ with the same source state but different target states, are disjoint. We basically label the edges by representatives from these sets, viewing them as formal symbols.

Let $w_1, \ldots, w_n$ be different well-matched words such that for each edge (q_1, q_2) (in $\mathrm{TG}_{\mathcal{M}}$ or $\mathrm{TG}_{\mathcal{M}_{\mathcal{B}}}$) there is at least one i such that $w_i \in W_{(q_1,q_2)}$. Consider the alphabet $\Theta = \{a_1, \ldots, a_n\}$ and define the finite parity automaton $\hat{\mathcal{A}} = (Q, \Theta, q_0, \hat{\delta}, \Omega)$ by $\hat{\delta}(q_1, a_i) = q_2$ if $w_i \in W_{(q_1,q_2)}$. If there is no such q_2, then $\hat{\delta}(q_1, a_i)$ is undefined. The finite parity automaton $\hat{\mathcal{B}}$ is defined in the same way.

Because $\mathcal{A}$ and $\mathcal{B}$ accept the same language as stair parity DVPAs, the two finite parity automata $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ accept the same language: For an infinite word $\alpha \in \Theta^\omega$ we obtain an infinite word $\beta \in \Sigma^\omega$ by replacing each a_i by w_i. The runs of $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ on α correspond to the states on the steps in the runs of $\mathcal{A}$ and $\mathcal{B}$ on β, respectively. Hence, α is accepted by $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ if, and only if, β is accepted by $\mathcal{A}$ and $\mathcal{B}$.

We now use results from [5] that admit to compute the minimal number of priorities for a finite deterministic parity automaton just by computing a new priority assignment on the same transition structure: Since there is a finite P-parity automaton equivalent to $\hat{\mathcal{A}}$ (namely $\hat{\mathcal{B}}$), there is also a mapping $\Omega' : Q \to P$ such

that $\hat{\mathcal{A}}$ accepts the same language as the same automaton $(Q, \Theta, q_0, \hat{\delta}, \Omega')$ with Ω replaced by Ω'. This implies that Ω and Ω' are equivalent on $\mathrm{TG}_{\mathcal{M}}$. □

Based on Lemma 12 we can now show how to solve the stair parity index problem.

Theorem 13. *The stair parity index problem for stair parity DVPAs can be solved in polynomial time.*

Proof. In [5] a polynomial time algorithm is given that computes for a given finite deterministic parity automaton the minimal number of priorities required for accepting the language. This is done by computing a new priority assignment on the given parity automaton that uses this minimal number of priorities. Applying this algorithm to $\mathrm{TG}_{\mathcal{M}}$ for the given stair parity DVPA $\mathcal{A} = (\mathcal{M}, \Omega)$ (and any deterministic labelling of the edges) results in a mapping Ω' that uses the least number of priorities while being equivalent to Ω on $\mathrm{TG}_{\mathcal{M}}$. Given Ω', one can easily decide whether there is a mapping $\Omega'' : Q \to P$ that is equivalent to Ω on $\mathrm{TG}_{\mathcal{M}}$. Applying Lemma 12 and Remark 11, we conclude that such an Ω'' exists if, and only if, there is stair P-parity DVPA equivalent to $\mathcal{A}$. □

4.3. *From stair conditions to standard conditions*

In this section, we consider the problem of deciding for a given stair parity DVPA whether there is an equivalent parity DVPA and to construct one if it exists. We show how to decide this problem in general for stair Büchi DVPAs. We comment on the full class of stair parity DVPAs at the end of this section.

Let us start with an example showing that there are stair Büchi DVPAs that can be translated into parity DVPAs but these require a high number of priorities.

Example 14. *Consider the alphabet consisting of $A_c = \{c\}$, $A_i = \{a, b\}$, and $A_r = \{r\}$. For a natural number h, let L_h be the ω-language containing the infinite words α with the following properties:*

- *In each finite prefix of α there are at most h unmatched calls (which means that the stack height never exceeds h).*
- *There is an i such that there are infinitely many prefixes wb of α with precisely i unmatched calls in w, and there are only finitely many prefixes of α that contain less than i unmatched calls (stated informally: there are infinitely many b on the smallest stack level that occurs infinitely often).*

We define a visibly pushdown machine and then provide a stair Büchi condition and a parity condition that both can be used to accept L_h. The machine uses states $\{q_0, q'_0, \ldots, q_h, q'_h\}$. The index of the states indicates how many unmatched calls are still allowed. Thus, q_h is the initial state. We only use one stack symbol that is pushed for every call and popped for every return. We can therefore omit it in the

definition of the transitions. Then the transitions look like those of a finite automaton, and are defined as follows (as usual, undefined transitions can be directed into a rejecting sink state) :

- $\delta(q_i, c) = \delta(q'_i, c) = q_{i-1}$ *for each* $i \in \{1, \ldots, h\}$
- $\delta(q_i, r) = \delta(q'_i, c) = q_{i+1}$ *for each* $i \in \{0, \ldots, h-1\}$
- $\delta(q_i, a) = \delta(q'_i, a) = q_i$
- $\delta(q_i, b) = \delta(q'_i, b) = q'_i$.

On this machine, we can either use the stair Büchi condition with $F = \{q'_0, \ldots, q'_h\}$, *or the parity condition with* $\Omega(q_i) = 2i + 1$ *and* $\Omega(q'_i) = 2i + 2$.

Clearly, both automata accept only input words in which at most h open calls occur. Consider such an input word α *and let i be the lowest stack level that occurs infinitely often. For the stair Büchi condition, only the final states on this level i are relevant. Hence, a word is accepted if on level i infinitely many b occur, as required by* L_h. *Similarly, for the parity condition the most important priorities that occur infinitely often are those on level i, and thus* α *is accepted if on level i infinitely many b occur.*

It is not difficult to see that a lower number of priorities is not sufficient for accepting L_h *with a parity condition. Hence, the example illustrates that the number of priorities of a parity DVPA that is equivalent to a given stair Büchi DVPA cannot be bounded.*

In Sec. 2, we described the language L_{su} of strictly unbounded words over $\langle\{c\}, \emptyset, \{r\}\rangle$, containing all words with an infinite number of unmatched calls. This language can be accepted by a stair Büchi DVPA but not by a parity DVPA [2]. We show that a language L accepted by a stair Büchi DVPA can

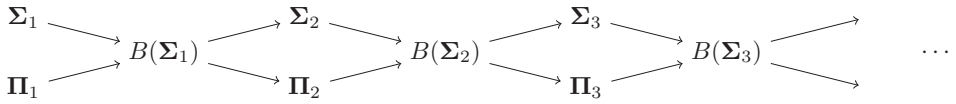
- either be accepted by a parity DVPA, or
- L is at least as complex as L_{su}.

To formalize the notion of "as complex as L_{su}", we need to introduce some terminology and results concerning the topological complexity of ω-languages.

We can view A^ω as a topological space by equipping it with the Cantor topology, where the open sets are those of the form LA^ω for $L \subseteq A^*$. Starting from the open sets one defines the finite Borel hierarchy as a sequence $\mathbf{\Sigma}_1, \mathbf{\Pi}_1, \mathbf{\Sigma}_2, \mathbf{\Pi}_2, \ldots$ of classes of ω-languages as follows (we omit the word "finite" and only refer to this hierarchy as Borel hierarchy in the following):

- $\mathbf{\Sigma}_1$ consists of the open sets.
- $\mathbf{\Pi}_i$ consists of the complements of the languages in $\mathbf{\Sigma}_i$.
- $\mathbf{\Sigma}_{i+1}$ consists of countable unions of languages in $\mathbf{\Pi}_i$.

If we denote by $B(\mathbf{\Sigma}_i)$ the closure of $\mathbf{\Sigma}_i$ under finite Boolean combinations, then we obtain the following relation between the classes of the Borel hierarchy, where an arrow indicates strict inclusion of the corresponding classes:



The above statement of a language L being at least as complex as L_{su} refers to the topological complexity. It is known that languages accepted by deterministic automata (independent of the specific automaton model) with a parity condition are included in $B(\mathbf{\Sigma}_2)$, and in [13] it is shown that languages accepted by stair parity DVPAS are in $B(\mathbf{\Sigma}_3)$. Furthermore, it is known that L_{su} is a true $\mathbf{\Sigma}_3$-set (it is complete for $\mathbf{\Sigma}_3$ for the reduction notion introduced below) [4]. In particular, it is not contained in $B(\mathbf{\Sigma}_2)$.

In our decidability proof we show that specific patterns in a stair parity DVPA induce a high topological complexity of the accepted language (namely being at least as complex as L_{su}). On the other hand, the absence of these patterns allows for the construction of an equivalent parity DVPA.

Before we introduce these patterns, we define the reducibility notion. Originally, it is defined using continuous functions. For our purposes it is easier to work with a different definition based on the Wadge game [24] (see also [4]).

Consider two alphabets A_1, A_2 and let $L_1 \subseteq A_1^\omega$ and $L_2 \subseteq A_2^\omega$. The Wadge game $W(L_1, L_2)$ is played between Players I and II as follows. In each round, Player I plays an element of A_1 and Player II replies with a finite word from A_2^* (the empty word is also possible). In the limit, Player I plays an infinite word x over A_1, and Player II a finite or infinite word y over A_2. Player II wins if y is infinite and $x \in L_1$ iff $y \in L_2$.

We write $L_1 \leq_{\mathrm{W}} L_2$ if Player II has a winning strategy in $W(L_1, L_2)$. The following theorem is a consequence of basic properties of $\leq_{\mathrm{W}}$.

Theorem 15 ([24]). *If $L_1 \leq_{\mathrm{W}} L_2$, then each class of the Borel hierarchy that contains L_2 also contains L_1.*

We use the following consequence of Theorem 15 and the properties of L_{su}.

Lemma 16. *If $L_{\mathrm{su}} \leq_{\mathrm{W}} L$, then L cannot be accepted by a parity DVPA.*

Proof. As mentioned above, the languages that can be accepted by parity DPDAs are contained in $B(\mathbf{\Sigma}_2)$. This is a folklore result for which we do not know a specific reference. We therefore briefly sketch how to prove it. We apply Theorem 15 using the following argument. Let $\mathcal{A}$ be a parity DPDA and let P be the set of priorities used by $\mathcal{A}$. Let $L_P \subseteq P^\omega$ be the sequences of priorities that satisfy the parity condition. Then $L_\omega(\mathcal{A}) \leq_{\mathrm{W}} L_P$ because in the Wadge game Player II can simply keep track of the run of $\mathcal{A}$ on the word played by Player I, and play the corresponding priorities of the states of $\mathcal{A}$. Then clearly the word played by I is in $L_\omega(\mathcal{A})$ iff the priority sequence of II satisfies the parity condition. Now, L_P is easily seen to be a Boolean combination of $\mathbf{\Sigma}_2$-sets (see Sec. 5 of [22]).

Since L_{su} is not contained in $B(\mathbf{\Sigma}_2)$ [4], we conclude from Theorem 15 that $L_{\mathrm{su}} \leq_{\mathrm{W}} L$ implies that L cannot be accepted by a parity DVPA. □

Forbidden patterns.

Fix a stair Büchi DVPA $\mathcal{A} = (Q, A, \Gamma, q_0, \delta, F)$ and let $L = L_\omega(\mathcal{A})$. Recall that L does not contain words with unmatched returns. We assume that all states of $\mathcal{A}$ are reachable.

For an input word u, states q, q', and stack contents σ, σ', we write $(q, \sigma) \xrightarrow{u} (q', \sigma')$ if there is a run for the input u from (q, σ) to (q', σ'). The notation $(q, \sigma) \xrightarrow[F]{u} (q', \sigma')$ means that at least one state from F occurs on a step in this run (for steps to be defined we assume that all prefixes of u are of non-negative stack height). Dual to that we write $(q, \sigma) \xrightarrow[\notin F]{u} (q', \sigma')$ to indicate that no state from F occurs on a step in this run. If we omit the input word u on top of the arrow, then this means that there exists some corresponding input word.

It is not difficult to see that $L_{\mathrm{su}} \leq_{\mathrm{W}} L$ if there are words u and u', a stack content $\sigma \in \Gamma^+$, and a state $q \in Q \setminus F$ such that

$$(q, \bot) \xrightarrow[F]{u} (q, \sigma) \xrightarrow{u'} (q, \bot)$$

and no final state occurs on steps in this run (in a run that starts and ends in the empty stack, the steps are the configurations with empty stack). To prove $L_{\mathrm{su}} \leq_{\mathrm{W}} L$, the corresponding winning strategy for Player II in the Wadge game is: $c \mapsto u$ and $r \mapsto u'$ (meaning that for c played by Player I, Player II responds with u, and for each r played by Player I, Player II responds with u'.

Unfortunately, the above condition is not necessary for $L_{\mathrm{su}} \leq_{\mathrm{W}} L$. Consider the stair Büchi DVPA $\mathcal{A}$ shown in Fig. 3 with one call symbol c and two return symbols r_1, r_2 (the initial state does not matter). In this automaton, the simple pattern described above cannot occur because the only non-final states are q and q'. For these two states, words u and u' as required in the pattern cannot exist for the following reasons:

- The state q can only be reached via calls and therefore $(q, \bot)$ is not reachable from $(q, \bot)$.
- From q' the symbol Z' is pushed onto the stack. But q' can only be reached on popping Z. Therefore $(q', \bot)$ is not reachable from $(q', \bot)$.

However, the example automaton $\mathcal{A}$ contains an extended pattern that guarantees that $L_{\mathrm{su}} \leq_{\mathrm{W}} L_\omega(\mathcal{A})$, as defined below and illustrated in Fig. 4.

Formally, we call $q, q' \in Q \setminus F$, $q'' \in Q$, $u, v, w, x, y, z \in A^*$, and $\sigma, \sigma' \in \Gamma^*$ a forbidden pattern of $\mathcal{A}$ if $uvwxyz \in L_{\mathrm{mwm}}$ and

$$(q, \bot) \xrightarrow[F]{u} (q, \sigma), \quad (q, \bot) \xrightarrow[\notin F]{v} (q', \bot), \ (q', \bot) \xrightarrow[\notin F]{w} (q, \sigma'),$$
$$(q, \bot) \xrightarrow{x} (q'', \bot), \ (q'', \sigma') \xrightarrow{y} (q'', \bot), \ (q'', \sigma) \xrightarrow{z} (q', \bot).$$

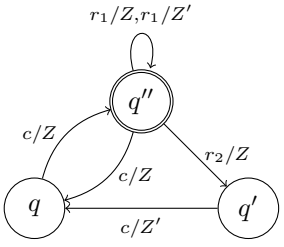


Fig. 3. A stair Büchi DVPA illustrating the definition of forbidden pattern.

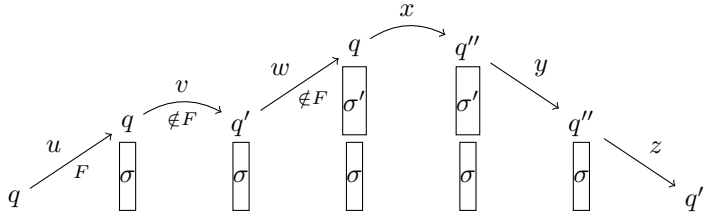


Fig. 4. Forbidden pattern.

Note that σ' might be empty. Since $uvwxyz$ is a minimally well-matched word, q is a non-final state, and a final state is seen on a step on the path $(q, \bot) \xrightarrow[F]{u} (q, \sigma)$, the stack content σ cannot be empty. Further, note that this pattern subsumes the first simple pattern: Given q, σ, and u, u' as in the simple pattern, choose $q' = q'' = q$, $v = w = x = y = \varepsilon$, and $z = u'$.

The example automaton from Fig. 3 contains a forbidden pattern with states q, q', q'', the words $u = cc$, $v = cr_2$, $w = c$, $x = cr_1$, $y = r_1$, $z = r_1 r_2$, and the stack contents $\sigma = ZZ$, $\sigma' = Z'$.

Lemma 17. *If $\mathcal{A}$ has a forbidden pattern, then $L_{\mathrm{su}} \leq_{\mathrm{W}} L_\omega(\mathcal{A})$.*

Proof. We describe a winning strategy f for Player II in the Wadge game. The basic idea is to play u whenever Player I plays c, and to match the last open u with z whenever Player I plays r. However, after playing z, the automaton $\mathcal{A}$ is in state q' (compare Fig. 4). Hence, to play u again, we first have to play w to reach q, producing a σ' on the stack. Therefore, it can happen that we first have remove these σ' from the stack before we can match the last open u with z. To keep track of this, we use words over $\{0, 1\}$ as memory for f representing an abstraction of the stack of $\mathcal{A}$ (0 corresponds to σ and 1 corresponds to σ').

To simplify the description of f, we construct the moves such that $\mathcal{A}$ is always in q' after reading a finite word generated by f. We also assume that q' is the initial state of $\mathcal{A}$. If this is not the case, Player II can simply prepend to the first move a word leading $\mathcal{A}$ to state q'.

Let $\eta \in \{0,1\}^*$ be the current memory content (the initial content being ε) for the strategy f. Then f works as follows:

- If Player I plays c, then play wuv and update the memory to 01η.
- If Player I plays r, then let $i \geq 0$ be such that η is of the form $1^i 0\eta'$. In this case, play $wxyy^iz$ and update the memory to η'.

Let $|\eta|_0$ denote the number of 0 occurring in η and let k be the number of final states seen on steps in the run $(q, \bot) \xrightarrow{u} (q', \sigma)$. Note that $k \geq 1$ by definition of forbidden pattern. By induction one shows that

(1) after each move of Player II the number of open calls in the word played by Player I corresponds to $|\eta|_0$,
(2) the number of final states seen on steps when $\mathcal{A}$ reads a finite word produced by f is $k \cdot |\eta|_0$.

This implies that $\mathcal{A}$ accepts the infinite word produced by Player II according to f iff the infinite word produced by Player I contains an unbounded number of unmatched calls. Thus, f is a winning strategy for II. □

Complexity of state pairs.

We now show that without forbidden patterns in $\mathcal{A}$ it is possible to construct a parity DVPA $\mathcal{A}'$ that is equivalent to $\mathcal{A}$. In order to find an upper bound on the number of required priorities, we start by defining a measure for the complexity of pairs of non-final states. The pair (q, q') from Fig. 4 would be of infinite complexity. If we now replace the states q and q' in the upper part of Fig. 4 by states p and p', as shown in Fig. 5, then this indicates that the possible runs between q and q' are at least as complex as those between p and p'. Since q'' is just an auxiliary state and not of particular importance, we replaced it by p'' to obtain a more consistent naming scheme. We show that this relation indeed defines a strict partial order on pairs of non-final states in the case that $\mathcal{A}$ does not contain forbidden patterns.

For $p, p', q, q' \in Q \setminus F$ define $(p, p') \prec (q, q')$ iff there exists $p'' \in Q$ and stack contents σ, σ' such that (see Fig. 5 for an illustration):

$$(q, \bot) \xrightarrow[F]{u} (p, \sigma), \quad (p, \bot) \xrightarrow[\notin F]{} (p', \bot), (p', \bot) \xrightarrow[\notin F]{} (p, \sigma'),$$
$$(p, \bot) \to (p'', \bot), (p'', \sigma') \to (p'', \bot), (p'', \sigma) \xrightarrow{z} (q', \bot),$$

and $uz \in L_{\text{mwm}}$. The words v, w, x, y from the definition of forbidden pattern are not made explicit in this definition because we never need to refer to them. As for forbidden patterns, σ' might be empty but σ must be non-empty.

Lemma 18. *If $\mathcal{A}$ does not have a forbidden pattern, then $\prec$ is a strict partial order on pairs of states.*

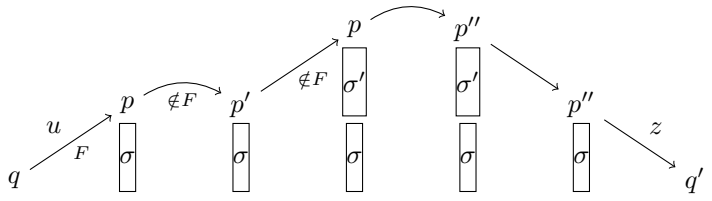


Fig. 5. The relation $(p, p') \prec (q, q')$.

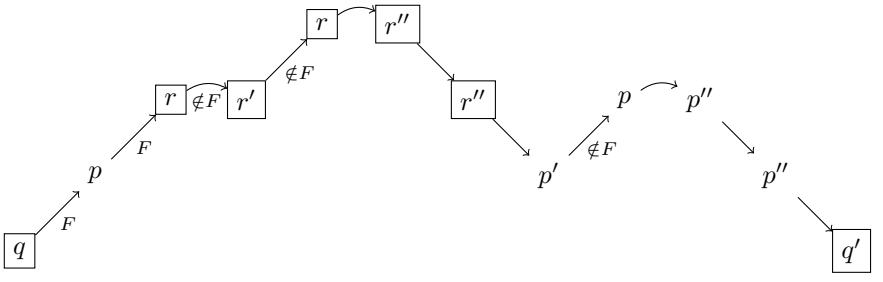


Fig. 6. Transitivity of $\prec$.

Proof. We have to show that $\prec$ is transitive and irreflexive (asymmetry follows from these two). The relation is obviously irreflexive because of the absence of forbidden patterns. Transitivity is illustrated in Fig. 6 for $(r, r') \prec (p, p') \prec (q, q')$ (the stack contents are omitted). The shown pattern is obtained from $(r, r') \prec (p, p')$ and $(p, p') \prec (q, q')$. The configurations with a frame lead to a pattern witnessing $(r, r') \prec (q, q')$. □

For $\mathcal{A}$ without forbidden patterns, we assign to each pair of states a number according to its height in the partial order, i.e., $ht : Q^2 \to \mathbb{N}$ is a mapping satisfying

$$ht(q, q') = \max(\{0\} \cup \{ht(p, p') \mid (p, p') \prec (q, q')\}) + 1.$$

We need the following simple observation.

Lemma 19. *Let $q_1, q_1', q_2, q_2' \in Q \setminus F$. If there is a stack content σ such that $(q_2, \bot) \xrightarrow{u} (q_1, \sigma)$ and $(q_1', \sigma) \xrightarrow{v} (q_2', \bot)$ with $uv \in L_{\mathrm{mwm}}$, then $ht(q_2, q_2') \geq ht(q_1, q_1')$.*

Proof. The condition $(q_2, \bot) \xrightarrow{u} (q_1, \sigma)$ and $(q_1', \sigma) \xrightarrow{v} (q_2', \bot)$ with $uv \in L_{\mathrm{mwm}}$ implies that whenever $(q, q') \prec (q_1, q_1')$, then also $(q, q') \prec (q_2, q_2')$. Thus, $ht(q_2, q_2') \geq ht(q_1, q_1')$ by definition of ht. □

To make use of $\prec$ and ht in the construction of $\mathcal{A}'$ we need the following lemma. Note that this statement does not assume that $\mathcal{A}$ as no forbidden patterns.

Lemma 20. *The relation* $\prec \subseteq (Q \setminus F)^2$ *can be computed in time polynomial in the size of* $\mathcal{A}$.

Proof. In [7] it is shown that for a given configuration $p\sigma$ of $\mathcal{A}$ one can compute in polynomial time the set $pre^*(p\sigma)$ of configurations from which there is a run to $p\sigma$, and the set $post^*(p\sigma)$ of configurations that are reachable from $p\sigma$ by a run. These sets of configurations are sets of words over Γ, starting with a symbol from Q, and can be represented by finite automata.

The algorithms from [7] can be modified to consider only runs that either see a final state on a step or do not see a final state on a step, resulting in the sets $pre^*_F(q\sigma)$, $pre^*_{\notin F}(q\sigma)$, and similarly for *post*.

For checking whether $(p, p') \prec (q, q')$ it is sufficient to check for each p'' if there are runs as required in the definition of $\prec$. This can be done by a suitable combination of the above mentioned algorithms. For example, as stack content σ any σ with $p\sigma \in post^*_F(q\bot)$, and $p''\sigma \in pre^*(q'\bot)$ can be used. Since $post^*_F(q\bot)$ and $pre^*(q'\bot)$ are represented by finite automata, the existence of such a σ can be decided efficiently. Similarly for σ'.

All these computations can be done in polynomial time, and there are only polynomially many combinations of states that have to tested. □

Informal description of the parity DVPA.

In a Büchi stair condition, a final state visited in a run is "erased" (in the sense that it is not considered for acceptance), if it is not on a step. If we construct a parity DVPA, then we cannot erase states like this. Instead, we use the mechanisms of different priorities to simulate erasing a state. Roughly, final states of the stair Büchi automaton are translated into even priorities. If a final state is erased, then this is compensated by visiting a higher odd priority. For the choice of the correct priorities we use the function *ht*.

In the description below, we use the terminology of "$\mathcal{A}$ closing a pair (q, q') of states". This means that $\mathcal{A}$ was in state q at some position and after reading a word from L_{mwm} it reached state q', i.e., $\mathcal{A}$ was in state q before reading a call and reached q' after the matching return.

As mentioned above, we somehow need to determine a priority for the final states that are visited. Assume that the automaton is in configuration (q, β) and reads a word that increases the stack height leading to some configuration $(p, \sigma\beta)$ and visiting some final states on steps during this run. We do not know if these final states remain on steps or will be erased at some point. But if we knew, e.g., that whenever we come back to the stack content β with, say, state q', that the pair (q, q') is of height at least i, then we could signal priority $2i$ for the final states that we have seen after (q, β) and signal priority $2i + 1$ if we indeed close a pair (q, q') on the level of β, and thus erasing all the final states.

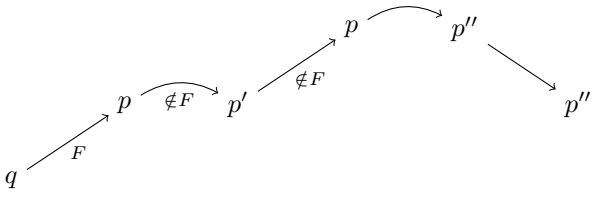


Fig. 7. The pattern for determining the priority of the states with $ht(p, p') = i$.

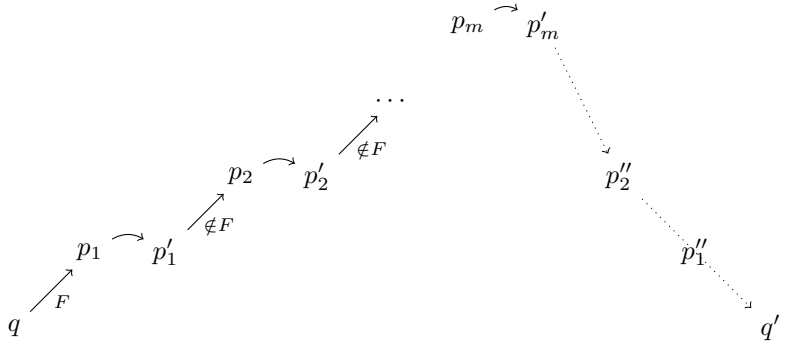


Fig. 8. Detecting that each pair with q is of height at least i.

Assume that we have already seen the pattern shown in Fig. 7, where (p, p') is a pair of height $i-1$. Then $ht(q, q') \geq i$ for every state q' that we could reach when coming back to the stack height of the configuration with q at the beginning of this pattern. In particular, if h is the maximal height of a pair of states, and (p, p') are of height h, then we know that the final states between q and p cannot all be deleted because this would require closing a pair of height $h+1$.

By a simple combinatorial argument, one can see that such a pattern as shown in Fig. 7 must occur if $\mathcal{A}$, before returning to the stack height of q, has successively closed $m := |Q|^3 + 1$ pairs $(p_1.p'_1), \ldots, (p_m.p'_m)$ of height $i-1$ without visiting final states on steps in between, as illustrated in Fig. 8 (in the picture the pairs are closed on increasing stack levels, however, they can also be on the same stack level). More precisely, assume that the pattern from Fig. 8 has occurred up to state p'_m. Now assume that q is closed with a state q' as indicated by the dotted lines. Going down from the level of p_m to the level of q in the picture, $\mathcal{A}$ visits states p''_i the next time it reaches the stack level of (p_i, p'_i) (the intermediate states along the dotted line in the picture). By the choice of m, one triple (p_j, p'_j, p''_j) of states must occur twice. Since we assumed that (p_j, p'_j) is of height $i-1$, this gives rise to a pattern witnessing that $ht(q, q') \geq i$.

To detect such situations, $\mathcal{A}'$ maintains a counter with range from 0 to m for each possible height of state pairs, and roughly behaves as follows:

- Whenever a pair of height i is closed by $\mathcal{A}$, then counter i is increased by one (and for technical reasons counter number 0 is increased whenever $\mathcal{A}$ visits a non-final state after reading a call or an internal symbol). To detect the closed pairs, $\mathcal{A}'$ stores the states of $\mathcal{A}$ on the stack, and the height of state pairs can be computed by Lemma 20.
- There is an additional flag for each $i \in \{0, \ldots, h\}$ indicating whether counter number i was reset because a final state of $\mathcal{A}$ has been visited (the flag is set to 1), or because it reached its maximal value m (the flag is set to 0).
- When counter number i reaches value m (if several counters reach m at the same time we take the maximal such i), then the automaton signals priority $2i + 2$ if the flag number i is set, and $2i + 1$ if the flag is not set. In the next transition the counter is reset.

Formal description of the parity DVPA.

Recall that $m := |Q|^3 + 1$ and that h is the maximal height of a pair of states from $Q \setminus F$.

- The states of $\mathcal{A}'$ are of the form (q, χ, f), where $q \in Q$ is a state of $\mathcal{A}$, $\chi : \{0, \ldots, h\} \to \{0, \ldots, m\}$ represents the counters, and $f : \{0, \ldots, h\} \to \{0, 1\}$ represents the flags mentioned in the informal description.
- The stack symbols of $\mathcal{A}'$ are of the form $[Z, (q, \chi, f)]$, where Z is a stack symbol of $\mathcal{A}$ and (q, χ, f) is a state of $\mathcal{A}'$.
- We now define when $\mathcal{A}'$ can move from state (q, χ, f) to state (q', χ', f'), depending on whether it reads a call, an internal action, or a return. In all cases, q' is the next state of $\mathcal{A}$, i.e., $\mathcal{A}'$ simulates $\mathcal{A}$ in its first component. If $q' \in F$, then $\chi' = 0$ and $f' = 1$, i.e., the constant functions mapping everything to 0 and 1, respectively. The other cases for δ' are listed below. For better readability, we write the state on top of the stack symbol in the specification of call and return transitions.

  Call: $(q, \chi, f) \xrightarrow{c} \begin{matrix}(q', \chi', f') \\ [Z, (q, \chi, f)]\end{matrix}$ if $\delta(q, c) = (Z, q')$, $q' \notin F$, and

$$\chi'(i) = \begin{cases} (\chi(i) \mod m) + 1 \text{ if } i = 0, \\ (\chi(i) \mod m) \text{ otherwise,} \end{cases} \qquad f'(i) = \begin{cases} f(i) \text{ if } \chi(i) < m, \\ 0 \text{ otherwise.} \end{cases}$$

  Internal action: $(q, \chi, f) \xrightarrow{a} (q', \chi', f')$ if $\delta(q, a) = q'$, $q' \notin F$, and χ' and f' are as in the case of a call symbol.

Return: $\begin{matrix}(q,\chi,f)\\ [Z,(q'',\chi'',f'')]\end{matrix} \xrightarrow{r} (q',\chi',f')$ if $\delta(q,Z,r)=q'$, $q'\notin F$, and

$$\chi'(i) = \begin{cases} (\chi''(i) \mod m)+1 \text{ if } q''\notin F \text{ and } i \le ht(q'',q'),\\ (\chi''(i) \mod m) \text{ otherwise,}\end{cases}$$

$$f'(i) = \begin{cases} f''(i) \text{ if } \chi''(i) < m,\\ 0 \text{ otherwise.}\end{cases}$$

- The priority function Ω' of $\mathcal{A}'$ is defined as follows

$$\Omega'(q,\chi,f) = \begin{cases} 0 \text{ if } \chi(i) < m \text{ for all } i,\\ 2d+1+f(d) \text{ if } d=\max\{i \mid \chi(i)=m\}.\end{cases}$$

- The initial state is (q_0,χ_0,f_0) with $\chi_0=0$ and $f_0=1$.

Lemma 21. *The parity DVPA $\mathcal{A}'$ is equivalent to $\mathcal{A}$.*

Proof. We note the following helpful fact on reachable states (q,χ,f) of $\mathcal{A}'$:

(1) If $f(i)=1$ for some i, then $f(j)=1$ and $\chi(i)\ge\chi(j)$ for all $j\ge i$. The initial state satisfies this property, and if we apply the definition of the transition function to a state satisfying the property, then one can easily verify that the resulting state also satisfies it.

Now consider an accepting run of $\mathcal{A}$. We show that the corresponding run of $\mathcal{A}'$ is also accepting. Let the kth state in this run of $\mathcal{A}'$ be (q_k,χ_k,f_k).

If ℓ is a step in the run and q_ℓ is a final state of $\mathcal{A}$, then all flags are set to 1 at this point. From the definition of δ' follows that these flags can only be set to 0 if the corresponding counter reaches value m (we assume that the final state occurs on a step and therefore the run never accesses the stack symbols below). Now assume that $\mathcal{A}'$ signals some odd priority $2i+1$ at some position k after this final state. This means that i is maximal with $\chi_k(i)=m$, and furthermore $f_k(i)=0$. But if $f_k(i)=0$, then there must be some k' with $\ell<k'<k$ such that $f_{k'}(i)=1$ and $\chi_{k'}(i)=m$ because this is the only situation in which the flag is set to 0.

From (1) we conclude that $f_{k'}(j)=1$ for all $j\ge i$ and hence $\Omega'(q_{k'},\chi_{k'},f_{k'})$ is an even priority bigger than $2i+1$. Thus, for each odd priority occurring after a final state on a step there is a bigger even priority also occurring after this final state. Hence, the run of $\mathcal{A}'$ is also accepting.

For the other direction, consider a non-accepting run of $\mathcal{A}$ and as before let (q_k,χ_k,f_k) be the kth state in the corresponding run of $\mathcal{A}'$. There is a position such that after this position no final states of $\mathcal{A}$ occur on a step. From now on we only consider this part of the run.

Consider the sequence $k_1,k_2,k_3,\ldots$ of steps. As no final state occurs on a step, we have the following relation between the counter values at two successive steps:

(i) If k_{j+1} was reached from k_j by reading a call or an internal symbol, then the only change of the counters is $\chi_{k_{j+1}}(0) = (\chi_{k_j}(0) \mod m)+1$. The other values remain the same.

(ii) If k_{j+1} was reached from k_j by reading a minimally well-matched word, then the counters are updated as follows:

$$\chi_{k_{j+1}}(i) = \begin{cases} (\chi_{k_j}(i) \mod m) + 1 \text{ if } i \leq ht(q_{k_j}, q_{k_{j+1}}), \\ (\chi_{k_j}(i) \mod m) \text{ otherwise.} \end{cases}$$

The flags between two successive steps are updated as follows:

$$f_{k_{j+1}}(i) = \begin{cases} f_{k_j}(i) \text{ if } \chi_{k_j}(i) < m, \\ 0 \text{ otherwise.} \end{cases}$$

Now let d be the highest counter that is infinitely often increased on a step (such a counter exists because counter 0 is increased for each call and each internal symbol). Then the highest priority occurring on a step is obviously $2d + 1$ because after the first reset of counter d to 0 the flag number d is 0 on all following steps.

We have to show that no even priority higher than $2d + 1$ can occur infinitely often. Restrict the part of the run under consideration further to the suffix on which no counter higher than d is incremented on a step. We can conclude that for successive steps connected by a minimally well-matched word we have that $ht(q_{k_j}, q_{k_{j+1}}) \leq d$.

We first assume that $d > 0$. At the end of the proof we briefly explain the case $d = 0$.

Pick j such that there is ℓ with $k_j < \ell < k_{j+1}$ and $\Omega'(q_\ell, \chi_\ell, f_\ell) = 2i + 2$ (if no such position exists, then the run of $\mathcal{A}'$ is clearly rejecting). For simplicity let $(q_{k_j}, \chi_{k_j}, f_{k_j}) = (q, \chi, f)$ and $(q_{k_{j+1}}, \chi_{k_{j+1}}, f_{k_{j+1}}) = (q', \chi', f')$.

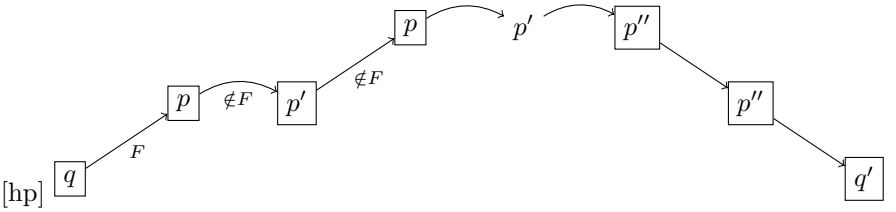
We now consider the part of the run from k_j to ℓ and show that $i < ht(q, q') \leq d$ and hence $2i + 2 < 2d + 1$.

Since $\Omega'(q_\ell, \chi_\ell, f_\ell) = 2i + 2$ we know that $f_\ell(i) = 1$ and i is maximal with $\chi_\ell(i) = m$. If $i = 0$ we know that $i < d$ by our assumption $d > 0$. If $i > 0$, at position ℓ a pair of states of height i is closed. From Lemma 19 we obtain that $d \geq ht(q, q') \geq i$.

There are two cases to consider. If flag number i was already set to 1 at position k_j, i.e., $f(i) = 1$, then $i \neq d$ (as we only consider the part of the run where the flag for d remains 0 forever on the steps). Together with $d \geq i$ we get $d > i$.

If $f(i) = 0$, then it must be reset to 1 by visiting a final state. At the same time the counters are reset to 0. Then m pairs of height i have to be closed to reach the value $\chi_\ell(i) = m$. Furthermore, these pairs have to be closed at positions that correspond to steps in the part of the run between k_j and ℓ (not steps in the whole run). Let these pairs be $(p_1, p'_1), (p_2, p'_2), \ldots, (p_m, p'_m)$ (see Fig. 8) and the corresponding pairs of positions be $(\ell_1, \ell'_1) \ldots, (\ell_m, \ell'_m)$. Now consider for each n the minimal position ℓ''_n with $\ell \leq \ell''_n \leq k_{j+1}$ such that the stack height at ℓ'_n and ℓ''_n is the same. Let p''_n denote the state at the corresponding position (see again Fig. 8).

By the choice of m, there are $n_1 \neq n_2$ such that $(p_{n_1}, p'_{n_1}, p''_{n_1}) = (p_{n_2}, p'_{n_2}, p''_{n_2})$. Denote the corresponding triple by (p, p', p''). This triple witnesses that $ht(q, q') > ht(p, p') = i$ as illustrated in the following picture (the framed positions define the pattern from Fig. 5).



It remains to consider the case $d = 0$. Consider only the suffix of the run after the position where the flag for counter 0 remains 0 on all steps and no other counter is increased on a step anymore. Then all pairs closed on steps are of height 0 and by Lemma 19, pairs closed between two successive steps are also of height 0. So the maximal priority that we can see on this part of the run would be 2. For this to happen, the flag for counter 0 must be 1 and counter 0 must have value m. The flags are only set to 1 if a final state of $\mathcal{A}$ is reached, and at the same time the counters are set to 0. Let q, q' be the states at two successive steps, and assume that in between a final state is seen. Let p be the state after the symbol following the final state. If this symbol is a call or an internal, then $(p, p) \prec (q, q')$ (choosing $p'' = p$), contradicting $ht(q, q') = 0$. Thus, each final state of $\mathcal{A}$ is immediately followed by a return. Thus, whenever the flag is set to 1 by a final state, it is immediately reset to 0 in the next transition, and thus priority 2 never occurs (on the considered part of the run). □

Combining Lemmas 17 and 21 we obtain the following.

Theorem 22. *A stair Büchi DVPA $\mathcal{A}$ is equivalent to a parity DVPA if, and only if, it does not contain any forbidden patterns.*

The relation $\prec$ can be computed and checked for irreflexivity in polynomial time (Lemma 20). Hence we get the following corollary.

Corollary 23. *For a stair Büchi DVPA $\mathcal{A}$ it is decidable in polynomial time if it is equivalent to some parity DVPA.*

A direct consequence of Lemma 21 is:

Theorem 24. *If a stair Büchi DVPA $\mathcal{A}$ is equivalent to some parity DVPA, then we can effectively construct such a parity DVPA.*

Using more complex forbidden patterns, it might be possible to lift the methods presented in this section to decide for general stair parity DVPAs whether the stair condition is required. We leave this open for future work.

A simpler question can be solved using the game theoretic approach from Sec. 4.1: Given a stair parity DVPA $\mathcal{A}$ and a set P of priorities, we can decide whether there is a parity DVPA using the priorities from P that accepts $L_\omega(\mathcal{A})$ by using the classification game. In this case, the classification game could be formalized using a combination of a classical parity and a stair parity condition. Pushdown games with such a winning condition can be solved with the methods from [13].

5. Conclusion

We have considered several decidability questions for ω-DPDAs. The regularity and equivalence problem are still open for the full class of ω-DPDAs. We have sketched some partial results from [14] showing the decidability for these two problems for the class of weak ω-DPDAs by a reduction to the corresponding problems for DPDAs on finite words. It seems that a decidability result for the full class of ω-DPDAs requires new ideas.

In the second part, we have analyzed the problem of simplifying the acceptance condition of ω-DPDAs. We have shown that the smallest number of priorities required for accepting the language of a given parity DPDA can be computed. For the standard parity condition we have used a game approach leading to an algorithm running in exponential time. Whether a matching lower bound exists, remains open. For stair parity DVPAs, the stair parity index problem can be solved by a polynomial time algorithm that uses a reduction to the computation of the parity index of a finite automaton.

We have also shown that for stair Büchi DVPAs it is decidable whether the stair condition is required or whether there exists an equivalent parity DVPA. It seems that the methods used in the proof can be generalized from stair Büchi conditions to arbitrary stair parity conditions. However, we have not yet solved the problem and leave it for future work. Besides the decision problems, it would also be interesting to understand the relations between the levels of the parity index hierarchy and the stair parity index hierarchy for DVPAs.

References

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

[2] R. Alur and P. Madhusudan, Visibly pushdown languages, *STOC '*04: *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing* (ACM Press, New York, USA, 2004), pp. 202–211.

[3] C. Baier and J.-P. Katoen, *Principles of Model Checking* (MIT Press, 2008).

[4] T. Cachat, J. Duparc and W. Thomas, Solving pushdown games with a Σ_3 winning condition, *Proceedings of the* 11*th Annual Conference of the European Association for Computer Science Logic, CSL* 2002, *Lecture Notes in Computer Science* **2471** (Springer, 2002), pp. 322–336.

[5] O. Carton and R. Maceiras, Computing the Rabin index of a parity automaton, *ITA* **33**(6) (1999) 495–506.

[6] R. S. Cohen and A. Y. Gold, Omega-computations on deterministic pushdown machines, *JCSS* **16**(3) (1978) 275–300.
[7] J. Esparza, D. Hansel, P. Rossmanith and S. Schwoon, Efficient algorithms for model checking pushdown systems, *CAV* (2000), pp. 232–247.
[8] W. Fridman, Formats of winning strategies for six types of pushdown games, *Proceedings of the First Symposium on Games, Automata, Logic, and Formal Verification, GandALF* 2010, eds. A. Montanari, M. Napoli and M. Parente **25** (Electronic Proceedings in Theoretical Computer Science, 2010), pp. 132–145.
[9] E. Grädel, W. Thomas and T. Wilke (eds.), *Automata, Logics, and Infinite Games: A Guide to Current Research* [*outcome of a Dagstuhl seminar, February* 2001] *Lecture Notes in Computer Science* **2500** (Springer, 2002).
[10] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison Wesley, 1979).
[11] M. Linna, A decidability result for deterministic *omega*-context-free languages, *Theor. Comput. Sci.* **4**(1) (1977) 83–98.
[12] C. Löding, Infinite games and automata theory, *Lectures in Game Theory for Computer Scientists*, eds. K. R. Apt and E. Grädel (Cambridge University Press, 2011).
[13] C. Löding, P. Madhusudan and O. Serre, Visibly pushdown games, *FSTTCS* 2004, *Lecture Notes in Computer Science* **3328** (Springer, 2004), pp. 408–420.
[14] C. Löding and S. Repke, Regularity problems for weak pushdown ω-automata and games, *Mathematical Foundations of Computer Science* 2012, *Lecture Notes in Computer Science* **7464** (Springer Berlin/Heidelberg, 2012), pp. 764–776.
[15] D. Perrin and J.-É. Pin, *Infinite words*, Pure and Applied Mathematics,, Vol. 141 (Elsevier, 2004).
[16] S. Repke, Simplification problems for automata and games, PhD thesis, RWTH Aachen, Germany (2014).
[17] G. Sénizergues, L(A)=L(B)? decidability results from complete formal systems, *Theor. Comput. Sci.* **251**(1–2) (2001) 1–166.
[18] L. Staiger, Finite-state ω-languages, *JCSS* **27**(3) (1983) 434–448.
[19] R. E. Stearns, A regularity test for pushdown machines, *Information and Control* **11**(3) (1967) 323–340.
[20] P. Stephan, Deterministic visibly pushdown automata over infinite words, diploma thesis, RWTH Aachen (2006).
[21] H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity* (Birkhäuser, Basel, Switzerland, 1994).
[22] W. Thomas, Automata on infinite objects, *Handbook of Theoretical Computer Science*, **B: Formal Models and Semantics** (Elsevier Science Publishers, Amsterdam, 1990), pp. 133–192.
[23] L. G. Valiant, Regularity and related problems for deterministic pushdown automata, *J. ACM* **22**(1) (1975) 1–10.
[24] W. W. Wadge, Reducibility and determinateness on the baire space, PhD thesis, University of California, Berkeley (1984).
[25] I. Walukiewicz, Pushdown processes: Games and model checking, *Information and Computation* **164**(2) (2001) 234–263.