Regularity Problems for Weak Pushdown ω-Automata and Games

Extended Version (2012-09-03)

Christof Löding and Stefan Repke*

Lehrstuhl für Informatik 7, RWTH Aachen, Germany
{loeding, repke}@automata.rwth-aachen.de

Abstract We show that the regularity and equivalence problems are decidable for deterministic weak pushdown ω-automata, giving a partial answer to a question raised by Cohen and Gold in 1978. We prove the decidability by a reduction to the corresponding problems for deterministic pushdown automata on finite words. Furthermore, we consider the problem of deciding for pushdown games whether a winning strategy exists that can be implemented by a finite automaton. We show that this problem is already undecidable for games defined by one-counter automata or visibly pushdown automata with a safety condition.

1 Introduction

Finite automaton and pushdown automaton are two of the most fundamental automaton models in computer science. Finite automata have good closure and algorithmic properties. For example, language equivalence and inclusion are decidable (see [9]), and for many subclasses of the regular languages it is decidable whether a given automaton accepts a language inside this subclass (see [17] 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 [9]), and it is also 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 [14] shows the decidability of the equivalence problem. Furthermore, the regularity problem for deterministic pushdown automata is also decidable [16, 18].

While automata on finite words are a very useful tool, some applications, in particular verification by model checking (see [2]), 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

* Supported by the DFG Research Training Group 1298 "Algorithmic Synthesis of Reactive and Discrete-Continuous Systems" (AlgoSyn).

because of the more complex acceptance conditions, many of the good properties of finite automata on finite words are preserved (see [11] 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.

Our first contribution addresses these questions. We prove the decidability of the two problems (regularity and equivalence) for the subclass of deterministic pushdown ω-automata with weak acceptance condition. Intuitively, an automaton with weak acceptance condition only allows a bounded number of alternations between accepting and rejecting states. This class of automata is capable of expressing boolean combinations of reachability and safety conditions. Our proof is based on a reduction to the corresponding questions for pushdown automata on finite words in the spirit of the minimization algorithm for deterministic weak ω-automata in [10].

We continue our investigations by considering the regularity problem in the extended setting of pushdown games. A pushdown game is given by a pushdown ω-automaton and a partition of the state space into states for Player 0 and Player 1. In a play, the two players build up an infinite sequence of configurations. The partition of the state space determines which player chooses the next successor configuration. In case this automaton is deterministic, the sequences of configurations and sequences of input letters are in one-to-one correspondence, and Player 0 wins if the infinite input word is accepted by the automaton. In this setting, a strategy for a player is a function that tells the player for a given finite sequence of input letters (corresponding to the previous moves of the play) which input letter to choose next (thereby determining the next configuration). It is a winning strategy if each play in which the player follows the strategy is winning for this player. For pushdown games with winning conditions like Büchi condition, or more generally Muller or parity conditions, it is decidable which of the players has a winning strategy, and it is known that such a strategy can be computed by a pushdown automaton with output function reading the letters of the play and outputting the next letter according to the strategy [19].

The regularity problem for pushdown games asks whether the player who has a winning strategy also has one that can be computed by a finite automaton. For example, the question studied in [13, 12], that asks whether for a given document type definition (DTD) one can decide if it is possible to validate streaming XML documents against this DTD with constant memory (by a finite automaton), can be expressed as a regularity problem for a pushdown game with a safety winning condition.

We show that the regularity problem for pushdown games is already undecidable in very simple cases, namely for one-counter automata (pushdown automata with a single stack symbol), and visibly pushdown automata (in which the type of the stack operation is determined by the input letter [1]), both with safety winning conditions. While this result does not transfer back to the constant memory validation question for DTDs, it shows that the latter problem cannot be solved by this more general approach.

The remainder of the paper is structured as follows. In Section 2, we give basic definitions on automata and games. In Section 3, we show the decidability of the regularity and equivalence problem for deterministic weak pushdown ω-automata by a reduction to automata on finite words, and in Section 4 we present our results on the regularity problem for pushdown games.

2 Preliminaries

The set of non-negative integers is $\mathbb{N} := \{0, 1, \ldots\}$. For a set S, we denote its cardinality by $|S|$. Let Σ be an alphabet, i.e., a finite set of symbols, then Σ^* (Σ^ω) is the set of *(ω-)words* over Σ, i.e., finite (countably infinite) sequences of Σ symbols. The subsets of Σ^* (Σ^ω) are called *(ω-)languages*. For a word $w = a_1 \cdots a_n \in \Sigma^*$, we define $|w| = n \in \mathbb{N}$ as its length and $w^R = a_n \cdots a_1 \in \Sigma^*$ as its *reversal*. The empty word ε is the word of length $|\varepsilon| = 0$. We assume the reader to be familiar with regular languages, i.e., the languages specified by regular expressions or equivalently by finite state automata. We are mainly concerned with deterministic pushdown automata in this work.

Definition 1. *A* deterministic pushdown machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ *consists of*

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

The set of *configurations* of $\mathcal{M}$ is $Q\Gamma^*\bot$ where $q_0\bot$ is the *initial* configuration. For a given input (ω-)word $w \in \Sigma^*$ ($w \in \Sigma^\omega$), a finite (infinite) sequence $q_0W_0, q_1W_1, \ldots$ of configurations with $q_0W_0 = q_0\bot$ is a *run* of w on $\mathcal{M}$ if there are $a_i \in \Sigma_\varepsilon$ with $w = a_1a_2\cdots$ and $\delta(q_i, A, a_{i+1}) = (q_{i+1}, U)$ is such that $W_i = AV$ and $W_{i+1} = UV$ for some stack suffix $V \in \Gamma_\bot^*$.

If the size $|\Gamma|$ of the stack alphabet is 1, then $\mathcal{M}$ is called a *one counter machine*. If the size $|\Gamma|$ of the stack alphabet is 0, then $\mathcal{M}$ is called a *finite state machine*, and we omit the components related to the stack from the notation of the machine and the transitions.

Automata and Languages. For finite words, we consider the model of a *deterministic pushdown automaton (DPDA)* $\mathcal{A} = (\mathcal{M}, F)$ consisting of a deterministic pushdown machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ and a set of final states $F \subseteq Q$. It *accepts* a word $w \in \Sigma^*$ if w induces a run ending in a final state. These words form the language $L_*(\mathcal{A}) \subseteq \Sigma^*$. For ω-words, we use the model of a *deterministic parity pushdown automaton (ω-DPDA)* $\mathcal{A} = (\mathcal{M}, \tau)$ consisting of a deterministic pushdown machine $\mathcal{M}$ as above and a function $\tau : Q \to \mathbb{N}$ assigning *colors* to the states of $\mathcal{M}$. An infinite word $\alpha \in \Sigma^\omega$ is *accepted* if it induces an infinite run such that the lowest color of the states occurring infinitely often is even. The accepted ω-words form the ω-language $L_\omega(\mathcal{A}) \subseteq \Sigma^\omega$. We call an ω-DPDA *weak* if colors never increase during a run. When restricting the color set of a (weak) ω-DPDA to $\{0, 1\}$ or $\{1, 2\}$, we end up with *Büchi (reachability)* and *coBüchi (safety)* acceptance, respectively.

To ensure that an infinite run reads an infinite word, we require that $\mathcal{A}$ has no infinite sequence of ε-transitions. The presence of such sequences can be tested, and they can be removed in polynomial time by redirecting some of the ε-transitions into sink states. Under this assumption, for a finite word w, we define $\delta_*(w)$ to be the last state q_n of a run $q_0 W_0, \ldots, q_n W_n$ on w such that there is no further ε-transition possible.

All restrictions in the type of the underlying pushdown machine carry over to the automata. Finite state (ω-)automata are denoted by (ω-)DFA. As usual, an (ω-)language is called regular if it can be accepted by a (ω-)DFA.

Games and Strategies. A *pushdown game (PDG)* $\mathcal{G} = (\mathcal{M}, \tau, Q_0)$ consists of an ω-DPDA $(\mathcal{M}, \tau)$ and a set $Q_0 \subseteq Q$. A *play* is an ω-word $\alpha = a_1 a_2 \cdots \in \Sigma^\omega$ successively build up by two players. After the prefix $a_1 \cdots a_i$, the next action $a_{i+1} \in \Sigma$ is chosen by Player 0 if $\delta_*(a_1 \cdots a_i) \in Q_0$, otherwise Player 1 chooses (since $\mathcal{M}$ is total). Player 0 wins the play α iff it is accepted by the ω-DPDA $(\mathcal{M}, \tau)$. This can be considered as the game with the (possibly infinite) configuration graph of $\mathcal{M}$ as arena. A *strategy* is a function $f : \Sigma^* \to \Sigma$ advising to choose action $f(w)$ after a finite play prefix $w \in \Sigma^*$. We call it *winning* for a Player if he wins a play as long as he obeys to f no matter what his opponent does. A game can be *won* by a player if he has a winning strategy. We are especially interested in winning strategies representable by automata. A *pushdown strategy (PDS)* $\mathcal{F} = (\mathcal{M}', \sigma)$ consists of a deterministic pushdown machine $\mathcal{M}'$ and a function $\sigma : Q' \to \Sigma$ advising actions according to the states of $\mathcal{M}'$. It defines the strategy $f(w) = \sigma\big(\delta_*(w)\big)$.

Again, the definitions carry over for the restricted classes of finite state and one counter machines. Finite state games and one counter games are denoted by FSG and 1CG. In general, the winning player of a PDG has a winning pushdown strategy [19], and the winner of an FSG has a winning finite state strategy (FSS) [3].

A pushdown game that is useful in several places in this work is the so called classification game associated to an ω-DPDA $\mathcal{A}$ and a set C of colors. The idea in this game is that one player plays an infinite input word and the other player plays an infinite sequence of colors from C. The configurations of the game

mimic the run of $\mathcal{A}$ on the input word played. The player who is in charge of the colors wins with an accepting color sequence iff the ω-DPDA accepts the input word played. Since we are interested in weak automata, we consider the weak classification game in which the player in charge of the colors is not allowed to increase the colors during a play.

Definition 2. *Let $C \subseteq \mathbb{N}$ be finite and $\mathcal{A} = (\mathcal{M}, \tau)$ with $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ be an ω-DPDA. We define the* weak classification game $\mathcal{G}_{\mathcal{A},C} = (\mathcal{M}', \tau', Q_0')$ *with $\mathcal{M}' = (Q', \Sigma', \Gamma', \delta', q_0', \bot')$ as follows:*[1]

- $Q' = Q \times C \times \{0,1\}$, $q_0' = \big(q_0, \max(C), 0\big)$, *and* $(q,i,x) \in Q_0'$ *iff* $x = 0$,
- $\Sigma' = \Sigma \uplus C$, $\Gamma' = \Gamma$, $\bot' = \bot$,
- $\delta'\big((q,i,0),A,j\big) = \big((q,j,1),A\big)$, *where* $j \in \{0,\dots,i\}$,
  $\delta'\big((q,i,1),A,\varepsilon\big) = \big((p,i,1),W\big)$, *if* $\delta(q,A,\varepsilon) = (p,W)$, *and otherwise*
  $\delta'\big((q,i,1),A,a\big) = \big((p,i,0),W\big)$, *if* $\delta(q,A,a) = (p,W)$ *for* $a \in \Sigma$,
- $\tau'\big((q,i,x)\big) = \big(\tau(q) + i\big)$

Note that the winning condition is a parity condition in general which is weak iff $\mathcal{A}$ is weak. Since the second component in the vertices representing the color from C can never increase, it remains fixed to some i from some point on in every play. Then the original parity condition of $\mathcal{A}$ is evaluated shifted by this fixed index i. If i is even, then Player 0 wins if the input word is accepted by $\mathcal{A}$. If i is odd, then Player 0 wins if the input word is rejected by $\mathcal{A}$. Thus, the acceptance status of the color sequence played by Player 0 has to correspond to the acceptance status of the played input word. Therefore, if Player 0 has a winning strategy in the game that can be implemented by some kind of machine (pushdown or finite state), then this strategy can directly be transformed into a weak automaton for $L_\omega(\mathcal{A})$ by using the color output of the strategy as colors for the acceptance condition.

A first application of this game is the observation that the computation power of the automaton model (pushdown or finite state) and the expressiveness of the acceptance condition are in some sense orthogonal. This statement can also be derived from the proof of Theorem 24 in [15].

Lemma 3. *Let $\mathcal{A}$ be a weak ω-DPDA such that $L_\omega(\mathcal{A})$ is regular. Then there exists a weak ω-DFA $\mathcal{A}'$ with the same colors as $\mathcal{A}$ such that $L_\omega(\mathcal{A}') = L_\omega(\mathcal{A})$.*

Proof. Let $\mathcal{A}$ and $L = L_\omega(\mathcal{A})$ be as above, and $\mathcal{A}''$ be an ω-DFA with $L_\omega(\mathcal{A}'') = L$. Consider the weak classification game $\mathcal{G}_{\mathcal{A}'',C}$ with $C = \tau(Q)$ being the colors of $\mathcal{A}$. Player 0 can easily win $\mathcal{G}_{\mathcal{A}'',C}$ by a pushdown strategy simulating $\mathcal{A}$, i.e., always playing the colors according to $\mathcal{A}$. Since $\mathcal{A}''$ is a finite automaton, $G_{\mathcal{A}'',C}$ is a finite state game, and Player 0 also has a finite state strategy [3] which can be used as a weak ω-DFA $\mathcal{A}'$ with colors C such that $L_\omega(\mathcal{A}') = L$. □

[1] For readability, we dropped the definition of a positive and negative sink state for cases in which a wrong action is played, and the corresponding player has to lose.

3 Regularity and Equivalence Testing for Weak ω-DPDA

In this section, we show how to decide the regularity and the equivalence problems for weak ω-DPDAs. The equivalence problem asks for two given weak ω-DPDAs whether they accept the same language, and the regularity problem asks for a given weak ω-DPDA whether it accepts a regular ω-language. Both problems are decidable in the case of DPDAs over finite words ([14] for equivalence and [16, 18] for regularity). We show that the problems for weak ω-DPDAs can be reduced to the case of finite words. Our technique is inspired by a normal form of weak ω-DFA which assigns minimal colors to each state [10]. Then certain decision problems for ω-DFA can be reduced to DFA on finite words. The same approach for weak ω-DPDA faces the difficulty that minimal colors also depend on the stack content. We overcome this by considering minimal colors for configurations which yields a regular coloring. This also allows us to normalize weak ω-DPDA such that we can apply known algorithms for finite words.

For the remainder of this section, let $\mathcal{A} = (\mathcal{M}, \tau)$ be a weak ω-DPDA with $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ and greatest color $k = \max\big(\tau(Q)\big)$. W.l.o.g., we assume $k \geq 1$. The first important step is to define a language of finite words from the ω-language of $\mathcal{A}$ by simply setting states with an accepting (even) color as final.

Definition 4. *The* finitary language $L_*(\mathcal{A}) \subseteq \Sigma^*$ *of* $\mathcal{A}$ *is the language of finite words accepted by the DPDA* $(\mathcal{M}, F)$ *with* $F = \big\{q \in Q \,\big|\, \tau(q) \text{ is even}\big\}$.

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

Remark 5. If $L_*(\mathcal{A}_{qW}) = L_*(\mathcal{A}_{pV})$, then $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$.

If the inverse 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. Unfortunately, this is not true in general, as illustrated by the following example.

Example 6. For $n \in \mathbb{N}$, consider the language $L_n \subseteq \Sigma^\omega$ over alphabet $\Sigma = \{a, b, c\}$ defined as follows:

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

Obviously, L_n is a regular ω-language since it is defined by a regular expression. Every ω-DFA recognizing L_n needs a state set of size at least exponential in n because before reading the first c it has to remember the last $n+1$ symbols. A weak ω-DPDA with a state set of size linear in n and a stack alphabet of constant size can recognize L_n by writing the string $w \in \{a, b\}$ that occurs before the first c onto the stack (using state q_0). Afterwards, one can check the reversal w^{R} of w with linearly many states by counting to $n+1$ (using the states $q_1, \ldots, q_{n+1}$), then moving to q_a or q_b depending on the letter on the stack, and comparing this letter with the bottom letter on the stack (moving to $q_\top$ or $q_\bot$ to indicate the

result of the comparison). To identify the bottom letter, it is stored as $\$_a$ or $\$_b$ while the other letters are stored as $\#_a$ or $\#_b$ on the stack.

The full definition of the weak ω-DPDA $\mathcal{A}_n = \big((Q, \Sigma, \Gamma, \delta, q_0, \bot), \tau\big)$ is: $Q = \{q_0, \dots, q_{n+1}, q_a, q_b, q_\top, q_\bot\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{\$_a, \$_b, \#_a, \#_b\}$, and transitions as follows (where $A \in \Gamma_\bot$, $x, y \in \{a, b\}$, $z \in \Sigma$, $i \in \{1, \dots, n\}$):

- $\delta(q_0, A, x) = \begin{cases} (q_0, \$_x A) & \text{if } A = \bot, \\ (q_0, \#_x A) & \text{if } A \neq \bot, \end{cases}$
- $\delta(q_0, A, c) = (q_1, A)$, $\delta(q_i, \#_x, z) = (q_{i+1}, \varepsilon)$, $\delta(q_{n+1}, \#_x, z) = (q_x, \varepsilon)$,
- $\delta(q_x, \#_y, z) = (q_x, \varepsilon)$, $\delta(q_x, \$_y, z) = \begin{cases} (q_\top, \varepsilon) & \text{if } x = y, \\ (q_\bot, \varepsilon) & \text{if } x \neq y, \end{cases}$
- $\delta(q_\top, A, z) = (q_\top, A)$, and for all other transitions: $\delta(q, A, z) = (q_\bot, A)$,

and reachability coloring $\tau : Q \to \{0, 1\}$ (where $i \in \{1, \dots, n+1\}$): $\tau(q_\top) = 0$ and $\tau(q_0) = \tau(q_i) = \tau(q_a) = \tau(q_b) = \tau(q_\bot) = 1$.

Then we have $L_\omega(\mathcal{A}_n) = L_n$. To see that the inverse of Remark 5 does not hold, note that in configurations with a state from $\{q_1, \dots, q_{n+1}, q_a, q_b\}$, it is already clear whether the remaining word is accepted or not (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 but the set of finite words accepted depends on the height of the stack and is thus different if W and V are of different length.

Furthermore, $L_*(\mathcal{A}_n)$ is not regular because a finite word is only accepted if it reaches $q_\top$, which requires as many letters after the first c as before the first c:

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

Our next goal is to normalize $\mathcal{A}$ such that the inverse of Remark 5 becomes true. Therefore, we redefine the coloring of configurations (not simply states) based on the sets K_i defined below. The intuition behind the definition is that each configuration should be assigned the lowest color possible.

Definition 7. *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 staying in K_i forever is accepting iff i is even, and
b) each run leaving K_i leaves it to $\bigcup_{j<i} K_j$.

Example 8. The classes K_i for $\mathcal{A}_n$ from Example 6 are (where $\# = \{\#_a, \#_b\}$):

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

$$K_1 \;=\; (Q\Gamma^*\bot) \setminus K_0, \quad \text{for } i \geq 2.$$

Note that K_0 are exactly the configurations from where every word is accepted. All other configurations belong to K_1, like the ones with the bottom state ($q_\bot \Gamma^* \bot \subseteq K_1$) but also the initial state ($q_0 \Gamma^* \bot \subseteq K_1$). Further, some states occur in both K_0 and K_1, 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 $q_1 \#_a \#^n \$_b \bot \in K_1$.

Definition 9. *$\mathcal{A}$ is in* normal form *if colors correspond to classes, i.e., $qW \in K_{\tau(q)}$ for all $qW \in Q\Gamma^*\bot$ that are reachable from the initial configuration.*

Example 8 shows that $\mathcal{A}_n$ is not in normal form. However, the classes K_0 and K_1 are regular sets of words. This is true in general and can be used to transform $\mathcal{A}$ into normal form.

Lemma 10. *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 $L_*(\mathcal{A}'') = L_*(\mathcal{A}')$, where $\mathcal{A}''$ has $\mathcal{O}(|Q|)$ states and $|\Gamma|\cdot 2^{\mathcal{O}(|Q|\cdot k)}$ stack symbols.*

Proof (Sketch). The core of the proof consists in constructing a deterministic finite state machine that reads a reversed configuration and assigns the corresponding class K_i to it. This can be achieved by considering the weak classification game $G_{\mathcal{A},C}$ where C is the set of colors of $\mathcal{A}$. Obviously, Player 0 has a winning strategy (by simply playing the colors from the unique run on the input word played by Player 1). From the definition of the K_i one can deduce that the smallest i such that Player 0 wins from a position $(qW, i, 0)$ is such that $qW \in K_i$. The set of configurations from which Player 0 has a winning strategy is regular and can be accepted by an alternating automaton with a number of states linear in the number of states of the pushdown machine defining $G_{\mathcal{A},C}$ [4, 8]. This alternating automaton can be turned into a DFA of exponential size reading the reversed configurations [5]. This DFA can then be simulated on the configurations along a run of $\mathcal{A}$ by storing its states on the stack in parallel to the actual stack symbols, which yields the normalized ω-DPDA $\mathcal{A}'$. Therefore, the blowup of the stack alphabet is exponential. The states of $\mathcal{A}'$ only contain the additional information on the class K_i the configuration is in. Finally, for the DPDA $\mathcal{A}''$, this information is reduced to whether the class is even or odd, resulting in $2\cdot|Q|$ many states. Each step of the computation can be done in exponential time. □

The example illustrates that the deterministic automaton reading the configurations in reverse has to be of exponential size in general. For weak ω-DPDA in normal form, the inverse of Remark 5 is true.

Lemma 11. *Let $\mathcal{A}$ be in normal form. If $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$, then $L_*(\mathcal{A}_{qW}) = L_*(\mathcal{A}_{pV})$.*

Proof (Sketch). Using basic properties on the sets K_i, a simple argument shows that if $\mathcal{A}$ is in normal form, then two configurations qW and pV with $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$ must have the same color, i.e., $\tau(q) = \tau(p)$. From this, the statement of the lemma easily follows. □

Remark 5 and Lemma 11 can be summarized as follows when considering the special case of initial configurations of two weak ω-DPDA.

Corollary 12. *Let $\mathcal{A}$ and $\mathcal{A}'$ be weak ω-DPDA in normal form. $L_*(\mathcal{A}) = L_*(\mathcal{A}')$ iff $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$.*

The next result is an immediate consequence since the equivalence problem for DPDA is decidable [14].

Corollary 13. *The equivalence problem for weak ω-DPDA is decidable.*

We can also use Corollary 12 to reduce the regularity problem to the case of finite words.

Theorem 14. *Let $\mathcal{A}$ be in normal form. $L_*(\mathcal{A})$ is regular iff $L_\omega(\mathcal{A})$ is regular.*

Proof. For the implication from finite to infinite words, suppose $L_*(\mathcal{A})$ is regular, i.e., $L_*(\mathcal{A}) = L_*(\mathcal{A}')$ for some DFA $\mathcal{A}'$. Viewing $\mathcal{A}'$ as a Büchi automaton (assigning color 1 to rejecting and color 0 to accepting states), we obtain that $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$ because $\mathcal{A}'$ visits a state of color 0 after the same prefixes as $\mathcal{A}$ visits a state with even color. Therefore, $L_\omega(\mathcal{A})$ is regular (this also shows that $\mathcal{A}'$ is a weak ω-DFA for an appropriate coloring).

If $L_\omega(\mathcal{A})$ is recognizable by an ω-DFA, then Lemma 3 shows that there is even a weak such ω-DFA $\mathcal{A}'$. We can assume that $\mathcal{A}'$ is in normal form (using our construction or the results in [10]). Applying Corollary 12, we obtain that $L_*(\mathcal{A}) = L_*(\mathcal{A}')$ is regular. This shows the inverse implication. □

Our main result uses the fact that regularity for DPDA is decidable [16, 18].

Corollary 15. *The regularity problem for weak ω-DPDA is decidable.*

Complexity. Our method to decide regularity is composed of transforming weak ω-DPDA into normal form and, applying the known regularity test for DPDA on its finitary language. Finally, in case of a positive result, the DFA being equivalent to the finitary language can be colored appropriately according to Theorem 14, to yield a weak ω-DFA that recognizes the original ω-language.

Lemma 10 normalizes a given weak ω-DPDA $\mathcal{A}$ in exponential time, and yields a DPDA $\mathcal{A}''$ that recognizes its finitary language, with $\mathcal{O}(|Q|)$ states and $|\Gamma|\cdot 2^{\mathcal{O}(|Q|^2)}$ stack symbols since we assume $k \in \mathcal{O}(|Q|)$. For the regularity test in the second step, we refer to [18] which gives better complexity bounds than [16]. According to Valiant's work, a DPDA that recognizes a regular language with n states, t stack symbols, and words of at most length h in transitions, can be transformed in doubly exponential time into an equivalent DFA 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. Both steps in composition run in triply exponential time. In case that the ω-language is regular, it yields a DFA, the number of states of which is bounded by $E^2\big(|Q|^2 \log |Q| + \log |\Gamma| + \log h\big)$, and which can be colored appropriately to yield a weak ω-DFA equivalent to $\mathcal{A}$.

Hence, the regularity test for ω-languages presented here is more expensive in terms of computation time than for languages of finite words. Nevertheless, in case of a regular ω-language, the resulting weak ω-DFA has the exact same bound on the number of states as for finite words. From Example 8, we see that an exponential blowup for the resulting weak ω-DFA is unavoidable.

4 Finite State Strategies for Pushdown Games

In this section, we are going to generalize the regularity problem to pushdown games. Testing for regularity in this setting means to ask for the existence of a winning finite state strategy. Let us first reconsider the weak classification game from Definition 2 which connects ω-languages and games in the context of regularity testing. We show that having a finite state representation for an ω-language naturally extends to having a finite state winning strategies.

Lemma 16. *Let $\mathcal{A}$ be a weak ω-DPDA with color set C. $L_\omega(\mathcal{A}) \subseteq \Sigma^\omega$ is regular iff Player 0 can win the weak classification game $\mathcal{G}_{\mathcal{A},C}$ with a finite state strategy.*

Proof. Let $\mathcal{A}$ and C be as above. The ω-language $L_\omega(\mathcal{A})$ is regular iff it is recognized by a weak ω-DFA $\mathcal{A}'$ with colors C according to Lemma 3. Such a weak ω-DFA naturally corresponds to a winning FSS for Player 0 in $\mathcal{G}_{\mathcal{A},C}$. □

In the rest of this section, we study the decision problem whether Player 0 can win a weak PDG with an FSS. We give a trivial answer in case of games with reachability condition but a negative answer for safety condition.

Lemma 17. *For a reachability PDG, Player 0 has a winning finite state strategy if he can win.*

Proof. Let $\mathcal{G} = (\mathcal{M}, \tau, Q_0)$ be a weak PDG with $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$, colors $\tau(Q) = \{0, 1\}$, and f 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 we arrange these play prefixes as a tree, then it is a finitely branching tree in which each branch is finite. According to König's Lemma, the tree must be finite. Using the nodes of this tree as states of a finite automaton yields a finite state winning strategy. □

Lemma 17 implies the following since PDGs are effectively determined [19].

Corollary 18. *For a reachability PDG, it is decidable whether Player 0 has a winning finite state strategy.*

This problem is not symmetric, meaning that it is undecidable from the perspective of Player 1, which is about having a winning FSS for a safety condition. To this end, we encode the run of a *2-register machine (2RM)* as a pushdown game. A 2RM can be seen as an input-free deterministic machine that is equipped with 2 counters which can be increased, decreased and tested for zero. The halting problem is undecidable for this machine model as it can encode Turing machines. For our next proof, we consider a similar problem which is to decide whether the unique run of a 2RM is *ultimately periodic (UP)*, i.e., whether the state sequence starting from the initial configuration forms an infinite word uv^ω where u, v are nonempty finite state sequences.

Lemma 19. *It is undecidable whether the run of a 2RM is ultimately periodic.*

Theorem 20. *For a safety 1CG, it is undecidable whether Player 0 has a winning finite state strategy.*

Proof (Sketch). We use Lemma 19 and construct for a given 2RM M a safety 1CG $\mathcal{G}$ such that Player 0 can win with an FSS iff M has a UP run. The main idea of $\mathcal{G}$ is divided into two phases. First, Player 1 increases and decreases the counter to an arbitrary value. Whether at this point the value is zero or not determines which register of M will be simulated by the counter of $\mathcal{G}$ in the second phase. There Player 0 has to play an infinite transition sequence of M that is correct just according to the simulated register. Thus, 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 run of M which simulates the according register correctly. This can be modeled by a safety winning condition where only an incorrect simulation in the second phase leads to an unsafe state.

Obviously, Player 0 has a winning strategy. But here we are interested in a winning FSS for Player 0. If he has a winning FSS f, then its limited memory cannot remember after the first phase whether the counter is zero or not, i.e., which register will be simulated by $\mathcal{G}$ in the second phase. But independent from that information, f is winning in the second phase and hence the transition sequence that f produces must be correct in both cases. This means that the sequence is indeed the unique run of M, and since it can be represented by the FSS f, it must be UP. For the inverse, assume that the run of M is UP. Then it can be represented by a finite state machine. Hence, a winning FSS for Player 0 is to ignore the first phase and to play in the second phase the run of M which is correct with respect to both registers. □

The undecidability of that problem can be shown in a similar way for visibly PDG, i.e., where the type of the stack operation is determined by the input symbol played in the game [1]. Due to this new restriction, the construction needs additional stack symbols to make an FSS of Player 0 not see what is happening on the stack.

Theorem 21. *For a safety visibly PDG, it is undecidable whether Player 0 has a winning finite state strategy.*

5 Conclusions

In Section 3, we considered weak pushdown ω-automata. We established a normal form that each weak pushdown ω-automaton can be effectively converted to, and revealed its close relation to languages of finite words. This allowed us to lift known decision procedures for equivalence [14] and regularity [18] from the case of finite words to ω-automata. Our regularity test can be performed in triply exponential time. The worst case size of an equivalent ω-DFA is between singly and doubly exponential. Especially the upper bound for ω-languages coincides with the one given by Valiant [18] for languages of finite words. He further gave singly exponential bounds for some restricted types of pushdown automata (like

ε-free, 1-counter). It is open whether they can be used to improve the test on respective types of ω-automata. Finally, the decidability of the regularity problem for (non-weak) pushdown ω-automata, as formulated in [6], remains open.

We dedicated Section 4 to an extension of the regularity problem for pushdown games, which asks for the existence of a finite state winning strategy. Beside a simple decidability result in the case of reachability winning conditions, we showed in the main result of that section the undecidability for safety conditions. The latter result extends to more complicated winning conditions of course. The decidability remains open in the case of visibly one-counter games.

Acknowledgements. With thanks to Wladimir Fridman for fruitful discussions.

References

[1] R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC*, pages 202–211, 2004.
[2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
[3] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138:295–311, 1969.
[4] T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, pages 704–715, 2002.
[5] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
[6] R. S. Cohen and A. Y. Gold. Omega-computations on deterministic pushdown machines. *JCSS*, 16(3):275–300, 1978.
[7] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV*, pages 232–247, 2000.
[8] M. Hague and C.-H. L. Ong. Winning regions of pushdown parity games: A saturation method. In *CONCUR*, pages 384–398, 2009.
[9] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.
[10] C. Löding. Efficient minimization of deterministic weak omega-automata. *Information Processing Letters*, 79(3):105–109, 2001.
[11] D. Perrin and J.-É. Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.
[12] L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *ICDT*, pages 299–313, 2007.
[13] L. Segoufin and V. Vianu. Validating streaming XML documents. In *PODS*, pages 53–64, 2002.
[14] G. Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theor. Comput. Sci.*, 251(1-2):1–166, 2001.
[15] L. Staiger. Finite-state ω-languages. *JCSS*, 27(3):434–448, 1983.
[16] R. E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11(3):323–340, 1967.
[17] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Basel, Switzerland, 1994.
[18] L. G. Valiant. Regularity and related problems for deterministic pushdown automata. *J. ACM*, 22(1):1–10, 1975.
[19] I. Walukiewicz. Pushdown processes: Games and model checking. *Information and Computation*, 164(2):234–263, 2001.

Appendix

A Proofs from Section 1

We justify the claim that the problem studied in [13, 12] can be expressed as a regularity problem for pushdown games.

We first introduce some terminology concerning the problem from [13, 12]. A DTD is basically a context-free grammar that does not distinguish terminal and non-terminal symbols and that can use regular expressions on the right hand side of the word. Derivation trees are defined in the standard way. A symbol a can be at a leaf if the language defined by the regular expression on the right hand side of the rule for the symbol contains the empty word.

For example, the rules $r \to a^*$, $a \to b$ and $b \to \varepsilon$ generate all (unranked) trees with r at the root. Below r there is a finite sequence of a (maybe empty, then the root is the only node), and below each a there is one b.

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

$$lin(t) = a \cdot lin(t_1) \cdots lin(t_n) \cdot \bar{a}.$$

The question studied in [13, 12] 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 $lin(D)$ is the set of all linearizations of derivation trees of D, then the question is whether there is a finite automaton $\mathcal{A}$ such that $L(\mathcal{A}) \cap Lin = lin(D)$.

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 $lin(D)$. 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 $lin(D)$ 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.

B Proofs from Section 2

We show how to detect and remove infinite sequences of ε-transitions for an ω-DPDA $\mathcal{A}$.

First, note that in each such infinite sequence there is a configuration qAW such that the top symbol A is never popped again during this sequence. This implies that from each configuration with state q and top stack symbol A there is an infinite sequence of ε-transitions. It is therefore enough to detect such pairs of q and A, which we call *diverging pairs*.

We find these pairs using existing algorithms for pushdown systems. Let $\mathcal{A}_\varepsilon$ denote the modification of $\mathcal{A}$ in which all non-ε-transitions are deleted.

Let $q \in Q$ and $A \in \Gamma_\perp$. If $A \neq \perp$, we compute a nondeterministic finite automaton (NFA) for the set $P_{q,A} = \text{post}^*_{\mathcal{A}_\varepsilon}(qA\perp)$ of configurations reachable from $qA\perp$ in $\mathcal{A}_\varepsilon$. If $A = \perp$, we compute an NFA for $P_{q,A} = \text{post}^*_{\mathcal{A}_\varepsilon}(q\perp)$. These NFAs can be computed in polynomial time [7].

The pair q, A is diverging if the following conditions are satisfied:

- If $A \neq \perp$, then no configuration of the form $p\perp$ is in $P_{q,A}$ (otherwise A is popped during the unique ε-sequence).
- There is no configuration $pBW \in P_{q,A}$ such that for p and B no outgoing ε-transitions is defined (the sequence of ε-transitions stops).

These conditions can be tested in polynomial time by some simple membership tests on the NFAs.

To remove the infinite ε-sequences from $\mathcal{A}$, it is enough to redirect the transitions for the detected diverging pairs into a new rejecting sink state looping on every input letter (we are interested in infinite words and an infinite sequence of ε-transitions cannot result in an accepted infinite word).

C Proofs from Section 3

For the remainder of this section, let $\mathcal{A} = (\mathcal{M}, \tau)$ be a weak ω-DPDA with $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \perp)$ and greatest color $k = \max\big(\tau(Q)\big)$. W.l.o.g., we assume $k \geq 1$. We start with a lemma stating some useful properties of the sets K_i from Definition 7.

Lemma 22. *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}.
c) If $qW \in K_i$, then $\tau(q) \geq i$.

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 7 (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.

c) Assume contrary that there is a configuration $qW \in K_i$ with $\tau(q) < i$ and choose it such that $\tau(q) + i$ is minimal. If $\tau(q) + i$ is odd (meaning that $\tau(q)$ is odd iff i is even), then pick an infinite run starting from qW staying in K_i according to (a). By Definition 7 (a), this run has to stabilize at a color with different parity than $\tau(q)$ and hence smaller than $\tau(q)$. Let $q'W' \in K_i$ be a configuration on the run such that $\tau(q') < \tau(q) < i$. This contradicts the minimality of $\tau(q) + i$.
If $\tau(q) + i$ is even, we get $\tau(q) < i - 1$ and $i \geq 2$. A run leading from $qW \in K_i$ to $q'W' \in K_{i-1}$ is guaranteed by (b) such that $\tau(q') \leq \tau(q) < i - 1$. Again, we get a contradiction to the minimality of $\tau(q) + i$. □

The next lemma states that we can construct a deterministic finite state machine (DFSM) that reads a reversed configuration and assigns the corresponding class K_i to it.

Lemma 23. *One can generate a DFSM* $\mathcal{M}^K = (Q^K, \Sigma^K, \delta^K, q_0^K)$ *with alphabet* $\Sigma^K = \Gamma_\perp$*, and a function* $f^K : Q \times Q^K \to \mathbb{N}$ *assigning the classes: for each configuration* $qW \in K_i$ *iff* $f^K\big(q, \delta_*^K(W^R)\big) = i$*. This computation takes time* $2^{\mathcal{O}(|Q| \cdot k)}$ *and yields size* $|Q^K| \in 2^{\mathcal{O}(|Q| \cdot k)}$.

Proof. We show this lemma by using the weak classification game $\mathcal{G}_{\mathcal{A},C}$ from Definition 2, where the colors $C = \{0, \ldots, k\}$ suffice according to Lemma 22 (c). As this game naturally represents Definition 7, it has the property that Player 0 can win from a configuration $(q, i, 0)W$ iff $qW \in K_j$ with $j \leq i$, as we explain in the following.

An obvious winning strategy for Player 0 from such a configuration 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 inverse, assume contrary 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 22 (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 $j \geq 2$. Hence, Player 1 can force the play to a configuration $(q', i', 0)W'$ with $q'W' \in K_{j-1}$ due to Lemma 22 (b). By assumption, Player 0 can still win from there which contradicts the minimality since $i' + j - 1 \leq i + j - 1 < i + j$.

Since $\mathcal{A}$ is weak, the winning condition of $G_{\mathcal{A},D}$ is also a weak parity condition. This can be rewritten as a Büchi condition redefining odd colors as 1 and even colors as 0. From [4], we know that the winning region for Player 0 in such a Büchi pushdown game is a regular set of configurations. To generate $\mathcal{M}^K$ with exponential size, we use an algorithm given in [8]. Based on $\mathcal{G}_{\mathcal{A},C}$ which has

$\mathcal{O}(|Q|\cdot k)$ many states, it yields a so called *alternating multi-automaton* $\mathcal{A}'$. It is an alternating automaton having the same states as $\mathcal{G}_{\mathcal{A},C}$ (plus two extra states), with the property that it has an accepting run from (q,i,x) for a word $W \in \Gamma^*\bot$ iff Player 0 can win from $(q,i,x)W$ in $\mathcal{G}_{\mathcal{A},C}$. Formally, $\mathcal{A}' = (Q', \Sigma', \Delta', F')$ is an alternating automaton with

a) states $Q' \supseteq Q \times C \times \{0,1\}$ of size $|Q'| \in \mathcal{O}(|Q|\cdot k)$, final states $F' \subseteq Q'$,
b) input alphabet $\Sigma' = \Gamma_\bot$, and
c) transition relation $\Delta' \subseteq Q' \times \Sigma' \times 2^{Q'}$.

To define the acceptance, let $q' \xrightarrow{\varepsilon} \{q'\}$, and $q' \xrightarrow{AW} Q'_1 \cup \ldots \cup Q'_n$ iff $q' \xrightarrow{A} \{q'_1, \ldots, q'_n\}$ and $q'_i \xrightarrow{W} Q'_i$. In this setting, no designated initial states are needed because we want to ask whether a run starting from $(q,i,x) \in Q'$ is accepted. Then the language accepted by $\mathcal{A}'$ is

$$L_*(\mathcal{A}') = \left\{ (q,i,x)W \in \big(Q\times C\times\{0,1\}\big)\Gamma^*\bot \;\middle|\; (q,i,x) \xrightarrow{W} P' \text{ for some } P' \subseteq F' \right\}.$$

A DFA for the reversal language $L_*(\mathcal{A}')^{\mathrm{R}}$ of an alternating automaton $\mathcal{A}'$ can be obtained by a reversal powerset construction [5]. In the following, we provide such a construction on $\mathcal{A}'$ to obtain the desired DFSM $\mathcal{M}^K$ with an exponential blowup. Let $\mathcal{M}^K = (Q^K, \Sigma^K, \delta^K, q_0^K)$ be a DFSM with

a) states $Q^K = 2^{Q'}$, initial state $q_0^K = F'$,
b) input alphabet $\Sigma^K = \Sigma' = \Gamma_\bot$, and
c) transition function $\delta^K : Q^K \times \Sigma^K \to Q^K$ where

$$\delta^K(P', A) = \left\{ q' \in Q' \;\middle|\; (q', A, P'') \in \Delta \text{ for some } P'' \subseteq P' \right\}.$$

By construction, we have that $(q,i,x) \in \delta_*^K(W^{\mathrm{R}})$ iff $(q,i,x) \xrightarrow{W} P'$ for some $P' \subseteq F'$ iff $(q,i,x)W \in L_*(\mathcal{A}')$ iff Player 0 can win from $(q,i,x)W$ iff $qW \in K_j$ for some $j \le i$. It remains to define the function $f^K : Q \times Q^K \to \mathbb{N}$ as the minimal such i: $f^K(q,P') = \min\{i \mid (q,i,0) \in P'\}$. Hence, $f^K\big(q, \delta_*^K(W^{\mathrm{R}})\big) = i$ iff $qW \in K_i$. □

Based on this, we can now prove Lemma 10.

Lemma 10. *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 $L_*(\mathcal{A}'') = L_*(\mathcal{A}')$, where $\mathcal{A}''$ has $\mathcal{O}(|Q|)$ states and $|\Gamma|\cdot 2^{\mathcal{O}(|Q|\cdot k)}$ stack symbols.*

Proof. Let $\mathcal{M}^K = (Q^K, \Sigma^K = \Gamma_\bot, \delta^K, q_0^K)$ be a DFSM with function f^K according to Lemma 23. We define a weak ω-DPDA $\mathcal{A}' = (\mathcal{M}', \tau')$ with $\mathcal{M}' = (Q', \Sigma, \Gamma', \delta', q'_0, \bot')$ and coloring $\tau' : Q' \to \mathbb{N}$ in such a way that it runs like $\mathcal{A}$ and annotates the stack with a simulation of $\mathcal{M}^K$ to use the information about the classes. The construction goes as follows:

a) states $Q' = Q \times \{0, \dots, k\}$, $q_0' = (q_0, c)$ where $c = f^K\big(\delta^K(q_0^K, \bot)\big)$, i.e., $q_0 \bot \in K_c$,
b) stack symbols $\Gamma' = \Gamma \times Q^K$, $\bot' = (\bot, q_0^K)$,
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 $i \in \{1, \dots, n\}$,
   iii) $d = f^K(p, p_n^K)$,
d) coloring $\tau'\big((q,i)\big) = i$.

This construction maps a configuration $qA_n \cdots A_1$ in $\mathcal{A}$ to an annotated configuration $(q,i)A_n' \cdots A_1'$ in $\mathcal{A}'$ where $i = f^K\big(q, \delta_*^K(A_1 \cdots A_n)\big)$, i.e., $qA_n \cdots A_1 \in K_i$ and $A_i' = \big(A_i, \delta_*^K(A_1 \cdots A_{i-1})\big)$. Hence, a bijection between the configurations of $\mathcal{A}$ and properly annotated configurations of $\mathcal{A}'$ is established. From Definition 7 (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 7 (b) follows that $\mathcal{A}'$ is weak and the claim on the complexity follows from the construction and Lemma 23.

It remains to show that $\mathcal{A}'$ is in normal form. 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 $\mathcal{A}$. Assume that the correspondence of the classes is already established for $K_0, \dots, K_{i-1}$. 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 from $(q,i)A_n' \cdots A_1'$ that leads to K_j for $j < i$ in $\mathcal{A}'$, the corresponding run from $qA_n \cdots A_1$ also leads to K_j in $\mathcal{A}$, 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 from both configurations the same language is accepted. Hence, the classes K_i in the two automata also coincide 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, resulting in $2 \cdot |Q|$ many states. Each step of the computation can be done in at most exponential time. □

The proof of Lemma 11 is based on the following lemma.

Lemma 24. *Let $\mathcal{A}$ be in normal form. If $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$, then $\tau(q) = \tau(p)$.*

Proof. Assume contrary that for some configurations $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$ but $\tau(q) < \tau(p)$ and choose the configurations such that $\tau(q) + \tau(p)$ is minimal. We consider two cases that lead to a contradiction. If $\tau(q) + \tau(p)$ is odd, then Lemma 22 (a) yields an ω-word α such that from pV only color $\tau(p)$ is visited. The run of α from qW has the same acceptance and hence, due to the weakness it stabilizes at a color smaller than $\tau(q)$. Let $q'W'$ and $p'V'$ be configurations on the run on a prefix w of α from qW and pV, respectively, such that $\tau(q') < \tau(q)$

and $\tau(p') = \tau(p)$. Since $L_\omega(\mathcal{A}_{q'W'}) = L_\omega(\mathcal{A}_{p'V'})$, we have a contradiction to the minimality of $\tau(q) + \tau(p)$.

Otherwise, $\tau(q) + \tau(p)$ is even which implies $\tau(q) < \tau(p) - 1$ and $\tau(p) \geq 2$. Lemma 22 (b) guarantees that there is a word w leading from pV to $p'V'$ with color $\tau(p') = \tau(p) - 1$. Let $q'W'$ be the configuration reached by w from qW, then $\tau(q') \leq \tau(q) < \tau(p) = \tau(p')$. Again $L_\omega(\mathcal{A}_{q'W'}) = L_\omega(\mathcal{A}_{p'V'})$, and we have a contradiction to the minimality of $\tau(q) + \tau(p)$. □

Based on this, the proof of Lemma 11 is straight forward.

Lemma 11. *Let $\mathcal{A}$ be in normal form. If $L_\omega(\mathcal{A}_{qW}) = L_\omega(\mathcal{A}_{pV})$, then $L_*(\mathcal{A}_{qW}) = L_*(\mathcal{A}_{pV})$.*

Proof. Assume contrary that w.l.o.g. there is a word $w \in L_*(\mathcal{A}_{qW}) \setminus L_*(\mathcal{A}_{pV})$. Hence $\tau(q') \neq \tau(p')$ for the configurations $q'W'$, and $p'V'$, reached by w from qW, and pV, respectively. Lemma 24 then yields $L_\omega(\mathcal{A}_{q'W'}) \neq L_\omega(\mathcal{A}_{p'V'})$ which implies $L_\omega(\mathcal{A}_{qW}) \neq L_\omega(\mathcal{A}_{pV})$. □

D Proofs from Section 4

Before proving Lemma 19 and Theorem 20, we formalize the concept of a 2-register machine. A *2RM* $M = (Q, \delta, q_0, F)$ consists of

a) a finite state set Q, initial state q_0, final states $F \subseteq Q$, and
b) an input-free deterministic transition function $\delta : Q \times \{0,1\}^2 \to Q \times \{-1, 0, +1\}^2$.

The configurations of M is the set $Q \times \mathbb{N}^2$, and δ leads from a configuration (p, n_0, n_1) to another $(q, n_0 + d_0, n_1 + d_1)$ iff $\delta\big(p, \mathrm{sgn}(n_0), \mathrm{sgn}(n_1)\big) = (q, d_0, d_1)$, i.e., depending on the current state and whether the registers are zero it leads to another state and increases or decreases the registers. W.l.o.g., we assume that $n_i + d_i \in \mathbb{N}$ for all $i \in \{0, 1\}$. The halting problem is undecidable for this machine model as it can encode Turing machines. For the proof of Theorem 20, we consider a similar problem that is to decide whether the unique run of a 2RM is *ultimately periodic (UP)*, i.e., whether the state sequence starting from $(q_0, 0, 0)$ forms an infinite word uv^ω where u, v are nonempty finite state sequences.

Lemma 19. *It is undecidable whether the run of a 2RM is ultimately periodic.*

Proof. Due to the computational power of 2RMs, there are some with a non-UP run. We show the lemma by a reduction of the halting problem. To this end, we transform a given 2RM M into two different instances M_0, M_1 of the considered problem. Both these machines first simulate M until it reaches a halting state. Then M_0 continues with some computation that has a UP run, whereas M_1 continues with one not being UP. Thus, by construction, the property of having a UP run differs for M_0, M_1 if and only if M reaches a halting state. □

Theorem 20. *For a safety 1CG, it is undecidable whether Player 0 has a winning finite state strategy.*

Proof. We will use Lemma 19 and construct for a given 2RM M a safety 1CG $\mathcal{G}$ such that Player 0 can win with an FSS iff M has a UP run. The main idea of $\mathcal{G}$ is divided into two phases. First, Player 1 increases and decreases the counter to an arbitrary value. Which register of M will be simulated by the counter of $\mathcal{G}$ in the second phase is determined by whether or not the value of the counter is zero now. Then in the second phase, Player 0 has to play an infinite transition sequence of M that is correct just according to the simulated register. Thus, 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 run of M which simulates the according register correctly. This can be realized by a safety winning condition where an incorrect simulation in the second phase leads to an unsafe state.

Formally, we define for a given 2RM $M = (Q^M, \delta^M, q_0^M, F^M)$ the weak 1CG $\mathcal{G} = \big((Q, \Sigma, \Gamma, \delta, q_0, \bot), \tau, Q_0\big)$ with

a) states $Q = Q_0 \cup Q_1$ where $Q_0 = \{q_\bot\} \cup Q^M \times \{0,1\}$ (state (q,i) indicates that M is in state q and register i is simulated) and $Q_1 = \{q_0, q_{1,0}, q_{1,1}, q_\top\}$,
b) safety coloring $\tau : Q \to \{1,2\}$ where $\tau(q) = 1$ iff it is the bad state $q = q_\bot$,
c) alphabets $\Sigma = \{a,b,c\} \cup Q^M \times \{0,1\}^2$ and $\Gamma = \{\$\}$,
d) transitions:
   i) first phase:
      1) $\delta(q_0, A, a) = (q_0, \$A)$,
      2) $\delta(q_0, A, b) = (q_0, W)$ where $W = \begin{cases} \bot & \text{if } A = \bot, \\ \varepsilon & \text{if } A = \$, \end{cases}$
      3) $\delta(q_0, A, c) = (q_{1,i}, A)$ where $i = \begin{cases} 0 & \text{if } A = \bot, \\ 1 & \text{if } A = \$, \end{cases}$
      4) $\delta(q_{1,i}, A, b) = (q_{1,i}, W)$ where $W = \begin{cases} \bot & \text{if } A = \bot, \\ \varepsilon & \text{if } A = \$, \end{cases}$
      5) $\delta(q_{1,i}, \bot, c) = \big((q_0^M, i), \bot\big)$,
   ii) second phase:
      1) $\delta\big((p,i), A, (p, r_0, r_1)\big) = \big((q,i), W\big)$ for $\delta^M(p, r_0, r_1) = (q, d_0, d_1)$

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

   iii) all other cases: $\delta(p, A, x) = (q, A)$ where $q = \begin{cases} q_\bot & \text{if } p \in Q_0, \\ q_\top & \text{if } p \in Q_1. \end{cases}$

The intention of this construct is the following. In the first phase Player 1 can play a finite sequence $a^n b^m c b^\ell c$ with $n \le m + \ell$. Then in the second phase Player 0 answers with an infinite sequence over $Q^M \times \{0,1\}^2$ which identifies M-transitions. Depending on whether $n = m$ held previously, the game simulates either register

0 or 1, i.e., the counter of the game mimics all increase and decrease operations and verifies zero-tests for the simulated registers. Player 0 looses iff his transition sequence is inappropriate with respect to the simulated register.

If M has a UP run, then Player 0 has a winning FSS by playing exactly this run no matter which register is simulated. For the converse, assume that Player 0 has a winning FSS $\mathcal{S}$ with s states. After an action sequence a^s, there must have been a state repetition in $\mathcal{S}$: $\delta^*_{\mathcal{S}}(a^x) = \delta^*_{\mathcal{S}}(a^y)$ with $0 \leq x < y \leq s$. Thus, $\mathcal{S}$ is in the same state when in the first phase Player 1 gives the sequence $a^x b^x c b^y c$ or $a^y b^x c b^y c$, respectively, and does not known which register is simulated in the second phase. But since $\mathcal{S}$ is winning, it must produce a UP transition sequence that correctly handles both registers. Hence, this sequence is the run of M. □

Next, we show a similar result for visibly PDG instead of 1CG by a variant of the latter proof. But let us first define the new restriction. The idea is that from the input letter $a \in \Sigma$ one can determine how the stack height is altered [1]. A deterministic pushdown machine is called *visibly* with respect to a partition of the input alphabet $\Sigma = \Sigma_i \uplus \Sigma_c \uplus \Sigma_r$ when for all transitions:

$$\delta(q, A, a) = (q', W') \text{ implies that } a \neq \varepsilon \text{ and } W' = \begin{cases} A & \text{if } a \in \Sigma_i, \\ A'A & \text{if } a \in \Sigma_c, \\ \varepsilon & \text{if } a \in \Sigma_r \text{ and } A \neq \bot, \\ \bot & \text{if } a \in \Sigma_r \text{ and } A = \bot. \end{cases}$$

This restriction is lifted to pushdown automata and games.

Theorem 21. *For a safety visibly PDG, it is undecidable whether Player 0 has a winning finite state strategy.*

Proof. The construction is basically the same as in the proof of Theorem 20. But due to the visibility restriction we have to change one detail since the transitions played by Player 0 should not directly control the stack height. To fix this, we put Player 1 in charge of updating the register value on the stack. Due to the visibility property, we still have to prevent Player 0 from knowing the number of counting symbols. For this reason, we introduce an additional dummy stack symbol # beside the counting symbol \$. Thus, in the second phase a word $W \in \{\#, \$\}^*$ on the stack represents the value $|W|_\$$, i.e., the number of \$ symbols in W. So after Player 0 chose an M-transition, it is on Player 1 to either disprove it or to update the stack accordingly. The update step is done in a similar way as the complete first phase of the game such that an FSS of Player 0 cannot remember what happens although it can see all changes in stack height. Afterwards, Player 0 continues with the next transition.

Formally, we define for a given 2RM $M = (Q^M, \delta^M, q_0^M, F^M)$ the weak 1CG $\mathcal{G} = \big((Q, \Sigma, \Gamma, \delta, q_0, \bot), \tau, Q_0\big)$ with

a) states $Q = Q_0 \cup Q_1$ where $Q_0 = \{q_\bot\} \cup Q_M \times \{0,1\}$, and $Q_1 = \{q_0, q_\top\} \cup Q_M \times \{0,1\} \times \{0,1\} \times \{-1,0,+1\} \cup Q_M \times \{0,1\} \times \{-1,0,+1\} \cup \{check_0, check_1\}$,

b) safety coloring $\tau : Q \to \{1,2\}$ where $\tau(q) = 1$ iff it is the bad state $q = q_\perp$,
c) alphabets $\Sigma = \Sigma_i \uplus \Sigma_c \uplus \Sigma_r$ with $\Sigma_i = \{c,d\} \cup Q_M \times \{0,1\}^2$, $\Sigma_c = \{a\}$, $\Sigma_r = \{b\}$, and $\Gamma = \{\$, \#\}$,
d) transitions:
   i) first phase:
      1) $\delta(q_0, A, a) = (q_0, \#A)$,
      2) $\delta(q_0, \#, b) = (q_0, \varepsilon)$,
      3) $\delta(q_0, A, c) = \big((q_0^M, i), A\big)$ where $i = \begin{cases} 0 & \text{if } A = \perp, \\ 1 & \text{if } A = \#, \end{cases}$
   ii) second phase:
      1) $\delta\big((p,i), A, (p, r_0, r_1)\big) = \big((q, i, r_i, d_i), A\big)$ for $\delta_M(p, r_0, r_1) = (q, d_0, d_1)$ to select an M-transition,
      2) $\delta\big((q,i,r,d), A, c\big) = \big((q,i,d), A\big)$ to push/pop \$ symbols according to d for the M-transition chosen by Player 0:
         I) $\delta\big((q,i,+1), A, a\big) = \big((q,i,0), \$A\big)$, (push one \$ if $d = +1$)
         II) $\delta\big((q,i,d), A, a\big) = \big((q,i,d), \#A\big)$ if $d \neq +1$, (otherwise push \#)
         III) $\delta\big((q,i,d), \#, b\big) = \big((q,i,d), \varepsilon\big)$, (ignore \#)
         IV) $\delta\big((q,i,-1), \$, b\big) = \big((q,i,0), \varepsilon\big)$, (pop one \$ if $d = -1$)
         V) $\delta\big((q,i,0), A, c\big) = \big((q,i), A\big)$, (end of simulation if $d = 0$)
      3) $\delta\big((q,i,r,d), A, d\big) = \big(check_r, A\big)$ to disprove that the M-transition chosen by Player 0 can be performed:
         I) $\delta\big(check_r, \#, b\big) = (check_r, \varepsilon)$, (ignore \#)
         II) $\delta\big(check_0, \$, b\big) = (q_\perp, \varepsilon)$, (disproof $r = 0$ by popping \$)
         III) $\delta\big(check_1, \perp, b\big) = (q'_\perp, \varepsilon)$, (disproof $r = 1$ by reaching $\perp$)
   iii) all other cases: $\delta(p, A, x) = (q, W)$

where $q = \begin{cases} q_\perp & \text{if } p \in Q_0, \\ q_\top & \text{if } p \in Q_1, \end{cases}$ and $W = \begin{cases} A & \text{if } x \in \Sigma_i, \\ \#A & \text{if } x \in \Sigma_c, \\ \varepsilon & \text{if } x \in \Sigma_r \text{ and } A \neq \perp, \\ \perp & \text{if } x \in \Sigma_r \text{ and } A = \perp. \end{cases}$

Thus, the first phase works as in the proof of before and determines whether register 0 or 1 is simulated in the second phase. Then each time after Player 0 played a transition Player 1 can choose to apply its according register update to the stack by playing c or to disprove it by playing d. In the latter case, the stack is popped to find out whether there is a \$-symbol above the bottom $\perp$ contrary to the claim of Player 0 that the register is empty, or vise-versa. In the first case, Player 1 can play a sequence $a^n b^m$. By this, one \$-symbol is pushed onto or popped from the stack whereas all other operations are only concerning the dummy \#-symbols. Similarly to the first phase, an FSS cannot know the new stack height. Hence, it does not know which register is simulated since the register operations appear all the same to it. Then rest of the proof is as for Theorem 20. □