Aalborg Universitet

A bit of nondeterminism makes pushdown automata expressive and succinct

Guha, Shibashis; Jecker, Ismaël; Lehtinen, Karoliina; Zimmermann, Martin

A BIT OF NONDETERMINISM MAKES PUSHDOWN AUTOMATA EXPRESSIVE AND SUCCINCT

SHIBASHIS GUHA ◉ [a], ISMAËL JECKER [b,c], KAROLIINA LEHTINEN ◉ [d],
AND MARTIN ZIMMERMANN ◉ [e]

[a] Tata Institute of Fundamental Research, Mumbai, India

[b] University of Warsaw, Warsaw, Poland

[c] Université de Franche-Comté, CNRS, FEMTO-ST, Besançon, France

[d] CNRS, Aix-Marseille University, LIS, Marseille, France

[e] Aalborg University, Aalborg, Denmark

ABSTRACT. We study the expressiveness and succinctness of history-deterministic pushdown automata (HD-PDA) over finite words, that is, pushdown automata whose nondeterminism can be resolved based on the run constructed so far, but independently of the remainder of the input word. These are also known as good-for-games pushdown automata.

We prove that HD-PDA recognise more languages than deterministic PDA (DPDA) but not all context-free languages (CFL). This class is orthogonal to unambiguous CFL. We further show that HD-PDA can be exponentially more succinct than DPDA, while PDA can be double-exponentially more succinct than HD-PDA. We also study HDness in visibly pushdown automata (VPA), which enjoy better closure properties than PDA, and for which we show that deciding HDness is ExpTime-complete. HD-VPA can be exponentially more succinct than deterministic VPA, while VPA can be exponentially more succinct than HD-VPA. Both of these lower bounds are tight.

We then compare HD-PDA with PDA for which composition with games is well-behaved, i.e. *good-for-games* automata. We show that these two notions coincide, but only if we consider potentially infinitely branching games.

Finally, we study the complexity of resolving nondeterminism in HD-PDA. Every HD-PDA has a positional resolver, a function that resolves nondeterminism and that is only dependent on the current configuration. Pushdown transducers are sufficient to implement the resolvers of HD-VPA, but not those of HD-PDA. HD-PDA with finite-state resolvers are determinisable.

1. Introduction

Nondeterminism adds both expressiveness and succinctness to deterministic pushdown automata. Indeed, the class of context-free languages (CFL), recognised by nondeterministic pushdown automata (PDA), is strictly larger than the class of deterministic context-free languages (DCFL), recognised by deterministic pushdown automata (DPDA), both over finite and infinite words. Even when restricted to languages in DCFL, there is no computable bound on the relative succinctness of PDA [Har80, Val76]. In other words, nondeterminism is remarkably powerful, even for representing deterministic languages. The cost of such succinct representations is algorithmic: problems such as universality and solving games with a CFL winning condition are undecidable for PDA [Fin01, HU79], while they are decidable for DPDA [Wal01]. Intermediate forms of automata that lie between deterministic and nondeterministic models have the potential to mitigate some of the disadvantages of fully nondeterministic automata while retaining some of the benefits of the deterministic ones.

Unambiguity and bounded ambiguity, for example, restrict nondeterminism by requiring words to have at most one or at most k, for some fixed k, accepting runs. Holzer and Kutrib survey the noncomputable succinctness gaps between unambiguous PDA and both PDA and DPDA [HK10], while Okhotin and Salomaa show that unambiguous visibly pushdown automata are exponentially more succinct than DPDA [OS15]. Universality of unambiguous PDA is decidable, as it is decidable for unambiguous context-free grammars [SS78], which are effectively equivalent [Her97]. However, to the best of our knowledge, unambiguity is not known to reduce the algorithmic complexity of solving games with a context-free winning condition.

Another important type of restricted nondeterminism that is known to reduce the complexity of universality and solving games has been studied under the names of good-for-games nondeterminism [HP06] and history-determinism [Col09] (HD)[1]. Intuitively, a nondeterministic automaton is HD if its nondeterminism can be resolved on-the-fly, i.e. without knowledge of the remainder of the input word to be processed.

For finite automata on finite words, where nondeterminism adds succinctness, but not expressiveness, HD nondeterminism does not even add succinctness: every HD-NFA contains an equivalent DFA [BKS17], which can be obtained by pruning transitions from the HD-NFA. Thus, HD-NFA cannot be more succinct than DFA. But for finite automata on infinite words, where nondeterminism again only adds succinctness, but not expressiveness, HD coBüchi automata can be exponentially more succinct than deterministic automata [KS15]. Finally, for certain quantitative automata over infinite words, HD nondeterminism adds as much expressiveness as arbitrary nondeterminism [Col09].

Recently, pushdown automata on infinite words with HD nondeterminism (ω-HD-PDA) were shown to be strictly more expressive than ω-DPDA, while universality and solving games for ω-HD-PDA are not harder than for ω-DPDA [LZ22]. Thus, HD nondeterminism adds expressiveness without increasing the complexity of these problems, i.e. pushdown automata with HD nondeterminism induce a novel and intriguing class of context-free ω-languages.

---

[1]After the publication of the conference version [GJLZ21], which uses the term good-for-games, it has come to light that the notions of history-determinism and good-for-gameness do not always coincide [BL21]. We therefore prefer to use the term history-determinism as it corresponds better to our definitions. In Section 8, we show that at least for pushdown automata over *finite* words and potentially infinitely branching games, the two notions coincide.

Here, we continue this work by studying the expressiveness *and* succinctness of PDA over finite words. While the decidability results on solving games and universality for ω-HD-PDA on infinite words also hold for HD-PDA on finite words, the separation argument between ω-HD-PDA and ω-DPDA depends crucially on combining HD nondeterminism with the coBüchi acceptance condition. Since this condition is only relevant for infinite words, the separation result does not transfer to the setting of finite words.

Nevertheless, we prove that HD-PDA are more expressive than DPDA, yielding the first class of automata on finite words where HD nondeterminism adds expressiveness. The language witnessing the separation is remarkably simple, in contrast to the relatively subtle argument for the infinitary result [LZ22]: the language $\{a^i\$a^j\$b^k\$ \mid k \leqslant \max(i,j)\}$ is recognised by an HD-PDA but not by a DPDA. This yields a new class of languages, those recognised by HD-PDA over finite words, for which universality and solving games are decidable. We also show that this class is incomparable with unambiguous context-free languages.

We then turn our attention to the succinctness of HD-PDA. We show that the succinctness gap between DPDA and HD-PDA is at least exponential, while the gap between HD-PDA and PDA is at least double-exponential. These results hold already for finite words.

To the best of our knowledge, both our expressiveness and our succinctness results are the first examples of history-determinism being used effectively over finite, rather than infinite, words (recall that all HD-NFA are determinisable by pruning). Also, this is the first succinctness result for history-deterministic automata that does not depend on the infinitary coBüchi acceptance condition, which was used to show the exponential succinctness of HD coBüchi automata, as compared to deterministic ones [KS15].

We then study an important subclass of HD-PDA, namely, HD visibly pushdown automata (VPA), in which the stack behaviour (push, pop, skip) is determined by the input letter only. HD-VPA enjoy the good closure properties of VPA (to which they are expressively equivalent): they are closed under complement, union and intersection. We show that there is an exponential succinctness gap between deterministic VPA (DVPA) and HD-VPA, as well as between HD-VPA and VPA. Both of these are tight, as VPA, and therefore HD-VPA as well, admit an exponential determinisation procedure [AM04]. Furthermore, we show that HDness of VPA is decidable in ExpTime. This makes HD-VPA a particularly interesting class of PDA as they are recognisable, succinct, have good closure properties and, as for HD-PDA in general, deciding universality and solving games are both in ExpTime. In contrast, solving ω-VPA games is 2ExpTime-complete [LMS04]. We also relate the problem of checking HDness with the *good-enough synthesis* [AK20] or *uniformization* problem [CL15], which we show to be ExpTime-complete for DVPA and HD-VPA.

In general, the good compositional properties of HD automata enable them to be used for solving games and synthesis. In the regular setting, HD automata are exactly those for which composition with games (of which the winning condition is recognised by the automaton) preserves the winner. Hence, the terms "good-for-games" and "history-deterministic" have often been used interchangeably. However, these two notions do not always coincide, for example for quantitative automata [BL21]. Here we examine the situation for HD-PDA and show that for PDA, like for (ω-)regular automata, HD automata are exactly those which compose well with games, that is, history-determinism coincides with good-for-gameness. However, while in the (ω-)regular setting composition with finitely branching and infinitely

branching games coincide, this is not the case for PDA: there are PDA that are not HD but enjoy compositionality with all finitely branching games.

Nondeterminism in HD automata is resolved on-the-fly, i.e. the next transition to be taken only depends on the run prefix constructed so far and the next letter to be processed. Thus, the complexity of a resolver, mapping run prefixes and letters to transitions, is a natural complexity measure for HD automata. For example, finite HD automata (on finite and infinite words) have a finite-state resolver [HP06]. For pushdown automata with their infinite configuration space, the situation is markedly different: On one hand, we show that HD-PDA admit positional resolvers, that is, resolvers that depend only on the current configuration, rather than on the entire run prefix produced so far. Note that this result only holds for HD-PDA over finite words, but not for ω-HD-PDA. Yet, positionality does not imply that resolvers are simple to implement. We show that there are HD-PDA that do not admit a resolver implementable by a pushdown transducer. In contrast, all HD-VPA admit pushdown resolvers, again showing that HD-VPA are better behaved than general HD-PDA. Finally, HD-PDA with finite-state resolvers are determinisable.

Structure of the paper. We begin with some preliminaries in Section 2 and introduce HD-PDA in Section 3. We study their expressiveness in Section 4 by comparing them to DPDA, PDA, and unambiguous PDA. Afterwards, in Section 5, we exhibit the succinctness of HD-PDA in relation to DPDA and PDA. The important subclass of HD-VPA is studied in Section 6, focusing on succinctness, decidability of HDness, and connections to the good-enough synthesis problem. Closure properties of HD-PDA are studied in Section 7 while we compare the notions of HDness and good-for-gameness for PDA on finite words in Section 8. Then, in Section 9, we solve synthesis with specifications given by HD-PDA as well as universality of such automata. Next, the resources required to resolve the nondeterminism in an HD-PDA are studied in Section 10 before we conclude with some open problems in Section 11.

Previous version. A conference version of this paper was presented at MFCS 2021 [GJLZ21]. The present article contains all proofs, a more detailed discussion of closure properties, and a new section on compositionality.

Related work. The notion of HD nondeterminism has emerged independently several times, at least as Colcombet's history-determinism [Col09], in Piterman and Henzinger's HD automata [HP06], and as Kupferman, Safra, and Vardi's nondeterminism for recognising derived languages, that is, the language of trees of which all branches are in a regular language [KSV06]. Related notions have also emerged in the context of XML document parsing. Indeed, preorder typed visibly pushdown languages and 1-pass preorder typeable tree languages, considered by Kumar, Madhusudan, and Viswanathan [KMV07] and Martens, Neven, Schwentick, and Bex [MNSB06] respectively, also consider nondeterminism which can be resolved on-the-fly. However, the restrictions there are stronger than simple HD nondeterminism, as they also require the typing to be unique, roughly corresponding to unambiguity in automata models and grammars. This motivates the further study of unambiguous HD automata, although this remains out of scope for the present paper. The XML extension AXML has also inspired Active Context Free Games [MSS06], in which one player, aiming to produce a word within a target regular language, chooses positions on a word and the other player chooses a rewriting rule from a context-free grammar. Restricting the strategies of the first player to moving from left to right makes finding the winner

decidable [MSS06, BSSK13]; however, since the player still knows the future of the word, this restriction is not directly comparable to HD nondeterminism.

Unambiguity, or bounded ambiguity, is an orthogonal way of restricting nondeterminism by limiting the number of permitted accepting runs per word. For regular languages, it leads to polynomial equivalence and containment algorithms [SI85]. Minimization remains NP-complete for both unambiguous automata [JR93, BM12] and HD automata [Sch20] (at least when acceptance is defined on states, see [RK19]). On pushdown automata, increasing the permitted degree of ambiguity leads to both greater expressiveness and unbounded succinctness [Her97]. Finally, let us mention two more ways of measuring–and restricting– nondeterminism in PDA: bounded nondeterminism, as studied by Herzog [Her97] counts the branching in the run-tree of a word, while the minmax measure [SY93, GLW05] counts the number of nondeterministic guesses required to accept a word. The natural generalisation of history-determinism as the *width* of an automaton [KM19] has not yet, to the best of our knowledge, been studied for PDA.

2. Preliminaries

An alphabet Σ is a finite nonempty set of letters. The empty word is denoted by ε, the length of a word w is denoted by $|w|$, and the nth letter of w is denoted by $w(n)$ (starting with $n = 0$). The set of (finite) words over Σ is denoted by Σ^*, the set of nonempty (finite) words over Σ by Σ^+, and the set of finite words of length at most n by $\Sigma^{\leqslant n}$. A language over Σ is a subset of Σ^*.

For alphabets Σ_1, Σ_2, we extend functions $f\colon \Sigma_1 \to \Sigma_2^*$ homomorphically to words over Σ_1 via $f(w) = f(w(0))f(w(1))f(w(2))\cdots$.

2.1. **Pushdown automata.** A pushdown automaton (PDA for short) $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ consists of a finite set Q of states with the initial state $q_I \in Q$, an input alphabet Σ, a stack alphabet Γ, a transition relation Δ to be specified, and a set F of final states. For notational convenience, we define $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\perp = \Gamma \cup \{\perp\}$, where $\perp \notin \Gamma$ is a designated stack bottom symbol. Then, the transition relation Δ is a subset of $Q \times \Gamma_\perp \times \Sigma_\varepsilon \times Q \times \Gamma_\perp^{\leqslant 2}$ that we require to neither write nor delete the stack bottom symbol from the stack: if $(q, \perp, a, q', \gamma) \in \Delta$, then $\gamma \in \perp \cdot (\Gamma \cup \{\varepsilon\})$, and if $(q, X, a, q', \gamma) \in \Delta$ for $X \in \Gamma$, then $\gamma \in \Gamma^{\leqslant 2}$. Given a transition $\tau = (q, X, a, q', \gamma)$ let $\ell(\tau) = a \in \Sigma_\varepsilon$. We say that τ is an $\ell(\tau)$-transition and that τ is a Σ-transition, if $\ell(\tau) \in \Sigma$. For a finite sequence ρ over Δ, the word $\ell(\rho) \in \Sigma^*$ is defined by applying ℓ homomorphically to every transition. We take the size of $\mathcal{P}$ to be $|Q| + |\Gamma|$.[2]

A stack content is a finite word in $\perp\Gamma^*$ (i.e. the top of the stack is at the end) and a configuration $c = (q, \gamma)$ of $\mathcal{P}$ consists of a state $q \in Q$ and a stack content γ. The initial configuration is $(q_I, \perp)$.

The set of modes of $\mathcal{P}$ is $Q \times \Gamma_\perp$. A mode (q, X) enables all transitions of the form (q, X, a, q', γ') for some $a \in \Sigma_\varepsilon$, $q' \in Q$, and $\gamma' \in \Gamma_\perp^{\leqslant 2}$. The mode of a configuration $c = (q, \gamma X)$ is (q, X). A transition τ is enabled by c if it is enabled by c's mode. In this case, we write $(q, \gamma X) \xrightarrow{\tau} (q', \gamma\gamma')$, where $\tau = (q, X, a, q', \gamma')$.

---

[2]Note that we prove exponential succinctness gaps, so the exact definition of the size is irrelevant, as long as it is polynomial in $|Q|$ and $|\Gamma|$. Here, we pick the sum for the sake of simplicity.
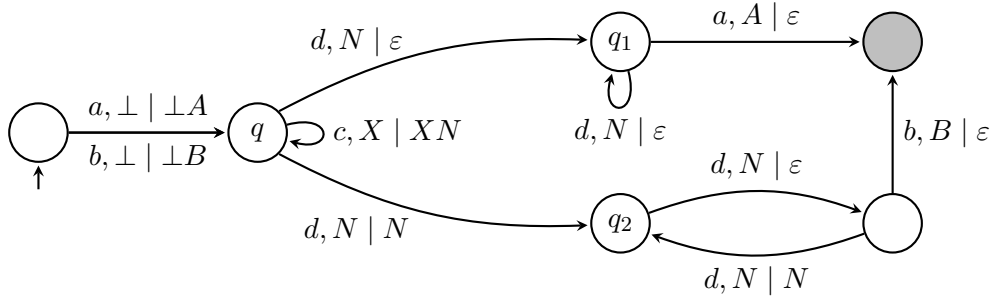
Figure 1: The PDA $\mathcal{P}$ from Example 2.2. Grey states are final, and X is an arbitrary stack symbol.

A run of $\mathcal{P}$ is a finite sequence $\rho = c_0\tau_0 c_1\tau_1 \cdots c_{n-1}\tau_{n-1}c_n$ of configurations and transitions with c_0 being the initial configuration and $c_{n'} \xrightarrow{\tau_{n'}} c_{n'+1}$ for every $n' < n$. The run ρ is a run of $\mathcal{P}$ on $w \in \Sigma^*$, if $w = \ell(\rho)$. We say that ρ is accepting if it ends in a configuration whose state is final. The language $L(\mathcal{P})$ recognized by $\mathcal{P}$ contains all $w \in \Sigma^*$ such that $\mathcal{P}$ has an accepting run on w.

Remark 2.1. Let $c_0\tau_0 c_1\tau_1 \cdots c_{n-1}\tau_{n-1}c_n$ be a run of $\mathcal{P}$. Then, the sequence $c_0 c_1 \cdots c_{n-1}c_n$ of configurations is uniquely determined by the sequence $\tau_0\tau_1 \cdots \tau_{n-1}$ of transitions. Hence, whenever convenient, we treat a sequence of transitions as a run if it indeed induces one (not every such sequence does induce a run, e.g. if a transition $\tau_{n'}$ is not enabled in $c_{n'}$).

We say that a PDA $\mathcal{P}$ is deterministic (DPDA) if
- every mode of $\mathcal{P}$ enables at most one a-transition for every $a \in \Sigma \cup \{\varepsilon\}$, and
- for every mode of $\mathcal{P}$, if it enables some ε-transition, then it does not enable any Σ-transition.

Hence, for every input and for every run prefix on it there is at most one enabled transition to continue the run. Still, due to the existence of ε-transitions, a DPDA can have more than one run on a given input. However, these only differ by trailing ε-transitions.

The class of languages recognized by PDA is denoted by CFL, the class of languages recognized by DPDA by DCFL.

Example 2.2. The PDA $\mathcal{P}$ depicted in Figure 1 recognises the language $\{ac^n d^n a \mid n \geqslant 1\} \cup \{bc^n d^{2n} b \mid n \geqslant 1\}$. Note that while $\mathcal{P}$ is nondeterministic, $L(\mathcal{P})$ is in DCFL.

3. History-deterministic Pushdown Automata

Here, we introduce history-deterministic pushdown automata on finite words (HD-PDA for short), nondeterministic pushdown automata whose nondeterminism can be resolved based on the run prefix constructed so far and on the next input letter to be processed, but independently of the continuation of the input beyond the next letter.

As an example, consider the PDA $\mathcal{P}$ from Example 2.2. It is nondeterministic, but knowing whether the first transition of the run processed an a or a b allows the nondeterminism to be resolved in a configuration of the form $(q, \gamma N)$ when processing a d: in the former case, take the transition to state q_1, in the latter case the transition to state q_2. Afterwards, there are no nondeterministic choices to make and the resulting run is accepting whenever the input is in the language. This automaton is therefore history-deterministic.

We implement this intuition with the notion of a *resolver*, that is, a function that, given the run so far, and the next input letter, produces the sequence of transitions (ε and otherwise) to be taken to process this letter. After the last letter has been processed, the run produced so far must be accepting, without trailing ε-transitions. We will show that this is not a serious restriction at the end of the section.

Fix a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$. We say that a run $c_0 \tau_0 \cdots \tau_n c_{n+1}$ of $\mathcal{P}$ processing some $w \in \Sigma^*$ has no trailing ε-transitions if

(1) $n = -1$ if $w = \varepsilon$, and
(2) $\ell(\tau_0 \cdots \tau_{n-1})$ is a strict prefix of w, if $w \neq \varepsilon$.

A (nondeterminism) resolver for $\mathcal{P}$ is a function $r \colon \Delta^* \times \Sigma \to \Delta$ such that for every $w \in L(\mathcal{P})$, there is an accepting run $\rho = c_0 \tau_0 \cdots \tau_n c_{n+1}$ on w that has no trailing ε-transitions and satisfies $\tau_{n'} = r(\tau_0 \cdots \tau_{n'-1}, w(|\ell(\tau_0 \cdots \tau_{n'-1})|))$ for all $0 \leqslant n' < n$. If w is nonempty, then $w(|\ell(\tau_0 \cdots \tau_{n'-1})|)$ is defined for all $0 \leqslant n' < n$, as ρ has no trailing ε-transitions. Note that ρ is unique if it exists. We say that $\mathcal{P}$ is history-deterministic (HD) if it has a resolver. We denote the class of languages recognised by HD-PDA by HD-CFL.

Note that the prefix processed so far can be recovered from r's input, i.e. it is $\ell(\rho)$. However, the converse is not true due to the existence of ε-transitions. This is the reason that the run prefix and not the input prefix is the argument for the resolver.

We require the run induced by a resolver to have no trailing ε-transitions, as a resolver requires the next letter to be processed as part of its input. This is obviously undefined once the input has ended. Nevertheless, at the end of this section, we study a variant of resolvers with end-of-word markers, which will allow trailing ε-transitions, showing that they do not add expressiveness.

Intuitively, every DPDA *should* be HD, as there is no nondeterminism to resolve during a run. However, in order to reach a final state, a run of a DPDA on some input w may traverse *trailing* ε-transitions after the last letter of w is processed. On the other hand, the run of an HD-PDA on w consistent with any resolver has to end with the transition processing the last letter of w. Hence, not every DPDA recognises the same language when viewed as an HD-PDA. Nevertheless, we show, using standard pushdown automata constructions, that every DPDA can be turned into an equivalent HD-PDA. As every HD-PDA is a PDA by definition, we obtain a hierarchy of languages.

Lemma 3.1. *DCFL $\subseteq$ HD-CFL $\subseteq$ CFL.*

Proof. We only consider the first inclusion, as the second one is trivial. So, let $L \in$ DCFL, say it is recognised by the DPDA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$. We say that a mode m of $\mathcal{P}$ is a reading mode if it does not enable an ε-transition. Hence, due to determinism, a reading mode m can only enable at most one a-transition for every $a \in \Sigma$.

Now, consider some nonempty word $w(0) \cdots w(n) \in L(\mathcal{P})$ (we take care of the empty word later on), say with accepting run ρ (treated, for notational convenience, as a sequence of transitions). This run can be decomposed as

$$\rho = \rho_0 \, \tau_0 \, \rho_1 \, \tau_1 \, \rho_2 \, \cdots \, \rho_n \, \tau_n \, \rho_{n+1}$$

where τ_i processes $w(i)$ and each ρ_i is a (possibly empty) sequence of ε-transitions. Each run prefix induced by some $\rho_0 \tau_0 \rho_1 \cdots \rho_i$ ends in a configuration with reading mode.

Intuitively, we have to eliminate the trailing ε-transitions in ρ_{n+1}. To do so, we postpone the processing of each letter $w(i)$ to the end of ρ_{i+1}. Instead, we guess that the next input is $w(i)$ by turning the original $w(i)$-transition τ_i into an ε-transition τ_i' that stores $w(i)$ in the

state space of the modified automaton. Then, ρ_{i+1} is simulated and a dummy transition τ_i^d processing the stored letter $w(i)$ is executed.

Hence, the resulting run of the modified automaton on w has the form

$$\rho_0 \, \tau_0' \, \rho_1 \, \tau_0^d \, \tau_1' \, \rho_2 \, \tau_1^d \, \cdots \, \rho_n \, \tau_{n-1}^d \, \tau_n' \, \rho_{n+1} \, \tau_n^d,$$

where each τ_i' is now an ε-transition, each ρ_i is a (possibly empty) sequence of ε-transitions, and each τ_i^d is a dummy transition processing $w(i)$. Hence, the run ends with the transition processing the last letter $w(n)$ of the input.

The resulting PDA is HD, as a resolver has access to the next letter to be processed, which is sufficient to resolve the nondeterminism introduced by the guessing of the next letter.

More formally, consider the PDA $\mathcal{P}' = (Q', \Sigma, \Gamma, q_I', \Delta', F')$ where

- $Q' = Q \cup (Q \times \Sigma)$,
- $q_I' = q_I$,
- $F' = F \cup I$ where $I = \{q_I\}$ if $\varepsilon \in L(\mathcal{P})$ and $I = \emptyset$, otherwise, and
- Δ' is the union of the following sets of transitions:
  - $\{\tau \in \Delta \mid \ell(\tau) = \varepsilon\}$, which is used to simulate the leading sequence of ε-transitions before the first letter is processed by $\mathcal{P}$, i.e. the transitions in ρ_0 above.
  - $\{(q, X, \varepsilon, (q', a), \gamma) \mid (q, X, a, q', \gamma) \in \Delta \text{ and } a \in \Sigma\}$, which are used to guess and store the next letter to be processed, i.e. the transitions τ_i' above.
  - $\{((q, a), X, \varepsilon, (q', a), \gamma) \mid (q, X, \varepsilon, q', \gamma) \in \Delta\}$, which are used to simulate ε-transitions after a letter has been guessed, but not yet processed, i.e. transitions in some ρ_i with $i > 0$ above.
  - $\{((q, a), X, a, q, X) \mid (q, X) \text{ is a reading mode}\}$, the dummy transitions used to actually process the guessed and stored letter.

Now, formalising the intuition given above, one can show that $\mathcal{P}'$ has a resolver witnessing that it recognises $L(\mathcal{P})$. In particular, the empty word is in $L(\mathcal{P}')$ if and only if it is in $L(\mathcal{P})$, as the run induced by the resolver on ε ends in the initial configuration, which is final if and only if $\varepsilon \in L(\mathcal{P})$. $\qquad\square$

HD-PDA are by definition required to end their run with the last letter of the input word. Instead, one could also consider a model where they are allowed to take some trailing ε-transitions after the last input letter has been processed. As a resolver has access to the next input letter, which is undefined in this case, we need resolvers with end-of-word markers to signal the resolver that the last letter has been processed. In the following, we show that HD-PDA with end-of-word resolvers are as expressive as standard HD-PDA, albeit exponentially more succinct.

Fix some distinguished end-of-word-marker $\#$, which takes the role of the next input letter to be processed, if there is none after the last letter of the input word is processed. Let $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ be a PDA with $\# \notin \Sigma$. An EoW-resolver for $\mathcal{P}$ is a function $r \colon \Delta^* \times (\Sigma \cup \{\#\}) \to \Delta$ such that for every $w \in L(\mathcal{P})$, there is an accepting run $c_0 \tau_0 \cdots \tau_n c_n$ on w such that $\tau_{n'} = r(\tau_0 \cdots \tau_{n'-1}, w\#(|\ell(\tau_0 \cdots \tau_{n'-1})|))$ for all $0 \leqslant n' < n$. Note that the second argument given to the resolver is a letter of $w\#$, which is equal to $\#$ if the run prefix induced by $\tau_0 \cdots \tau_{n'-1}$ has already processed the full input w.

Lemma 3.2. *HD-PDA with EoW-resolvers are as expressive as HD-PDA.*

Proof. A (standard) resolver can be turned into an EoW-resolver that ignores the EoW-marker. Hence, every HD-PDA is an HD-PDA with EoW-resolver recognizing the same language. So, it only remains to consider the other inclusion.

To this end, let $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ be a PDA with EoW-resolver. The language

$$C_{acc} = \{\gamma q \mid q \in F \text{ and } \gamma \in \perp\Gamma^*\} \subseteq \perp\Gamma^* Q$$

encoding final configurations of $\mathcal{P}$ is regular. Hence, the language

$$C = \{\gamma q \in \perp\Gamma^* Q \mid \text{ there is a run infix } (q, \gamma)\tau_0 \cdots \tau_{n-1} c_n$$
$$\text{with } \ell(\tau_0 \cdots \tau_{n-1}) = \varepsilon \text{ and } c_n \in C_{acc}\}$$

can be shown to be regular as well by applying saturation techniques [Bü64][3] to the restriction of $\mathcal{P}$ to ε-transitions. If $\mathcal{P}$ reaches a configuration $c \in C$ after processing an input w, then $w \in L$, even if c's state is not final.

Let $\mathcal{A} = (Q_{\mathcal{A}}, \Gamma_{\perp} \cup Q, q_I^{\mathcal{A}}, \delta_{\mathcal{A}}, F_{\mathcal{A}})$ be a DFA recognizing C. We extend the stack alphabet of $\mathcal{P}$ to $\Gamma \times Q_{\mathcal{A}} \times (Q_{\mathcal{A}} \cup \{u\})$, where u is a fresh symbol. Then, we extend the transition relation such that it keeps track of the unique run of $\mathcal{A}$ on the stack content: If $\mathcal{P}$ reaches a stack content $\perp(X_1, q_1, q_1')(X_2, q_2, q_2') \cdots (X_s, q_s, q_s')$, then we have

$$q_j = \delta_{\mathcal{A}}^*(q_I^{\mathcal{A}}, \perp X_1 \cdots X_j)$$

for every $1 \leqslant j \leqslant s$ as well as $q_j' = q_{j-1}$ for every $2 \leqslant j \leqslant s$ and $q_1' = u$. Here, $\delta_{\mathcal{A}}^*$ is the standard extension of $\delta_{\mathcal{A}}$ to words. The adapted PDA is still HD, as no new nondeterminism has been introduced, and keeps track of the state of $\mathcal{A}$ reached by processing the stack content as well as the shifted sequence of states of $\mathcal{A}$, which is useful when popping the top stack symbol: If the topmost stack symbol (X, q, q') is popped of the stack then q' is the state of $\mathcal{A}$ reached when processing the remaining stack.

Now, we double the state space of $\mathcal{P}$, making one copy final, and adapt the transition relation again so that a final state is reached whenever $\mathcal{P}$ would reach a configuration in C. Whether a configuration in C is reached can be determined from the current state of $\mathcal{P}$ being simulated, as well as the top stack symbol containing information on the run of $\mathcal{A}$ on the current stack content. The resulting PDA $\mathcal{P}'$ recognises $L(\mathcal{P})$ and has on every word $w \in L(\mathcal{P})$ an accepting run without trailing ε-transitions. Furthermore, an EoW-resolver for $\mathcal{P}$ can be turned into a (standard) resolver for $\mathcal{P}'$, as the tracking of stack contents and the doubling of the state space does not introduce nondeterminism. $\square$

As $\mathcal{A}$ has at most exponential size, $\mathcal{P}'$ is also exponential (both in the size of $\mathcal{P}$). This exponential blowup incurred by removing the end-of-word marker is in general unavoidable. In Theorem 5.1, we show that the language L_n of bit strings whose n^{th} bit from the end is a 1 requires exponentially-sized HD-PDA. On the other hand, it is straightforward to devise a polynomially-sized HD-PDA $\mathcal{P}_{\text{EoW}}$ with EoW-marker recognizing L_n: the underlying PDA stores the input word on the stack, guesses nondeterministically that the word has ended, uses n (trailing) ε-transitions to pop of the last $n - 1$ letters stored on the stack, and then checks that the topmost stack symbol is a 1. With an EoW-resolver, the end of the input does not have to be guessed, but is marked by the EoW-marker. Hence, $\mathcal{P}_{\text{EoW}}$ is HD.

Finally, let us remark that the history-determinism of PDA and context-free languages is undecidable. These problems were shown to be undecidable for ω-HD-PDA and ω-HD-CFL by reductions from the inclusion and universality problem for PDA on finite words [LZ22,

---

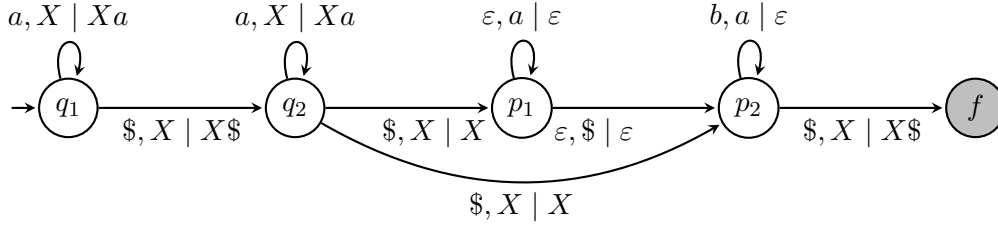[3]Also, see the survey by Carayol and Hague [CH14] for more details.

Figure 2: A PDA $\mathcal{P}_{B_2}$ recognising B_2. Grey states are final, and X is an arbitrary stack symbol.

Theorem 6.1]. The same reductions also show that these problems are undecidable over PDA on finite words.

Theorem 3.3. *The following problems are undecidable:*

(1) *Given a PDA $\mathcal{P}$, is $\mathcal{P}$ an HD-PDA?*
(2) *Given a PDA $\mathcal{P}$, is $L(\mathcal{P}) \in$ HD-CFL?*

4. Expressiveness

Here we show that HD-PDA are more expressive than DPDA but less expressive than PDA.

Theorem 4.1. *DCFL $\subsetneq$ HD-CFL $\subsetneq$ CFL.*

To show that HD-PDA are more expressive than deterministic ones, we consider the language $B_2 = \{a^i \$ a^j \$ b^k \$ \mid k \leqslant \max(i, j)\}$. It is recognised by the PDA $\mathcal{P}_{B_2}$ depicted in Figure 2, hence $B_2 \in$ CFL. We show that $B_2 \in$ HD-CFL by proving that $\mathcal{P}_{B_2}$ is history-deterministic: the only nondeterministic choice, between moving to p_1 or to p_2 upon reading the second \$, can be made only based on the prefix $a^i \$ a^j$ processed so far, which deterministically lead to q_2.

Lemma 4.2. *$B_2 \in$ HD-CFL.*

Proof. Let us summarise the behaviour of the pushdown automaton $\mathcal{P}_{B_2}$ recognising B_2 (see Figure 2). First, the automaton copies the two blocks of a's on the stack. Then, when it processes the second \$, it transitions nondeterministically to either p_1 or p_2. In p_1, it erases the second a-block from the stack, so that the first block is at the top of the stack, and then transitions to p_2. In p_2, the automaton compares the number of b's in the input with the number of a's in the topmost block of the stack. If the latter is larger than or equal to the former, $\mathcal{P}_{B_2}$ pops one a for each b in the input, and then transitions to the final state when it processes the third \$.

When processing the second \$, knowing whether the first or second block of the prefix contains more a's allows the nondeterminism to be resolved: if the first block contains more a's, take the transition to the state p_1, if the second block contains more a's, take the transition to the state p_2. $\qquad \square$

Now, in order to show that B_2 is not in $DCFL$, we prove that its complement B_2^c is not a context-free language. Since $DCFL$ is closed under complementation, this implies the desired result.

Lemma 4.3. *The complement B_2^c of B_2 is not in CFL.*

Proof. Assume, for the sake of contradiction, that the complement B_2^c of B_2 is in CFL. Now consider the regular language

$$A = \{a^i\$a^j\$b^k\$ | i, j, k \in \mathbb{N}\}.$$

Since the intersection of a context-free language and a regular language is context-free, we have that $B_2^c \cap A \in CFL$. Therefore, $B_2^c \cap A$ satisfies the pumping lemma for context-free languages (see, e.g. [HU79]): there exists $m \in \mathbb{N}$ such that the word $z = a^m\$a^m\$b^{m+1}\$ \in B_2^c \cap A$ can be decomposed as $z = uvwxy$ such that

(1) $|vx| \geqslant 1$;
(2) $uv^n wx^n y \in B_2^c \cap A$ for every $n \geqslant 0$.

Note that Item 2 directly implies that both v and x are in the language $\{a\}^* \cup \{b\}^*$, as otherwise $uv^2 wx^2 y$ is not in A. On top of that, Item 1 implies that either v or x is in $\{a\}^+ \cup \{b\}^+$. We conclude by proving, through a case distinction, that Item 2 cannot hold as either uwy or $uv^2 wx^2 y$ is in B_2.

- Assume that neither v nor x is in $\{b\}^+$. Then either v or x is in $\{a\}^+$, hence $uv^2 wx^2 y = a^{m_1}\$a^{m_2}\$b^{m+1}\$$ for some $m_1, m_2 \geqslant m$ such that either $m_1 > m$ or $m_2 > m$. In both cases, we get $uv^2 wx^2 y \in B_2$.
- Assume that either v or x is in $\{b\}^+$. Then pumping v and x down in z reduces the size of at most one of the a-blocks: we have that $uwy = a^{m_1}\$a^{m_2}\$b^{m_3+1}\$$ for $m_1, m_2, m_3 \leqslant m$ such that $m_3 < m$, and either $m_1 = m$ or $m_2 = m$. In both cases, we get $uwy \in B_2$.

As every possible case results in a contradiction, our initial hypothesis is false: $B_2^c \notin \text{CFL}$. $\square$

The previous two lemmata and Lemma 3.1 yield DCFL $\subsetneq$ HD-CFL.

Finally, to show that PDA are more expressive than HD-PDA, we consider the language $L = \{a^n b^n \mid n \geqslant 0\} \cup \{a^n b^{2n} \mid n \geqslant 0\}$. We note that $L \in$ CFL while we show below $L \notin$ HD-CFL using arguments similar to the classical proof showing that L is not DCFL. Hence, the following lemma completes the proof of Theorem 4.1.

Lemma 4.4. $L \notin$ *HD-CFL.*

Proof. We show that there does not exist an HD-PDA recognising L. In fact, we show that if there exists an HD-PDA $\mathcal{P}$ recognising L, then we can construct a PDA $\widehat{\mathcal{P}}$ recognising the language $\widehat{L} = L \cup \{a^n b^n c^n \mid n \geqslant 0\}$. A straightforward application of the pumping lemma for context-free languages (see, e.g. [HU79]) to words of the form $a^n b^n c^n$ shows that $\widehat{L}$ is not in CFL. Thus, we reach a contradiction.

The idea behind the construction is to replicate the part of the control unit of $\mathcal{P}$ which processes the suffix b^n of an input word $a^n b^{2n}$ with the difference that in the newly added parts, the transitions caused by input symbol b are replaced with similar ones for input symbol c. This new part of the control unit may be entered after $\widehat{\mathcal{P}}$ has processed $a^n b^n$.

We now construct the PDA $\widehat{\mathcal{P}}$ from $\mathcal{P}$ as follows. Let $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ with $Q = \{q_0, q_1, \ldots, q_n\}$, and let $q_0 = q_I$. Now consider $\widehat{\mathcal{P}} = (Q \cup \widehat{Q}, \Sigma, \Gamma, q_I, \Delta \cup \widehat{\Delta}, F \cup \widehat{F})$ with $\widehat{Q} = \{\widehat{q}_0, \widehat{q}_1, \ldots, \widehat{q}_n\}$, $\widehat{F} = \{\widehat{q}_i \mid q_i \in F\}$, and $\widehat{\Delta}$ includes the following additional transitions:

1. $\{(q_f, X, \varepsilon, \widehat{q}_f, X) \mid q_f \in F, X \in \Gamma_\perp\}$: switch from the original final states to the new states.
2. $\{(\widehat{q}_i, X, c, \widehat{q}_j, \gamma) \mid (q_i, X, b, q_j, \gamma) \in \Delta\}$: replicate the original b-transitions by c-transitions in the new states.

3. $\{(\widehat{q_i}, X, \varepsilon, \widehat{q_j}, \gamma) \mid (q_i, X, \varepsilon, q_j, \gamma) \in \Delta\}$: replicate all ε-transitions.

Now we show that $L(\widehat{\mathcal{P}}) = \widehat{L}$. First we show that $L(\widehat{\mathcal{P}}) \subseteq \widehat{L}$. Consider a word $w \in L(\widehat{\mathcal{P}})$. There may be two cases:

(i) Assume $\widehat{\mathcal{P}}$ has an accepting run on w that does not visit a state in $\widehat{Q}$. In this case, we have that w is in $L(\mathcal{P}) = L \subseteq \widehat{L}$.

(ii) Assume there exists an accepting run of $\widehat{\mathcal{P}}$ on w that visits a state in $\widehat{Q}$. Since $\mathcal{P}$ recognises L, and by construction of $\widehat{\mathcal{P}}$, a state $\widehat{q_i} \in \widehat{Q}$ can be reached from a state $q_i \in Q$ only if $\widehat{q_i} \in \widehat{F}$ and $q_i \in F$, and the corresponding transition is an ε-transition, we have that starting from the initial configuration $(q_I, \bot)$, a state in $\widehat{Q}$ is reached for the first time only after processing an input prefix $a^n b^n$ or $a^n b^{2n}$ for some $n \geqslant 0$. If this prefix of w is $a^n b^{2n}$, then $w = a^n b^{2n}$. This is because if $w = a^n b^{2n} c^m$ for some $m > 0$ (recall that after visiting a state $\widehat{q_i}$ in $\widehat{Q}$, the only non-ε transitions possible are on the letter c), then by the construction of $\widehat{\mathcal{P}}$, we have that $\mathcal{P}$ can accept the word $a^n b^{2n} b^m$ which is not in the language L. On the other hand, let the prefix be $a^n b^n$ when a state $\widehat{q_i} \in \widehat{Q}$ is visited for the first time. Note that $\widehat{q_i} \in \widehat{F}$, and let $(\widehat{q_i}, \gamma_i)$ be the corresponding configuration. If a sequence of transitions $\widehat{\tau_i}, \ldots, \widehat{\tau_j}$ from $(\widehat{q_i}, \gamma_i)$ to $(\widehat{q_j}, \gamma_j)$ is possible such that not all of $\widehat{\tau_i}, \ldots, \widehat{\tau_j}$ are ε-transitions, that is, the transitions process c^m for some $m \in \mathbb{N}$, and $\widehat{q_j} \in \widehat{F}$, then a sequence of transitions $\tau_i, \ldots, \tau_j$ of the same length processing b^m is possible from (q_i, γ_i) to (q_j, γ_j) with $q_j \in F$. Since this leads to an accepting run from $(q_I, \bot)$ to (q_j, γ_j) while visiting only the states in Q on processing $a^n b^n b^m$ with $m > 0$, we have $m = n$, and hence $w = a^n b^n c^n \in \widehat{L}$.

On the other hand, if all transitions $\widehat{\tau_i}, \ldots, \widehat{\tau_j}$ are ε-transitions, then $w = a^n b^n \in \widehat{L}$.

Now we prove the other direction, that is $\widehat{L} \subseteq L(\widehat{\mathcal{P}})$. Here, we rely on the fact that the accepting run of $\mathcal{P}$ on $a^n b^n$ induced by a resolver r is a prefix of the accepting run of $\mathcal{P}$ on $a^n b^{2n}$ induced by r. This allows to switch to the copied states $\widehat{Q}$ after processing $a^n b^n$ and then process c^n instead of b^n.

Consider a word $w \in \widehat{L}$ such that $w \in L$. By construction of $\widehat{\mathcal{P}}$, we have that $w \in L(\widehat{\mathcal{P}})$ since $\widehat{\mathcal{P}}$ accepts all words that are also accepted by $\mathcal{P}$. Now suppose that $w \in \widehat{L}$ but $w \notin L$, that is, w is of the form $a^n b^n c^n$ for some $n \geqslant 1$. Since by assumption, we have that $\mathcal{P}$ is an HD-PDA recognising the language L, there exists a resolver r that for every word in L induces an accepting run of the word in L. Let (q_i, γ_i) be the configuration of $\mathcal{P}$ reached after processing the prefix $a^n b^n$ in the run induced by r on the input $a^n b^{2n}$.

Note that $q_i \in F$ since r also induces an accepting run for the input $a^n b^n$. Now if for the input $a^n b^{2n}$, the sequence of transitions chosen by r from (q_i, γ_i) after processing $a^n b^n$ is $\tau_i, \tau_{i+1}, \ldots, \tau_j$ with $(q_i, \gamma_i) \xrightarrow{\tau_i} (q_{i+1}, \gamma_{i+1}) \cdots (q_{j-1}, \gamma_{j-1}) \xrightarrow{\tau_j} (q_j, \gamma_j)$, with $q_j \in F$, and the sequence $\tau_i, \ldots, \tau_j$ processes b^n, then by the construction of $\widehat{\mathcal{P}}$, there exists a sequence of transitions $\widehat{\tau_i}, \widehat{\tau_{i+1}}, \ldots, \widehat{\tau_j}$ with $(\widehat{q_i}, \gamma_i) \xrightarrow{\widehat{\tau_i}} (\widehat{q_{i+1}}, \gamma_{i+1}) \cdots (\widehat{q_{j-1}}, \gamma_{j-1}) \xrightarrow{\widehat{\tau_j}} (\widehat{q_j}, \gamma_j)$ and with $\widehat{q_j} \in \widehat{F}$ such that there is an ε-transition from (q_i, γ_i) to $(\widehat{q_i}, \gamma_i)$ and the sequence $\widehat{\tau_i}, \widehat{\tau_{i+1}}, \ldots, \widehat{\tau_j}$ processes c^n, and hence $w \in L(\widehat{\mathcal{P}})$.

Thus we have that $\widehat{L} = L(\widehat{\mathcal{P}})$. Hence we show that if $\mathcal{P}$ is an HD-PDA, then we can construct a PDA $\widehat{\mathcal{P}}$ recognising $\widehat{L}$ which is not a CFL, thus leading to a contradiction to our assumption that L is in HD-CFL. □

Unambiguous context-free languages, i.e. those generated by grammars for which every word in the language has a unique leftmost derivation, are another class sitting between DCFL and CFL. Thus, it is natural to ask how unambiguity and history-determinism are related: To conclude this section, we show that both notions are independent.

Theorem 4.5. *There is an unambiguous context-free language that is not in HD-CFL and a language in HD-CFL that is inherently ambiguous.*

An unambiguous grammar for the language $\{a^n b^n \mid n \geqslant 0\} \cup \{a^n b^{2n} \mid n \geqslant 0\} \notin$ HD-CFL is easy to construct and we show that the language $B = \{a^i b^j c^k \mid i, j, k \geqslant 1, k \leqslant \max(i, j)\}$ is inherently ambiguous. Its inclusion in HD-CFL is easily established using a similar argument as for the language $B_2 = \{a^i \$ a^j \$ b^k \$ \mid k \leqslant \max(i, j)\}$ above. The dollars add clarity to the HD-PDA but are cumbersome in the proof of inherent ambiguity.

We show that $B = \{a^i b^j c^k \mid i, j, k \geqslant 1, k \leqslant \max(i, j)\}$ is inherently ambiguous, i.e. for every grammar generating B there is at least one word that has two different leftmost derivations.

We use standard definitions and notation for context-free grammars as in [HU79, Section 4.2]. We say that a grammar is reduced, if every variable is reachable from the start variable, every variable can be reduced to a word of terminals, and for no variable A, it holds that $A \overset{*}{\Longrightarrow} A$.

Let $D(G) = \{A \in V \mid A \overset{*}{\Longrightarrow} xAy \text{ for some } x, y \text{ with } xy \neq \varepsilon\}$. An unambiguous CFG G is called *almost-looping*, if

(1) G is reduced,
(2) all variables, possibly other than the start variable S, belong to $D(G)$, and
(3) either $S \in D(G)$ or S occurs only once in the leftmost derivation of any word in $L(G)$.

Now we state the following lemma from [Mau69].

Lemma 4.6. *For every unambiguous CFG G, there exists an unambiguous almost-looping CFG G' such that $L(G) = L(G')$.*

An example of an almost-looping grammar for language B is the following:

$$S \to S_1 \mid S_2 \qquad\qquad S_2 \to aS_2 \mid aD$$
$$S_1 \to aS_1 \mid aS_1 c \mid aBc \qquad D \to bDc \mid bD \mid bc$$
$$B \to bB \mid b$$

Figure 3: An example CFG for language B

Now we prove the following, using techniques inspired by Maurer's proof that $\{a^i b^j c^k \mid i, j, k \geqslant 1, i = j \lor j = k\}$ is inherently ambiguous [Mau69].

Lemma 4.7. *The language B is inherently ambiguous.*

Proof. Assume, towards a contradiction, that G is an unambiguous grammar for B, which, from Lemma 4.6, we can assume, without loss of generality, to be an almost-looping grammar. Let A be a variable of G.

(1) A is of Type 1 if there is a derivation $A \overset{*}{\Longrightarrow} xAy$ where $xy = a^{n_{A,1}}$ for some $n_{A,1} > 0$.
(2) A is of Type 2 if there is a derivation $A \overset{*}{\Longrightarrow} xAy$ where $xy = b^{n_{A,2}}$ for some $n_{A,2} > 0$.
(3) A is of Type 3 if there is a derivation $A \overset{*}{\Longrightarrow} xAy$ where $x = a^{\ell_{A,3}}$ and $y = c^{r_{A,3}}$ for some $\ell_{A,3} \geqslant r_{A,3} > 0$.

(4) A is of Type 4 if there is a derivation $A \overset{*}{\Longrightarrow} xAy$ where $x = b^{\ell_{A,4}}$ and $y = c^{r_{A,4}}$ for some $\ell_{A,4} \geqslant r_{A,4} > 0$.

(5) A is of Type 5 if there is a derivation $A \overset{*}{\Longrightarrow} xAy$ where $x = a^{\ell_{A,5}}$ and $y = b^{r_{A,5}}$ for some $\ell_{A,5}, r_{A,5} > 0$.

Note that some variables may be of multiple types (e.g. the variable D in Figure 3 has Type 2 and Type 4).

First, we show that each variable in $D(G)$ has at least one of these five types. So, let $A \in D(G)$. Then, there exists a derivation $A \overset{*}{\Longrightarrow} xAy$ with $xy \neq \varepsilon$. Note that both x and y belong to a^*, b^*, or c^* since otherwise, due to G being reduced, one could derive words that are not in the language. Next, we note that the cases where x belongs to c^*, and y belongs to a^* or b^* cannot happen. Similarly, the case where x belongs to b^*, and y belongs to a^* cannot happen. Also we cannot have xy in c^*, since this will allow us to have words with an arbitrary number of c's which can be more than the number of a's and b's and such a word is not in the language.

Further, we cannot have $x = a^\ell$ and $y = c^r$ with $0 < \ell < r$. Otherwise, consider a derivation of some word in B that uses A, i.e.

$$S \overset{*}{\Longrightarrow} \alpha A \beta \overset{*}{\Longrightarrow} a^s b^u c^v \quad \text{with } v \leqslant \max(s, u).$$

Now, towards a contradiction assume we indeed have

$$A \overset{*}{\Longrightarrow} xAy \quad \text{with } x = a^\ell, y = c^r \text{ and } \ell < r.$$

Then, pumping q copies of x and y, for some suitable $q \in \mathbb{N}$, yields a derivation

$$S \overset{*}{\Longrightarrow} \alpha A \beta \overset{*}{\Longrightarrow} \alpha x^q A y^q \beta \overset{*}{\Longrightarrow} a^{s+\ell q} b^u c^{v+rq} \quad \text{such that } v + rq > \max(s + \ell q, u),$$

i.e. we have derived a word that is not in B. Similarly, we cannot have $x = b^\ell$ and $y = c^r$ for some $0 < \ell < r$. Altogether, this implies that A indeed has at least one of the five types stated above.

Moreover, we claim that there is a $t \in \mathbb{N}$ such that the following three properties are true for every word $w \in B$:

Property 1: If w has more than t c's, then the (unique) leftmost derivation of w has the form

$$S \overset{*}{\Longrightarrow} \alpha A \beta \overset{*}{\Longrightarrow} \alpha x A y \beta \overset{*}{\Longrightarrow} w \quad \text{such that } xy \text{ contains a } c.$$

Thus, A has type 3 or type 4.

Property 2: If w has more than t a's, then the (unique) leftmost derivation of w has the form

$$S \overset{*}{\Longrightarrow} \alpha A \beta \overset{*}{\Longrightarrow} \alpha x A y \beta \overset{*}{\Longrightarrow} w \quad \text{such that } xy \text{ contains an } a.$$

Thus, A has type 1, type 3, or type 5.

Property 3: If w has more than t b's, then the (unique) leftmost derivation of w has the form

$$S \overset{*}{\Longrightarrow} \alpha A \beta \overset{*}{\Longrightarrow} \alpha x A y \beta \overset{*}{\Longrightarrow} w \quad \text{such that } xy \text{ contains a } b.$$

Thus, A has type 2, type 4, or type 5.

We prove these properties as follows: we denote by d the *width* of the grammar G which is the maximum number of symbols appearing on the right side of some production rule of G. Further, we denote by m the number of variables appearing in G. We argue that $t = d^{m+1}$ satisfies the three properties above. We focus on Property 1, the two other proofs are similar. Suppose that w contains more than d^{m+1} c's and consider the derivation tree of

that word. The *weight* $\omega(v)$ of a vertex v in the derivation tree is defined as the number of c's in the subtree rooted at v. Hence, the root of the derivation tree has at least weight d^{m+1}. We build a finite path $v_0, v_1, \ldots, v_k$ from the root of this tree to one of its leaves as follows: The initial vertex v_0 is the root and at each step, we choose as successor of v_i its child v_{i+1} with the largest weight. A vertex v_i of this path is *decreasing* if $\omega(v_i) > \omega(v_{i+1})$. There are are at least $m+1$ decreasing vertices on the path because $\omega(v_0) = d^{m+1}$, $\omega(v_k) = 1$, and $\omega(v_{i+1}) \geqslant \frac{1}{d} \cdot \omega(v_i)$. Thus, there are two decreasing vertices on the path that are labelled by the same variable A such that there is a derivation of the form $A \overset{*}{\Longrightarrow} xAy$ with some c in xy.

Let $p > t$ be a positive integer divisible by the least common multiple of the $n_{A,i}, \ell_{A,i}$ and $r_{A,i}$ for all $A \in D(G)$ and $i \in \{1, \ldots, 5\}$, where we define $n_{A,1} = 1$ if A is not of Type 1, and similarly for all other $i > 1$. We show that the word $w = a^{2p}b^{2p}c^{2p} \in B$ has two leftmost derivations.

First consider the derivation of the word $w_b = a^{2p}b^p c^{2p} \in B$. As we have more than t c's in w_b Property 1 shows that the derivation contains a variable of Type 3 or Type 4. Next, we argue that it cannot contain a variable of Type 4: The occurrence of such a variable would allow us to either produce a word that is not in $a^* b^* c^*$ or to inject $b^p c^r$ for $p \geqslant r > 0$ leading to the derivation of $a^{2p}b^{2p}c^{2p+r}$, which is not in the language. Thus, the derivation of w_b uses at least one variable of Type 3. Also, since w_b has $p > t$ b's, Property 3 implies that the (unique) leftmost derivation of w_b has the form

$$S \overset{*}{\Longrightarrow} \alpha A \beta \overset{*}{\Longrightarrow} \alpha x A y \beta \overset{*}{\Longrightarrow} w_b \quad \text{such that } xy \text{ contains a } b.$$

Thus A is a variable of Type 2 or Type 5 (note that we have already ruled out Type 4 above). More precisely, we have that x belongs to a^+ or b^* and $y = b^j$ for some $j \in \mathbb{N}$. Now we show that the case where x belongs to a^+ is not possible. Assume for contradiction that $x = a^i$ for some $i > 0$. Then we also have the derivation

$$S \overset{*}{\Longrightarrow} \alpha A \beta \overset{*}{\Longrightarrow} a^{2p-i}b^{p-j}c^{2p} \notin B.$$

Therefore, A is a Type 2 variable that is used in the derivation of w_b, which can be used to inject another b^p, yielding a derivation of w. Thus, we have exhibited a derivation of w that uses a variable of Type 3.

Now consider a derivation of the word $w_a = a^p b^{2p} c^{2p}$. Such a derivation cannot contain a variable of Type 3 since this allows us either to produce a word that is not in $a^* b^* c^*$ or to inject $a^p c^r$ for $p \geqslant r > 0$, leading to the derivation of $a^{2p}b^{2p}c^{2p+r} \notin B$. Further, arguing as above, some variable of Type 1 must appear in the derivation of w_a that is used to obtain sufficient number of a's in the derivation of w_a. Such a variable of Type 1 can be used to inject a^p into w_a which leads to the derivation of w. Thus, we have exhibited a derivation of w that does not contain a variable of Type 3.

Altogether, there are two different leftmost derivations of the word w. Thus, G is not unambiguous, yielding the desired contradiction. □

5. Succinctness

We show that HD-PDA are not only more expressive than DPDA, but also more succinct. Similarly, we show that PDA are more succinct than HD-PDA. Recall that the size of a PDA is the sum of its state-space and stack alphabet.

Theorem 5.1. *HD-PDA can be exponentially more succinct than DPDA, and PDA can be double-exponentially more succinct than HD-PDA.*

We first show that HD-PDA can be exponentially more succinct than DPDA. To this end, we construct a family $(C_n)_{n\in\mathbb{N}}$ of languages such that C_n is recognised by an HD-PDA of size $O(n)$, yet every DPDA recognising C_n has at least exponential size in n.

Let $c_n \in (\${0,1\}^n)^*$ be the word describing an n-bit binary counter counting from 0 to $2^n - 1$. For example, $c_2 = \$00\$01\$10\$11$. We consider the family of languages $C_n = \big\{w \in \{0,1,\$,\#\}^* \mid w \neq c_n\#\big\} \subseteq \{0,1,\$,\#\}^*$ of bad counters. Each C_n is recognised by a DPDA since it is co-finite.

We show that the language C_n is recognised by an HD-PDA of size $O(n)$ and that every DPDA $\mathcal{D}$ recognising C_n has exponential size in n. Observe that this result implies that even HD-PDA that are equivalent to DPDA are not determinisable by pruning. In contrast, for NFA, HDness implies determinisability by pruning [BKS17].

Lemma 5.2. *The language C_n is recognised by an HD-PDA of size $O(n)$.*

Proof. We define a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ that recognises C_n. The automaton $\mathcal{P}$ operates in three phases: a push phase, followed by a check phase, and then a final phase. These phases work as follows. Suppose that $\mathcal{P}$ receives an input $w \in \{0,1,\$,\#\}^*$. During the first phase, $\mathcal{P}$ pushes the input processed onto the stack until the sequence 1^n appears. If it never appears, the input is accepted. During this phase, $\mathcal{P}$ also checks whether the prefix w' of w processed up to this point is a sequence of counter values starting with 0^n, i.e. whether w' is in the language

$$L_c = \{\$d_0\$d_1\$\cdots\$d_m \mid d_0 = 0^n, d_i = 1^n \Leftrightarrow i = m, \text{ and } d_i \in \{0,1\}^n \text{ for all } 1 \leqslant i \leqslant m\}.$$

If $w' \notin L_c$, then $\mathcal{P}$ immediately accepts. Otherwise, $\mathcal{P}$ moves to the second phase. During the check phase, $\mathcal{P}$ pops the stack. At any point, $\mathcal{P}$ can nondeterministically guess that the top symbol of the stack is evidence of bad counting (details are described below). It then accepts the input if the guess was correct. If $\mathcal{P}$ completely pops the stack without correctly guessing an error in the counter, it moves to the final phase. Since the prefix w' processed up to this point ends with the sequence 1^n, if $\mathcal{P}$ now processes any suffix different from a single $\#$, then the input is not equal to $c_n\#$, and can be accepted.

The stack alphabet of $\mathcal{P}$ has constant size 3. The push phase requires $3(n+1)$ states:

- First, $\mathcal{P}$ checks whether $\$0^n$ is a prefix of the input. This can be done with $n+2$ states.
- Then, $\mathcal{P}$ checks whether the following $\{0,1\}^*$ segments are n-bits wide, and only the last one is 1^n. This can be done with $2n+1$ additional states: repeatedly, $\mathcal{P}$ processes $n+1$ symbols, checks whether only the first of them is a $\$$, and keeps track of whether at least one of them is 0.

We now show that $6(n+1)$ additional states are enough for the check phase. To this end, we study the errors that $\mathcal{P}$ needs to check. Note that, to increment the counter correctly, we need to change the value of all the bits starting from the last 0, and leave the previous bits unchanged. Therefore, $\mathcal{P}$ can recognise with $6(n+1)$ states whether the top symbol of the stack does not correspond to a correct counter increment: $\mathcal{P}$ pops the top $n+1$ stack symbols while keeping in memory

- the value of the first symbol popped;
- whether we have not yet popped a $\$$ (there is exactly one $\$$ in the top $n+1$ stack symbols, as the stack content is in L_c), or a $\$$ but no 0 afterwards, or a $\$$ and at least one 0 afterwards.

The input is accepted whenever the first symbol popped and the top stack symbol after popping match yet no 0 has been popped between the $ and the last symbol, or they differ yet at least one 0 has been popped between the $ and the last symbol.

Finally, only three states are needed for the final phase: when the bottom of the stack is reached, $\mathcal{P}$ transitions to a new state, and from there it checks whether the suffix is in the language $\{0, 1, \$, \#\}^* \setminus \{\#\}$.

To conclude, note that $\mathcal{P}$ is history-deterministic: the only nondeterministic choice happens during the check phase, and the resolver knows which symbols of the stack are evidence of bad counting. Note that this choice only depends on the current stack content.   □

As mentioned above, each C_n is recognised by a DPDA; now we show that such a DPDA must be large.

Lemma 5.3. *Every DPDA recognising the language C_n has at least exponential size in n.*

Proof. It is known that every DPDA can be complemented at the cost of multiplying its number of states by three [HU79]. Therefore, to prove the statement, we show that even every PDA recognising the complement $\{c_n\#\}$ of C_n has at least exponential size in n:

Claim 5.4. *Every PDA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ recognising $\{c_n\#\}$ has a size greater than $2^{(n-1)/3}$.*

To prove the claim, we transform $\mathcal{P}$ into a context-free grammar generating the singleton language $\{c_n\#\}$, and then we show that such a grammar requires exponentially many variables. This is a direct consequence of the mk Lemma [CLL$^+$05], but proving it directly using similar techniques yields a slightly better bound.

Before changing $\mathcal{P}$ into a grammar, we slightly modify its acceptance condition: we add to $\mathcal{P}$ a fresh final state f in which the stack can be completely popped *including the bottom of stack symbol $\perp$* (which normally cannot be touched according to our definition of PDA). Moreover, we allow $\mathcal{P}$ to transition towards f nondeterministically from all of its other final states. This new automaton, which accepts by empty stack, is easily transformed into a grammar $\mathcal{G}$ using the standard transformation [HU79]:

- The terminals of $\mathcal{G}$ are 0, 1, $ and #.
- The variables of $\mathcal{G}$ are the triples (p, X, q), for every state $p, q \in Q \cup \{f\}$ and stack symbol $X \in \Gamma_\perp$.
- The initial variable is $(q_I, \perp, f)$, where q_I is the initial state of $\mathcal{P}$ and f is the fresh final state.
- Each transition $(p, X, a, q, \gamma) \in \Delta$ yields production rules as follows:
  (1) If $\gamma = \varepsilon$, then $\mathcal{G}$ has the production rule $(p, X, q) \to a$;
  (2) If $\gamma = Y$, then $\mathcal{G}$ has the production rule $(p, X, q_1) \to a(q, Y, q_1)$ for all $q_1 \in Q$;
  (3) If $\gamma = YZ$, then $\mathcal{G}$ has the production rule $(p, X, q_2) \to a(q, Y, q_1)(q_1, Z, q_2)$ for all $q_1, q_2 \in Q$.

The variables can be interpreted as follows: for every $p, q \in Q$ and $X \in \Gamma$, the variable (p, X, q) can be derived into any input word $w \in \{0, 1, \$, \#\}^*$ that $\mathcal{P}$ can process starting in state p and ending in state q while consuming the symbol X from the top of the stack. Therefore, in particular, since the initial variable is $(q_I, \perp, f)$, $\mathcal{G}$ generates the same language as $\mathcal{P}$.

We now prove that the grammar $\mathcal{G}$ has at least 2^{n-1} distinct variables, hence $(|Q| + 1)^2(|\Gamma| + 1) \geqslant 2^{n-1}$, which implies that the size $|Q| + |\Gamma|$ of $\mathcal{P}$ is at least $\geqslant 2^{(n-1)/3}$. To this end, we study a (directed and ordered) derivation tree T of the word $c_n\#$.

Remember that $c_n = \$d_0\$d_1\$\cdots\$d_{2^n-1}$ represents an n-bit binary counter counting from 0 to $2^n - 1$. For each $0 \leqslant i \leqslant 2^n - 1$, let us consider the vertex v_i of T such that the counter value d_i is an infix of the derivation of v_i, but of none of its children. In other words, d_i is split between the derivations of the children of v_i. By definition of the grammar $\mathcal{G}$, each vertex of T has at most three children, hence at most two counter values can be split amongst the children of a given vertex. Therefore, for every $1 \leqslant i \leqslant 2^n - 2$, while $v_i = v_{i-1}$ or $v_i = v_{i+1}$ might hold, we know that $v_i \neq v_j$ for each $j \in \{1, 2, \ldots, 2^n - 1\} \setminus \{i-1, i, i+1\}$. As a consequence, the vertices $v_0, v_2, v_4, \ldots, v_{2^n-2}$ are all distinct. Finally, since $c_n\#$ is the only word recognised by $\mathcal{P}$ and each counter value $d \in \{0, 1\}^n$ appears a single time as an infix of $c_n\#$, the 2^{n-1} variables labelling these vertices need to be distinct. $\square$

Now, we consider the gap between HD-PDA and PDA. We show that there exists a family $(L_n)_{n>0}$ of languages such that L_n is recognised by a PDA of size $O(\log n)$ while every HD-PDA recognising this language has at least exponential size in n.

Formally, we set $L_n = (0+1)^*1(0+1)^{n-1}$, that is, the nth bit from the end is a 1. Here, we count starting from 1, so that the last bit is the 1st bit from the end. Note that this is the standard example for showing that NFA can be exponentially more succinct than DFA, and has been used for many other succinctness results ever since.

Lemma 5.5. *There exists a PDA of size $O(\log n)$ recognising L_n.*

Proof. We describe a PDA $\mathcal{P}_n$ that recognises L_n. The PDA $\mathcal{P}_n$ nondeterministically guesses the nth bit from the end, checks that it is a 1 and switches to a counting gadget that checks that the word ends in n steps, as follows:

(i) It pushes the binary representation of $n-2$ onto the stack. For example, if $n = 8$, then 110 is pushed onto the stack with 0 at the top. Note that $\log(n-2)$ states suffice for pushing the binary representation of $n-2$. If $n = 1$, then instead of pushing anything onto the stack, the automaton directly moves to a final state without any enabled transitions.

(ii) Then $\mathcal{P}_n$ moves to a state that attempts to decrement the counter by one for each successive input letter, as follows: When an input letter is processed, it pops 0's until 1 is at the top of the stack, say m 0's. Then, it replaces the 1 with a 0, and finally pushes m 1's back onto the stack before processing the next letter. If the stack empties before a 1 is at the top of the stack, then the counter value is 0 and the automaton moves to a final state with no enabled transitions. Note that $O(\log n)$ states again suffice for this step.

Thus, $\mathcal{P}_n$ has $O(\log n)$ states. Note that for all n, $\mathcal{P}_n$ uses only three stack symbols, 0, 1, and $\perp$. Altogether, the size of $\mathcal{P}_n$ is $O(\log n)$, and $\mathcal{P}_n$ recognises L_n. $\square$

Lemma 5.6. *Every HD-PDA recognising L_n has at least exponential size in n.*

Towards proving this, we define the following notions. We say that a word w of length n is *rotationally equivalent* to a word w' if w' is obtained from w by rotating it. For example, the word $w = 1101$ is rotationally equivalent to $w' = 1110$ since w' can be obtained from w by rotating it once to the right. Note that the words that are rotationally equivalent form an equivalence class, and thus rotational equivalence partitions $\{0, 1\}^n$. Since the size of each class is at most n, the number of equivalence classes is at least $\frac{2^n}{n}$.

Now, we define the *stack height* of a configuration $c = (q, \gamma)$ as $\text{sh}(c) = |\gamma| - 1$, and we define *steps* of a run as usual: Consider a run $c_0\tau_0c_1\tau_1\cdots c_{n-1}\tau_{n-1}c_n$. A position $s \in$

$\{0, \ldots, n\}$ is a step if for all $s' \geqslant s$, we have that $\mathrm{sh}(c_{s'}) \geqslant \mathrm{sh}(c_s)$, that is, the stack height is always at least $\mathrm{sh}(c_s)$ after position s. Any infinite run of a PDA has infinitely many steps. We have the following observation.

Remark 5.7. If two runs of a PDA have steps s_0 and s_1, respectively, with the same mode, then the suffix of the run following the step s_0 can replace the suffix of the other run following the step s_1, and the resulting run is a valid run of the PDA.

Now, we are ready to prove Lemma 5.6. Here, we work with infinite inputs for HD-PDA. The run induced by a resolver on such an input is the limit of the runs on the prefixes.

Proof. Let $\mathcal{P}$ be an HD-PDA with resolver r that recognises L_n. We show that $|Q| \cdot |\Gamma| \geqslant \frac{2^n}{n}$, where Q is the set of states and Γ is the stack alphabet of $\mathcal{P}$.

Towards a contradiction, assume that $|Q| \cdot |\Gamma| < \frac{2^n}{n}$. Then there exist two words w_0 and w_1 of length n that are not rotationally equivalent and such that the runs ρ_0 and ρ_1 of $\mathcal{P}$ induced by r on w_0^ω and w_1^ω contain steps with the same mode, at positions s_0 and s_1 in ρ_0 and ρ_1 respectively, such that at least n letters are processed before s_0 and s_1. Now consider in each of these two runs the sequence of input letters of length n preceding and including the step position. Let these n-letter words be w_0' and w_1' respectively. Since w_0 and w_1 are not rotationally equivalent, w_0' and w_1' differ in at least one position $j \leqslant n$.

W.l.o.g., assume that for w_0', the bit at position j is 0, while it is 1 at position j for w_1'. Since the resolver chooses a run such that for every word where the nth letter from the end is a 1 is accepted, this implies that ρ_0 does not visit a final state after processing $j-1$ letters after s_0, while ρ_1 visits a final state after processing $j-1$ letters after s_1.

Now we reach a contradiction as follows. The suffix of ρ_0 starting from position $s_0 + 1$ can be replaced with the suffix of ρ_1 starting from position $s_1 + 1$. By Remark 5.7, this yields a valid run ρ of $\mathcal{P}$. However, since the state that occurs after $j-1$ letters are processed after position s_1 in ρ_1 is final, after the replacement, the state that occurs after $j-1$ letters are processed after position s_0 in ρ is final as well. However the nth letter from the end of the word processed by this accepting run of $\mathcal{P}$ is a 0, contradicting that $\mathcal{P}$ recognises L_n.

Thus, we have that $|Q| \cdot |\Gamma|$ is at least equal to the number of rotationally equivalent classes, that is, $|Q| \cdot |\Gamma| \geqslant \frac{2^n}{n}$. Hence, the size of $\mathcal{P}$ is at least $\frac{2^{\frac{n}{2}}}{\sqrt{n}}$. $\square$

6. History-deterministic Visibly Pushdown Automata

As we shall see in Section 7, one downside of HD-PDA is that, like ω-HD-PDA, they have poor closure properties and checking HDness is undecidable. We therefore consider a well-behaved class of HD-PDA, namely HD visibly pushdown automata, HD-VPA for short, that is closed under union, intersection, and complementation.

Let Σ_c, Σ_r and Σ_{int} be three disjoint sets of *call* symbols, *return* symbols and *internal* symbols respectively. Let $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{\mathsf{int}}$. A *visibly pushdown automaton* [AM04] (VPA) $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ is a restricted PDA that pushes onto the stack only when it reads a call symbol, it pops the stack only when a return symbol is read, and does not use the stack when there is an internal symbol. Formally,

- A letter $a \in \Sigma_{\mathsf{c}}$ is only processed by transitions of the form (q, X, a, q', XY) with $X \in \Gamma_\perp$, i.e. some stack symbol $Y \in \Gamma$ is pushed onto the stack.

- A letter $a \in \Sigma_r$ is only processed by transitions of the form $(q, X, a, q', \varepsilon)$ with $X \neq \bot$ or $(q, \bot, a, q', \bot)$, i.e. the topmost stack symbol is removed, or if the stack is empty, it is left unchanged.
- A letter $a \in \Sigma_{\mathsf{int}}$ is only processed by transitions of the form (q, X, a, q', X) with $X \in \Gamma_\bot$, i.e. the stack is left unchanged.
- There are no ε-transitions.

Intuitively, the stack height of the last configuration of a run processing some $w \in (\Sigma_c \cup \Sigma_r \cup \Sigma_s)^*$ only depends on w.

We denote by HD-VPA the VPA that are history-deterministic. Every VPA (and hence every HD-VPA) can be determinised, i.e. all three classes of automata recognise the same class of languages, denoted by VPL, which is a strict subset of DCFL [AM04].

6.1. **Succinctness.** While all three classes of VPA are equally expressive, VPA can be exponentially more succinct than deterministic VPA (DVPA) [AM04]. We show that there is an exponential gap both between the succinctness of HD-VPA and DVPA and between VPA and HD-VPA. The proof of the former gap again uses a language of bad counters, similar to C_n used in Theorem 5.1, which we adapt for the VPA setting by adding a suffix allowing the automaton to pop the stack. Furthermore, for the gap between VPA and HD-VPA, we similarly adapt the language L_n of words where the nth bit from the end is a 1, from the proof of Theorem 5.1, by making sure that the stack height is always bounded by 1. Such an HD-VPA is essentially an HD-NFA, and therefore determinisable by pruning, which means that it is as big as a deterministic automaton for the language.

Theorem 6.1. *HD-VPA can be exponentially more succinct than DVPA and VPA can be exponentially more succinct than HD-VPA.*

We split the proof into two parts.

Lemma 6.2. *HD-VPA can be exponentially more succinct than DVPA.*

Proof. We construct a family $(C'_n)_{n\in\mathbb{N}}$ of languages such that there is an HD-VPA of size $O(n)$ recognising C'_n, yet every DVPA recognising C'_n has at least exponential size in n. This family is obtained by adapting the family $(C_n)_{n\in\mathbb{N}}$ that we used to prove the succinctness of HD-PDA in Section 5: Once again, we consider the word $c_n \in (\${0,1\}^n)^*$ describing an n-bit binary counter counting from 0 to $2^n - 1$. We consider the languages

$$C'_n = \left\{ w \in \{0,1,\$,\#\}^* \mid w \neq c_n \#^{2^n(n+1)} \right\} \subseteq \{0,1,\$,\#\}^*$$

of bad counters, where 0, 1 and \$ are call symbols and \# is a return symbol. The only difference with C_n is that the forbidden word is $c_n \#^{2^n(n+1)}$ instead of $c_n \#$. An HD-VPA of size $O(n)$ recognising C'_n is obtained by a small modification of the construction presented in the proof of Lemma 5.2. We adapt the construction of the automaton $\mathcal{P}$ recognising C_n as follows:

- The push phase is identical;
- The check phase is performed by consuming the \# symbols instead of having ε-transitions. While the stack is not empty, $\mathcal{P}$ accepts even if it has not found evidence of bad counting yet. Moreover, $\mathcal{P}$ transitions towards a final sink state if a non-\# symbol is read. Once the stack is empty, it transitions towards the final phase;

- In the final phase, since the prefix processed up to this point ends with an empty stack, if the suffix left to read is non-empty then the input is not equal to $c_n \#^{2^n(n+1)}$, and can be accepted.

Finally, we can prove that every DPDA (and in particular every DVPA) recognising C'_n has at least exponential size in n in the exact same way as we proved Lemma 5.3: The proof only uses the fact that c_n is an infix of the single word rejected by C_n, thus C'_n can be treated identically. Note that this lower bound is independent of the partition of the letters into calls, returns, and internals. □

Lemma 6.3. *VPA can be exponentially more succinct than HD-VPA.*

We show that there exists a family $(L'_n)_{n \in \mathbb{N}}$ of languages such that there exists a VPA of size $O(n)$ recognising L'_n while every HD-VPA recognising the same language has size at least $2^{n/6}$.

Towards this we consider a language L'_n of words in $(01 + 10)^* \cdot (\varepsilon + 0 + 1)$ with the nth last letter being 1. We first note that L'_n can be recognised by a VPA with $O(n)$ states, which checks that the input is in $(01 + 10)^* \cdot (\varepsilon + 0 + 1)$ and nondeterministically guesses the nth last letter and verifies that it is a 1.

First, we note that every DFA recognising L'_n has exponential size, which can be shown by counting the equivalence classes of the Myhill-Nerode congruence of L'_n (see, e.g. [HU79]).

Remark 6.4. Every DFA recognising L'_n has at least $2^{\lceil n/2 \rceil}$ states.

Using this, we obtain an exponential lower bound on the size of HD-VPA recognising L'_n, thereby completing the proof of Lemma 6.3.

Lemma 6.5. *Every HD-VPA recognising L'_n has at least size $2^{\lceil n/6 \rceil}$.*

Proof. The proof is based on the fact that HD-NFA can be determinised by pruning [BKS17], that is, they always contain an equivalent DFA, i.e. the lower bound of Remark 6.4 is applicable to HD-NFA as well.

Let $\mathcal{P}$ be an HD-VPA recognising L'_n. We consider the following cases:

(1) *Both 0 and 1 are either a return symbol or an internal symbol*: The HD-VPA $\mathcal{P}$ in this case can essentially be seen as an HD-NFA with the same set of states, since the stack is not used (it is always equal to $\bot$). Given that HD-NFA are determinisable by pruning, by Remark 6.4 such an HD-NFA has at least $2^{\lceil n/2 \rceil}$ states.

(2) *At least one of 0 and 1 is a call symbol while the other one is a call or an internal symbol*: Let Q be the set of states and Γ be the stack alphabet of $\mathcal{P}$. Since the height of the stack is nondecreasing, $\mathcal{P}$ has only access to the top stack symbol. We can thus construct an equivalent HD-NFA over finite words with states in $Q \times \Gamma$. Since HD-NFA are determinisable by pruning, and using Remark 6.4 again, we have that $|Q| \cdot |\Gamma| \geqslant 2^{\lceil n/2 \rceil}$. Thus either $|Q| \geqslant 2^{n/4}$ or $|\Gamma| \geqslant 2^{n/4}$. Hence for this case, we have that the size of the HD-VPA is at least $2^{n/4}$.

(3) *One of 0 and 1 is a call symbol while the other one is a return symbol*: Note that since a word in L'_n is composed of sequences of 10 and 01, the stack height can always be restricted to 2. Thus the configuration space of $\mathcal{P}$, restricted to configurations on accepting runs, is finite, and there is an equivalent HD-NFA of size at most $|Q| \cdot |\Gamma|^2$. Thus $|Q| \cdot |\Gamma|^2 \geqslant 2^{\lceil n/2 \rceil}$ giving either $|Q| \geqslant 2^{\lceil n/6 \rceil}$ or $|\Gamma| \geqslant 2^{\lceil n/6 \rceil}$. Again by the determinizability by pruning argument, we have that the size of the HD-VPA $\mathcal{P}$ is at least $2^{\lceil n/6 \rceil}$. □

6.2. **Deciding HDness of VPA.** We now turn to the question of deciding whether a given VPA is HD. We show decidability using the *one-token game*, introduced by Bagnol and Kuperberg [BK18]. It is easier to decide than the game-based characterisation of HDness of ω-regular automata by Henzinger and Piterman [HP06]. While the one-token game does not characterise the HDness of Büchi automata [BK18], here we show that it suffices for VPA.

Theorem 6.6. *The following problem is* ExpTime-*complete: Given a VPA $\mathcal{P}$, is $\mathcal{P}$ HD?*

Proof. We first define the *one-token game*, introduced by Bagnol and Kuperberg [BK18] in the context of regular languages, for VPA. Fix a VPA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$, which we assume w.l.o.g. to be complete, i.e. every mode has at least one enabled a-transition for every $a \in \Sigma$. The positions of the one-token game consist of pairs of configurations (c, c'), starting from the pair containing the initial configuration of $\mathcal{P}$ twice. At each round i from position (c_i, c_i'):

- Player 1 picks a letter $a_i \in \Sigma$, then
- Player 2 picks an a_i-transition $\tau_i \in \Delta$ enabled in c_i, leading to a configuration c_{i+1}, then
- Player 1 picks an a_i-transition $\tau_i' \in \Delta$ enabled in c_i', leading to a configuration c_{i+1}'.
- Then round i ends. In round $i + 1$, the play then proceeds from the position (c_{i+1}, c_{i+1}').

Note that due to completeness of $\mathcal{P}$, there is always at least one move available for each player.

The moves of the two players during a play induce an infinite word $a_0 a_1 \cdots \in \Sigma^\omega$ and two runs $c_0 \tau_0 c_1 \tau_1 c_2 \cdots$ and $c_0' \tau_0' c_1' \tau_1' c_2' \cdots$ built by Player 2 and Player 1 respectively. Player 2 wins if for all $n \geqslant 0$, whenever the run $c_0' \tau_0' \cdots \tau_n' c_{n+1}'$ constructed by Player 1 is accepting then the run $c_0 \tau_0 \cdots \tau_n c_{n+1}$ constructed by Player 2 is accepting as well. Recall that VPA do not have ε-transitions, so the two runs proceed in lockstep, i.e. both runs process $a_0 \cdots a_n$.

Observe that this game can be seen as a safety game on an infinite arena induced by the configuration graph of a visibly pushdown automaton, obtained by taking the product of $\mathcal{P}$ with itself. This in turn is solvable in exponential time [Wal01].

It now suffices to argue that this game characterises whether the VPA $\mathcal{P}$ is HD: We argue that $\mathcal{P}$ is HD if and only if Player 2 wins the one-token game on $\mathcal{P}$. One direction is immediate: if $\mathcal{P}$ is HD, then the resolver induces a strategy for Player 2 in the one-token game which ensures that the run constructed by Player 2 is accepting whenever Player 1 picks a word that is accepted by $\mathcal{P}$. This covers in particular the cases when the run constructed by Player 1 is accepting, which suffices for Player 2 to win.

For the converse direction, we show that a winning strategy for Player 2 in the one-token game can be turned into a resolver for $\mathcal{P}$. To this end, consider the family of *copycat strategies* for Player 1 that copy the transition chosen by Player 2 until she plays an a-transition from a configuration c to a configuration c' such that there is a word aw that is accepted from c but w is not accepted from c'. We call such transitions non-residual. If Player 2 plays such a non-residual transition, then the copycat strategies stop copying and instead play the letters of w and the transitions of an accepting run over aw from c.

If Player 2 wins the one-token game with a strategy σ, she wins, in particular, against every copycat strategy for Player 1. Observe that copycat strategies win any play along which Player 2 plays a non-residual transition. Therefore, σ must avoid ever playing a non-residual transition. We can now use σ to construct a resolver r_σ for $\mathcal{P}$: r_σ maps a sequence of transitions over a word w and a letter a to the transition chosen by σ in the one-token game where Player 1 played wa and used a copycat strategy to construct his run. Then, r_σ never produces a non-residual transition. As a result, if a word w is in $L(\mathcal{P})$, then

the run induced by r_σ over every prefix v of w leads to a configuration that accepts the remainder of w. This is in particular the case for w itself, for which r_σ induces an accepting run. This concludes our argument that r_σ is indeed a resolver, and $\mathcal{P}$ is therefore HD.

Thus, to decide whether a VPA $\mathcal{P}$ is HD it suffices to solve the one-token game on $\mathcal{P}$, which can be done in exponential time. The matching lower bound follows from a reduction from the inclusion problem for VPA, which is EXPTIME-hard [AM04], to HDness (see [LZ22, Theorem 6.1(1)] for details of the reduction in the context of ω-HD-PDA).   □

Thus, HDness for VPA is decidable. On the other hand, the problem of deciding whether a language of a VPA is history-deterministic, i.e. accepted by some HD-VPA, is trivial, as VPA are determinisable [AM04].

6.3. **Good-enough Synthesis.** Finally, we relate the HDness problem to the *good-enough synthesis* problem [AK20], also known as the *uniformization* problem [CL15], which is similar to Church's synthesis problem[4], except that the system is only required to satisfy the specification on inputs in the projection of the specification on the first component.

Let $w \in \Sigma_1$ and $w' \in \Sigma_2$ with $|w| = |w'|$. Then, for the sake of readability, we write $\binom{w}{w'}$ for the word $\binom{w(0)}{w'(0)} \cdots \binom{w(|w|-1)}{w'(|w|-1)}$ over $\Sigma_1 \times \Sigma_2$.

Definition 6.7 (GE-synthesis). Given a language $L \subseteq (\Sigma_1 \times \Sigma_2)^*$, is there a function $f : \Sigma_1^* \to \Sigma_2$ such that for each $w \in \{w \mid \exists w' \in \Sigma_2^*.\binom{w}{w'}) \in L\}$ the word $\binom{w}{w'}$ is in L, where $w'(n) = f(w(0) \cdots w(n))$ for each $0 \leqslant n < |w|$.

We now prove that the GE-synthesis problem for HD-VPA and DVPA is as hard as the HDness problem for VPA, giving us the following corollary of Theorem 6.6.

Corollary 6.8. *The* GE-*synthesis problem for inputs given by HD-VPA or DVPA is* EXPTIME-*complete.*

Proof. We show that deciding GE-synthesis and HDness, which is EXPTIME-complete (see Theorem 6.6), are polynomially equivalent. We show the upper bound for HD-VPA and the lower bound for DVPA.

We first reduce the good-enough synthesis problem to the HDness problem. Given an HD-VPA $\mathcal{P} = (Q, \Sigma_1 \times \Sigma_2, \Gamma, q_I, \Delta, F)$, with resolver r, let $\mathcal{P}'$ be $\mathcal{P}$ projected onto the first component: $\mathcal{P}' = (Q, \Sigma_1, \Gamma, q_I, \Delta', F)$ has the same states, stack alphabet and final states as $\mathcal{P}$, but the transitions of $\mathcal{P}'$ are obtained by replacing each label $\binom{a}{b}$ of a transition of $\mathcal{P}$ by a. Let each transition of $\mathcal{P}'$ be annotated with a Σ_2-letter of a corresponding $\mathcal{P}$-transition. Thus, $\mathcal{P}'$ recognises the projection of $L(\mathcal{P})$ on the first component.

We show that $\mathcal{P}$ has a GE-synthesis function if and only if $\mathcal{P}'$ is HD. First, a GE-synthesis function f for $\mathcal{P}$, combined with r, induces a resolver r' for $\mathcal{P}'$ by using f to choose output letters and r to choose which transition of $\mathcal{P}$ to use; together these uniquely determine a transition in $\mathcal{P}'$. Then, if $w \in L(\mathcal{P}')$, f guarantees that the annotation of the run induced by r' in $\mathcal{P}'$ is a witness w' such that $\binom{w}{w'} \in \mathcal{P}$, and then r guarantees that the run is accepting, since the corresponding run in $\mathcal{P}$ over $\binom{w}{w'}$ must be accepting. Conversely, assume $\mathcal{P}'$ is HD. Then, a resolver for $\mathcal{P}'$ induces a GE-synthesis function for $\mathcal{P}$ by reading the Σ_2-annotation of the chosen transitions in $\mathcal{P}'$. Indeed, the resolver produces an accepting run with annotation

---

[4]Compare the following definitions to that of Gale-Stewart games in Section 9, which capture Church's synthesis problem.

w' of $\mathcal{P}'$ for every word w in the projection of $L(\mathcal{P})$ on the first component. The same run is an accepting run in $\mathcal{P}$ over $\binom{w}{w'}$ which is therefore in $L(\mathcal{P})$.

For the lower bound, we reduce the HDness problem of a VPA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ to the GE-synthesis problem of a DVPA $\mathcal{P}' = (Q, \Sigma \times \Delta, \Gamma, q_I, \Delta', F)$. The deterministic automaton $\mathcal{P}'$ is as $\mathcal{P}$ except that each transition $\tau \in \Delta$ over a letter $a \in \Sigma$ is replaced with the same transition over $\binom{a}{\tau}$ in Δ'. In other words, $\mathcal{P}'$ recognises the accepting runs of $\mathcal{P}$ and its GE-synthesis problem asks whether there is a function that constructs on-the-fly an accepting run for every word in $L(\mathcal{P})$, that is, whether $\mathcal{P}$ has a resolver. $\qquad\square$

This result should be compared to the case of LTL specifications, for which the GE-synthesis problem is 2ExpTime-complete [AK20].

7. Closure properties

We show that, like ω-HD-PDA, HD-PDA have poor closure properties, but we start with the positive results and consider closure under intersection, union, and set difference with regular languages.

Theorem 7.1. *If L is in HD-CFL and R is regular, then $L \cup R$, $L \cap R$ and $L \backslash R$ are also in HD-CFL, but $R \backslash L$ is not necessarily in HD-CFL.*

Proof. Consider an HD-PDA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ recognising L, and a resolver r for $\mathcal{P}$. By definition, r only has to induce a run on every $w \in L(\mathcal{P})$, but does not necessarily induce a run on $w \notin L(\mathcal{P})$. First, we turn $\mathcal{P}$ into an equivalent HD-PDA $\mathcal{P}'$ that has a resolver that induces a run on every input $w \in \Sigma^*$. This property allows us then to take the product of $\mathcal{P}'$ and a DFA for R.

To this end, we add a fresh nonfinal sink state q_s with a self-loop (q_s, X, a, q_s, X) for every input letter $a \in \Sigma$ and every stack symbol $X \in \Gamma_\perp$. Also, we add transitions so that every configuration has, for every $a \in \Sigma$, an enabled a-transition to the sink. The resulting PDA $\mathcal{P}'$ is equivalent to $\mathcal{P}$ and r is still a resolver for it. But, we can also turn r into a resolver r' that induces a run on every possible input as follows: If $\ell(\tau_0 \cdots \tau_n)$ is a prefix of a word in $L(\mathcal{P})$, then we define $r'(\tau_0 \cdots \tau_n, a) = r(\tau_0 \cdots \tau_n, a)$ for every $a \in \Sigma$. Otherwise, if $\ell(\tau_0 \cdots \tau_n)$ is not a prefix of a word in $L(\mathcal{P})$, we define $r'(\tau_0 \cdots \tau_n, a) = (q, X, a, q_s, X)$, where (q, X) is the mode of the last configuration of the run induced by $\tau_0 \cdots \tau_n$. Thus, as soon as the input can no longer be extended to a word in $L(\mathcal{P})$, the run induced by r' moves to the sink state and processes the remaining input.

Now, let $\mathcal{A}$ be a DFA recognising R. For $L \cup R$, we construct the product PDA $\mathcal{P}_\cup$ of $\mathcal{P}'$ and $\mathcal{A}$ that simulates a run of $\mathcal{P}'$ and the unique run of $\mathcal{A}$ simultaneously on an input word and accepts if either the run of $\mathcal{P}$ or the run of $\mathcal{A}$ is accepting. We note that when an ε-transition is chosen in $\mathcal{P}'$ by the resolver, then no move is made in $\mathcal{A}$. As $\mathcal{P}'$ has a run on every input, the product PDA $\mathcal{P}_\cup$ has one as well.

For $L \cap R$, we construct a PDA $\mathcal{P}_\cap$ which is similar to $\mathcal{P}_\cup$ with the difference that $\mathcal{P}_\cap$ accepts when each of $\mathcal{P}'$ and $\mathcal{A}$ has an accepting run on the input word.

The PDA $\mathcal{P}_\cup$ and $\mathcal{P}_\cap$ are both history-deterministic, since only the nondeterminism of $\mathcal{P}'$ needs to be resolved.

Finally, since $L \backslash R = L \cap R^c$ and the complement R^c is regular, it follows that $L \backslash R$ is recognised by an HD-PDA if there is an HD-PDA recognising L. For $R \backslash L$, note that Σ^* is a regular language and $\Sigma^* \backslash L = L^c$. In Theorem 7.2, we show that HD-CFL is not closed under complementation, which implies that $R \backslash L$ is not necessarily in HD-CFL. $\qquad\square$

Now, we consider the closure properties of HD-PDA.

Theorem 7.2. *HD-PDA are not closed under union, intersection, complementation, set difference, concatenation, Kleene star and homomorphism.*

Proof. The proofs for union, intersection, complementation, and set difference are similar to those used for ω-HD-PDA [LZ22, Theorem 5.1]. We state them here for completeness.

To show non-closure under union, consider the languages $L_1 = \{a^n b^n \mid n \geqslant 0\}$ and $L_2 = \{a^n b^{2n} \mid n \geqslant 0\}$ respectively. There exist a DPDA recognising L_1 and a DPDA recognising L_2. Hence by Lemma 3.1, there also exist an HD-PDA recognising L_1 and an HD-PDA recognising L_2. However, we show in Lemma 4.4 that $L_1 \cup L_2$ is not recognised by any HD-PDA.

For intersection, consider the languages $L_3 = \{a^n b^n c^m \mid m, n \geqslant 0\}$ and $L_4 = \{a^m b^n c^n \mid m, n \geqslant 0\}$. There exist DPDA recognising L_3 and L_4. Hence by Lemma 3.1 there exist HD-PDA recognising L_3 and L_4. Now let $L = L_3 \cap L_4 = \{a^n b^n c^n \mid n \geqslant 0\}$. As L is not a CFL there does not exist any HD-PDA recognising L.

For non-closure under complementation, recall the language

$$B_2 = \{a^i \$ a^j \$ b^k \$ \mid k \leqslant \max(i,j)\}.$$

We prove in Lemma 4.2 that $B_2 \in$ HD-CFL, yet Lemma 4.3 shows that its complement B_2^c is not even a context-free language.

Closure under set difference implies closure under complementation since for every language L over alphabet Σ, we have that the complement L^c is equal to $\Sigma^* \backslash L$.

Now we show that HD-CFL is not closed under concatenation. Consider again the languages $L_1 = \{a^n b^n \mid n \geqslant 0\}$, and $L_2 = \{a^n b^{2n} \mid n \geqslant 0\}$. We showed that $L_1 \cup L_2$ is not an HD-CFL. Now let $L_5 = cL_1 \cup L_2$. Clearly L_5 is a DCFL and hence an HD-CFL since the initial letter determines whether the word is in cL_1 or in L_2. Now the language represented by the regular expression c^* is regular, and hence an HD-CFL. We argue that $c^* L_5$ is not an HD-CFL, and hence HD-CFLs are not closed under concatenation.

Assume for contradiction that $c^* L_5$ is an HD-CFL. Let $L_6 = c^* L_5 \cap ca^* b^* = cL_1 \cup cL_2$. Since HD-CFLs are closed under intersection with regular languages (see Theorem 7.1), L_6 is also an HD-CFL. Now if L_6 is an HD-CFL, then there exists an HD-PDA $\mathcal{P}_6$ with a resolver r such that for a word $w = cv \in L_6$ where $v \in L_1 \cup L_2$, the resolver r after reading c reaches some state q of $\mathcal{P}_6$ from which it induces an accepting run on v. This allows us to construct from $\mathcal{P}_6$ an HD-PDA $\mathcal{P}_{1 \cup 2}$ accepting $L_1 \cup L_2$ with q as the initial state. We reach a contradiction since $L_1 \cup L_2$ is not an HD-CFL.

Now we show that HD-CFLs are also not closed under Kleene star. Again we consider the language L_5 above which is an HD-CFL. We show that L_5^* is not an HD-CFL. Had L_5^* been an HD-CFL, this would imply that $L_5^* \cap ca^* b^*$ is an HD-CFL. However $L_5^* \cap ca^* b^* = cL_1 \cup cL_2$ which is the language L_6 defined above. Now as we have already shown that the language L_6 is not an HD-CFL, this implies that L_5^* is also not an HD-CFL.

For non-closure under homomorphism, consider the language

$$L = \left\{ \begin{pmatrix} a \\ 1 \end{pmatrix}^n \begin{pmatrix} b \\ \# \end{pmatrix}^n \,\middle|\, n \geqslant 0 \right\} \cup \left\{ \begin{pmatrix} a \\ 2 \end{pmatrix}^n \begin{pmatrix} b \\ \# \end{pmatrix}^{2n} \,\middle|\, n \geqslant 0 \right\}$$

is recognised by a DPDA, and hence by an HD-PDA using Lemma 3.1, but its projection (which is a homomorphism)

$$\{a^n b^n \mid n \geqslant 0\} \cup \{a^n b^{2n} \mid n \geqslant 0\}$$

cannot be recognised by an HD-PDA (see Lemma 4.4).   □

8. Compositionality with games

One of the appeals of history-deterministic automata consists of the good compositional properties that they enjoy, which allows them to be used to solve games. Namely, we consider arena-based games, in which two players, called Player 1 and Player 2, build a path, with the owner of the current position choosing an outgoing edge at each turn, and Player 2 wins if the label of the path is in the winning condition. If the winning condition is the language of a deterministic or history-deterministic automaton $\mathcal{D}$ with acceptance condition C, taking the product (or composition) of the arena with $\mathcal{D}$ yields an arena which, when treated as a game with winning condition C, has the same winner as the original game with winning condition $L(\mathcal{D})$. In short, this product construction reduces games with winning conditions recognised by deterministic or history-deterministic C-automata into C-games. For example, it can be used to reduce games with ω-regular winning conditions, which are recognised by history-deterministic and deterministic parity automata, into parity games. In this sense, history-deterministic automata are just as good as deterministic automata when it comes to solving games via composition, making them appealing for applications such as reactive synthesis, which can be solved via games.

Here we first show that HD-PDA, like regular history-deterministic automata, enjoy this compositional property, both with respect to finite and infinite arenas. We then consider the converse question, namely, whether all PDA that behave well with respect to composition, are HD-PDA. This is the case for (ω-)regular automata [BL19], but not for quantitative automata [BL21]. We show that for PDA over finite words, those that enjoy compositionality with infinitely branching games are exactly those that are history-deterministic. On the other hand, unlike for the (ω-)regular setting, compositionality with finitely branching games does not suffice to guarantee that the automaton is history-deterministic, even for automata over finite words.

Our proof of equivalence of compositionality with games and history-determinism for PDA over finite words uses the determinacy of safety games. Generalising this proof to automata over infinite words would require determinacy of games with ω-context-free winning conditions, which is a large cardinal assumption [Fin12].

A Σ-arena $A = (V, V_1, V_2, E, \iota, \ell)$ consists of a directed graph (V, E) of which the positions V are partitioned into those belonging to Player 1, V_1, and those belonging to Player 2, V_2, of which the edges E are labelled with $\Sigma \cup \{\mathtt{end}\}$ via a labelling $\ell \colon E \to \Sigma \cup \{\mathtt{end}\}$), and rooted at an initial position $\iota \in V$. We assume that all $\mathtt{end}$-edges lead to a terminal position, that is, without outgoing edges, that all other positions have at least one successor, and that all edges leading to a terminal position are labelled $\mathtt{end}$.

A play on A is a finite path ending in a terminal position, or an infinite path. A strategy for a player is a mapping from finite paths ending in a position that belongs to that player to one of its outgoing edges. A play is consistent with Player i's strategy σ if for all of its prefixes π ending in V_i, the next edge is $\sigma(\pi)$.

An (arena-based) game (A, L) consists of a Σ-arena A and a language $L \subseteq \Sigma^*$. A finite play is winning for Player 2 if it is labelled with a word in $L \cdot \mathtt{end}$. Note that Player 2 loses in particular all infinite plays. Winning strategies are defined in the usual way, and we say that a player wins the game if they have a winning strategy.

We now consider the composition of an arena and an automaton, that is, a product game in which Player 2 not only has to guarantee that the label of the play in the arena is in the language of the automaton, but she also has to build an accepting run over that word, transition by transition, as the play in the arena progresses.

Formally, the positions of the product game $G(A, \mathcal{P})$ of a Σ-arena $A = (V, V_1, V_2, E, \iota, \ell)$ and a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ consists of a position in V and a configuration of $\mathcal{P}$, the initial position being $(\iota, q_I, \bot)$. At each round i from position (v, q, γ), the player who owns the position v chooses an outgoing edge $(v, v') \in E$, labelled by some letter $\ell(v, v') = a$. If $a \neq \mathtt{end}$, Player 2 then chooses a possibly empty sequence of ε-transitions followed by one a-transition, so that the combined sequence induces a finite run from (q, γ) leading to some (q', γ'). Then, the play proceeds in round $i + 1$ from (v', q', γ'). If $a = \mathtt{end}$, then the play ends in (v', q, γ), as v' is by assumption a terminal position of the arena. Player 2 wins by reaching a position (v, q, γ) where v is the terminal position and q is final. Again, strategies, winning strategies and winning the game are defined as expected.

Observe that $G(A, \mathcal{P})$ may have an infinite number of positions, and may have infinite branching, due to unbounded ε-transition sequences. If A is finitely representable, so is $G(A, \mathcal{P})$. In particular, if A is finite, then $G(A, \mathcal{P})$ is a reachability game on the configuration graph of a pushdown automaton.

Definition 8.1. Let $\mathcal{P}$ be a PDA over an alphabet Σ.
- $\mathcal{P}$ is compositional if the following holds for all (finitely or infinitely branching) Σ-arenas A: Player 2 wins the game $(A, L(\mathcal{P}))$ if and only if she wins $G(A, \mathcal{P})$.
- $\mathcal{P}$ is weakly compositional if the following holds for all finitely branching Σ-arenas A: Player 2 wins the game $(A, L(\mathcal{P}))$ if and only if she wins $G(A, \mathcal{P})$.

Note that compositionality implies weak compositionality. In the remainder of this section, we compare history-determinism and (weak) compositionality.

Lemma 8.2. *Every HD-PDA is compositional.*

Proof. Let $\mathcal{P}$ be an HD-PDA over Σ and let A be a Σ-arena. If Player 2 wins the game $(A, L(\mathcal{P}))$ with a strategy σ, then she wins $G(A, \mathcal{P})$ by using σ in the A-component of $G(A, \mathcal{P})$ and the resolver r that witnesses the history-determinism of $\mathcal{P}$ in the $\mathcal{P}$-component of $G(A, \mathcal{P})$. Then, σ guarantees that the word built in A is in $L(\mathcal{P}) \cdot \mathtt{end}$, and r guarantees that Player 2 reaches a final state at the end of the word in $L(\mathcal{P})$, before $\mathtt{end}$ is encountered.

Conversely, a winning strategy for Player 2 in $G(A, \mathcal{P})$ projected onto the A-component is a winning strategy for Player 2 in $(A, L(\mathcal{P}))$. $\square$

We now consider the converse: whether it is the case that all (weakly) compositional PDA are HD-PDA. In the regular setting, automata are weakly compositional if and only if they are compositional [BL19]. Here, however, we observe that the answer depends on whether infinite branching is permitted in the arenas considered.

Lemma 8.3. *There exists a PDA that is weakly compositional, but not history-deterministic.*

Proof. Consider the 3-state PDA in Fig. 8 that recognises a^*b, by first guessing an upper bound m on the number of a's in the word, then processing a no more than m times, followed by a b. The automaton accepts $a^n b$ by pushing $m \geqslant n$ elements onto the stack before reading n times a, followed by b.

It is clearly not an HD-PDA, as the resolver would have to predict the number of a's to be seen. However, for all finitely branching Σ-arenas A, if Player 2 wins the game $(A, L(\mathcal{P}))$,
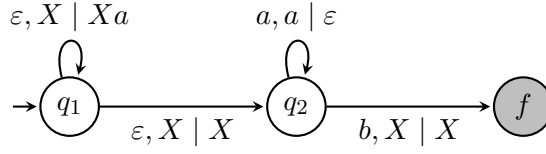
Figure 4: A PDA recognising a^*b that is weakly compositional but not history-deterministic. Grey states are final, and X is an arbitrary stack symbol.

she also wins $G(A, \mathcal{P})$. Indeed, if Player 2 wins $(A, L(\mathcal{P}))$ with a strategy σ, then by König's lemma, there is some bound n on the number of a's in any play that is consistent with σ. Then, in $G(A, \mathcal{P})$, Player 2 wins by first pushing n letters onto the stack in $\mathcal{P}$, then playing σ in the A-component. Since σ guarantees that there are at most n occurrences of a in the play of $L(\mathcal{P})$, this strategy is winning for Player 2. $\qquad\square$

In contrast, HD-PDA are exactly the PDA that preserve winners when composed with infinitely branching arenas.

Theorem 8.4. *A PDA is history-deterministic if and only if it is compositional.*

Proof. One direction of the equivalence was shown in Lemma 8.2: every HD-PDA is compositional.

For the other direction, we show that from a PDA that is not history-deterministic, we can construct an arena that witnesses the failure of compositionality. To characterise automata that are not history-deterministic, we rely on the so-called *letter game*, a game-based characterisation of history-determinism due to Henzinger and Piterman [HP06].

Given a PDA $\mathcal{P}$, the letter game is played by two players, called Challenger and Resolver, and $\mathcal{P}$ is history-deterministic if and only if Resolver wins the letter game induced by $\mathcal{P}$. The game is played in rounds: in each round i, Challenger chooses a letter a_i and then Resolver responds with ρ_i, the concatenation of a (potentially empty) sequence of ε-transitions and an a_i-transition. Resolver's winning condition is a safety condition: before each round i, either the word $w = a_0 a_1 \cdots a_{i-1}$ built by Challenger so far is not in the language $L(\mathcal{P})$ or the sequence $\rho_0 \rho_1 \cdots \rho_{i-1}$ played by Resolver so far induces an accepting run of $\mathcal{P}$ over w. This game characterises history-determinism, as a winning strategy for Resolver is exactly a resolver witnessing the history-determinism of $\mathcal{P}$.

We now crucially rely on the determinacy of the letter game, which we can easily show for PDA over finite words. Indeed, the winning condition for Resolver is a safety condition, in the sense that all losing plays have a finite prefix of which all continuations are losing. The winning condition is therefore Borel, and, by Martin's theorem [Mar75], determined. This implies that if $\mathcal{P}$ is not history-deterministic, then Challenger has a winning strategy σ_C in the letter game.

We now build an arena A_{σ_C}, based on σ_C, such that Player 2 wins $(A_{\sigma_C}, L(\mathcal{P}))$, but loses the composition game $G(A_{\sigma_C}, \mathcal{P})$, thus witnessing that $\mathcal{P}$ is not compositional.

The arena A_{σ_C} emulates the strategy tree for σ_C, in the sense that all the plays of the arena are plays that are consistent with the strategy σ_C. Furthermore, the branching in the arena corresponds to the branching of the strategy tree of σ_C, which represents Resolver's choices in the letter game. All positions of A_{σ_C} belong to Player 1.

Note that a play in the letter game has the form $a_0 \rho_0 a_1 \rho_1 a_2 \rho_2 \cdots$ where each a_i is a letter in Σ and each ρ_i is a (possibly empty) sequence of ε-transitions concatenated with

an a_i-transition. Furthermore, as Resolver picks the ρ_i, it is her turn to make a move after a prefix of the form $a_0\rho_0a_1\cdots\rho_{i-1}a_i$ for $i \geqslant 0$. Now, the positions of A_{σ_C} are the empty play prefix, which is the initial position, and all play prefixes that are consistent with σ_C and end with a letter, i.e. it is Resolver's turn. There is an a-labelled edge from the initial position ε to the position a, where a is the letter picked by σ_C in round zero. Furthermore, for every position of the form $a_0\rho_0a_1\cdots\rho_{i-2}a_{i-1}\rho_{i-1}a_i$ for $i \geqslant 1$, there is an a_i-labelled edge from $a_0\rho_0a_1\cdots\rho_{i-2}a_{i-1}$ to $a_0\rho_0a_1\cdots\rho_{i-2}a_{i-1}\rho_{i-1}a_i$.

Consider a position $a_0\rho_0a_1\cdots\rho_{i-1}a_i$ for $i \geqslant 1$. If $w = a_0\cdots a_{i-1}$ is in $L(\mathcal{P})$, but $\rho_0\cdots\rho_{i-1}$ is not an accepting run processing w, then we replace the label a_i of the edge from $a_0\rho_0a_1\cdots\rho_{i-2}a_{i-1}$ to $a_0\rho_0a_1\cdots\rho_{i-2}a_{i-1}\rho_{i-1}a_i$ by **end** and remove all outgoing transitions of $a_0\rho_0a_1\cdots\rho_{i-2}a_{i-1}\rho_{i-1}a_i$.

Note that A_{σ_C} may be infinitely branching, if Resolver can pick arbitrarily long sequences of ε-transitions (e.g. recall the PDA presented in the proof of Lemma 8.3).

Recall that σ_C is a winning strategy for Challenger, i.e. during every play that is consistent with it, there is a round at which the word picked by Challenger is in $L(\mathcal{P})$, but Resolver has not picked an accepting run on that word. Thus, by construction, there is no infinite play in A_{σ_C} and every maximal play ends in a terminal position and is labelled by a word in $L(\mathcal{P}) \cdot$ **end**. Hence, Player 2 wins $(A_{\sigma_C}, L(\mathcal{P}))$.

Now, towards a contradiction, assume that she also wins $G(A_{\sigma_C}, \mathcal{P})$, say with a strategy σ_2. We argue that Resolver can mimic σ_2 in the letter game, and thereby win against σ_C, a contradiction. Indeed, σ_2 only chooses transitions in $\mathcal{P}$, since all positions in A_{σ_C} belong to Player 1. However, note that σ_2 in $G(A_{\sigma_C}, \mathcal{P})$ can base its decisions on the sequence of transitions picked by Player 1 during the play. By only considering those plays where Player 1's choices mimic the choices made by σ_2, which is possible due to the construction of A_{σ_C}, we eliminate this additional information.

So, let us define the strategy for the letter game mimicking σ_2 inductively: In round zero, Challenger picks some a according to σ_C in the letter game. Now, let ρ_0 be the sequence of transitions picked by σ_2 in $G(A_{\sigma_C}, \mathcal{P})$ in reaction to the initial move, which leads via an a-labelled edge from ε to a. Then, Resolver picks ρ_0 during the first round of the letter game.

Now, assume we are in round $i > 0$ of the letter game, Challenger has picked $a_0\cdots a_i$ so far, Resolver has answered with $\rho_0\cdots\rho_{i-1}$, and now has to pick ρ_i. Consider the position $a_0\rho_0\cdots\rho_{i-1}a_i$ of A_{σ_C}. As A_{σ_C} is a tree, there is a unique play prefix ending in that position. By construction of our strategy, this play prefix is consistent with σ_2. Now, let ρ_i be the sequence of transitions picked by σ_2 for that play prefix. Then, Resolver picks ρ_i during the ith round of the letter game.

Recall that σ_2 is a winning strategy for Player 2 in $G(A_{\sigma_C}, \mathcal{P})$. So, for every maximal play in A_{σ_C}, which by construction is labelled by a word in $L(\mathcal{P}) \cdot$ **end**, σ_2 constructs an accepting run of $\mathcal{P}$ processing that word. Thus, the mimicking strategy in the letter game is winning for Resolver. Thus, we have derived the desired contradiction.

We conclude that if $\mathcal{P}$ is not history-deterministic, then we can build, from a winning strategy for Challenger in the letter game, an arena such that Player 2 wins in the arena, but not in the composition with $\mathcal{P}$, witnessing that $\mathcal{P}$ is not compositional. $\square$

Observe that infinite branching is key for this proof to work, as exhibited by the counterexample in the proof of Lemma 8.3. Another ingredient we use is the determinacy of the letter game. While this is easy enough to obtain for PDA over finite words, for which the

letter game is a safety game (albeit with a potentially complex safe region), this becomes trickier for PDA over infinite words. Indeed, then the letter game's winning condition for Resolver is no longer a safety condition and it involves the negation of an ω-context-free language. The determinacy of games with a winning condition given by a Büchi one-counter automaton is a large cardinal assumption [Fin12], so obtaining determinacy for these letter games on ω-PDA is likely to be challenging.

9. Solving Games and Universality

One of the motivations for HD automata is that solving games with winning conditions given by HD automata is easier than for nondeterministic automata. This makes them appealing for applications such as the synthesis of reactive systems, which can be modelled as a game between an antagonistic environment and the system. Solving games is undecidable for PDA in general [Fin01], both over finite and infinite words, while for DPDA [Wal01] and ω-HD-PDA, it is ExpTime-complete [LZ22]. As a corollary, universality is also decidable for ω-HD-PDA, while it is undecidable for PDA, both over finite and infinite words [HU79].

Here, we consider Gale-Stewart games [GS53], abstract games induced by a language in which two players alternately pick letters, thereby constructing an infinite word. One player aims to construct a word that is in the language while the other aims to construct one that is not in the language. We use Gale-Stewart games rather than the arena-based games from the previous section since they are more convenient to work with. This is not a restriction, since arena-based games with finite arenas and Gale-Stewart games, both with winning conditions recognised by PDA, are effectively equivalent: a finite arena can be simulated by the PDA recognising the winning condition while a Gale-Stewart game can be seen as an arena-based game with a trivial arena.

Formally, given a language $L \subseteq (\Sigma_1 \times \Sigma_2)^*$ of sequences of letter pairs, the game $G(L)$ is played between Player 1 and Player 2 in rounds $i = 0, 1, \dots$ as follows: At each round i, Player 1 plays a letter $a_i \in \Sigma_1$ and Player 2 answers with a letter $b_i \in \Sigma_2$. A play of $G(L)$ is an infinite word $\binom{a_0}{b_0}\binom{a_1}{b_1}\cdots$ and Player 2 wins such a play if and only if each of its prefixes is in the language L. A strategy for Player 2 is a mapping from Σ_1^+ to Σ_2 that gives for each prefix played by Player 1 the next letter to play. A play is consistent with a strategy σ if for each i, $b_i = \sigma(a_0 a_1 \dots a_i)$. Player 2 wins $G(L)$ if she has a strategy such that all plays that are consistent with it are winning for Player 2. Observe that Player 2 loses whenever the projection of L onto its first component is not universal. Finally, universality reduces to solving these games: $\mathcal{P}$ is universal if and only if Player 2 wins $G(L)$ for $L = \{\binom{w(0)}{\#}\cdots\binom{w(n)}{\#} \mid w(0)\cdots w(n) \in L(\mathcal{P})\}$.

We now argue that solving games for HD-PDA easily reduces to the case of ω-HD-PDA, which are just HD-PDA over infinite words, where acceptance is not determined by final state, since runs are infinite, but rather by the states or transitions visited infinitely often. Here, we only need safety ω-HD-PDA, in which every infinite run is accepting (i.e. rejection is implemented via missing transitions). The infinite Gale-Stewart game over a language L of infinite words, also denoted by $G(L)$, is as above, except that victory is determined by whether the infinite word built along the play is in L.

Lemma 9.1. *Given an HD-PDA $\mathcal{P}$, there is a safety ω-HD-PDA $\mathcal{P}'$ no larger than $\mathcal{P}$ such that Player 2 wins $G(L(\mathcal{P}))$ if and only if she wins $G(L(\mathcal{P}'))$.*

Proof. Let $\mathcal{P}'$ be the PDA obtained from $\mathcal{P}$ by removing all transitions (q, X, a, q', γ) of $\mathcal{P}$ with $a \in \Sigma$ and with non-final q'. Note that all ε-transitions are preserved.

With a safety condition, in which every infinite run is accepting, $\mathcal{P}'$ recognises exactly those infinite words whose prefixes are all accepted by $\mathcal{P}$. Hence, the games $G(L(\mathcal{P}))$ and $G(L(\mathcal{P}'))$ have the same winning player. Note that the correctness of this construction crucially relies on our definition of HD-PDA, which requires a run on a finite word to end as soon as the last letter is processed. Then, the word is accepted if and only if the state reached by processing this last letter is final.

Finally, since $\mathcal{P}$ is HD, so is $\mathcal{P}'$. Consider an infinite input in $L(\mathcal{P}')$. Then, every prefix w has an accepting run of $\mathcal{P}$ induced by its resolver, which implies that the last transition of this run (which processes the last letter of w) is not one of those that are removed to obtain $\mathcal{P}'$. Now, an induction shows that the same resolver works for $\mathcal{P}'$ as well, relying on the fact that if w and w' with $|w| < |w'|$ are two such prefixes, then the resolver-induced run of $\mathcal{P}$ on w is a prefix of the resolver-induced run of $\mathcal{P}$ on w'. $\qquad\square$

Our main results of this section are now direct consequences of the corresponding results on ω-words [LZ22, Theorem 4.2 & Corollary 4.3].

Corollary 9.2. *Given an HD-PDA $\mathcal{P}$, deciding whether $L(\mathcal{P}) = \Sigma^*$ and whether Player 2 wins $G(L(\mathcal{P}))$ are both in* EXPTIME.

Determining whether, for a given HD-PDA $\mathcal{P}$, Player 2 wins $G(L(\mathcal{P}))$ is in fact EXPTIME-complete, which can be deduced from Walukiewicz's result showing that solving games with ω-DCFL winning conditions is EXPTIME-hard [Wal01, Proposition 31 & Theorem 26]. Whether universality of HD-PDA is also EXPTIME-hard is left open, as it is for HD-PDA over ω-words [LZ22].

10. RESOLVERS

The definition of a resolver does not put any restrictions on its complexity. In this section we study the complexity of the resolvers that HD-PDA need. On one hand, we show that resolvers can always be chosen to be *positional*, that is, dependent only on the current configuration of the automaton (which, for pushdown automata, includes the current state and the current stack content). Note that this is not the case for ω-regular automata[5], let alone ω-HD-PDA. On the other hand, we show that they are not always implementable by pushdown transducers.

More formally, a resolver r is positional, if for any two sequences ρ and ρ' of transitions inducing runs ending in the same configuration, $r(\rho, a) = r(\rho', a)$ for all $a \in \Sigma$.

Lemma 10.1. *Every HD-PDA has a positional resolver.*

Proof. Let r' be a (not necessarily positional) resolver for $\mathcal{P}$. We define a resolver r such that for each configuration and input letter, it makes a choice consistent with r' for some input leading to this configuration. In other words, for every configuration c reachable by a run induced by r', let ρ_c be an input to r' inducing a run ending in c. Then, we define $r(\rho, a) = r'(\rho_c, a)$, where c is the last configuration of the run induced by ρ. Note that every configuration reachable by a run induced by r is also reachable by a run induced by r'.

---

[5]A positional resolver for ω-regular automata implies determinisability by pruning, and we know that this is not always possible [BKS17].

We claim that r, which is positional by definition, is a resolver. Towards a contradiction, assume that this is not the case, i.e. there is a word $w \in L(\mathcal{P})$ such that the run ρ induced by r is rejecting. Since this run is finite and $w \in L(\mathcal{P})$, there is some last configuration c along the run ρ from which the rest of the word, say u, is accepted[6] (by some other run of $\mathcal{P}$ having the same prefix as ρ up to configuration c). Let τ be the next transition along ρ from c. Since r chose τ, the resolver r' also chooses τ after some history leading to c, over some word v. Since u is accepted from c, the word vu is in $L(\mathcal{P})$; since r' is a resolver, there is an accepting run over u from c starting with τ, contradicting that c is the last position on ρ from where the rest of the word could be accepted. ☐

Contrary to the case of finite and ω-regular automata, since HD-PDA have an infinite configuration space, the existence of positional resolvers does not imply determinisability. On the other hand, if an HD-PDA has a resolver which only depends on the mode of the current configuration, then it is *determinisable by pruning*, as transitions that are not used by the resolver can be removed to obtain a deterministic automaton. However, not all HD-PDA are determinisable by pruning, e.g. the HD-PDA for the languages C_n used to prove Theorem 5.1.

As even positional resolvers for HD-PDA are infinite objects mapping configurations to transitions, they are not necessarily finitely representable. We now turn our attention to whether HD-PDA always have finitely representable resolvers, i.e., to what kind of memory a *machine* needs in order to implement a resolver for an HD-PDA. First, we introduce transducers as a way to implement a resolver. A transducer is an automaton with outputs instead of acceptance, i.e. it computes a function from input sequences to outputs. A pushdown resolver is a pushdown transducer that implements a resolver.

Note that a resolver has to pick enabled transitions in order to induce accepting runs for all inputs in the language. To do so, it needs access to the mode of the last configuration. However, to keep track of this information on its own, the pushdown resolver would need to simulate the stack of the HD-PDA it controls. This severely limits the ability of the pushdown resolver to implement computations on its own stack. Thus, we give a pushdown resolver access to the current mode of the HD-PDA via its output function, thereby freeing its own stack to implement further functionalities.

Formally, a pushdown transducer (PDT for short) $\mathcal{T} = (\mathcal{D}, \lambda)$ consists of a DPDA $\mathcal{D}$ augmented with an output function $\lambda : Q^{\mathcal{D}} \to \Theta$ mapping the states $Q^{\mathcal{D}}$ of $\mathcal{D}$ to an output alphabet Θ. The input alphabet of $\mathcal{T}$ is the input alphabet of $\mathcal{D}$.

Then, given a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$, a pushdown resolver for $\mathcal{P}$ consists of a pushdown transducer $\mathcal{T} = (\mathcal{D}, \lambda)$ with input alphabet Δ and output alphabet $Q \times \Gamma_{\perp} \times \Sigma \to \Delta$ such that the function $r_{\mathcal{T}}$, defined as follows, is a resolver for $\mathcal{P}$: $r_{\mathcal{T}}(\tau_0 \ldots \tau_k, a) = \lambda(q_{\mathcal{T}})(q_{\mathcal{P}}, X, a)$ where

- $q_{\mathcal{T}}$ is the state of the last configuration of the longest run of $\mathcal{D}$ on $\tau_0 \ldots \tau_k$ (recall that while $\mathcal{D}$ is deterministic, it may have several runs on an input which differ on trailing ε-transitions);
- $(q_{\mathcal{P}}, X)$ is the mode of the last configuration of the run of $\mathcal{P}$ induced by $\tau_0 \ldots \tau_k$.

---

[6]Observe that this is no longer true over infinite words as an infinite run can stay within configurations from where an accepting run exists without being itself accepting. In fact, the lemma does not even hold for coBüchi automata [KS15] as the existence of positional resolvers implies determinisability by pruning.
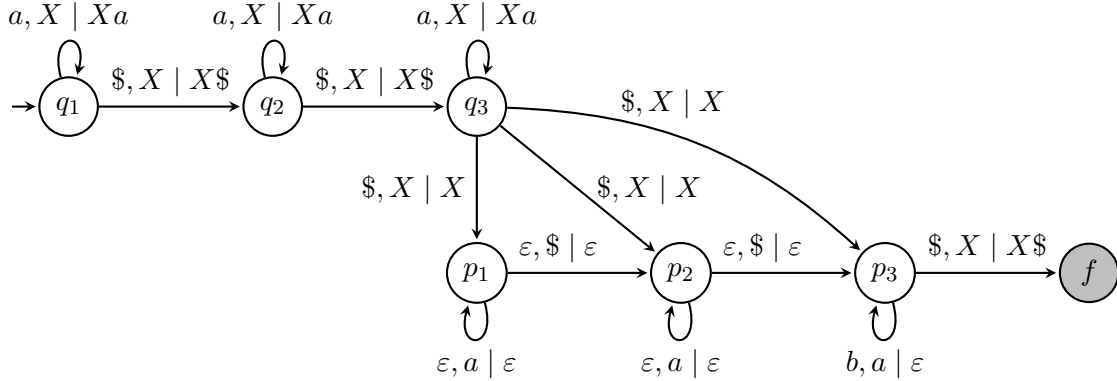
Figure 5: The PDA $\mathcal{P}_{B_3}$ for B_3. Grey states are final, and X is an arbitrary stack symbol.

In other words, a transducer implements a resolver by processing the run so far, and then uses the output of the state reached and the state and top stack symbol of the HD-PDA to determine the next transition in the HD-PDA.

We now give an example of an HD-PDA which does not have a pushdown resolver. The language in question is the language $B_3 = \{a^i\$a^j\$a^k\$b^l\$ \mid l \leqslant \max(i, j, k)\}$. Compare this to the language B_2 in Section 4 which *does* have a pushdown resolver. Let $\mathcal{P}_{B_3}$ be the automaton in Figure 5, which works analogously to the automaton for B_2 in Figure 2.

This automaton recognises B_3: for a run to end in the final state, the stack, and therefore the input, must have had an a-block longer than or equal to the final b-block; conversely, if the b-block is shorter than or equal to some a-block, the automaton can nondeterministically pop the blocks on top of the longest a-block off the stack before processing the b-block. Furthermore, this automaton is HD: the nondeterminism can be resolved by removing from the stack all blocks until the *longest* a-block is at the top of the stack, and this choice can be made once the third $\$$ is processed.

We now argue that this HD-PDA needs more than a pushdown resolver. The reason is that a pushdown resolver needs to be able to determine which of the three blocks is the longest while processing a prefix of the form $a^*\$a^*\$a^*$. However, at least one of the languages induced by these three choices is not context-free, yielding the desired contradiction.

Lemma 10.2. *The HD-PDA $\mathcal{P}_{B_3}$ has no pushdown resolver.*

Proof. Towards a contradiction, assume that there is a pushdown resolver r for $\mathcal{P}_{B_3}$, implemented by a PDT $\mathcal{T} = (\mathcal{D}, \lambda)$.

We construct, for each $i \in \{1, 2, 3\}$, a PDA $\mathcal{D}_i$ that recognises the language of words $w \in a^+\$a^+\$a^+$ such that $\mathcal{T}$ chooses the transition of $\mathcal{P}_{B_3}$ going from q_3 to p_i when constructing a run on $w\$$. Then, we show that at least one of the languages $L(\mathcal{D}_i)$ is not context-free, thereby obtaining the desired contradiction.

To this end, we first restrict the possible inputs $\mathcal{D}$ can process. Let τ_i for $i \in \{1, 2, 3\}$ be the self-loop processing a on state q_i of $\mathcal{P}_{B_3}$ and let τ'_i for $i \in \{1, 2\}$ be the $\$$-transition from q_i to q_{i+1}. Now, let $\mathcal{A}$ be a DFA for the regular language $\tau_1^+\tau'_1\tau_2^+\tau'_2\tau_3^+$ and let $\mathcal{D}'$ be obtained by taking the product of $\mathcal{D}$ with $\mathcal{A}$. Note that $\mathcal{D}'$ does not process any ε-transitions nor b-transitions of $\mathcal{P}_{B_3}$.

Now the PDA $\mathcal{D}_i$ for $i \in \{1, 2, 3\}$ is obtained from $\mathcal{D}'$ by modifying each transition of $\mathcal{D}'$ processing a transition τ of $\mathcal{P}_{B_3}$ to now process $\ell(\tau) \in \{a, \$\}$, and final states are defined as follows: Recall that states of $\mathcal{D}'$ are of the form (q, q') where q is a state of $\mathcal{D}$ and q' is a state of $\mathcal{A}$. Such a state is final in $\mathcal{D}_i$ if q' is final in $\mathcal{A}$, i.e. a word of the form $a^+\$a^+\$a^+$ has been processed, and $\lambda(q)(q_3, a, \$) = (q_3, \$, a, p_i, a)$, i.e. $\mathcal{T}$ chooses the transition of $\mathcal{P}_{B_3}$ going from q_3 to p_i.

The language $a^+\$a^+\$a^+$ is partitioned among the languages recognized by $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$, which implies that there is at least one $i \in \{1, 2, 3\}$ such that $\mathcal{D}_i$ accepts $a^m\$a^m\$a^m$ for infinitely many m's. Let us fix such an index i for the rest of the proof. Note that for every $m_1, m_2, m_3 \in \mathbb{N}$, the resolver $\mathcal{D}$ needs to be able to prolong the run selected over the input $a^{m_1}\$a^{m_2}\$a^{m_3}\$$ into an accepting run if the suffix $b^{\max(m_1, m_2, m_3)}$ is encountered. This is only possible from the states q_j satisfying $\max(m_1, m_2, m_3) = m_j$, which implies that every word $a^{m_1}\$a^{m_2}\$a^{m_3}$ accepted by $\mathcal{D}_i$ satisfies $\max(m_1, m_2, m_3) = m_i$.

To reach a contradiction, we now argue that this $\mathcal{D}_i$ recognises a language that is not context-free. Indeed, if it were, then by applying the pumping lemma for context-free languages, there would be a large enough m such that the word $a^m\$a^m\$a^m \in L(\mathcal{D}_i)$ could be decomposed as $uvwyz$ such that $|vy| \geqslant 1$ and uv^nwy^nz is in the language of $\mathcal{D}_i$ for all $n \geqslant 0$. In this decomposition, v and y must be $\$$-free. Then, if either v or y occurs in the ith block and is non-empty, by setting $n = 0$ we obtain a contradiction as the ith block is no longer the longest. Otherwise, we obtain a similar contradiction by setting $n = 2$. In either case, this shows that $\mathcal{T}$ is not a pushdown resolver for $\mathcal{P}$. $\qquad\square$

Another restricted class of resolvers are finite-state resolvers, which can be seen as pushdown resolvers that do not use their stack. Similarly to the case of ω-HD-PDA [LZ22, Theorem 7.2], the product of an HD-PDA and a finite-state resolver yields a DPDA for the same language.

Remark 10.3. Every HD-PDA with a finite-state resolver is determinisable.

Note that the converse does not hold. For example, consider the regular, and therefore deterministic context-free, language L_{10} of words $w\#$ with $w \in \{a, b\}^*$ with infix a^{10}. An HD-PDA $\mathcal{P}_{10}$ recognising L_{10} can be constructed as follows: $\mathcal{P}_{10}$ pushes its input onto its stack until processing the first $\#$. Before processing this letter, $\mathcal{P}_{10}$ uses ε-transitions to empty the stack again. While doing so, it can nondeterministically guess whether the next 10 letters removed from the stack are all a's. If yes, it accepts; in all other cases (in particular if the input word does not end with the first $\#$ or the infix a^{10} is not encountered on the stack) it rejects. This automaton is history-deterministic, as a resolver has access to the whole prefix before the first $\#$ when searching for a^{10} while emptying the stack. This is sufficient to resolve the nondeterminism. On the other hand, there is no finite-state resolver for $\mathcal{P}_{10}$, as resolving the nondeterminism, intuitively, requires to keep track of the whole prefix before the first $\#$ (recall that a finite-state resolver only has access to the topmost stack symbol).

We consider another model of pushdown resolver, namely one that does not only have access to the mode of the HD-PDA, but can check the full stack for regular properties. Thus, we consider a more general model where the transducer can use information about the whole stack when determining the next transition. More precisely, we consider a regular abstraction of the possible stack contents by fixing a DFA running over the stack and allowing the transducer to base its decision on the state reached by the DFA as well.

Then, given a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$, a pushdown resolver with regular stack access $\mathcal{T} = (\mathcal{D}, \mathcal{A}, \lambda)$ consists of a DPDA $\mathcal{P}$ with input alphabet Δ, a DFA $\mathcal{A}$ over $\Gamma_\perp$ with state set $Q^\mathcal{A}$, and an output function λ with output alphabet $Q \times Q^\mathcal{A} \times \Sigma \to \Delta$ such that the function $r_\mathcal{T}$ defined as follows, is a resolver for $\mathcal{P}$:

$$r_\mathcal{T}(\tau_0 \ldots \tau_k, a) = \lambda(q_\mathcal{T})(q_\mathcal{P}, q_\mathcal{A}, a)$$

where

- $q_\mathcal{T}$ is the state of the last configuration of the longest run of $\mathcal{D}$ on $\tau_0 \ldots \tau_k$ (recall that while $\mathcal{D}$ is deterministic, it may have several runs on an input which differ on trailing ε-transitions).
- Let c be the last configuration of the run of $\mathcal{P}$ induced by $\tau_0 \ldots \tau_k$. Then, $q_\mathcal{P}$ is the state of c and $q_\mathcal{A}$ is the state of $\mathcal{A}$ reached when processing the stack content of c.

Every pushdown resolver with only access to the current mode is a special case of a pushdown resolver with regular stack access. On the other hand, having regular access to the stack is strictly stronger than having just access to the mode. However, by adapting the underlying HD-PDA, one can show that the *languages* recognised by HD-PDA with pushdown resolvers does not increase when allowing regular stack access.

Lemma 10.4. *Every HD-PDA with a pushdown resolver with regular stack access can be turned into an equivalent HD-PDA with a pushdown resolver.*

Proof. Let $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ be an HD-PDA and let $(\mathcal{D}, \mathcal{A}, \lambda)$ be a pushdown resolver with stack access for $\mathcal{P}$. We keep track of the state $\mathcal{A}$ reaches on the current stack as in the proof of Lemma 3.2: If a stack content $\perp(X_1, q_1) \cdots (X_s, q_s)$ is reached, then q_j is the unique state of $\mathcal{P}$ reached when processing $\perp X_1 \cdots X_j$. Now, it is straightforward to turn $(\mathcal{D}, \mathcal{A}, \lambda)$ into a pushdown resolver for $\mathcal{P}$ that has only access to the top stack symbol. $\square$

Finally, for HD-VPA, the situation is again much better. Using the one-token game discussed in the proof of Theorem 6.6 and known results [LMS04] about VPA games having VPA strategies, we obtain our final theorem.

Theorem 10.5. *Every HD-VPA has a (visibly) pushdown resolver that can be computed in exponential time.*

Proof. Recall the one-token game that characterises HDness of a VPA $\mathcal{P}$ from the proof of Theorem 6.6. We argued there that the one-token game is a safety game played on the configuration graph of a deterministic VPA whose size is polynomial in the size of $\mathcal{P}$. Therefore, Player 2, when she wins, has a winning strategy that can be implemented by a visibly pushdown transducer, which can be computed in exponential time [Fri10, Wal01]. The proof of Theorem 6.6 shows that such a strategy for Player 2 in the one-token game for VPA induces a resolver. We conclude that HD-VPA have visibly pushdown resolvers computable in exponential time. $\square$

Note that in the above proof, we could also compute directly a strategy to the letter game, rather than the one-token game, by using the closure of VPA under complementation and union to encode the letter game as a safety game on the configuration graph of a VPA. However, the resulting VPA would be of exponential, rather than polynomial, size.

11. Conclusion

We have continued the study of history-deterministic pushdown automata initiated by Lehtinen and Zimmermann [LZ22], focusing here on expressiveness and succinctness. In particular, we have shown that HD-PDA are not only more expressive than DPDA (as had already been shown for the case of infinite words), but also more succinct than DPDA: We have introduced the first techniques for using HD nondeterminism to succinctly represent languages that do not depend on the coBüchi condition. Similarly, for the case of VPA, for which deterministic and nondeterministic automata are equally expressive, we proved a (tight) exponential gap in succinctness.

Solving games and universality are decidable for HD-PDA, but HDness is undecidable and HD-PDA have limited closure properties. On the other hand, HDness for VPA is decidable and they inherit the closure properties of VPA, e.g. union, intersection and complementation, making HD-VPA an exciting class of pushdown automata. Finally, we have studied the complexity of resolvers for HD-PDA, showing that positional ones always suffice, but that they are not always implementable by pushdown transducers. Again, HD-VPA are better-behaved, as they always have a resolver implementable by a VPA.

Let us conclude by mentioning some open problems raised by our work.

- Is universality for HD-PDA ExpTime-hard? Note that the analogous problem for HD-PDA over ω-words is also open [LZ22].
- It is known that the succinctness gap between PDA and DPDA is noncomputable [Har80, Val76], i.e. there is no computable function f such that any PDA of size n that has some equivalent DPDA also has an equivalent DPDA of size $f(n)$. Due to our hierarchy results, at least one of the succinctness gaps between PDA and HD-PDA and between HD-PDA and DPDA has to be uncomputable, possibly both.
- We have shown that some HD-PDA do not have pushdown resolvers, but there are no known upper bounds on the complexity of resolvers. It is even open whether every HD-PDA has at least a computable resolver.
- On the level of languages, it is open whether every language in HD-CFL has an HD-PDA recognising it with a resolver implementable by a pushdown transducer.
- We have shown that HDness is undecidable, both for PDA and for context-free languages. Is it decidable whether a given HD-PDA has an equivalent DPDA?
- Equivalence of DPDA is famously decidable [Sén01] while it is undecidable for PDA [HU79]. Is equivalence of HD-PDA decidable?
- Does every HD-PDA that is equivalent to a DPDA have a finite-state resolver with regular stack access?
- There is a plethora of fragments of context-free languages one can compare HD-CFL to, let us just mention a few interesting ones: Height-deterministic context-free languages [NS07], context-free languages with bounded nondeterminism [Her97] and preorder typeable visibly pushdown languages [KMV07].
- We have shown that history-determinism and good-for-gameness coincide for PDA over finite words; it remains open whether this is also the case for PDA over infinite words.

References

[AK20]   Shaull Almagor and Orna Kupferman. Good-enough synthesis. In Shuvendu K. Lahiri and Chao Wang, editors, *CAV 2020, Part II*, volume 12225 of *LNCS*, pages 541–563. Springer, 2020. doi:10.1007/978-3-030-53291-8_28.

[AM04]   Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *STOC 2004*, pages 202–211. ACM, 2004. `doi:10.1145/1007352.1007390`.

[BK18]   Marc Bagnol and Denis Kuperberg. Büchi Good-for-Games Automata Are Efficiently Recognizable. In Sumit Ganguly and Paritosh Pandya, editors, *FSTTCS 2018*, volume 122 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.FSTTCS.2018.16`.

[BKS17]   Udi Boker, Orna Kupferman, and Michał Skrzypczak. How deterministic are good-for-games automata? In Satya V. Lokam and R. Ramanujam, editors, *FSTTCS 2017*, volume 93 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPICS.FSTTCS.2017.18`.

[BL19]   Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In Wan Fokkink and Rob van Glabbeek, editors, *CONCUR 2019*, volume 140 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CONCUR.2019.19`.

[BL21]   Udi Boker and Karoliina Lehtinen. History determinism vs. good for gameness in quantitative automata. In Mikołaj Bojańczy and Chandra Chekuri, editors, *FSTTCS 2021*, volume 213 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.FSTTCS.2021.38`.

[BM12]   Henrik Björklund and Wim Martens. The tractability frontier for NFA minimization. *J. Comput. Syst. Sci.*, 78(1):198–210, 2012. `doi:10.1016/J.JCSS.2011.03.001`.

[BSSK13]   Henrik Björklund, Martin Schuster, Thomas Schwentick, and Joscha Kulbatzki. On optimum left-to-right strategies for active context-free games. In Wang-Chiew Tan, Giovanna Guerrini, Barbara Catania, and Anastasios Gounaris, editors, *ICDT 2013*, pages 105–116. ACM, 2013. `doi:10.1145/2448496.2448510`.

[Bü64]   J. Richard Büchi. Regular canonical systems. *Archiv für mathematische Logik und Grundlagenforschung*, 6(3):91–111, Apr 1964. `doi:10.1007/BF01969548`.

[CH14]   Arnaud Carayol and Matthew Hague. Saturation algorithms for model-checking pushdown systems. In Zoltán Ésik and Zoltán Fülöp, editors, *AFL 2014*, volume 151 of *EPTCS*, pages 1–24, 2014. `doi:10.4204/EPTCS.151.1`.

[CL15]   Arnaud Carayol and Christof Löding. Uniformization in automata theory. In *LMPS 2015*. 2015. URL: `https://hal.archives-ouvertes.fr/hal-01806575`.

[CLL+05]   Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005. `doi:10.1109/TIT.2005.850116`.

[Col09]   Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP 2009, (Part II)*, volume 5556 of *LNCS*, pages 139–150. Springer, 2009. `doi:10.1007/978-3-642-02930-1\_12`.

[Fin01]   Olivier Finkel. Topological properties of omega context-free languages. *Theor. Comput. Sci.*, 262(1):669–697, 2001. `doi:10.1016/S0304-3975(00)00405-9`.

[Fin12]   Olivier Finkel. The Determinacy of Context-Free Games. In Christoph Dürr and Thomas Wilke, editors, *STACS 2012*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 555–566, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.STACS.2012.555`.

[Fri10]   Wladimir Fridman. Formats of winning strategies for six types of pushdown games. In Angelo Montanari, Margherita Napoli, and Mimmo Parente, editors, *GANDALF 2010*, volume 25 of *EPTCS*, pages 132–145, 2010. `doi:10.4204/EPTCS.25.14`.

[GJLZ21]   Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. In Filippo Bonchi and Simon J. Puglisi, editors, *MFCS 2021*, volume 202 of *LIPIcs*, pages 53:1–53:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.MFCS.2021.53`.

[GLW05]   Jonathan Goldstine, Hing Leung, and Detlef Wotschke. Measuring nondeterminism in pushdown automata. *J. Comput. Syst. Sci.*, 71(4):440–466, 2005. `doi:10.1016/J.JCSS.2005.04.001`.

[GS53]   David Gale and F. M. Stewart. Infinite games with perfect information. In Harold William Kuhn and Albert William Tucker, editors, *Contributions to the Theory of Games (AM-28), Volume II*, chapter 13, pages 245–266. Princeton University Press, 1953.

[Har80]   Juris Hartmanis. On the succinctness of different representations of languages. *SIAM J. Comput.*, 9(1):114–120, 1980. `doi:10.1137/0209010`.

[Her97]   Christian Herzog. Pushdown automata with bounded nondeterminism and bounded ambiguity. *Theor. Comput. Sci.*, 181(1):141–157, 1997. `doi:10.1016/S0304-3975(96)00267-8`.

[HK10]   Markus Holzer and Martin Kutrib. Descriptional complexity of (un)ambiguous finite state machines and pushdown automata. In Antonín Kucera and Igor Potapov, editors, *RP 2010*, volume 6227 of *LNCS*, pages 1–23. Springer, 2010. `doi:10.1007/978-3-642-15349-5\_1`.

[HP06]   Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *CSL 2006*, volume 4207 of *LNCS*, pages 395–410. Springer, 2006. `doi:10.1007/11874683\_26`.

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

[JR93]   Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993. `doi:10.1137/0222067`.

[KM19]   Denis Kuperberg and Anirban Majumdar. Computing the width of non-deterministic automata. *Log. Methods Comput. Sci.*, 15(4), 2019. `doi:10.23638/LMCS-15(4:10)2019`.

[KMV07]   Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Visibly pushdown automata for streaming XML. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW 2007*, pages 1053–1062. ACM, 2007. `doi:10.1145/1242572.1242714`.

[KS15]   Denis Kuperberg and Michal Skrzypczak. On determinisation of good-for-games automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP 2015 (Part II)*, volume 9135 of *LNCS*, pages 299–310. Springer, 2015. `doi:10.1007/978-3-662-47666-6\_24`.

[KSV06]   Orna Kupferman, Shmuel Safra, and Moshe Y. Vardi. Relating word and tree automata. *Ann. Pure Appl. Log.*, 138(1-3):126–146, 2006. `doi:10.1016/J.APAL.2005.06.009`.

[LMS04]   Christof Löding, P. Madhusudan, and Olivier Serre. Visibly pushdown games. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004*, volume 3328 of *LNCS*, pages 408–420. Springer, 2004. `doi:10.1007/978-3-540-30538-5\_34`.

[LZ22]   Karoliina Lehtinen and Martin Zimmermann. Good-for-games ω-pushdown automata. *Log. Methods Comput. Sci.*, 18(1), 2022. `doi:10.46298/lmcs-18(1:3)2022`.

[Mar75]   Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. URL: `http://www.jstor.org/stable/1971035`.

[Mau69]   Hermann A. Maurer. A direct proof of the inherent ambiguity of a simple context-free language. *J. ACM*, 16(2):256–260, 1969. `doi:10.1145/321510.321517`.

[MNSB06] Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. Expressiveness and complexity of XML schema. *ACM Trans. Database Syst.*, 31(3):770–813, 2006. `doi:10.1145/1166074.1166076`.

[MSS06]   Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Active context-free games. *Theory Comput. Syst.*, 39(1):237–276, 2006. `doi:10.1007/S00224-005-1278-3`.

[NS07]   Dirk Nowotka and Jirí Srba. Height-deterministic pushdown automata. In Ludek Kucera and Antonín Kucera, editors, *MFCS 2007*, volume 4708 of *LNCS*, pages 125–134. Springer, 2007. `doi:10.1007/978-3-540-74456-6\_13`.

[OS15]   Alexander Okhotin and Kai Salomaa. Descriptional complexity of unambiguous input-driven pushdown automata. *Theor. Comput. Sci.*, 566:1–11, 2015. `doi:10.1016/j.tcs.2014.11.015`.

[RK19]   Bader Abu Radi and Orna Kupferman. Minimizing GFG Transition-Based Automata. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *ICALP 2019*, volume 132 of *LIPIcs*, pages 100:1–100:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ICALP.2019.100`.

[Sch20]   Sven Schewe. Minimising Good-For-Games Automata Is NP-Complete. In Nitin Saxena and Sunil Simon, editors, *FSTTCS 2020*, volume 182 of *LIPIcs*, pages 56:1–56:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.FSTTCS.2020.56`.

[Sén01]   Géraud Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theor. Comput. Sci.*, 251(1-2):1–166, 2001. `doi:10.1016/S0304-3975(00)00285-1`.

[SI85]   Richard Edwin Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.*, 14(3):598–611, 1985. `doi:10.1137/0214044`.

[SS78]   Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer, 1978. `doi:10.1007/978-1-4612-6264-0`.

[SY93]   Kai Salomaa and Sheng Yu. Limited nondeterminism for pushdown automata. *Bull. EATCS*, 50:186–193, 1993.

[Val76]   Leslie G. Valiant. A note on the succinctness of descriptions of deterministic languages. *Inf. Control.*, 32(2):139–145, 1976. `doi:10.1016/S0019-9958(76)90173-X`.

[Wal01]   Igor Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001. `doi:10.1006/inco.2000.2894`.