# A Bit of Nondeterminism Makes Pushdown Automata Expressive and Succinct

**Shibashis Guha**
Tata Institute of Fundamental Research, Mumbai, India

**Ismaël Jecker**
IST Austria, Klosterneuburg, Austria

**Karoliina Lehtinen**
CNRS, Aix-Marseille University and University of Toulon, LIS, Marseille, France

**Martin Zimmermann**
University of Liverpool, UK

## Abstract

We study the expressiveness and succinctness of good-for-games pushdown automata (GFG-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.

We prove that GFG-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 GFG-PDA can be exponentially more succinct than DPDA, while PDA can be double-exponentially more succinct than GFG-PDA. We also study GFGness in visibly pushdown automata (VPA), which enjoy better closure properties than PDA, and for which we show GFGness to be ExpTime-complete. GFG-VPA can be exponentially more succinct than deterministic VPA, while VPA can be exponentially more succinct than GFG-VPA. Both of these lower bounds are tight.

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

**2012 ACM Subject Classification** Theory of computation → Formal languages and automata theory

**Keywords and phrases** Pushdown Automata, Good-for-games, Synthesis, Succintness

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2021.53

**Related Version** *Full Version*: https://arxiv.org/abs/2105.02611 [14]

**Funding** *Ismaël Jecker*: Funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 754411.
*Karoliina Lehtinen*: Funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 892704.

## 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 [15, 38]. 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 [11, 19], while they are decidable for DPDA [39].


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 [18], while Okhotin and Salomaa show that unambiguous visibly pushdown automata are exponentially more succinct that DPDA [31]. Universality of unambiguous PDA is decidable, as it is decidable for unambiguous context-free grammars [33], which are effectively equivalent [17]. 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 (GFG) nondeterminism [16] and history-determinism [10]. Intuitively, a nondeterministic automaton is GFG 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, GFG nondeterminism does not even add succinctness: every GFG-NFA contains an equivalent DFA [6], which can be obtained by pruning transitions from the GFG-NFA. Thus, GFG-NFA cannot be more succinct than DFA. But for finite automata on infinite words, where nondeterminism again only adds succintness, but not expressiveness, GFG coBüchi automata can be exponentially more succinct than deterministic automata [23]. Finally, for certain quantitative automata over infinite words, GFG nondeterminism adds as much expressiveness as arbitrary nondeterminism [10].

Recently, pushdown automata on infinite words with GFG nondeterminism ($\omega$-GFG-PDA) were shown to be strictly more expressive than $\omega$-DPDA, while universality and solving games for $\omega$-GFG-PDA are not harder than for $\omega$-DPDA [25]. Thus, GFG nondeterminism adds expressiveness without increasing the complexity of these problems, i.e. pushdown automata with GFG nondeterminism induce a novel and intriguing class of context-free $\omega$-languages.

Here, we continue this work by studying the expressiveness *and* succinctness of PDA over finite words. While the decidability results for $\omega$-GFG-PDA on infinite words also hold for GFG-PDA on finite words, the separation argument between $\omega$-GFG-PDA and $\omega$-DPDA depends crucially on combining GFG 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 GFG-PDA are more expressive than DPDA, yielding the first class of automata on finite words where GFG nondeterminism adds expressiveness. The language witnessing the separation is remarkably simple, in contrast to the relatively subtle argument for the infinitary result [25]: the language $\{a^i\$a^j\$b^k\$ \mid k \leqslant \max(i,j)\}$ is recognised by a GFG-PDA but not by a DPDA. This yields a new class of languages, those recognised by GFG-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 succinctness of GFG-PDA. We show that the succinctness gap between DPDA and GFG-PDA is at least exponential, while the gap between GFG-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 good-for-games nondeterminism being used effectively over finite, rather than infinite, words (recall that all GFG-NFA are determinisable by pruning). Also, this

is the first succinctness result for good-for-games automata that does not depend on the infinitary coBüchi acceptance condition, which was used to show the exponential succinctness of GFG coBüchi automata, as compared to deterministic ones [23].

We then study an important subclass of GFG-PDA, namely, GFG visibly pushdown automata (VPA), in which the stack behaviour (push, pop, skip) is determined by the input letter only. GFG-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 GFG-VPA, as well as between GFG-VPA and VPA. Both of these are tight, as VPA, and therefore GFG-VPA as well, admit an exponential determinisation procedure [2]. Furthermore, we show that GFGness of VPA is decidable in ExpTime. This makes GFG-VPA a particularly interesting class of PDA as they are recognisable, succinct, have good closure properties and deciding universality and solving games are both in ExpTime. In contrast, solving $\omega$-VPA games is 2ExpTime-complete [27]. We also relate the problem of checking GFGness with the *good-enough synthesis* [1] or *uniformization* problem [9], which we show to be ExpTime-complete for DVPA and GFG-PDA.

Nondeterminism in GFG 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 GFG automata. For example, finite GFG automata (on finite and infinite words) have a finite-state resolver [16]. For pushdown automata with their infinite configuration space, the situation is markedly different: On one hand, we show that GFG-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 GFG-PDA over finite words, but not for $\omega$-GFG-PDA. Yet, positionality does not imply that resolvers are simple to implement. We show that there are GFG-PDA that do not admit a resolver implementable by a pushdown transducer. In contrast, all GFG-VPA admit pushdown resolvers, again showing that GFG-VPA are better behaved than general GFG-PDA. Finally, GFG-PDA with finite-state resolvers are determinisable.

All proofs omitted due to space restrictions can be found in the full version [14].

## Related work

The notion of GFG nondeterminism has emerged independently several times, at least as Colcombet's history-determinism [10], in Piterman and Henzinger's GFG automata [16], 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 [24]. 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 [21] and Martens, Neven, Schwentick, and Bex [28] respectively, also consider nondeterminism which can be resolved on-the-fly. However, the restrictions there are stronger than simple GFG 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 GFG automata, although this remains out of scope for the present paper. The XML extension AXML has also inspired Active Context Free Games [29], 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 [29, 5]; however, since the player still knows the future of the word, this restriction is not directly comparable to GFG 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 [37]. Minimization remains NP-complete for both unambiguous automata [20, 4] and GFG automata [35] (at least when acceptance is defined on states, see [32]). On pushdown automata, increasing the permitted degree of ambiguity leads to both greater expressiveness and unbounded succinctness [17]. Finally, let us mention two more ways of measuring–and restricting–nondeterminism in PDA: bounded nondeterminism, as studied by Herzog [17] counts the branching in the run-tree of a word, while the minmax measure [34, 13] counts the number of nondeterministic guesses required to accept a word. The natural generalisation of GFGness as the *width* of an automaton [22] has not yet, to the best of our knowledge, been studied for PDA.

## 2 Preliminaries

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

For alphabets $\Sigma_1, \Sigma_2$, we extend functions $f\colon \Sigma_1 \to \Sigma_2^*$ homomorphically to words over $\Sigma_1$ 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 $\Sigma$, a stack alphabet $\Gamma$, a transition relation $\Delta$ 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 $\Delta$ 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 $\tau$ is an $\ell(\tau)$-transition and that $\tau$ is a $\Sigma$-transition, if $\ell(\tau) \in \Sigma$. For a finite sequence $\rho$ over $\Delta$, the word $\ell(\rho) \in \Sigma^*$ is defined by applying $\ell$ homomorphically to every transition. We take the size of $\mathcal{P}$ to be $|Q| + |\Gamma|$.[1]

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 $\gamma$. 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 $(q, X, a, q', \gamma')$ 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 $\tau$ 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 $\tau = (q, X, a, q', \gamma')$.

A run of $\mathcal{P}$ is a finite sequence $\rho = c_0\tau_0c_1\tau_1\cdots c_{n-1}\tau_{n-1}c_n$ 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 $\rho$ is a run of $\mathcal{P}$ on $w \in \Sigma^*$, if $w = \ell(\rho)$. We say that $\rho$ 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 $w$.

[1] 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.

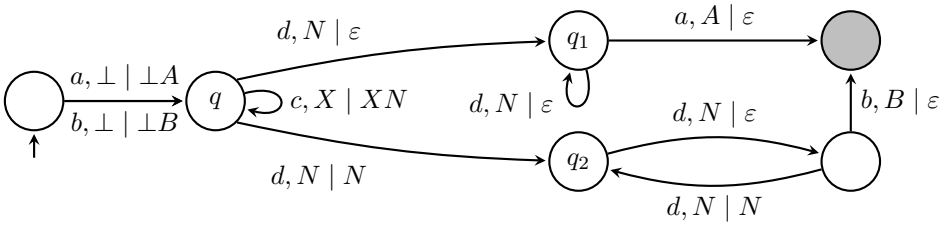


**Figure 1** The PDA $\mathcal{P}$ from Example 2. Grey states are final, and $X$ is an arbitrary stack symbol.

▶ Remark 1. Let $c_0\tau_0c_1\tau_1\cdots c_{n-1}\tau_{n-1}c_n$ be a run of $\mathcal{P}$. Then, the sequence $c_0c_1\cdots c_{n-1}c_n$ 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 $\varepsilon$-transition, then it does not enable any $\Sigma$-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 $\varepsilon$-transitions, a DPDA can have more than one run on a given input. However, these only differ by trailing $\varepsilon$-transitions.

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

▶ **Example 2.** The PDA $\mathcal{P}$ depicted in Figure 1 recognizes the language $\{ac^nd^na \mid n \geqslant 1\} \cup \{bc^nd^{2n}b \mid n \geqslant 1\}$. Note that while $\mathcal{P}$ is nondeterministic, $L(\mathcal{P})$ is in DCFL.

## 2.2 Good-for-games Pushdown Automata

Here, we introduce good-for-games pushdown automata on finite words (GFG-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. 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 good-for-games.

Formally, a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ is good-for-games if there is a (nondeterminism) resolver for $\mathcal{P}$, a function $r\colon \Delta^* \times \Sigma \to \Delta$ such that for every $w \in L(\mathcal{P})$, there is an accepting run $\rho = c_0\tau_0\cdots\tau_nc_n$ on $w$ that has no trailing $\varepsilon$-transitions, i.e.

1. $n = 0$ if $w = \varepsilon$ (which implies that $c_0$ is accepting), and
2. $\ell(\tau_0\cdots\tau_{n-1})$ is a strict prefix of $w$, if $w \neq \varepsilon$,

and $\tau_{n'} = r(\tau_0\cdots\tau_{n'-1}, w(|\ell(\tau_0\cdots\tau_{n'-1})|))$ for all $0 \leqslant n' < n$. If $w$ is nonempty, then $w(|\ell(\tau_0\cdots\tau_{n'-1})|)$ is defined for all $0 \leqslant n' < n$ by the second requirement. Note that $\rho$ is unique if it exists.

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 $\varepsilon$-transitions. This is the reason that the run prefix and not the input prefix is the argument for the resolver. We denote the class of languages recognised by GFG-PDA by GFG-CFL.

Intuitively, every DPDA *should* be good-for-games, 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* $\varepsilon$-transitions after the last letter of $w$ is processed. On the other hand, the run of a GFG-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 a GFG-PDA. Nevertheless, we show, using standard pushdown automata constructions, that every DPDA can be turned into an equivalent GFG-PDA. As every GFG-PDA is a PDA by definition, we obtain a hierarchy of languages.

▶ **Lemma 3.** *DCFL* $\subseteq$ *GFG-CFL* $\subseteq$ *CFL.*

Instead of requiring that GFG-PDA end their run with the last letter processed, one could add an end-of-word marker that allows traversing trailing $\varepsilon$-transitions after the last letter has been processed. In Appendix A.1, we show that this alternative definition does not increase expressiveness, which explains our (arguably simpler) definition.

Finally, let us remark that GFGness of PDA and context-free languages is undecidable. These problems were shown to be undecidable for $\omega$-GFG-PDA and $\omega$-GFG-CFL by reductions from the inclusion and universality problem for PDA on finite words [25]. The same reductions also show that these problems are undecidable over PDA on finite words.

▶ **Theorem 4.** *The following problems are undecidable:*

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

## 2.3 Games and Universality

One of the motivations for GFG automata is that solving games with winning conditions given by a GFG automaton 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 [11], both over finite and infinite words, while for $\omega$-GFG-PDA, it is ExpTime-complete [25]. As a corollary, universality is also decidable for $\omega$-GFG-PDA, while it is undecidable for PDA, both over finite and infinite words [19].

Here, we consider Gale-Stewart games [12], 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. Note that these games are different, but related, to games played on configuration graphs of pushdown automata [39].

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 $i = 0, 1, \ldots$ 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 $\Sigma_1^+$ to $\Sigma_2$ that gives for each prefix played by Player 1 the next letter to play. A play agrees with a strategy $\sigma$ if for each $i$, $b_i = \sigma(a_0 a_1 \ldots a_i)$. Player 2 wins $G(L)$ if she has a strategy that only agrees with plays that 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 GFG-PDA easily reduces to the case of $\omega$-GFG-PDA, which are just GFG-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 $\omega$-GFG-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 5.** *Given a GFG-PDA $\mathcal{P}$, there is a safety $\omega$-GFG-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 $(q, X, a, q', \gamma)$ of $\mathcal{P}$ with $a \in \Sigma$ and with non-final $q'$.

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 GFG-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 GFG, 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 $w'$. ◀

Our main results of this section are now direct consequences of the corresponding results on $\omega$-words [25].

▶ **Corollary 6.** *Given a GFG-PDA $\mathcal{P}$, deciding whether $L(\mathcal{P}) = \Sigma^*$ and whether Player 2 wins $G(L(\mathcal{P}))$ are both in ExpTime.*

## 2.4 Closure properties

Like $\omega$-GFG-PDA, GFG-PDA have poor closure properties.

▶ **Theorem 7.** *GFG-PDA are not closed under union, intersection, complementation, set difference and homomorphism.*

The proofs are similar to those used for $\omega$-GFG-PDA and relegated to the full version [14]. There, we also study the closure properties under these operations with regular languages: If $L$ is in GFG-CFL and $R$ is regular, then $L \cup R$, $L \cap R$ and $L \backslash R$ are also in GFG-CFL, but $R \backslash L$ is not necessarily in GFG-CFL.

# 3 Expressiveness

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

▶ **Theorem 8.** *DCFL $\subsetneq$ GFG-CFL $\subsetneq$ CFL.*

To show that GFG-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. The first two states $q_1$ and $q_2$ deterministically push the input onto the stack, until the occurrence of the second \$. When the second \$ is processed, there is a nondeterministic choice to move to $p_1$ or $p_2$ and erase along $\varepsilon$-transitions 1 or 0 blocks

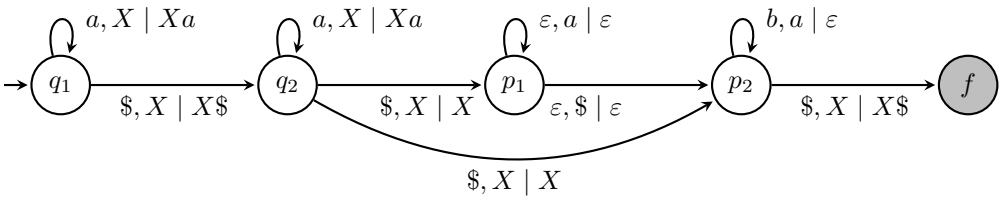


**Figure 2** A PDA $\mathcal{P}_{B_2}$ recognising $B_2$. Grey states are final, and $X$ is an arbitrary stack symbol.

from the stack, so that the 1st or 2nd block of $a$'s respectively remains at the top of the stack. Then, the automaton compares the length of the $b$-block in the input with the length of the $a$-block at the top of the stack and accepts if the $b$-block is shorter, i.e. the third \$ is processed before the whole $a$-block is popped off the stack. If the input has not the form $a^i\$a^j\$b^k\$$, then it is rejected.

We show that $B_2 \in$ GFG-CFL by proving that $\mathcal{P}_{B_2}$ is good-for-games: the nondeterministic choice between moving to $p_1$ or to $p_2$ can be made only based on the prefix $a^i\$a^j$ processed so far. This is straightforward, as a resolver only needs to know which of $i$ and $j$ is larger, which can be determined from the run prefix constructed thus far. Then, 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.

Finally, to show that PDA are more expressive than GFG-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 $L \notin$ GFG-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 GFGness are related: To conclude this section, we show that both notions are independent.

▶ **Theorem 9.** *There is an unambiguous context-free language that is not in GFG-CFL and a language in GFG-CFL that is inherently ambiguous.*

An unambiguous grammar for the language $\{a^n b^n \mid n \geqslant 0\} \cup \{a^n b^{2n} \mid n \geqslant 0\} \notin$ GFG-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 GFG-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 GFG-PDA but are cumbersome in the proof of inherent ambiguity.

## 4 Succinctness

We show that GFG-PDA are not only more expressive than DPDA, but also more succinct. Similarly, we show that PDA are more succinct than GFG-PDA.

▶ **Theorem 10.** *GFG-PDA can be exponentially more succinct than DPDA, and PDA can be double-exponentially more succinct than GFG-PDA.*

We first show that GFG-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 a GFG-DPDA of size $O(n)$, yet every DPDA regognising $C_n$ 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 $C_n = \{w \in \{0, 1, \$, \#\}^* \mid w \neq c_n\#\} \subseteq \{0, 1, \$, \#\}^*$ of bad counters.

We show that the language $C_n$ is recognised by a GFG-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 GFG-PDA that are equivalent to DPDA are not determinisable by pruning. In contrast, for NFA, GFGness implies determinisability by pruning [6].

We conclude this section by showing that PDA can be double-exponentially more succinct than GFG-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 GFG-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 $n^{\text{th}}$ bit from the end is a 1. We count starting from 1, so that the last bit is the $1^{\text{st}}$ 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.

## 5 Good-for-games Visibly Pushdown Automata

One downside of GFG-PDA is that, like $\omega$-GFG-PDA, they have poor closure properties and checking GFGness is undecidable. We therefore consider a well-behaved class of GFG-PDA, namely GFG visibly pushdown automata, GFG-VPA for short, that is closed under union, intersection, and complementation.

Let $\Sigma_c, \Sigma_r$ and $\Sigma_{\mathsf{int}}$ 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* [2] (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 $a \in \Sigma_{\mathsf{r}}$ is only processed by transitions of the form $(q, X, a, q', \varepsilon)$ with $X \neq \perp$ or $(q, \perp, a, q', \perp)$, 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 $X \in \Gamma_\perp$, i.e. the stack is left unchanged.
- There are no $\varepsilon$-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 GFG-VPA the VPA that are good-for-games. Every VPA (and hence every GFG-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 [2]. However, VPA can be exponentially more succinct than deterministic VPA (DVPA) [2]. We show that there is an exponential gap both between the succinctness of GFG-VPA and DVPA and between VPA and GFG-VPA. The proof of the former gap again uses a language of bad counters, similar to $C_n$ used in Theorem 10, 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 GFG-VPA, we similarly adapt the language $L_n$ of words where the $n^{\text{th}}$ bit from the end is a 1, from the proof of Theorem 10, by making sure that the stack height is always bounded by 1. Then, a GFG-VPA is essentially a GFG-NFA, and therefore determinisable by pruning, which means that it is as big as a deterministic automaton for the language.

▶ **Theorem 11.** *GFG-VPA can be exponentially more succinct than DVPA and VPA can be exponentially more succinct than GFG-VPA.*

We now turn to the question of deciding whether a given VPA is GFG. We show decidability using the *one-token game*, introduced by Bagnol and Kuperberg [3]. It modifies the game-based characterisation of GFGness of $\omega$-regular automata by Henzinger and Piterman [16]. While the one-token game does not characterise the GFGness of Büchi automata, here we show that it suffices for VPA. The matching lower bound follows from a reduction from the inclusion problem for VPA, which is ExpTime-hard [2], to GFGness (see [25] for details of the reduction in the context of $\omega$-GFG-PDA).

▶ **Theorem 12.** *The following problem is ExpTime-complete: Given a VPA $\mathcal{P}$, is $\mathcal{P}$ GFG?*

We first define the *one-token game*, introduced by Bagnol and Kuperberg [3] in the context of regular languages, for VPA. Given a VPA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$, the positions of the one-token game consist of pairs of configurations $(c_i, c'_i)$, starting from the initial configuration of $\mathcal{P}$. At each round $i$:

- Player 1 picks a letter $a_i \in \Sigma$,
- Player 2 picks an $a_i$-transition $\tau_i \in \Delta$ enabled in $c_i$, leading to a configuration $c_{i+1}$,
- Player 1 picks an $a_i$-transition $\tau'_i \in \Delta$ enabled in $c'_i$, leading to a configuration $c'_{i+1}$,
- The game proceeds from the configuration $(c_{i+1}, c'_{i+1})$.

A play consists of an infinite word $a_0 a_1 \cdots \in \Sigma^\omega$ and two sequences of transitions $\tau_0 \tau_1 \cdots$ and $\tau'_0 \tau'_1 \cdots$ built by Players 2 and 1 respectively. Player 1 wins if for some $n$, $\tau'_0 \cdots \tau'_n$ is an accepting run of $\mathcal{P}$ over $a_0 \dots a_n$ and $\tau_0 \cdots \tau_n$ is not. Recall that VPA do not have $\varepsilon$-transitions, so the two runs proceed in lockstep.

Observe that this game can be seen as a safety game on a visibly pushdown arena and can therefore be encoded as a Gale-Stewart game with a DCFL winning condition. This in turn is solvable in ExpTime [39]. To prove Theorem 12, it now suffices to argue that this game characterises whether the VPA $\mathcal{P}$ is GFG.

**Proof.** We now argue that $\mathcal{P}$ is GFG if and only if Player 2 wins the one-token game on $\mathcal{P}$. One direction is immediate: if $\mathcal{P}$ is GFG, then the resolver is also a strategy for Player 2 in the one-token game.

For the converse direction, 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 $s$, she wins, in particular, against this family of copycat strategies for Player 1. Observe that copycat strategies win any play along which Player 2 plays a non-residual transition. Therefore $s$ must avoid ever playing a non-residual transition. We can now use $s$ to induce a resolver $r_s$ for $\mathcal{P}$: $r_s$ maps a sequence of transitions over a word $w$ to the transition chosen by $s$ in the one-token game where Player 1 played $w$ and a copycat strategy. Then, $r_s$ never produces a non-residual transition. As a result, if a word $w$ is in $L(\mathcal{P})$, then the run induced by $r_s$ 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 $r_s$ induces an accepting run. This concludes our argument that $r_s$ is indeed a resolver, and $\mathcal{P}$ is therefore GFG.

Thus, to decide whether a VPA $\mathcal{P}$ is GFG it suffices to solve the one-token game on $\mathcal{P}$, which can be done in exponential time. ◀

Finally, we relate the GFGness problem to the *good-enough synthesis* problem [1], also known as the *uniformization* problem [9], which is similar to the Church synthesis problem, 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 13** (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 GFG-VPA and DVPA is as hard as the GFGness problem for VPA, giving us the following corollary of Theorem 12.

▶ **Corollary 14.** *The GE-synthesis problem for inputs given by GFG-VPA, and in particular for DVPA, is ExpTime-complete.*

**Proof.** We first reduce the good-enough synthesis problem to the GFGness problem. Given a GFG-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 has an $a$-transition for some $a \in \Sigma_1$ whenever $\mathcal{P}$ has the same transition over $\binom{a}{b}$ for some $b \in \Sigma_2$. Let each transition of $\mathcal{P}'$ be annotated with the $\Sigma_2$-letter of the corresponding $\mathcal{P}$-transition. Thus $\mathcal{P}'$ recognises the projection of $L(\mathcal{P})$ on the first component.

A resolver for $\mathcal{P}'$ induces a GE-synthesis function for $\mathcal{P}$ by reading the $\Sigma_2$-annotation of the chosen transitions in $\mathcal{P}'$. Indeed, the resolver produces an accepting run with annotation $w'$ 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 $L(\mathcal{P})$. Conversely 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.

We now reduce the GFGness 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 $\tau$ over a letter $a$ in $\Delta$ is replaced with the same transition over $\binom{a}{\tau}$ in $\Delta'$. 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. ◀

In contrast, for LTL specifications, the GE-synthesis problem is 2ExpTime-complete [1].

## 6 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 GFG-PDA need. We consider two somewhat orthogonal notions of complexity: memory and machinery. On one hand, we show that resolvers can always be chosen to be *positional*, that is, dependent on the current state

and stack configuration only. Note that this is not the case for $\omega$-regular automata[2], let alone $\omega$-GFG-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 $\rho$ and $\rho'$ of transitions inducing runs ending in the same configuration, $r(\rho, a) = r(\rho', a)$ for all $a \in \Sigma$.

▶ **Lemma 15.** *Every GFG-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 reachable configuration $c$, let $\rho_c$ 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 $\rho$.

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 $\rho$ 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 $\rho$ from which the rest of the word, say $u$, is accepted[3] (by some other run of $\mathcal{P}$ having the same prefix as $\rho$ up to configuration $c$). Let $\tau$ be the next transition along $\rho$ from $c$. Since $r$ chose $\tau$, the resolver $r'$ also chooses $\tau$ 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 $\tau$, contradicting that $c$ is the last position on $\rho$ from where the rest of the word could be accepted. ◀

Contrary to the case of finite and $\omega$-regular automata, since GFG-PDA have an infinite configuration space, the existence of positional resolvers does not imply determinisability. On the other hand, if a GFG-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 GFG-PDA are determinisable by pruning, e.g. the GFG-PDA for the languages $C_n$ used to prove Theorem 10.

We now turn to how powerful resolvers for GFG-PDA need to be. 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 GFG-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 GFG-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 $\Theta$. The input alphabet of $\mathcal{T}$ is the input alphabet of $\mathcal{D}$.

---

[2] A positional resolver for $\omega$-regular automata implies determinisability by pruning, and we know that this is not always possible [6].

[3] 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 [23] as the existence of positional resolvers implies determinisability by pruning.

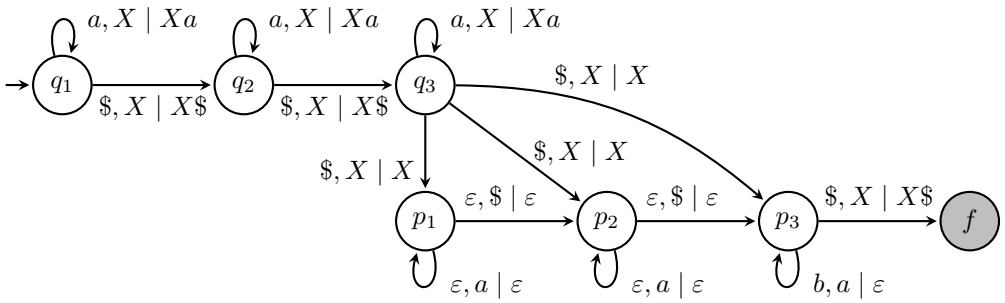


**Figure 3** The PDA $\mathcal{P}_{B_3}$ for $B_3$. Grey states are final, and $X$ is an arbitrary stack symbol.

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 $\Delta$ 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 $\mathcal{P}$: $r_\mathcal{T}(\tau_0 \dots \tau_k, a) = \lambda(q_\mathcal{T})(q_\mathcal{P}, X, a)$ where

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

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 GFG-PDA to determine the next transition in the GFG-PDA.

We now give an example of a GFG-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 3 which *does* have a pushdown resolver. Let $\mathcal{P}_{B_3}$ be the automaton in Figure 3, 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 GFG: 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 GFG-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 16.** *The GFG-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 $\mathcal{T} = (\mathcal{D}, \lambda)$.

From $\mathcal{T}$, for each $i \in \{1, 2, 3\}$, we can construct a PDA $\mathcal{D}_i$ that recognises the language of words $w \in a^*\$a^*\$a^*$ such that $\mathcal{T}$ chooses from $q_3$ the transition of $\mathcal{P}_{B_3}$ going to $p_i$ when constructing a run on $w\$$: this is simply the pushdown automaton $\mathcal{D}$ underlying $\mathcal{T}$ where transitions of $\mathcal{D}$ processing non-$\varepsilon$ transitions $\tau$ of $\mathcal{P}_{B_3}$ are modified to now process $\ell(\tau) \in \{a, b, \$\}$, transitions of $\mathcal{D}$ processing $\varepsilon$-transitions of $\mathcal{P}_{B_3}$ are removed, and states $q$ of $\mathcal{D}$ such that $\lambda(q)(q_3, X, \$) = (q_3, \$, X, p_i, X)$ are made final, intersected with a DFA checking that the input is in $a^*\$a^*\$a^*$.

Since $\mathcal{T}$ implements a resolver for $\mathcal{P}$, each $\mathcal{D}_i$ only accepts words of the form $a^{m_1}\$a^{m_2}\$a^{m_3}$ such that $\max(m_1, m_2, m_3) = m_i$. Furthermore, at least for one $i \in \{1, 2, 3\}$, $\mathcal{D}_i$ accepts $a^m\$a^m\$a^m$ for infinitely many $m$.

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 $i^{\text{th}}$ block and is non-empty, by setting $n = 0$ we obtain a contradiction as the $i$th 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 $\mathcal{P}$. ◀

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 $\omega$-GFG-PDA [26], the product of a GFG-PDA and a finite-state resolver yields a DPDA for the same language.

▶ Remark 17. Every GFG-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 $a^{10}$. A GFG-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 $\varepsilon$-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 good-for-games, 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).

In Appendix A.2 we consider another model of pushdown resolver, namely one that does not only have access to the mode of the GFG-PDA, but can check the full stack for regular properties. We show that this change does not increase the class of good-for-games context-free languages that are recognised by a GFG-PDA with a pushdown resolver.

Finally, for GFG-VPA, the situation is again much better. The classical game-based characterisation of GFGness of $\omega$-regular automata by Henzinger and Piterman [16] can be lifted to VPA. Then, using known results [27] about VPA games having VPA strategies, we obtain our final theorem.

▶ **Theorem 18.** *Every GFG-VPA has a (visibly) pushdown resolver.*

**Proof.** Fix a VPA $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ and consider the following two-player game $\mathcal{G}(\mathcal{P})$, introduced by Henzinger and Piterman to decide GFGness of $\omega$-automata [16]. In each round, first Player 1 picks a letter from $\Sigma$ or ends the play. If he has not ended the play, then Player 2 picks a transition of $\mathcal{P}$. Hence, once Player 1 has stopped the play, Player 1 has picked an input word $w$ over $\Sigma^*$ and Player 2 has indicated a run $\rho$ of $\mathcal{P}$. A finite play with outcome $(w, \rho)$ is winning for Player 2 if either $w \notin L(\mathcal{P})$ or $\rho$ induces an accepting run of $\mathcal{P}$ on $w$. A strategy for Player 2 in this game is a mapping $\sigma\colon \Sigma^+ \to \Delta$ and an outcome $(w(0) \cdots w(k), \rho(0) \cdots \rho(k))$ is consistent with $\sigma$, if $\rho(j) = \sigma(w(0) \cdots w(j))$ for every $0 \leqslant j \leqslant k$. We say that $\sigma$ is winning for Player 2, if every outcome of a finite play that is consistent with $\sigma$ is winning for her (note that we disregard infinite plays).

Now, Player 2 wins $\mathcal{G}(\mathcal{P})$ if and only if $\mathcal{P}$ is a GFG-VPA. This follows as every winning strategy for Player 2 can be turned into a resolver and vice versa.

Now, as the class of languages recognized by VPA, is closed under complementation and union [2], one can encode $\mathcal{G}(\mathcal{P})$ as a Gale-Stewart game with a VPL winning condition. Such games can be solved effectively [27] and the winner always has a winning strategy implemented by a (visibly) PDT. Thus, if $\mathcal{P}$ is a GFG-VPA, i.e. Player 2 wins $\mathcal{G}(\mathcal{P})$, then she has a winning strategy implemented by a (visibly) PDT, which can easily be turned into a (visibly) pushdown resolver for $\mathcal{P}$. ◀

## 7 Conclusion

We have continued the study of good-for-games pushdown automata, focusing on expressiveness and succinctness. In particular, we have shown that GFG-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 GFG 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 GFG-PDA, but GFGness is undecidable and GFG-PDA have limited closure properties. On the other hand, GFGness for VPA is decidable and they inherit the closure properties of VPA, e.g. union, intersection and complementation, making GFG-VPA an exciting class of pushdown automata. Finally, we have studied the complexity of resolvers for GFG-PDA, showing that positional ones always suffice, but that they are not always implementable by pushdown transducers. Again, GFG-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.

- It is known that the succinctness gap between PDA and DPDA is noncomputable [15, 38], 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 GFG-PDA and between GFG-PDA and DPDA has to be uncomputable, possibly both.
- We have shown that some GFG-PDA do not have pushdown resolvers. It is even open whether every GFG-PDA has a computable resolver.
- On the level of languages, it is open whether every language in GFG-CFL has a GFG-PDA recognising it with a resolver implementable by a pushdown transducer.
- We have shown that GFGness is undecidable, both for PDA and for context-free languages. Is it decidable whether a given GFG-PDA has an equivalent DPDA?
- Equivalence of DPDA is famously decidable [36] while it is undecidable for PDA [19]. Is equivalence of GFG-PDA decidable?
- Does every GFG-PDA that is equivalent to a DPDA have a finite-state resolver with regular stack access (see Appendix A.2 for definitions)?
- There is a plethora of fragments of context-free languages one can compare GFG-CFL to, let us just mention a few interesting ones: Height-deterministic context-free languages [30], context-free languages with bounded nondeterminism [17] and preorder typeable visibly pushdown languages [21].

## References


1 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.

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

3 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.

4 Henrik Björklund and Wim Martens. The tractability frontier for NFA minimization. *Journal of Computer and System Sciences*, 78(1):198–210, 2012.

5 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.

6 Udi Boker, Orna Kupferman, and Michał Skrzypczak. How deterministic are good-for-games automata? *arXiv*, 2017. arXiv:1710.04115.

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

8 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.

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

10 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.

11 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.

12 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.

13 Jonathan Goldstine, Hing Leung, and Detlef Wotschke. Measuring nondeterminism in pushdown automata. *Journal of Computer and System Sciences*, 71(4):440–466, 2005.

14 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. *arXiv*, 2021. arXiv:2105.02611.

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

16 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.

17 Christian Herzog. Pushdown automata with bounded nondeterminism and bounded ambiguity. *Theor. Comput. Sci.*, 181(1):141–157, 1997.

18 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.

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

**20** Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

**21** 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.

**22** 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.

**23** 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.

**24** Orna Kupferman, Shmuel Safra, and Moshe Y Vardi. Relating word and tree automata. *Annals of Pure and Applied Logic*, 138(1-3):126–146, 2006.

**25** Karoliina Lehtinen and Martin Zimmermann. Good-for-games $\omega$-pushdown automata. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS 2020*, pages 689–702. ACM, 2020. doi:10.1145/3373718.3394737.

**26** Karoliina Lehtinen and Martin Zimmermann. Good-for-games $\omega$-pushdown automata. *arXiv*, 2020. arXiv:2001.04392.

**27** 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.

**28** Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. Expressiveness and complexity of XML schema. *TODS*, 31(3):770–813, 2006.

**29** Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Active context-free games. *Theory of Computing Systems*, 39(1):237–276, 2006.

**30** 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.

**31** 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.

**32** 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.

**33** 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.

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

**35** 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.

**36** 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.

**37** Richard Edwin Stearns and Harry B Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611, 1985.

**38** 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.

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

# A Appendix

## A.1 Resolvers with End-of-word Markers

As mentioned in the main part, GFG-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 $\varepsilon$-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 GFG-PDA with end-of-word resolvers are as expressive as standard GFG-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 19.** *GFG-PDA with EoW-resolvers are as expressive as GFG-PDA.*

**Proof.** A (standard) resolver can be turned into an EoW-resolver that ignores the EoW-marker. Hence, every GFG-PDA is a GFG-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

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

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

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

can be shown to be regular as well by applying saturation techniques [7][4] to the restriction of $\mathcal{P}$ to $\varepsilon$-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 $\mathcal{A} = (Q_\mathcal{A}, \Gamma_\bot \cup Q, q_I^\mathcal{A}, \delta_\mathcal{A}, F_\mathcal{A})$ 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 $\bot(X_1, q_1, q_1')(X_2, q_2, q_2') \cdots (X_s, q_s, q_s')$, then we have

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

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 good-for-games, 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.

[4] Also, see the survey by Carayol and Hague [8] for more details.

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 $\mathcal{P}'$ recognizes $L(\mathcal{P})$ and has on every word $w \in L(\mathcal{P})$ an accepting run without trailing $\varepsilon$-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. ◀

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 10, we show that the language $L_n$ of bit strings whose $n^{\text{th}}$ bit from the end is a 1 requires exponentially-sized GFG-PDA. On the other hand, it is straightforward to devise a polynomially-sized GFG-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) $\varepsilon$-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 good-for-games.

## A.2 Pushdown Resolvers with Regular Stack Access

Recall that pushdown transducers implementing a resolver have access to the mode of the GFG-PDA whose nondeterminism it resolves. Here, 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 a DPDA $\mathcal{P}$ with input alphabet $\Delta$, a DFA $\mathcal{A}$ over $\Gamma_\perp$ with state set $Q^{\mathcal{A}}$, and an output function $\lambda$ 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 $\mathcal{P}$:

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

where

- $q_{\mathcal{T}}$ is the state of the last configuration of the longest run of $\mathcal{D}$ on $\tau_0 \dots \tau_k$ (recall that while $\mathcal{D}$ is deterministic, it may have several runs on an input which differ on trailing $\varepsilon$-transitions).
- Let $c$ be the last configuration of the run of $\mathcal{P}$ induced by $\tau_0 \dots \tau_k$. 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 GFG-PDA, one can show that the *languages* recognised by GFG-PDA with pushdown resolvers does not increase when allowing regular stack access.

▶ **Lemma 20.** *Every GFG-PDA with a pushdown resolver with regular stack access can be turned into an equivalent GFG-PDA with a pushdown resolver.*

**Proof.** Let $\mathcal{P} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$ be a GFG-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 19: If a stack content $\bot(X_1, q_1) \cdots (X_s, q_s)$ is reached, then $q_j$ is the unique state of $\mathcal{P}$ reached when processing $\bot X_1 \cdots X_j$. 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. ◀