Explorability in Pushdown Automata

Ayaan Bedi ✉ 🏠 🆔
Chennai Mathematical Institute, India

Karoliina Lehtinen ✉ 🏠 🆔
CNRS Researcher, Aix-Marseille Université, LIS, France

─── **Abstract** ───────────────────────────

We study *explorability*, a measure of nondeterminism in pushdown automata, which generalises history-determinism. An automaton is k-explorable if, while reading the input, it suffices to follow k concurrent runs, built step-by-step based only on the input seen so far, to construct an accepting one, if it exists. We show that the class of explorable PDAs lies strictly between history-deterministic and fully nondeterministic PDAs in terms of both expressiveness and succinctness. In fact increasing explorability induces an infinite hierarchy: each level k defines a strictly more expressive class than level $k-1$, yet the entire class remains less expressive than general nondeterministic PDAs. We then introduce a parameterized notion of explorability, where the number of runs may depend on input length, and show that exponential explorability precisely captures the context-free languages. Finally, we prove that explorable PDAs can be doubly exponentially more succinct than history-deterministic ones, and that the succinctness gap between deterministic and 2-explorable PDAs is not recursively enumerable. These results position explorability as a robust and operationally meaningful measure of nondeterminism for pushdown systems.

1 Introduction

Determinism and nondeterminism are central themes in theoretical computer science, particularly in the study of computational models in automata theory and complexity theory. Nondeterminism often brings either increased expressive power or more succinct representations. For example, in the case of finite automata, while nondeterminism does not increase expressive power, it allows for exponentially more succinct representations. In contrast, for Turing machines, while expressiveness remains unchanged, the distinction between deterministic and nondeterministic variants leads to the foundational open problem of the field: *P vs. NP*. Pushdown automata (PDAs), which are strictly more powerful than finite automata, exhibit both increased expressiveness and non recursively ennumerable succinctness when nondeterminism is introduced.

While deterministic models are often seen as overly restrictive, unrestricted nondeterminism functions as a computational "superpower", capable of always making the correct choice. This raises the question of whether is there a meaningful spectrum between determinism and full nondeterminism and how to usefully quantify the degree of nondeterminism.

A variety of measures of nondeterminism have been suggested for pushdown automata. Goldstine et al. quantified nondeterminism by counting the minimum number of guesses (in bits) needed to accept a word, which induces an infinite hierarchy of complexity classes and language subclasses [2], while Han et al. measure computation tree width and ambiguity [4].

45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025).
Editors: C. Aiswarya, Ruta Mehta, and Subhajit Roy; Article No. 12; pp. 12:1–12:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Another approach is to restrict nondeterminism to bounded or context-dependent forms [1]. For example, Herzog characterizes pushdown automata with at most k nondeterministic choices as unions of k deterministic context-free languages, noting tradeoffs in descriptive complexity that are sometimes not recursively bounded [10]. Models with regulated non-determinism (using control sets or stack-content constraints) describe language classes that lie strictly between the deterministic and fully nondeterministic cases.

One well-studied intermediate form of nondeterminism is *bounded ambiguity*, which restricts the number of accepting runs per input – for instance, to at most one (unambiguity), or at most k, for some fixed constant k. In parallel, another notable form of restricted nondeterminism that simplifies decision problems like universality and game solving is *history-determinism* (HD), also known as *good-for-games* (GFG) nondeterminism [9]. The key idea behind HD automata is that their nondeterministic choices can be resolved in a forward, step-by-step manner, based only on the input processed so far, without any need to anticipate future input symbols [3].

Recently, history-deterministic pushdown automata (HD-PDAs) have been shown to be more expressive than their deterministic counterparts [3]. Moreover, HD-PDAs were shown to enjoy at least exponential succinctness over deterministic PDAs. However, when compared to general nondeterministic PDAs, HD-PDAs are strictly less expressive and also less succinct, by at least doubly exponential factor [3]. Furthermore, at least one of these succinctness gaps is far from tight, since the succinctness gap between nondeterministic and deterministic PDAs is not recursively enumerable (RE) [6] – the question of which one, or both, was left open.

This significant gap between HD-PDAs and general nondeterministic PDAs motivates the exploration of intermediate classes of languages. By identifying and characterizing such classes, we aim to better understand the trade-offs between expressiveness, succinctness, and the degree of nondeterminism permitted in pushdown computations. Therefore we study the notion of *explorability*, which was introduced for finite state automata by Hazard and Kuperberg [8] as a generalisation of history-determinism. Intuitively, k-explorability corresponds to having k separate runs that are being resolved on-the-fly, with the requirement that for any word in the language, at least one of the runs should be accepting.

The problem of recognizing explorable NFAs is ExpTime-complete for automata over finite words and for Büchi automata. In the setting of infinite words, recognizing ω-explorable automata is also ExpTime-complete for safety and co-Büchi acceptance conditions [8]. Moreover Hazard, Idir and Kuperberg showed that there is a trade-off between the explorability of an automaton and the number of priorities needed to recognise all ω-regular languages [7].

Our contributions.   In this work, we extend the study of PDAs by investigating their expressiveness and succinctness across increasing levels of explorability. We show that *explorable* PDAs are strictly more expressive than HD-PDAs. Furthermore, we establish an infinite hierarchy within the class of explorable PDAs: k-explorable PDAs are strictly less expressive than $(k + 1)$-explorable ones. We also prove that the class of explorable PDAs is strictly less expressive than the class of general nondeterministic PDAs.

We then ask under what conditions does the class of explorable PDAs match the expressive power of the class of fully nondeterministic PDAs? We answer this by introducing a *parameterized* notion of explorability, where the number of concurrent runs may depend on the length of the input word. In this setting, we demonstrate a hierarchy between constant-bounded, linearly-bounded, and exponentially-bounded explorability. Exponentially explorable PDAs capture exactly the class of context-free languages, and are therefore

equivalent in expressiveness to general nondeterministic PDAs. While we only study this notion over PDAs, the notion of parameterised explorability might be of independent interest, as is the case for parameterised ambiguity.

We then turn our attention to the succinctness of explorable PDAs. We prove that the succinctness gap between HD-PDAs and general explorable PDAs is at least doubly exponential. Moreover, we establish that a similar doubly exponential gap persists even when explorability is restricted to a constant. Delving further into the non-RE separation between deterministic and nondeterministic models, we show that the succinctness gap between DPDAs and 2-explorable PDAs is not recursively enumerable. That is, there exists no recursive bound on the size blow-up required to simulate certain 2-explorable PDAs by DPDAs.

2 Preliminaries

Pushdown Automaton

▶ **Definition 1.** *A* pushdown automaton *(PDA) is a computational model that extends finite automata with a stack, providing additional memory. Formally, a PDA is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where:*
- *Q is a finite set of states,*
- *Σ a finite input alphabet,*
- *Γ a finite stack alphabet,*
- *$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^{\leq 2}}$ is the transition function,*
- *$q_0 \in Q$ is the initial state,*
- *$Z_0 \in \Gamma$ is the initial stack symbol, and*
- *$F \subseteq Q$ is the set of accepting states.*

At any point during computation, the *mode* of a PDA is of the form $Q \times \Gamma$ and is determined by its current state and the symbol at the top of the stack. The behavior of the PDA is defined by the transition function δ, which may allow multiple transitions from the same mode. The transitions enabled at a mode, runs, accepting runs, and language of an automaton are defined as usual [11]. The *size* of a PDA is the product of the number of states $|Q|$ and the number of stack symbols $|\Gamma|$.

PDAs accept exactly the class of languages known as *context-free languages* (CFLs). A PDA is said to be *deterministic* if for all $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, and $X \in \Gamma$, the set $\delta(q, a, X)$ contains at most one element, and if $\delta(q, \varepsilon, X) \neq \emptyset$, then $\delta(q, a, X) = \emptyset$ for all $a \in \Sigma$. The class of languages accepted by deterministic PDAs (DPDAs) is called the class of *deterministic context-free languages* (DCFLs), and it is known that DCFL $\subsetneq$ CFL, i.e., DCFLs form a strict subset of CFLs [11].

3 Explorability of Pushdown Automata

The *k*-Explorability Game for Pushdown Automata

As described by Hazard et al. [8], *explorability* restricts or measures the amount of non-determinism an automaton needs to accept a language, while allowing more flexibility than history determinism (HD). For a given $k \in \mathbb{N}$, an automaton is said to be *k-explorable* if, when processing an input, it is sufficient to keep track of most k runs in order to construct an accepting run, if one exists. This generalizes the idea of HD automata, which corresponds to the special case where $k = 1$.

We now introduce the *k-explorability game* [8] over a PDA, which generalises the game-based definition of HD-automata, and captures a hierarchy of nondeterministic behavior in pushdown automata (PDAs). This game is played between two players – *Spoiler* and *Determiner* – on a nondeterministic PDA A.

▶ **Definition 2** (*k*-Explorability Game for PDAs)**.** *Let* $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *be a pushdown automaton, and let* $k \in \mathbb{N}$ *be a fixed integer. The k-explorability game on P is played on the arena* $Q^k \times (\Gamma^*)^k$, *where each of k tokens maintains its own copy of the control state and stack content. We can imagine a configuration here as k tokens say 1... k placed in the space* $Q \times \Gamma^*$ *and arena is the set of all possible configurations.*

The game proceeds as follows:

▬ *Initialization: The initial configuration is a k-tuple of identical PDA configurations:*

$$S_0 = ((q_0, Z_0), \dots, (q_0, Z_0))$$

▬ *Gameplay: At each step* $i \geq 1$, *from configuration* S_{i-1}:
   1. *Spoiler chooses an input symbol* $a_i \in \Sigma$.
   2. *Determiner responds by selecting the next configuration*

   $$S_i = ((q_i^1, \gamma_i^1), \dots, (q_i^k, \gamma_i^k)) \in (Q \times \Gamma^*)^k$$

   such that for each $l \in \{1, \dots, k\}$, *there exists a path of transitions in P from* $(q_{i-1}^l, \gamma_{i-1}^l)$ *to* (q_i^l, γ_i^l) *of the form:*

   $$(q_{i-1}^l, \gamma_{i-1}^l) \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \xrightarrow{a_i} (q_i^l, \gamma_i^l)$$

   where each step is valid under the PDA transition relation δ.

▬ *The play continues for an infinite number of steps and a word* $w = a_1 a_2 \dots$ *is formed.*

A play *is an infinite sequence* $\pi = S_0\, t_1\, S_1\, t_2\, S_2 \dots$ *such that:* $S_0 = ((q_0, Z_0), \dots, (q_0, Z_0)) \in (Q \times \Gamma^*)^k$, *is a k-tuple of initial state* q_0 *and initial stack configuration* Z_0. *And for all* $i \geq 1$, *and for each* $l \in \{1, \dots, k\}$, *the transition* $(q_{i-1}^l, \gamma_{i-1}^l) \xrightarrow{t_i^l} (q_i^l, \gamma_i^l)$ *is valid according to the PDA's transition relation* δ.

The play is won by Determiner *if for every finite prefix of* $w = a_1 a_2 \dots$ *that is in the language of* L, *i.e.* $w' = a_1 \dots a_n \in L(P)$, *there exists at least one token* $l \in \{1, \dots, k\}$ *such that the sequence of configurations for token l forms an accepting run of P on* w'. *Otherwise,* Spoiler *wins the game.*

We say that a PDA P is *k-explorable* (Expl_k-PDA) if Determiner has a winning strategy in the *k*-explorability game on P. If P is *k*-explorable for some $k \in \mathbb{N}$, we say that P is *explorable* (Expl-PDA). When $k = 1$ then it's exactly the game characterising HD ([9, Section 3]) and therefore HD is 1-explorable.

▶ **Definition 3.** *We define* Expl_k-CFL *as the class of languages recognized by k-explorable nondeterministic pushdown automata. That is,*

$$\mathsf{Expl}_k\text{-CFL} = \{L \subseteq \Sigma^* \mid L \text{ is recognized by a k-explorable PDA}\}.$$

▶ **Definition 4.** *We define* Expl-CFL *as the class of languages recognized by* explorable *pushdown automata. Formally,* Expl-CFL $= \bigcup_{k \in \mathbb{N}} \mathsf{Expl}_k$-CFL.

▶ **Definition 5** (*k*-Run in a *k*-Explorable PDA)**.** *Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a nondeterministic pushdown automaton, and let $k \in \mathbb{N}$. A k-run of P on an input word $w = a_1 a_2 \ldots a_n \in \Sigma^*$ in the k-explorability setting is a sequence of k-tuples of configurations:*

$$\mathcal{R} = (C_0^1, \ldots, C_0^k), (C_1^1, \ldots, C_1^k), \ldots, (C_n^1, \ldots, C_n^k)$$

such that:
- *For all $1 \le l \le k$, $C_0^l = (q_0, Z_0)$ is the initial configuration (same across all tokens),*
- *For each step $i = 1, \ldots, n$ and each token $l = 1, \ldots, k$, there exists a valid PDA path:*

$$C_{i-1}^l \xrightarrow{a_i} C_i^l,$$

A k-run $\mathcal{R}$ is said to be accepting *if there exists at least one token $l \in \{1, \ldots, k\}$ such that the final configuration $C_n^l = (q_n^l, \gamma_n^l)$ satisfies $q_n^l \in F$ (i.e., the control state is accepting).*

Parameterised Explorability

The notion of *k*-explorability captures bounded nondeterminism by allowing the Determiner to explore at most k computational branches in parallel. However, in many settings, the amount of explorability required to resolve nondeterminism may grow with the size of the input. To capture this, we extend the explorability to a *parameterized* framework, where the number of tokens available to Determiner is not fixed but instead given by a function $f(n)$ of the input length n.

▶ **Definition 6** ($f(n)$-Parameterized Explorability Game)**.** *Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a nondeterministic pushdown automaton, and let $f : \mathbb{N} \to \mathbb{N}$ be a computable function. The $f(n)$-parameterized explorability game on P is played between two players,* **Spoiler** *and* **Determiner***, as follows:*
- **Initialization:** *Spoiler announces a number $n \in \mathbb{N}$, declaring the maximum length of the word he will play. Determiner is then given $f(n)$ tokens, each representing a copy of the PDA configuration.*
- **Gameplay:** *is similar to as defined for k-Explorability Game.*

The play is won by Determiner *if for every prefix of the Spoiler's word in $w \in \Sigma^{\le n} \cap L(P)$, there exists at least one token $l \in \{1, \ldots, f(n)\}$ such that the sequence of configurations for token l forms an accepting run of P on w. Otherwise, Spoiler wins.*

A PDA P is said to be $f(n)$-*explorable* if Determiner has a winning strategy in the $f(n)$-parameterized explorability game on P. Let C be a class of functions (such as $O(n)$); we define the language class:

$$\mathsf{Expl}_C\text{-}\mathsf{CFL} = \{L \subseteq \Sigma^* \mid \exists \text{ an } f(n)\text{-explorable PDA } P \text{ such that } L = L(P) \text{ and } f(n) \in C\}.$$

4 Expressiveness

It is well known that deterministic pushdown automata (DPDA) recognize the class of deterministic context-free languages (DCFL), which is strictly contained within the class of context-free languages (CFL) recognized by nondeterministic pushdown automata (PDA). Recent work by Guha et al. [3] shows that HD-PDA defines a language class (HD-CFL) that properly extends DCFL but is still strictly contained within CFL.

▶ **Theorem 7** (Theorem 4.1 [3])**.** *DCFL $\subsetneq$ HD-CFL $\subsetneq$ CFL*

We now turn our attention to the expressive power of *explorable pushdown automata*. Analogous to the results on history-deterministic pushdown automata, we investigate the hierarchy induced by allowing increasing, but bounded, explorability. In particular, we show that k-explorability induces an infinite hierarchy, which is strictly contained within CFL:

▶ **Theorem 8.** *For all k,* $\mathsf{Expl}_k\text{-CFL} \subsetneq \mathsf{Expl}_{k+1}\text{-CFL} \subsetneq CFL$.

4.1 Expl_k-CFL $\subsetneq$ Expl_{k+1}-CFL

To prove that $(k + 1)$-explorable pushdown automata are strictly more expressive than k-explorable ones, we consider, for each $i \in \{1, \ldots, k\}$, the language $L_i := \{a^n b^{in} \mid n \in \mathbb{N}\}$. We then define the language $L := \bigcup_{i=1}^{k+1} L_i$.

We claim that L is accepted by a $k + 1$-explorable pushdown automaton but cannot be recognized by any k-explorable pushdown automaton. Intuitively, accepting L requires distinguishing among k different linear relations between the number of a's and b's, each corresponding to a different multiplicative factor i. This requires k distinct tokens, one for each L_i, which cannot be simulated with fewer runs.

▶ **Lemma 9.** *The language L is recognized by a $(k + 1)$-explorable pushdown automaton.*

Proof. For each i, L_i is recognized by a DPDA D_i that pushes i symbols onto the stack at each a and then compares the number of b's to the stack height by popping the stack as it reads b. The automaton that initially nondeterministically chooses among ε-transitions to the initial states of each D_i for $i \in [1..k + 1]$ clearly recognises L and is $k + 1$-explorable since Determiner can win the $k + 1$-explorability game by having each D_i explored by a distinct token.   ◀

L is not k-explorable

We observe that a k-explorable PDA P would have to accept some words $a^n b^\ell$ and their continuation $a^n b^{\ell'}$ along the same run, as Determiner does not have enough tokens to check membership to each L_i with a disjoint run. Therefore, there would be some accepting runs that can be extended into another accepting run via more b-transitions. Then, we build a PDA P' which is similar to P, except that after seeing an accepting state, it can also read c's instead of b's. We show, using a pumping argument, that this automaton would recognise a language that is not CFL, leading to a contradiction. The difficulty in this proof is that we do not have an exact description for the language recognised by P' as it depends on exactly which pairs of words P accepts along a single run, and the hypothesis of k-explorability only tells us that this must occur for some pairs ℓ, ℓ' for each n. However, we can describe $L(P')$ in enough detail to show that it is, in any case, not CFL.

We define the language $L_{i,j,n} := \{a^n b^{in} c^{jn}\}$, where $i, j, n \in \mathbb{N}$. Each $L_{i,j,n}$ consists of strings with block sizes linearly dependent on n, with coefficients i and j for the lengths of the b and c blocks, respectively.

Now we define a language L_S such that for each $n \in \mathbb{N}$, the set of words of the form $a^n b^m c^k \in L'$ is contained in a finite and nonempty union of such languages $L_{i,j}, n$, but the choice of the coefficents is and js may vary with n. Formally, given $S = (S_n)_{n \in \mathbb{N}}$ where each $S_n \subseteq \mathbb{Z}^+ \times \mathbb{Z}^+$ is finite and non-empty.

$$L_S := \bigcup_{n \in \mathbb{N}} \left(\{a^n\} \cdot \bigcup_{(i,j) \in S_n} \{b^{in} c^{jn}\} \right),$$

That is, for every $n \in \mathbb{N}$, L_S consists of a finite union of languages of the form $L_{i,j,n}$. In this way, L_S represents a "non-uniform" or "n-dependent" union of linearly structured languages.

▶ **Lemma 10.** *Given $S = (S_n)_{n \in \mathbb{N}}$ where each $S_n \subseteq \mathbb{Z}^+ \times \mathbb{Z}^+$ is finite and non-empty, the language L_S is not context-free.*

The proof is a pumping argument, detailed in the Appendix.

▶ **Lemma 11.** *The language $L := \bigcup_{i=1}^{k+1} L_i$, where $L_i := \{a^n b^{in} \mid n \in \mathbb{N}\}$, is not recognized by any k-explorable PDA.*

Proof. Assume, towards contradiction, that there exists a k-explorable PDA $\mathcal{A}$ that recognizes L. Since L includes strings of the form $a^n b^{in}$ for $i \in \{1, 2, \ldots, k+1\}$, and we only have k tokens, by the pigeonhole principle, for any fixed n, for any winning Determiner strategy, there must be at least one token that accepts two different strings $a^n b^{in}$ and $a^n b^{jn}$ for some $i < j$. That means that the run built by this token visits an accepting state at least twice. We define S_n as the pairs i, j such that there is an accepting run over $a^n b^{jn}$ of which a prefix accepts $a^n b^{in}$. By the pigeonhole principle, that S_n is non-empty for all n.

We use this fact to construct a new PDA $\mathcal{A}_c$ that accepts the language L_S, for $S = (S_n)_{n \in \mathbb{N}}$ as described above. As previously shown, L_S is not context-free, so the existence of such a PDA will lead to a contradiction.

Let $\mathcal{A} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$, with $Q = \{q_0, q_1, \ldots, q_n\}$ and $q_I = q_0$. We define a modified PDA $\mathcal{A}_c = (Q \cup Q', \Sigma', \Gamma, q_I, \Delta', F')$, where:

- $\Sigma' = \{a, b, c\}$,
- $Q' = \{q'_0, q'_1, \ldots, q'_n\}$ is a copy of Q,
- $F' = \{q'_f \mid q_f \in F\}$,
- $\Delta' = \Delta \cup \Delta_c$, with Δ_c defined as:
  1. For every $q_f \in F$, add $(q_f, X, \varepsilon, q'_f, X)$ for all $X \in \Gamma_\perp$ (transferring control to the new state space),
  2. For every transition $(q_i, X, b, q_j, \gamma) \in \Delta$, add $(q'_i, X, c, q'_j, \gamma) \in \Delta_c$ – replacing the input symbol b with c in the copied state space.
  3. For every $(q_i, X, \varepsilon, q_j, \gamma) \in \Delta$, add $(q'_i, X, \varepsilon, q'_j, \gamma) \in \Delta_c$. - to replicated ε-transitions

Intuitively, the modified PDA simulates the original input $a^n b^{in}$ on $\mathcal{A}$, and then, upon acceptance, continues in the copied state space simulating a second phase where each b transition is replaced by a c-transition, thereby accepting strings of the form $a^n b^{in} c^{(j-i)n}$.

For each pair in some S_n, there is an accepting run in P that extends into another accepting run. Therefore P' accepts the word $a^n b^{in} c^{(j-i)n}$ for each $(i, j) \in S_n$. Hence A_c accepts $L_S \cup L$. This implies that L_S is context-free (by intersection with the regular language $a^* b^* c^*$) – a contradiction, since we have already shown in Lemma 4.2 that $L_S \notin \mathsf{CFL}$.

Thus, no k-explorable PDA can recognize L. ◀

We conclude the first half of **Theorem 8**: For all $k \in \mathbb{N}$, the class of languages recognized by $(k+1)$-explorable pushdown automata strictly contains the class of languages recognized by k-explorable pushdown automata. That is, $\mathsf{Expl}_k\text{-CFL} \subsetneq \mathsf{Expl}_{k+1}\text{-CFL}$.

4.2 Expl-CFL $\subsetneq$ CFL

To separate the class of explorable context-free languages from the full class of context-free languages, we consider the following language:

$$L_{block} := \{(a^* \#)^* b^n \mid n = \text{length of some } a\text{-block}\}.$$

Intuitively, L_{block} consists of strings formed by a sequence of a-blocks separated by $\#$, followed by a block of b's whose length matches the length of some previous a-block. For example, the string $a^3\#a^5\#a^2\#b^5 \in L_{block}$, since the final b-block has length 5, which matches the length of the second a-block.

While L_{block} is clearly CFL, the intuition is that each of Determiner's tokens can only compare one block of a's to the final block of b's, and as the number of a-blocks is unbounded, so is the number of tokens that she would need to win the explorability game, making this language not explorable.

▶ **Lemma 12.** $L_{block} \in \boldsymbol{CFL}$

Proof. A nondeterministic PDA can guess and store the length of one of the a-blocks on the stack, then skip the rest of the input until it reaches the b-block, and verify the count by popping the stack.   ◀

▶ **Lemma 13.** *The language* $L_{block} := \{(a^*\#)^* b^n \mid n = \text{length of some } a\text{-block}\}$ *is not recognized by an explorable PDA.*

Proof. Assume, towards a contradiction, that for some k, $L_{block} \in \mathsf{Expl}_k\text{-CFL}$, i.e., it is recognized by an k-explorable PDA. Observe that $\mathsf{Expl}_k\text{-CFL}$ is closed under intersection with regular languages since taking a product with a deterministic finite automaton does not increase the explorability of the PDA.

Let us take the regular language $R = (a\#)^{k+1} b^*$. Then the intersection

$$L_{k-block} := L_{block} \cap R = \left\{ (a^*\#)^{k+1} b^n \mid n = \text{length of some } a\text{-block} \right\}$$

must also be in $\mathsf{Expl}_k\text{-CFL}$ under our assumption.

So the problem comes down to showing that $L_{k-block} \notin \mathsf{Expl}_k\text{-CFL}$. Assume, towards a contradiction, that $L_{k-block}$ is recognized by a k-explorable PDA A. Let w be a word in $(a^*\#)^{k+1}$. In the k-explorability game, since there are $k+1$ a-blocks but only k tokens, if all the blocks are of distinct lengths then, at least one token will accept both a word and its prefix. Therefore, for each such w, there are some accepting runs that can be extended into another accepting run via more b-transitions. Let S'_w be the set of pairs of integers such that both wb^i and wb^j are accepted along the same run. Note that for each $(i,j) \in S'_w$, both i and j match the length of some a-block in w and $j > i$. Call $S' = (S'_w)_{w \in W}$ where W consists of $(a^*\#)^{k+1}$ such that S'_w is non-empty. Note that S'_w is necessarily non-empty for w in which the block lengths are all distinct, and may or may not be empty otherwise.

As in Lemma 11 we can then build a second PDA, in which ε-transitions move from the final states of a first copy of A to the same state in another copy, in which b-transitions are replaced by c-transitions. Taking the product with a DFA for $(a^*\#)^{k+1} b * c*$, this results in a PDA A' for the language:

$$L_{block-S'} := \left\{ wb^i c^{i-j} \mid w \in (a^*\#)^{k+1}, (i,j) \in S'_w \right\}.$$

By construction, if $L_{k-block} \in \mathsf{Expl}_k\text{-CFL}$, then $L_{block-S'}$ is accepted by the A'. However, from Lemma 14, this language is not CFL. This contradicts that it is recognized by a PDA.

Hence, $L_{k-block} \notin \mathsf{Expl}_k\text{-CFL}$, and therefore $L_{block} \notin \mathsf{Expl}_k\text{-CFL}$ as well.   ◀

▶ **Lemma 14.** *Given* $S' = (S'_w)_{w \in (a^*\#)^{k+1}}$ *in which some* S'_w *are non-empty, then the language* $L_{block-S'}$ *is not context-free.*

The proof, detailed in the appendix, consists of a pumping argument.

▶ **Theorem 15.** *Expl-CFL $\subsetneq$ CFL*

Proof. Follows directly from Lemma 12 and Lemma 13: we have exhibited a context-free language L_{block} which is not in Expl_k-CFL, for any k, establishing the strict inclusion. ◀

5 Parametrized Explorability and Its Expressiveness

In the previous section, we established the existence of an expressiveness gap across different fixed levels of exploration. However, in that analysis, the exploration was treated as a constant, independent of the input. In this section, we investigate the consequences of parameterising the level of exploration as a function of the input length, as characterised by the parameterised explorability game. This allows us to study how expressiveness scales when the exploration level is allowed to grow with the size of the input.

We are led to the following conjecture, which captures the hierarchy induced by varying exploration bounds:

$$\mathsf{Expl}_{O(1)}\text{-CFL} \subsetneq \mathsf{Expl}_{O(n)}\text{-CFL} \subsetneq \mathsf{Expl}_{exp}\text{-CFL} = \mathsf{CFL}$$

Here, we write *exp* for the set of exponential functions, that is, in $O(2^{p(n)})$ for some polynomial p. Intuitively $\mathsf{Expl}_{O(f(n))}$-CFL is the class of languages recognised by a pushdown automaton that is allowed to have at most $O(f(n))$ tokens for *exploration*, where n is the length of the input. Observe that $\mathsf{Expl}_{O(1)}$-CFL is precisely the the class Expl-CFL discussed so far.

The conjecture above posits a strict hierarchy of language classes recognized by explorable pushdown automata under increasing exploration bounds. At the lower end, constant-bounded exploration ($O(1)$) is provably limited in expressiveness, while allowing exploration to grow linearly with input length ($O(n)$) yields strictly greater power. Ultimately, permitting exponential exploration suffices to recover the full class of context-free languages.

▶ **Lemma 16.** $\mathsf{Expl}_{O(1)}$-CFL $\subsetneq$ $\mathsf{Expl}_{O(n)}$-CFL

Proof. In the previous section, we saw that the language

$$L_{block} := \{(a^*\#)^* b^n \mid n = \text{length of some } a\text{-block}\}$$

cannot be recognized by any k-explorable pushdown automaton for any $k \in \mathbb{N}$ (Lemma 13). Consequently, we conclude that $L_{block} \notin \mathsf{Expl}_{O(1)}$-CFL.

However, it is straightforward to observe that $L_{block} \in \mathsf{Expl}_{O(n)}$-CFL. Indeed, we can construst a PDA that nondeterministically chooses an a-block and compares it with the b-block. Since the total number of such a-blocks is bounded by the length of the input word, it suffices to use $O(n)$ runs to cover all possible cases. Within each run, the automaton can compare the length of an a-block with the number of b's at the end of the input. If a match is found, the input is accepted. Hence, the inclusion $\mathsf{Expl}_{O(1)}$-CFL $\subsetneq$ $\mathsf{Expl}_{O(n)}$-CFL is strict. ◀

▶ **Lemma 17.** $\mathsf{Expl}_{\exp}$-CFL = CFL

Proof. Let $L \in$ CFL. Then there exists a PDA M that accepts L. It is well known that for any PDA, there exists an equivalent PDA that accepts the same language but does not use ε-transitions[13], so w.l.o.g. we assume M to have no ε-transitions.

Let m be the maximum number of nondeterministic transitions possible from any state in M'. Consider an input word w of length n. Since the machine can make at most m choices at each step, the number of possible runs of M' on input w is at most m^n.

Therefore, the total number of possible runs grows at most exponentially with the length of the input. Therefore exponentially many tokens are enough to explore each of the possible runs. Hence, $\mathsf{CFL} \subseteq \mathsf{Expl}_{\exp}\text{-}\mathsf{CFL}$.

The reverse inclusion is trivial. Therefore, $\mathsf{CFL} = \mathsf{Expl}_{\exp}\text{-}\mathsf{CFL}$. ◄

6 Succinctness

We now shift our focus to the succinctness of pushdown automata. Recall that the size of a PDA is the product $|Q| \cdot |\Gamma|$, where Q is the set of states and Γ is the stack alphabet. We argue that significant succinctness gaps exist between different levels of exploration. We build on several results from the work of Guha et al. [3].

We demonstrate that *explorable* PDAs are not only more expressive than history-deterministic PDAs (HD-PDAs) and deterministic PDAs (DPDAs), but also more succinct. Furthermore, we show that fixing the degree of exploration yields a *double-exponential gap* in succinctness when compared to unrestricted explorable PDAs.

6.1 Explorable vs. HD-PDA

Guha et al [3] showed that a nondeterministic pushdown automaton (PDA) can recognize the language $L_n = (0+1)^*1(0+1)^{n-1}$ using only $O(\log(n))$ space, while any history-deterministic PDA (HD-PDA) requires space exponential in n. This establishes a double-exponential gap in succinctness between PDAs and HD-PDAs.

In this work, we demonstrate that the same double exponential gap occurs already between HD-PDAs and explorable PDAs. Specifically, we show that an explorable PDA can recognize L_n using only $O(\log(n))$ space, highlighting the succinctness advantage of explorability over history-determinism.

▶ **Lemma 18.** *There exists an explorable PDA of size $O(\log n)$ recognising Ln.*

Proof. Guha et al. [3] described a PDA P_n that recognises L_n in $O(\log(n))$ space. We adjust their construction to get $\mathcal{P'}_n$ that is also n-explorable. The PDA $\mathcal{P}_n$ nondeterministically guesses the nth bit from the end of the input, checks that it is a 1 or a 0 and stored that in the state space, then switches to a counting gadget that verifies the input ends in exactly n steps, as follows:

1. 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.
2. 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 0s until a 1 is at the top of the stack, say m 0s. Then, it replaces the 1 with a 0, and finally pushes m 1s back onto the stack before processing the next letter. If the stack empties before a 1 is found, then the counter value is 0, and the automaton moves to an accepting state or dummy state depending on the initial 1-check. Note that $O(\log n)$ states again suffice for this step.
3. To make the automaton P'_n, we keep the description of P_n and add an ε-transition from the final state and dummy state (as deribed above) back to the initial state.

Now, consider the n tokens, numbered $0, 1, \ldots, n-1$, and an input word $w = w_1 w_2 \ldots w_m$. Each token (i) starts the logarithmic counting at input positions j such that $j \bmod n = i$. Therefore, if the input contains a 1 at the nth position from the end, then there exists at

least one token that reaches the final state of the PDA. Furthermore, if an additional input symbol is read after reaching the final state, the ε-transition takes the token to the initial state, and the log-space counter is reinitialized. This allows the token to repeat the counting procedure from the next position.

Thus, $\mathcal{P}'_n$ has $O(\log n)$ states and is n-explorable. 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. ◀

We can now state the following succinctness result.

▶ **Theorem 19.** Expl-CFL *can be double-exponentially more succinct than* HD-PDA.

6.2 k-Explorable PDA vs. Explorable PDA

We now compare the succinctness gap between PDA of bounded and unbounded explorability, using the following family of languages.

$$L_{mod\ n} := (0+1)^{<n}(1(0+1)^{n-1})^*$$

That is, a word belongs to $L_{mod\ n}$ if every position that is a multiple of n, counting from the end, contains a 1.

▶ **Lemma 20.** *Any k-explorable pushdown automaton (PDA) that accepts the language $L_{mod\ n}$ must have size at least $\frac{\sqrt[k]{2^n}}{n}$*

Proof. 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 follows. Consider a run $c_0 \tau_0 c_1 \tau_1 \cdots c_{n-1} \tau_{n-1} c_n$. A position $s \in \{0, \ldots, n\}$ is a *step* if for all $s' \geq s$, we have that $\text{sh}(c_{s'}) \geq \text{sh}(c_s)$, that is, the stack height is always at least $\text{sh}(c_s)$ after position s. Any infinite run of a PDA has infinitely many steps. We have the following observation:

▶ Remark 21. 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.

Consider an infinite run of a pushdown automaton (PDA) P on some input word w^ω where w is of length n. Each position of the run is characterised by an *indexed mode*, which is a tuple consisting of the current state, the top symbol of the stack, and the current position in w (an integer in $[0, n-1]$). Since the set of states Q, the stack alphabet Γ, and the input positions $[0, n]$ are all finite, the total number of distinct indexed modes is bounded by $|Q| \cdot |\Gamma| \cdot n$.

Since an infinite run must contain infinitely many steps and there are only finitely many indexed modes, by the pigeonhole principle, some indexed mode must appear infinitely often as a step. That is, there exists a mode $m = (q, \gamma, i)$, where $q \in Q$, $\gamma \in \Gamma$, and $i \in \{0, 1, \ldots, n\}$, such that m occurs at infinitely many points in the run as a step.

Now consider a k-explorable pushdown automaton. Determiner's strategy witnessing this induces a tuple of k runs in parallel on a given input word. As in the previous argument, in each individual run, some indexed mode must appear infinitely often as a step. Since there are at most $|Q| \cdot |\Gamma| \cdot n$ possible indexed modes for a single run, the number of possible combinations of indexed modes across all k runs is at most $(|Q| \cdot |\Gamma| \cdot n)^k$.

This gives an upper bound on the number of distinct indexed mode combinations that can occur as a step infinitely many times in a k-run of a k-explorable PDA.

Now, observe that there are 2^n distinct binary words of length n. Suppose the number of distinct modes is strictly less than $\frac{\sqrt[k]{2^n}}{n}$. Then, by the pigeonhole principle, there must exist at least two distinct words, say w_1 and w_2, such that the infinite k-run (k-tuple of

infinite runs) $\overline{\rho}_1$ and $\overline{\rho}_2$, induced by the k-explorable PDA on inputs w_1^ω and w_2^ω, respectively, contain infinitely many steps at the same index modulo n, and with identical step modes. Let these step modes be $m_1, m_2, \ldots, m_k$, one for each of the k runs for both w_1 and w_2, occurring at positions $s_1, s_2, \ldots, s_k$ in $\overline{\rho}_1$, and at corresponding positions $s'_1, s'_2, \ldots, s'_k$ in $\overline{\rho}_2$, such that for each $\alpha \in [k]$, we have

$$s_\alpha \bmod n = s'_\alpha \bmod n \quad \& \quad |s_\alpha| > n \quad \& \quad |s'_\alpha| > n$$

Since w_1 and w_2 are distinct words, they differ at some position, say j. WLOG, assume that $w_1[j] = 1$ while $w_2[j] = 0$. For w_1 at every instance where the input length modulo the word length (n) is $j - 1$, one of runs in $\overline{\rho}_1$ must reach an accepting state (since the bit at position j in w_1 is 1). However, in the run ρ_2, at every such instance (i.e., when the position modulo n is $j - 1$), no accepting state can be reached in $\overline{\rho}_2$, since the corresponding bit in w_2 is 0.

Let $s = max(s_1, \ldots, s_k)$, and consider a position p where the modulo n is $j - 1$ and $p > s$. At position p, ρ_1 should have an accepting state in one of the explorable runs as explained before. Now, the contradiction is as follows: WLOG, assume that the accepting state is at the first run in $\overline{\rho}_1$. This means that the suffix of first run $\overline{\rho}_2$ starting from position $s'_1 + 1$ can be replaced with the suffix of first run in $\overline{\rho}_1$ starting from position $s_1 + 1$ and this will yield a valid run (by remark 4.1.1) and accepting run (see figure 1). After replacement, the resultant run is on a word which is some concatenation of substrings of w_2^* and w_1^*; however since s_1 and s'_1 share the same index, the index of the final state also remains the same. As $s'_1 > n$ the initial occurrences of $w_2[j]$ will lie at some $(0 \bmod n)$th bit from the end; therefore, the resultant word is not in $L_{mod\ n}$. However, after the replacement, P accept the word, leading to a contradiction. ◄

▶ **Theorem 22.** Expl-CFL *can be double-exponentially more succinct than* Expl_k-CFL.

Proof. As in the Lemma 18, an explorable PDA (more specifically an n-explorable PDA) can recognise $L_{mod\ n}$ using $\log(n)$ space. In contrast, we showed that for each fixed k, any k-explorable PDA recognizing $L_{mod\ n}$ must use at least $\frac{\sqrt[k]{2^n}}{n}$ size. This establishes a double exponential gap in succinctness between fully explorable PDAs and their k-explorable counterparts. ◄

6.3   Deterministic vs. Explorable PDA

Theorem 5.1 in the work of Guha et al. [3] establishes that HD-PDAs can be exponentially more succinct than DPDAs. Since HD-PDAs correspond precisely to 1-explorable PDAs, it follows that the same exponential succinctness gap holds between DPDAs and explorable PDAs. Hartmanis [6] proved that the relative succinctness of representing deterministic context-free languages (DCFLs) using deterministic versus nondeterministic pushdown automata (PDAs) is not bounded by any recursive function. In this work, we adapt and extend his proof technique to establish a similar result for deterministic and explorable PDAs – showing that the succinctness gap between them is likewise non-recursive.

Let M be a Turing machine, for an input x, let $\text{ID}_0(x)$ denote the initial configuration of M on input x, and let $\text{ID}_1(x), \text{ID}_2(x), \ldots$ denote the successive configurations during the computation.

If $x = a_1 a_2 \ldots a_n$, we define the reversal of x as: $x^r = a_n a_{n-1} \ldots a_1$.
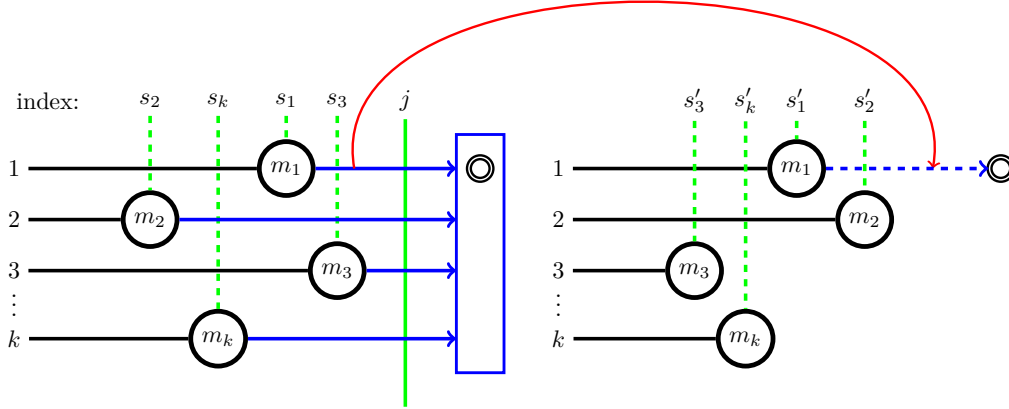
Figure 1 The figure illustrates two k-runs of a k-explorable PDA P. On the left, we depict the k-run $\bar{\rho}_1$, where at a position p an accepting state is reached along the run. On the right, we show $\bar{\rho}_2$, in which an identical modes appears at the same index modulo n, allowing a suffix of ρ_1 (beginning from the first run's step following the matching mode) to be replayed on ρ_2.

We now define $\text{VALC}[M]$ as the set of valid computations of M, represented with alternating reversals of configurations, as follows:

$$\{\text{ID}_0(x)\#[\text{ID}_1(x)]^r\#\text{ID}_2(x)\#\cdots\#\text{ID}_{2k}(x) \mid \text{ID}_{2k}(x) \text{ is a halting configuration of } M\}.$$

Each valid computation alternates between standard and reversed configurations, ending in a halting configuration. We also assume that $2k > 3$, that is TM takes at least 3 steps before halting.

We also define the set of invalid computations as: $\text{INVALC}[M] = \Gamma^* \setminus \text{VALC}[M]$,

INVALC is Explorable

More specifically, in this section we show that INVALC is 2-explorable.

Given a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ and an input word $w \in \Sigma^*$, a string α belongs to $\text{VALC}(M)$ if and only if the following conditions hold:

1. α is of the form $\alpha = \#C_0\#C_1\#C_2\#\cdots\#C_N\#$ where each C_i is a configuration encoded as a string over $(\Gamma \cup Q)^*$.
2. Each configuration C_i contains exactly one symbol from Q, indicating that the machine is in exactly one state at each computation step.
3. C_0 encodes the initial configuration of M on input w: the tape contains w followed by blanks, the head is at the leftmost position, and the machine is in state q_0.
4. C_N is a halting configuration – i.e., the state in C_N is either q_{accept} or q_{reject}.
5. For every i such that $0 \leq i < N$, the configuration C_{i+1} is a valid successor of C_i under M's transition function δ. And alternate between standard and reversed representations (i.e., C_i is reversed if i is odd).

We observe that among the five conditions defining membership in $\text{VALC}[M]$, the first four are *regular*. Therefore, when checking for $\text{INVALC}[M]$ only checking (5) – ensuring correct transitions between adjacent configurations – does not hold requires deeper computational power.

Verifying Condition 5 with 2-Explorability

We claim that a 2-explorable PDA suffices to verify that condition (5) does not hold. To do so, we define a 2-explorable machine consisting of a nondeterministic choice between two sub-automata, each of which checks the validity of half the pairs of configurations:

- One sub-automaton processes the sequence in blocks of the form:

$$\#\underline{C_0\#C_1}\#\underline{C_2\#C_3}\#\cdots\#C_N\#$$

  verifying that each reversed configuration C_{2i+1} is a valid successor of the standard configuration C_{2i}.
- The other sub-automaton processes the alternating pattern:

$$\#C_0\#\overline{C_1\#C_2}\#\overline{C_3\#C_4}\#\cdots\#C_N\#$$

  verifying that each standard configuration C_{2i+2} is a valid successor of the reversed configuration C_{2i+1}.

Each sub-automaton independently verifies consistency across transitions. Note that checking whether a configuration C' follows validly from C (according to M's transition function) can be done by a DPDA as follows:

- Push the entire configuration C onto the stack while reading.
- Then, read the next configuration C' and pop from the stack, comparing symbols and using the transition function δ to ensure that the head movement, state change, and tape content evolve correctly.
- Any mismatch leads to acceptance

Thus, the only nondeterminism is in the initial choice between the two sub-automata, after which each is deterministic. This means that only two runs suffice for Determiner in the 2-exploration game, and hence the machine 2-explorable.

Since the first four conditions are regular and the fifth is 2-explorable, we conclude that the set of invalid computations $\mathsf{INVALC}[M]$ is recognizable by a 2-explorable PDA.

▶ **Lemma 23** ([6, Lemma 1]). *INVALC[M] is a deterministic context-free language (DCFL) if and only if $L(M)$ is finite.*

Proof. If $L(M)$ is finite, then the set of valid accepting computations VALC[M] is finite and so its complement INVALC[M] is co-finite. Since the class of regular languages is closed under complementation, and all finite or co-finite languages are regular, it follows that INVALC[M] is regular and hence a DCFL.

Conversely, if $L(M)$ is infinite, then VALC[M] is not context-free (since checking the validity of more than two consecutive Turing Machine configurations is not context-free and our machines take at least three steps) and since DCFLs are closed under complement, its complement INVALC[M] cannot be a DCFL. ◀

▶ **Definition 24.** *We say that the relative succinctness of representing deterministic context-free languages (DCFLs) by 2-explorable and deterministic pushdown automata is* not *recursively bounded if there is no recursive function F such that for every 2-explorable-PDA A accepting a DCFL, there exists an equivalent deterministic PDA D such that $|D| \leq F(|A|)$, where $|A|$ and $|D|$ denote the descriptional sizes of A and D, respectively.*

▶ **Theorem 25.** *The relative succinctness of representing deterministic context-free languages by explorable and deterministic PDAs is not recursively bounded.*

Proof. Suppose, for contradiction, that there exists a recursive function F such that for every 2-explorable-PDA A accepting a DCFL, there exists a deterministic PDA D equivalent to A with $|D| \le F(|A|)$.

Then given a TM M, let A be the 2-explorable PDA that recognizes INVALC$[M]$. We can compute $F(|A|)$ and enumerate all deterministic PDAs $D_1, D_2, \ldots, D_s$ of sizes up to $F(|A|)$. Then $L(A)$ is not deterministic iff none of these D_j is equivalent to A, which is semi-decidable, by comparing memberships on all strings in Σ^*. If none of them is equivalent to A, we conclude that $L(A)$ is not a DCFL. Therefore, by Lemma 23 we can enumerate the set $\{M \mid L(M)$ is infinite $\}$ which would contradict Rice's theorem [12]. Hence, R is not recursively enumerable. Therefore, such a function F cannot exist. ◄

7 Conclusion

In this work, we introduced and studied the notion of *explorability* for PDAs, refining the landscape between deterministic and nondeterministic computation. Building on the concept of history-determinism, we defined the class of k-explorable PDAs and established a strict expressiveness hierarchy based on the number of concurrent explorations permitted.

On the succinctness front, we established that explorable PDAs can be doubly exponentially more succinct than PDAs with fixed constant bound on explorability. Furthermore, the succinctness gap between DPDAs and 2-explorable PDAs is not recursively enumerable, extending the classic non-RE gap between deterministic and nondeterministic PDAs.

Despite these advances, several intriguing questions remain open. The introduction of parameterized explorability gives rise to a rich landscape of intermediate classes – such as $\log n$-explorable, *poly*-explorable or $\sqrt{n}$-explorable PDAs – whose expressiveness and succinctness properties are yet to be systematically studied. Furthermore, while we have established tight bounds in several cases, the exact succinctness gap between general explorable PDAs and fully nondeterministic PDAs remains unresolved. Addressing these questions may not only deepen our understanding of the complexity-theoretic trade-offs between determinism and nondeterminism in the context of PDAs, but could also shed new light on longstanding questions regarding the structure of context-free languages. While we do not have any hope that explorability of PDA is decidable because HD is undecidable, it seems likely that in the context of visibly pushdown automata, explorability might have some better algorithmic properties, like recognizability, lower complexity universality, Gale-Stewart games and inclusion.

References

1 Zuzana Bednárová, Viliam Geffert, Carlo Mereghetti, and Beatrice Palano. Removing nondeterminism in constant height pushdown automata. *Descriptional Complexity of Formal Systems - 14th International Workshop, DCFS 2012, Braga, Portugal, July 23-25, 2012. Proceedings*, 7386:76–88, 2012. `doi:10.1007/978-3-642-31623-4_6`.

2 Jonathan Goldstine, Hing Leung, and Detlef Wotschke. Measuring nondeterminism in pushdown automata. *STACS 97, 14th Annual Symposium on Theoretical Aspects of Computer Science, Lübeck, Germany, February 27 - March 1, 1997, Proceedings*, 1200:295–306, 1997. `doi:10.1007/BFB0023467`.

3 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. *Log. Methods Comput. Sci.*, 20(1), 2024. `doi:10.46298/LMCS-20(1:3)2024`.

4 Yo-Sub Han, Sang-Ki Ko, and Kai Salomaa. Limited nondeterminism of input-driven pushdown automata: Decidability and complexity. In Michal Hospodár, Galina Jirásková, and Stavros

Konstantinidis, editors, *Descriptional Complexity of Formal Systems - 21st IFIP WG 1.02 International Conference, DCFS 2019, Košice, Slovakia, July 17-19, 2019, Proceedings*, volume 11612 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 2019. `doi:10.1007/978-3-030-23247-4_12`.

5   Michael. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1978.

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

7   Emile Hazard, Olivier Idir, and Denis Kuperberg. Explorable parity automata. *CoRR*, abs/2410.23187, 2024. `doi:10.48550/arXiv.2410.23187`.

8   Emile Hazard and Denis Kuperberg. Explorable automata. *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, 252:24:1–24:18, 2023. `doi:10.4230/LIPICS.CSL.2023.24`.

9   Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2006. `doi:10.1007/11874683_26`.

10 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`.

11 John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation, 2nd edition. *ACM SIGACT News*, 32(1):60–65, March 2001. `doi:10.1145/568438.568455`.

12 H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 74:358–366, 1953.

13 Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of formal languages.* Springer, Berlin, Germany, November 2012.

A   Turing Machine

Let M be a Turing machine. Then M is a 7-tuple:$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

- Q is a finite set of states,
- Σ is the input alphabet, not containing the blank symbol $\sqcup$,
- Γ is the tape alphabet, where $\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma$,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_{\text{accept}} \in Q$ is the accept state,
- $q_{\text{reject}} \in Q$ is the reject state, with $q_{\text{reject}} \neq q_{\text{accept}}$.

B   Pumping Proofs

Ogden's Lemma

Ogden's Lemma is a generalization of the *pumping lemma* for context-free languages. It is particularly useful in proving that certain languages are not context-free.

▶ **Lemma 26** (Ogden's Lemma [5])**.** *Let L be a context-free language. Then there exists a constant $k \geq 1$ such that for every string $w \in L$ with at least k marked positions, there exists a decomposition $w = uvxyz$ such that:*

1. *vxy contains at most k marked positions,*
2. *v and y together contain at least one marked position, and*
3. *for all $i \geq 0$, the string $uv^i xy^i z \in L$.*

▶ **Lemma 10.** *Given $S = (S_n)_{n \in \mathbb{N}}$ where each $S_n \subseteq \mathbb{Z}^+ \times \mathbb{Z}^+$ is finite and non-empty, the language L_S is not context-free.*

Proof. Assume, towards a contradiction, that L'_S is context-free. Then, by Ogden's Lemma as defined in Section 2, let us choose $w = a^n b^{in} c^{jn}$, where $n > p$, $i, j > 0$, and $(i, j) \in S_n$ are fixed. We mark all n positions in the a-block (i.e., all occurrences of a).

We now consider the possible forms of the substrings v and y. Since only the a-block contains marked positions, at least one of v or y must be a substring of the a-block. Also, in order to preserve the structure of strings in L_S, the decomposition $uvxyz$ must ensure that pumping does not mix symbols from different blocks – otherwise, $uv^i xy^i z$ would no longer be of the form $a^* b^* c^*$, violating the block structure required by L_S.

Thus, v must be entirely within the a-block, and y must lie within a single block – either a, b, or c. In any case, one of the b- or c-blocks will be left untouched by the pumping.

Without loss of generality, suppose y lies in the b-block. Then pumping increases the number of a's (via v) and the number of b's (via y), but leaves the number of c's unchanged. Let us pump $k = jn + 1$ times. Since we pumped a non-empty portion of the a-block (required by the Ogdens lemma), the number of a's becomes greater than $jn + 1$, while the number of c's remains exactly jn. This means the resulting string cannot belong to any $L_{i',j'}$ with $i', j' \in \mathbb{N}$, because the ratio of c's to a's is no longer an integer.

Thus, $uv^{jn+1} xy^{jn+1} z \notin L_S$, contradicting Ogden's Lemma. Therefore, L_S is not context-free. ◀

▶ **Lemma 14.** *Given $S' = (S'_w)_{w \in (a^* \#)^{k+1}}$ in which some S'_w are non-empty, then the language $L_{block-S'}$ is not context-free.*

Proof. Assume for contradiction that $L_{\text{block-}S'}$ is context-free. We apply *Ogden's Lemma*.

Let p be the constant from Ogden's Lemma. Define the string:

$$s = a^p \# a^{p+1} \# a^{p+2} \# \cdots \# a^{p+k}.$$

That is, s consists of $k + 1$ distinct a-blocks, each separated by $\#$, and each longer than the previous one. Choose two block lengths i, j occurring in s such that $j > i$, so that $(i, j) \in S'_s$.

Now consider the string:

$$w = s \# b^i c^{j-i} \in L_{\text{block-}S'}.$$

We mark all positions corresponding to the b's in w. There are $i \geq p$ such positions, so Ogden's Lemma applies: there exists a decomposition $w = uvxyz$ such that:

1. vxy contains at most k marked positions,
2. v and y together contain at least one marked position, and
3. for all $i \geq 0$, the string $uv^m xy^m z \in L_{block-S'}$.

Ovserve that if any of v, y cannot contain seperators ($\#$) or 2 distinct letters as it will break the regular strucute of the language. We now consider possible pumping effects:

Case 1: Pumping occurs entirely within the $b^i c^{j-i}$ suffix. If we pump only b's or both b's and c's, we can increase the number of b's and/or c's. However, the prefix s remains unchanged. Since the set S'_s depends only on s, and these are fixed, there is a maximum length $j_{\max}$ of any a-block in s. If $m > p + k$, the pumped string has more b's than any a-block in s, meaning there is no pair $(i', j') \in S'_s$ satisfying the new b/c lengths, contradicting $w_n \in L_{\text{block-}S'}$.

Case 2: Pumping pair v, y lies in an a-block and the b-block.   Suppose the pumped region includes some a-block in the prefix s and part of the b-block. Then, pumping increases both the length of that a-block and the number of b's.

We can take $m > p + k$ so that the number of b's becomes larger than all a-blocks in s except possibly the one being pumped. Observe that only one a-block is large, so there is no a block to match the sum of the b and c blocks. This means that the new b-block length must matches the (pumped) a-block's length. In such a situation, the new pair (i', j') extracted from the pumped string, such that i' is the length of a b-block, but as $j' > i'$ but there is no a-block whose length is more than i', the pumped string does not belong to $L_{\text{block-}S'}$. This contradicts Ogden's Lemma.

In all cases, pumping leads to a string not in the language. Thus, our assumption that $L_{\text{block-}S'}$ is context-free must be false.   ◀