Theoretical Computer Science 996 (2024) 114503

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

On the power of pushing or stationary moves for input-driven pushdown automata ☆,☆☆

Martin Kutrib *, Andreas Malcher, Matthias Wendlandt

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany

ARTICLE INFO

Keywords:
Input-driven pushdown automata
Deterministic pushdown automata
Representation by inverse homomorphism
Computational capacity
Decidability questions

ABSTRACT

Input-driven pushdown automata (IDPDAs) are pushdown automata where the next action on the pushdown store (push, pop, nothing) is solely governed by the input symbol. Nowadays such devices are usually defined such that every push operation pushes exactly one additional symbol on the pushdown store and, in addition, stationary moves are not allowed so that the devices work in real time. Here, we relax this strong definition and consider IDPDAs that may push more than one symbol in one step (push-IDPDA) or may perform stationary moves (stat-IDPDA). We study the computational power of the extended variants both in the deterministic and nondeterministic case, we investigate several decidability questions for the new automata classes, and we obtain interesting representations by inverse homomorphisms. Namely, every (1) deterministic, (2) real-time deterministic, and (3) nondeterministic context-free language can be characterized as the inverse homomorphic image of a language accepted by a (1) stat-IDPDA, (2) push-IDPDA, and (3) nondeterministic push-IDPDA.

1. Introduction

Input-driven pushdown automata (IDPDAs) have been introduced in [13] and their motivation stems from the search for an upper bound for the space needed for the recognition of deterministic context-free languages. IDPDAs are a subclass of pushdown automata that work in real time and, more importantly, in an input-driven way. That is, no moves on empty input are allowed, and the actions on the pushdown store are dictated by the input symbols. To this end, the input alphabet is partitioned into three subsets, where one subset contains symbols on which the automaton pushes a symbol onto the pushdown store, one subset contains symbols on which the automaton pops a symbol, and one subset contains symbols on which the automaton leaves the pushdown unchanged and makes a state change only. The basic results in [13] and its follow-up papers [3,7] are the equivalence of nondeterministic and deterministic models and the proof that the membership problem is solvable in logarithmic space.

Input-driven pushdown automata have been revisited in [1,2], where such devices are called visibly pushdown automata or nested word automata. Some of the results comprise descriptional complexity aspects for the determinization as well as closure properties and decidability questions which turned out to be similar to those of finite automata. Further aspects such as the minimization of

☆ This article belongs to Section A: Algorithms, automata, complexity and games, Edited by Paul Spirakis.

☆☆ A preliminary version of this work was presented at the *26th International Conference on Implementation and Application of Automata (CIAA 2022), Rouen, France, June 28–July 1, 2022, and it is published in [11].*

* Corresponding author.
E-mail addresses: kutrib@informatik.uni-giessen.de (M. Kutrib), andreas.malcher@informatik.uni-giessen.de (A. Malcher), matthias.wendlandt@informatik.uni-giessen.de (M. Wendlandt).

https://doi.org/10.1016/j.tcs.2024.114503
Received 24 November 2022; Received in revised form 1 March 2024; Accepted 11 March 2024

Available online 15 March 2024

IDPDAs and a comparison with other subfamilies of deterministic context-free languages have been studied in [5,6]. A recent survey with many valuable references on complexity aspects of input-driven pushdown automata may be found in [14]. An extension of input-driven pushdown automata towards pushdown automata synchronized by a finite transducer has been discussed in [4].

The renewed interest in input-driven pushdown automata has also affected the definition of the devices. While in the early papers IDPDAs are defined as ordinary real-time pushdown automata whose behavior on the pushdown is solely driven by the input symbols, the definition in [1] stipulates that at most one symbol is pushed in a single push operation. Moreover, instead of getting stuck when popping from the empty pushdown it is allowed to pop from the empty pushdown by simply defining that popping from the empty pushdown results in an empty pushdown. The latter restriction has been given up in [9] where so-called *digging input-driven pushdown automata* have been introduced and studied in depth. Such automata are basically IDPDAs that, when forced to pop from the empty pushdown, dig a hole of the shape of the popped symbol in the bottom of the pushdown. Popping further symbols from a pushdown having a hole at the bottom deepens the current hole furthermore. The hole can only be filled up by pushing symbols previously popped. It turned out that digging IDPDAs have similar properties as IDPDAs with regard to determinization, closure properties, and decidability questions.

In this paper, our goal is to consider variants of input-driven pushdown automata whose definition is closer to that of classical deterministic pushdown automata. To this end, we will further relax the restrictions of allowing only at most one symbol to be pushed in a single step and the processing of the input in real time. In detail, we consider *pushing input-driven pushdown automata* (push-IDPDAs) that are allowed to push more than one symbol in a single step. Moreover, to relax the real-time condition we consider *stationary input-driven pushdown automata* (stat-IDPDAs) where the input symbols still dictate which operation on the pushdown has to be carried out, but the transition functions determine whether the head remains on the current input symbol or moves to the right as usual. As a first result, which is in contrast to classical IDPDAs and digging IDPDAs, we obtain the result that nondeterministic push-IDPDAs and stat-IDPDAs are computationally stronger than their deterministic counterparts. In addition, it can be shown in the deterministic as well as nondeterministic case that stat-IDPDAs are computationally stronger than push-IDPDAs, which in turn are computationally stronger than classical IDPDAs. On the other hand, classical deterministic and nondeterministic pushdown automata are computationally stronger than the corresponding stat-IDPDAs. Another interesting result is that the introduced variants are well-suited for a representation by inverse homomorphism. In detail, it is shown that there is a homomorphism h_r such that every context-free language L can be represented as the inverse homomorphic image $h_r^{-1}(L') = L$ of a language L' accepted by a nondeterministic push-IDPDA. Using the same homomorphism it is also possible to obtain similar characterizations of the real-time deterministic context-free languages by deterministic push-IDPDAs and of general deterministic context-free languages by deterministic stat-IDPDAs. These characterizations can in particular be used to obtain undecidability results. It turns out that the inclusion problem for two deterministic push-IDPDAs with compatible signatures is not even semidecidable, whereas the problem is decidable for classical IDPDAs. Hence, the restriction to allow at most one symbol to be pushed is essential in order to get decidability. In the nondeterministic case, we can show that the questions of universality, equivalence, and regularity are not semidecidable for push-IDPDAs and stat-IDPDAs.

The paper is organized as follows. In Section 2, we compile the necessary definitions and give an example how a non-real-time deterministic context-free language is accepted by a stat-IDPDA. The homomorphic characterization of the (real-time) deterministic and general context-free languages is provided in Section 3. In Section 4, the computational power of the newly introduced automata classes is investigated and a complete picture concerning the relation to the classical automata classes can be given. Finally, we discuss in Section 5 decidability questions of the newly introduced automata classes.

2. Definitions and preliminaries

We denote the set of non-negative integers $\{0, 1, 2, \dots\}$ by $\mathbb{N}$. Let Σ^* denote the set of all words over the finite alphabet Σ. The *empty word* is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The set of words of length at most $n \geq 0$ is denoted by $\Sigma^{\leq n}$. The *reversal* of a word w is denoted by w^R. For the *length* of w we write $|w|$. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*. We write $|S|$ for the cardinality of a set S. We say that two language families $\mathscr{L}_1$ and $\mathscr{L}_2$ are *incomparable* if $\mathscr{L}_1$ is not a subset of $\mathscr{L}_2$ and vice versa.

A classical pushdown automaton is called input-driven if the next input symbol defines the next action on the pushdown store, that is, pushing a symbol onto the pushdown store, popping a symbol from the pushdown store, or changing the state without modifying the pushdown store. To this end, the input alphabet Σ is partitioned into the sets Σ_D, Σ_R, and Σ_N, that control the actions push (D), pop (R), and state change only (N). These notions are inspired by the automatic gearbox of a car where D means "drive", R means "reverse", and N means "neutral". We note that if the next input symbol forces the input-driven pushdown automaton to pop a symbol from the empty pushdown, then the computation does not get stuck but continues with an empty pushdown. Moreover, an input-driven pushdown automaton always works in real-time, that is, it is forced to move its input head in each step. Finally, an input-driven pushdown automaton must not push more than one symbol in a single step. In the following, we use the notation `push`(x) to indicate that the operation on the pushdown store replaces the topmost symbol by x, such that the pushdown height increases, and the notation `top`(x) to indicate that the operation on the pushdown store exchanges the topmost symbol by x, such that the pushdown height remains the same.

A *nondeterministic input-driven pushdown automaton (NIDPDA)* is a system $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$, where

1. Q is the finite set of *internal states*,
2. Σ is the finite set of *input symbols* partitioned into the sets Σ_D, Σ_R, and Σ_N,
3. Γ is the finite set of *pushdown symbols*,

4. $q_0 \in Q$ is the *initial state*,
5. $F \subseteq Q$ is the set of *accepting states*,
6. $\bot \notin \Gamma$ is the *empty-pushdown symbol*,
7. δ_D is the partial transition function mapping $Q \times \Sigma_D \times \Gamma$ to the subsets of $Q \times \{ \mathtt{push}(x) \mid x \in \Gamma^2 \}$, and $Q \times \Sigma_D \times \{\bot\}$ to the subsets of $Q \times \{ \mathtt{push}(x) \mid x \in \Gamma \}$,
8. δ_R is the partial transition function mapping $Q \times \Sigma_R \times (\Gamma \cup \{\bot\})$ to the subsets of Q,
9. δ_N is the partial transition function mapping $Q \times \Sigma_N \times (\Gamma \cup \{\bot\})$ to the subsets of $Q \times \{ \mathtt{top}(x) \mid x \in \Gamma \cup \{\bot\} \}$, where $(q', \mathtt{top}(\bot)) \in \delta_N(q, a, z)$ if and only if $z = \bot$.

A *configuration* of an NIDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ is a triple (q, w, s), where $q \in Q$ is the current state, $w \in \Sigma^*$ is the unread part of the input, and $s \in \Gamma^*$ denotes the current pushdown content, where the leftmost symbol is at the top of the pushdown store. The *initial configuration* for an input string w is set to (q_0, w, λ). During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $q, q' \in Q$, $a \in \Sigma$, $w \in \Sigma^*$, $z, z', z'' \in \Gamma$, and $s \in \Gamma^*$. We set

1. $(q, aw, zs) \vdash (q', w, z'z''s)$, if $a \in \Sigma_D$ and $(q', \mathtt{push}(z'z'')) \in \delta_D(q, a, z)$,
2. $(q, aw, \lambda) \vdash (q', w, z')$, if $a \in \Sigma_D$ and $(q', \mathtt{push}(z')) \in \delta_D(q, a, \bot)$,
3. $(q, aw, zs) \vdash (q', w, s)$, if $a \in \Sigma_R$ and $q' \in \delta_R(q, a, z)$,
4. $(q, aw, \lambda) \vdash (q', w, \lambda)$, if $a \in \Sigma_R$ and $q' \in \delta_R(q, a, \bot)$,
5. $(q, aw, zs) \vdash (q', w, z's)$, if $a \in \Sigma_N$ and $(q', \mathtt{top}(z')) \in \delta_N(q, a, z)$,
6. $(q, aw, \lambda) \vdash (q', w, \lambda)$, if $a \in \Sigma_N$ and $(q', \mathtt{top}(\bot)) \in \delta_N(q, a, \bot)$.

So, whenever the pushdown store is empty, the successor configuration is computed by the transition functions with the special empty-pushdown symbol $\bot$. As usual, we define the reflexive and transitive closure of $\vdash$ by $\vdash^*$.

Next, we turn to the variants of NIDPDAs that are closer to classical deterministic pushdown automata having the property that the next input symbol defines the next action on the pushdown store. First, we consider the variant that may push more than one symbol in a single step.

An input-driven pushdown automaton is a *pushing input-driven pushdown automaton* (push-NIDPDA) if its transition function δ_D maps $Q \times \Sigma_D \times \Gamma$ to the finite subsets of $Q \times \{ \mathtt{push}(x) \mid x \in \Gamma^+\Gamma \}$, and $Q \times \Sigma_D \times \{\bot\}$ to the finite subsets of $Q \times \{ \mathtt{push}(x) \mid x \in \Gamma^+ \}$.

Second, we consider the variant that is not forced to work in real time. To this end, it must be possible to keep the input head at its current position. Since λ-steps are somehow in conflict with the idea that the input symbol defines the action on the pushdown store, we consider stationary moves. So, an input-driven pushdown automaton is a *stationary input-driven pushdown automaton* (stat-NIDPDA) if its transition functions δ_D, δ_R, and δ_N result in additional components from $\{0, 1\}$. These components determine if the head moves to the right (1) as usual, or if it resides stationary on the current input symbol (0). In the second case, the symbol is read again in the next step. This implies that the action on the pushdown store dictated by this symbol is applied again. The relation $\vdash$ is straightforwardly extended to cover these cases.

Finally, some variant of input-driven pushdown automata is said to be deterministic (IDPDA, push-IDPDA, respectively stat-IDPDA) if $|\delta_x(q, a, z)| \leq 1$, for $x \in \{D, N, R\}$ and all $q \in Q$, $a \in \Sigma_x$, and $z \in \Gamma \cup \{\bot\}$.

The language accepted by some variant of an input-driven pushdown automaton M is the set $L(M)$ of words for which there exists some computation beginning in the initial configuration and ending in a configuration in which the whole input is read and an accepting state is entered. Formally:

$$L(M) = \{ w \in \Sigma^* \mid (q_0, w, \lambda) \vdash^* (q, \lambda, s) \text{ with } q \in F, s \in \Gamma^* \}.$$

In general, the family of all languages accepted by automata of some type X will be denoted by $\mathscr{L}(X)$.

Some properties of language families implied by classes of input-driven pushdown automata may depend on whether all automata involved share the same partition of the input alphabet. For easier writing, we call the partition of an input alphabet a *signature*, and say that two signatures $\Sigma = \Sigma_D \cup \Sigma_R \cup \Sigma_N$ and $\Sigma' = \Sigma'_D \cup \Sigma'_R \cup \Sigma'_N$ are *compatible* if and only if

$$\bigcup_{j \in \{D,R,N\}} (\Sigma_j \setminus \Sigma'_j) \cap \Sigma' = \emptyset \quad \text{and} \quad \bigcup_{j \in \{D,R,N\}} (\Sigma'_j \setminus \Sigma_j) \cap \Sigma = \emptyset.$$

So, for two compatible signatures $\Sigma = \Sigma_D \cup \Sigma_R \cup \Sigma_N$ and $\Sigma' = \Sigma'_D \cup \Sigma'_R \cup \Sigma'_N$ the symbols shared by both alphabets have the same effect on the pushdown stores.

Example 1. The language $L = \{ a^n b^l c^m d^n \mid l, m, n \geq 0 \text{ and } l \geq m \}$ is accepted by the stat-IDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ with the state set $Q = \{q_0, q_a, q_b, r_b, q_c, r_c, q_d, q_s, q_+, q_-\}$, $\Sigma_D = \{a, b\}$, $\Sigma_R = \{c, d\}$, $\Sigma_N = \emptyset$, the set of pushdown symbols $\Gamma = \{A, A', B\}$, the set of accepting states $F = \{q_0, q_+, r_b, r_c\}$, and the transition functions specified as:

(1) $\delta_D(q_0, a, \bot) = (q_a, \mathtt{push}(A'), 1)$
(2) $\delta_D(q_0, b, \bot) = (r_b, \mathtt{push}(B), 1)$
(3) $\delta_R(q_0, c, \bot) = (q_-, 1)$
(4) $\delta_R(q_0, d, \bot) = (q_-, 1)$
(5) $\delta_D(q_a, a, A') = (q_a, \mathtt{push}(AA'), 1)$
(6) $\delta_D(q_a, a, A) = (q_a, \mathtt{push}(AA), 1)$
(7) $\delta_D(q_a, b, A') = (q_b, \mathtt{push}(BA'), 1)$
(8) $\delta_D(q_a, b, A) = (q_b, \mathtt{push}(BA), 1)$
(9) $\delta_R(q_a, d, A') = (q_+, 1)$
(10) $\delta_R(q_a, d, A) = (q_d, 1)$
(11) $\delta_D(q_b, b, B) = (q_b, \mathtt{push}(BB), 1)$
(12) $\delta_R(q_b, c, B) = (q_c, 1)$
(13) $\delta_R(q_b, d, B) = (q_s, 0)$
(14) $\delta_R(q_c, c, B) = (q_c, 1)$
(15) $\delta_R(q_c, d, B) = (q_s, 0)$
(16) $\delta_R(q_c, d, A') = (q_+, 1)$
(17) $\delta_R(q_c, d, A) = (q_d, 1)$
(18) $\delta_R(q_d, d, A) = (q_d, 1)$
(19) $\delta_R(q_d, d, A') = (q_+, 1)$
(20) $\delta_R(q_s, d, B) = (q_s, 0)$
(21) $\delta_R(q_s, d, A') = (q_+, 1)$
(22) $\delta_R(q_s, d, A) = (q_d, 1)$
(23) $\delta_D(r_b, b, B) = (r_b, \mathtt{push}(BB), 1)$
(24) $\delta_R(r_b, c, B) = (r_c, 1)$
(25) $\delta_R(r_c, c, B) = (r_c, 1)$

At the beginning of the computation three cases can be distinguished. First, if the input is empty, M accepts in state q_0. The second case is that there is some a in the input, that is, $n \geq 1$ (Transition (1)). In the third case, the first input symbol is a b, that is, $n = 0$ (Transition (2)). If the first input symbol is c, the condition $l \geq m$ is violated and the computation blocks rejecting in state q_- (Transition (3)). Similarly, if the first input symbol is d, the number of d's is not equal to the number of a's and the computation blocks rejecting in state q_- (Transition (4)).

In the third case the computation continues in states r_b and r_c by applying Transitions (23–25). Both states are accepting and the computation does not block as long as the number of c's does not exceed the number of b's. Should this happen, the computation blocks without having processed the input entirely and, thus, rejects.

Let us now consider the second case. The states that may be entered are q_a, q_b, q_c, and q_d meaning that M processes the a-block, b-block, c-block, or d-block. The state q_s is used to pop some symbols B from the top of the pushdown with stationary moves. The state q_+ is used to halt the computation in an accepting state. If the input has been processed entirely, this means accept, otherwise this means reject. In detail, in the first step, a primed A is stored on the pushdown store (Transition (1)). It indicates the bottom-most symbol on the pushdown store and is used in the final phase to ensure that the correct number of d's is read. Afterwards, for every a read, an A is pushed (Transitions (5–6)) and, similarly, for every following b read a B is pushed (Transitions (7–8,11)). Should there follow some c's after the b's, for each c one symbol B is popped (Transitions (12,14)). If there are more c's than B's on the pushdown, the computation blocks rejecting since the condition $l \geq m$ is violated. If the number of c's equals the number of B's, that is $l = m$, then the first d in the input sends M into the accepting and blocking state q_+ if the topmost pushdown symbol is A', that is $n = 1$ (Transitions (9,16)). Otherwise the check whether the number of d's equals n is started (Transition (10,17)). If the number of c's is less than the number of B's, that is $l > m$, then state q_s is entered to pop the remaining B's at the top of the pushdown with stationary moves (Transitions (13,15,20)). After having popped the last B, M starts checking whether the number of d's equals n (Transitions (21–22)). To this end, if A' appears on the pushdown then we have $n = 1$ and M is sent into the accepting and blocking state q_+ (Transition (21)). Otherwise, if an A appears on the pushdown then state q_d is entered, which is used to check whether the number of d's equals the number pushdown symbols left. ■

3. Homomorphic reduction

In [10] the family of context-free languages is characterized by the closure of $\mathscr{L}$(IDPDA) under λ-free homomorphisms. Such results open the possibility to characterize certain language families by, in some sense, simpler ones and some kind of operations. Besides they shed some light on the structure of the family itself, they may be used as a powerful reduction tool in order to simplify some proofs or constructions.

Here we turn to show that all context-free languages can be represented as preimages of languages from $\mathscr{L}$(push-NIDPDA) under a simple injective homomorphism. Similarly, the family of deterministic context-free languages can be represented as preimages of languages from $\mathscr{L}$(stat-IDPDA) and the family of deterministic real-time context-free languages can be represented as preimages of languages from $\mathscr{L}$(push-IDPDA). Let Σ be some alphabet and # be a fresh symbol. Then we define the homomorphism $h_r : \Sigma^* \to (\Sigma \cup \{\#\})^*$ as $h_r(a) = a\#^2$ for all $a \in \Sigma$. Clearly, h_r depends on Σ but it is always clear from the context what Σ is.

First, we show the representation of deterministic real-time context-free languages. The constructions already contain a large part of the ideas, but have later to be extended in order to involve λ-steps.

Theorem 2. *Let $L \subseteq \Sigma^*$ be accepted by some deterministic real-time pushdown automaton M. Then there exists a push-IDPDA M' such that $L = h_r^{-1}(L(M'))$.*

Proof. Let the language L be given by a deterministic real-time pushdown automaton $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta \rangle$. We may safely assume that M never removes or changes the bottom-of-pushdown symbol at the bottom of the pushdown, nor does it push the bottom-of-pushdown symbol anywhere else in the pushdown. The transition function δ maps $Q \times \Sigma \times (\Gamma \cup \{\bot\})$ to $Q \times (\Gamma \cup \{\bot\})^*$ obeying the restrictions concerning $\bot$.

We construct a push-IDPDA $M' = \langle Q, \Sigma \cup \{\#\}, \Gamma \cup \{d\}, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ that on inputs $h_r(w)$, for $w \in \Sigma^*$, simulates the computation of M on input w. Here, $d \notin \Gamma$ is a new dummy symbol. We set $\Sigma_D = \Sigma$, $\Sigma_R = \{\#\}$, and $\Sigma_N = \emptyset$.

First, for $p \in Q$ we define

(1) $\delta_R(p, \#, d) = p$.

So, after having processed some input symbol from Σ, the following two symbols # force M' to pop two dummy symbols d without changing its state. This implies the essential behavior of M' on symbols from Σ. If M' wishes to simulate a push or top operation of M, it pushes two more dummy symbols. If M' wishes to simulate a pop operation of M, it pushes just one additional symbol. More precisely, we construct the transition function δ_D by translating every transition of M.

For $p \in Q$, $a \in \Sigma$, and $z \in \Gamma$ we define δ_D as follows.

If $\delta(p, a, z) = (q, s)$ is a move of M, then we set

(2) $\delta_D(p, a, z) = (q, \mathtt{push}(dds))$,

and if $\delta(p, a, \bot) = (q, s'\bot)$, then

(3) $\delta_D(p, a, \bot) = (q, \mathtt{push}(dds'))$.

Note that for $s = \lambda$ the pop move of M is simulated by replacing the topmost pushdown symbol by dd and subsequently popping dd again on the input ##.

So, in this way, M' on input $a\#\#$ eventually performs the same transition as M on input $a \in \Sigma$. That is,

$$(p, aw, s) \vdash_M (q, w, s') \text{ if and only if } (p, a\#\#w, s) \vdash_{M'} (q, w, s').$$

However, so far, the construction works fine only if M' is fed with inputs of the form $(\Sigma\#\#)^*$. So, the final construction step is to extend M' such that it rejects any input not belonging to the regular language $(\Sigma\#\#)^*$. This check can straightforwardly be implemented in the finite control. So, we obtained $(q_0, w, \lambda) \vdash_M^* (q, \lambda, s)$ if and only if $(q_0, h_r(w), \lambda) \vdash_{M'}^* (q, \lambda, s)$. This together with $L(M') \subseteq (\Sigma\#\#)^*$ implies $L = h_r^{-1}(L(M'))$. □

Since in the nondeterministic case any context-free language is accepted by some nondeterministic pushdown automaton in real time, the proof of Theorem 2 shows almost literally the representation of context-free languages by using push-NIDPDAs.

Corollary 3. *Let $L \subseteq \Sigma^*$ be accepted by some nondeterministic pushdown automaton M. Then there exists a push-NIDPDA M' such that $L = h_r^{-1}(L(M'))$.*

Next, we turn to involve λ-transitions of the deterministic pushdown automaton. It may happen that the number of consecutive λ-transitions in some computation exceeds any fixed constant. So, we cannot provide enough # symbols to simulate them by a push-IDPDA. Instead, we are going to use stationary moves (stat-IDPDA) to simulate a possible sequence of λ-transitions on an input symbol #.

Additionally, we may assume that all λ-transitions of M are popping. It is well known that any deterministic pushdown automaton can be simulated by some other obeying this condition (see, for example, Exercise 10.2 in [8]). Intuitively, the idea of this simulation is as follows. Given an arbitrary deterministic pushdown automaton, we can precompute its behavior on consecutive λ-transitions. The number of consecutive λ-transitions in accepting computations that perform push or top operations is bounded by the number of states and pushdown symbols. Once a bound is exceeded, the automaton runs into an infinite loop on λ-transitions and cannot accept anymore. So, the precomputed behavior (up to a pop operation) can be simulated in one step. This immediately implies that any λ-transition is a pop move. However, in this way of simulation the number of symbols pushed in one step is increased. We can cope with this problem by standard techniques that compress the pushdown content, where multiple symbols are combined to a single meta-symbol.

However, even if we assume that all λ-transitions are pop moves, the construction is more involved since a possible sequence of λ-transitions must be simulated on some input symbol # and, thus, M' has to be able to predict that M would perform λ-transitions. To this end, in certain situations the topmost pushdown symbol unequal to d has to be stored into the states.

Theorem 4. *Let $L \subseteq \Sigma^*$ be accepted by some deterministic pushdown automaton M. Then there exists a stat-IDPDA M' such that $L = h_r^{-1}(L(M'))$.*

Proof. Let the language L be given by a deterministic pushdown automaton $M = \langle Q, \Sigma, \Gamma, p, F, \vartriangle, \delta\rangle$. As in the proof of Theorem 2 let M never remove or change the bottom-of-pushdown symbol at the bottom of the pushdown, nor push the bottom-of-pushdown symbol anywhere else in the pushdown. Moreover, we may assume that in every step M adds at most one symbol to the pushdown store. So, δ maps $Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\vartriangle\})$ to $Q \times (\Gamma \cup \{\vartriangle\})^{\leq 2}$ obeying the restrictions concerning $\vartriangle$. Additionally, we may assume that all λ-transitions of M are popping. We construct a stat-IDPDA $M' = \langle Q', \Sigma \cup \{\#\}, \Gamma \cup \{d, \vartriangle\}, p_\vartriangle, F', \bot, \delta_D, \delta_R, \delta_N\rangle$ that on inputs $h_r(w)$, for $w \in \Sigma^*$, simulates the computation of M on input w. Again, $d \notin \Gamma$ is a new dummy symbol and we set $\Sigma_D = \Sigma$, $\Sigma_R = \{\#\}$, and $\Sigma_N = \emptyset$. We define

$$Q' = \{ q_z \mid q \in Q, z \in \Gamma \cup \{\triangle\} \} \cup \{ q_z^{(i)} \mid q \in Q, i \in \{1,2\}, z \in \Gamma \cup \{\triangle\} \},$$

where the superscript expresses how many symbols d are still to be pushed and the subscript stores the topmost pushdown symbol below possible symbols d. The set of accepting states F' is then $\{ q_z \mid q \in F, z \in \Gamma \cup \{\triangle\} \}$.

By the properties of M, its first move is not a λ-transition. So, the computation of M' starts in the state $p_\triangle$, that is, in the initial state of M that additionally stores the topmost pushdown symbol of M's initial configuration.

Next, we define the transition function δ_D. The idea is similar to that in the proof of Theorem 2. However, here we have to handle the fact that M' must not push more than one additional symbol, and we have to implement the update of the subscript of the states.

For $p \in Q$, $a \in \Sigma$, and $z \in \Gamma$, we define δ_D for all $z' \in \Gamma \cup \{\triangle\}$ as follows.

If $\delta(p,a,z) = (q, xy)$ is a push move of M, then we set

(1) $\delta_D(p_z, a, z') = (q_x^{(2)}, \mathtt{push}(yz'), 0)$,

and if $\delta(p,a,\triangle) = (q, x\triangle)$, then

(2) $\delta_D(p_\triangle, a, \bot) = (q_x^{(2)}, \mathtt{push}(\triangle), 0)$.

If $\delta(p,a,z) = (q,x)$ is a top move of M, then we set

(3) $\delta_D(p_z, a, z') = (q_x^{(1)}, \mathtt{push}(dz'), 0)$,

and if $\delta(p,a,\triangle) = (q,\triangle)$, then

(4) $\delta_D(p_\triangle, a, \bot) = (q_\triangle^{(1)}, \mathtt{push}(d), 0)$.

If $\delta(p,a,z) = (q,\lambda)$ is a pop move of M, then we set

(5) $\delta_D(p_z, a, z') = (q_{z'}, \mathtt{push}(dd), 1)$.

Furthermore, the states $q^{(1)}$ and $q^{(2)}$ are used to push one and two more dummy symbols, respectively. So, for all $q \in Q$, $a \in \Sigma$, and $z, z' \in \Gamma \cup \{\triangle\}$ we set

(6) $\delta_D(q_z^{(2)}, a, z') = (q_z^{(1)}, \mathtt{push}(dz'), 0)$,
(7) $\delta_D(q_z^{(1)}, a, d) = (q_z, \mathtt{push}(dd), 1)$.

Note that by definition, $z, z' \neq d$.

Let us summarize what the construction yields so far. It shows that

$$(p, aw, xs) \vdash_M (q, w, ys') \text{ if and only if } (p_x, aw, s) \vdash_{M'}^{+} (q_y, w, dds').$$

Now, we turn to the definition of δ_R. On the first of two adjacent symbols #, M' just pops one dummy symbol in a non-stationary move. So, for all $p \in Q$ and $z, z' \in \Gamma \cup \{\triangle\}$ we set

(8) $\delta_R(p_z, \#, d) = (p_z^{(1)}, 1)$.

On the second of the two adjacent symbols #, M has to pop the remaining dummy symbols (in fact, by construction, there is exactly one d left). If the next move to be simulated is a λ-transition of M, then M' performs a stationary move; otherwise it does not. The information about which case applies comes from the state $p_z^{(1)}$, that is, the information if $\delta(p,\lambda,z)$ is defined or not.

If $\delta(p,\lambda,z)$ is undefined, then we set

(9) $\delta_R(p_z^{(1)}, \#, d) = (p_z, 1)$.

If $\delta(p,\lambda,z) = (q,\lambda)$ is defined, then we set

(10) $\delta_R(p_z^{(1)}, \#, d) = (p_z^{(1)}, 0)$.

In this way, there is always one λ-transition pending, and M' can decide whether the next but one transition is again a λ-transition. So, M' can perform a non-stationary move while simulating the last λ-transition in this sequence.

Let $\delta(p,\lambda,z) = (q,\lambda)$. If $\delta(q,\lambda,z')$ is undefined, then we set

(11) $\delta_R(p_z^{(1)}, \#, z') = (q_{z'}, 1)$.

Let $\delta(p,\lambda,z)=(q,\lambda)$. If $\delta(q,\lambda,z')=(r,\lambda)$ is defined, then we set

(12) $\delta_R(p_z^{(1)},\#,z')=(q_{z'}^{(1)},0)$.

Let us summarize what the extended construction yields. It shows that

$(p,aw,xs)\vdash_M^+(q,w,ys')$ such that $\delta(q,\lambda,y)$ is undefined if and only if $(p_x,a\#\#w,s)\vdash_{M'}^+(q_y,w,s')$.

As before, the construction of M' is finally extended such that it accepts only inputs of the form $(\Sigma\#\#)^*$. We conclude $(p,w,\triangle)\vdash_M^*(q,\lambda,zs)$ if and only if $(p_\triangle,h_r(w),\lambda)\vdash_{M'}^*(q_z,\lambda,s)$. This together with $L(M')\subseteq(\Sigma\#\#)^*$ implies $L=h_r^{-1}(L(M'))$. □

4. Computational capacity

Here we consider the computational capacities of the different types of input-driven pushdown automata. It turns out that they form a hierarchy as shown in Fig. 1 at the end of the section. In particular, for the classes beyond classical IDPDAs nondeterminism is better than determinism. Moreover, the possibility to perform stationary moves implies stronger devices than the possibility to push more than one symbol. This is true for the deterministic as well as for the nondeterministic classes. Further relations with (non)deterministic (real-time) pushdown automata are established.

We precede the comparisons by a useful technical lemma.

Lemma 5. *Let M be any IDPDA, stat-NIDPDA, push-NIDPDA, stat-IDPDA, or push-IDPDA with input alphabet Σ that contains the symbols a and b. If $L(M)\cap a^*b^*$ is not regular then $a\in\Sigma_D$ and $b\in\Sigma_R$.*

Proof. We show the lemma for M being a stat-NIDPDA. The other cases follow almost literally.

Assume that the signature of $M=\langle Q,\Sigma,\Gamma,q_0,F,\bot,\delta_D,\delta_R,\delta_N\rangle$ does not obey $a\in\Sigma_D$ and $b\in\Sigma_R$. We are going to show that in this case $L(M)\cap a^*b^*$ is regular.

First, let $a\in\Sigma_R\cup\Sigma_N$. Then we can construct a nondeterministic finite automaton that may perform stationary moves $M'=\langle Q',\{a,b\},q_0',F',\delta'\rangle$ accepting $L(M)\cap a^*b^*$. The idea is to simulate M whereby the topmost pushdown symbol is kept as part of the state. In addition, M' has to ensure that only words of the form a^*b^* are accepted. So, we set $Q'=Q\times(\Gamma\cup\{\bot\})\times\{r_a,r_b\}$, $q_0'=(q_0,\bot,r_a)$, and $F'=F\times(\Gamma\cup\{\bot\})\times\{r_a,r_b\}$. The transition function δ' maps $Q'\times\{a,b\}$ to the subsets of $Q'\times\{0,1\}$. It is specified as follows, where $p,q\in Q$, $z,z'\in\Gamma$, and $d\in\{0,1\}$.

(1) $\delta'((p,\bot,r_a),a)\ni((q,\bot,r_a),d)$ if $a\in\Sigma_R$ and $\delta_R(p,a,\bot)\ni(q,d)$
(2) $\delta'((p,\bot,r_a),a)\ni((q,\bot,r_a),d)$ if $a\in\Sigma_N$ and $\delta_N(p,a,\bot)\ni(q,\mathtt{top}(\bot),d)$
(3) $\delta'((p,\bot,r_a),b)\ni((q,z',r_b),d)$ if $b\in\Sigma_D$ and $\delta_D(p,b,\bot)\ni(q,\mathtt{push}(z'),d)$
(4) $\delta'((p,\bot,r_a),b)\ni((q,\bot,r_b),d)$ if $b\in\Sigma_R$ and $\delta_R(p,b,\bot)\ni(q,d)$
(5) $\delta'((p,\bot,r_a),b)\ni((q,\bot,r_b),d)$ if $b\in\Sigma_N$ and $\delta_N(p,b,\bot)\ni(q,\mathtt{top}(\bot),d)$
(6) $\delta'((p,z,r_b),b)\ni((q,z',r_b),d)$ if $b\in\Sigma_D$ and $\delta_D(p,b,z)\ni(q,\mathtt{push}(z'z''),d)$
(7) $\delta'((p,\bot,r_b),b)\ni((q,\bot,r_b),d)$ if $b\in\Sigma_R$ and $\delta_R(p,b,\bot)\ni(q,d)$
(8) $\delta'((p,\bot,r_b),b)\ni((q,\bot,r_b),d)$ if $b\in\Sigma_N$ and $\delta_N(p,b,\bot)\ni(q,\mathtt{top}(\bot),d)$

It is not hard to see that M' in fact accepts $L(M)\cap a^*b^*$. Since M' can only accept regular languages, $L(M)\cap a^*b^*$ must be regular.

The remaining case to consider is $a\in\Sigma_D$ and $b\in\Sigma_D\cup\Sigma_N$. We proceed as in the first case and construct the transition function of the nondeterministic finite automaton M' as follows.

(1) $\delta'((q_0,\bot,r_a),a)\ni((q,z',r_a),d)$ if $\delta_D(q_0,a,\bot)\ni(q,\mathtt{push}(z'),d)$
(2) $\delta'((p,z,r_a),a)\ni((q,z',r_a),d)$ if $\delta_D(p,a,z)\ni(q,\mathtt{push}(z'z''),d)$
(3) $\delta'((q_0,\bot,r_a),b)\ni((q,z',r_b),d)$ if $b\in\Sigma_D$ and $\delta_D(q_0,b,\bot)\ni(q,\mathtt{push}(z'),d)$
(4) $\delta'((q_0,\bot,r_a),b)\ni((q,\bot,r_b),d)$ if $b\in\Sigma_N$ and $\delta_N(q_0,b,\bot)\ni(q,\mathtt{top}(\bot),d)$
(5) $\delta'((p,z,r_a),b)\ni((q,z',r_b),d)$ if $b\in\Sigma_D$ and $\delta_D(p,b,z)\ni(q,\mathtt{push}(z'z''),d)$
(6) $\delta'((p,z,r_a),b)\ni((q,z',r_b),d)$ if $b\in\Sigma_N$ and $\delta_N(p,b,z)\ni(q,\mathtt{top}(z'),d)$
(7) $\delta'((p,z,r_b),b)\ni((q,z',r_b),d)$ if $b\in\Sigma_D$ and $\delta_D(p,b,z)\ni(q,\mathtt{push}(z'z''),d)$
(8) $\delta'((p,\bot,r_b),b)\ni((q,\bot,r_b),d)$ if $b\in\Sigma_N$ and $\delta_N(p,b,\bot)\ni(q,\mathtt{top}(\bot),d)$
(9) $\delta'((p,z,r_b),b)\ni((q,z',r_b),d)$ if $b\in\Sigma_N$ and $\delta_N(p,b,z)\ni(q,\mathtt{top}(z'),d)$

Again, it follows immediately from the construction that M' accepts the language $L(M)\cap a^*b^*$ and, thus, $L(M)\cap a^*b^*$ must be regular. □

We continue with a result that says that the restriction to push at most one symbol in each step is a serious one. As mentioned before, a DPDA can always be transformed into an equivalent one that obeys this part of a normal form, but an input-driven pushdown

automaton cannot. So, this property is an essential restriction. Later it will be shown that this restriction is also unavoidable if the neat properties of IDPDAs should be obtained. A witness language for the following proper inclusion is $\{ a^n b^{2n} \mid n \geq 1 \}$.

Theorem 6. *The family of languages accepted by push-IDPDAs is a proper superset of the family of languages accepted by IDPDAs.*

Proof. Trivially, we have the inclusion $\mathscr{L}(\text{IDPDA}) \subseteq \mathscr{L}(\text{push-IDPDA})$. In order to show the properness, we consider the witness language $L = \{ a^n b^{2n} \mid n \geq 1 \}$. Since L is not regular, we obtain from Lemma 5 that any IDPDA accepting L must have the signature $\Sigma_D = \{a\}$, $\Sigma_R = \{b\}$, and $\Sigma_N = \emptyset$. However, any IDPDA with this signature runs into a configuration with empty pushdown after processing an input prefix $a^n b^n$ and thus runs through loops while processing the remaining input suffix b^n. Therefore, it would accept inputs not belonging to L by running through the loop once more.

It remains to be shown that L is accepted by some push-IDPDA. To this end, we construct the push-IDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$ with state set $Q = \{q_0, q_1, q_2\}$, $\Sigma_D = \{a\}$, $\Sigma_R = \{b\}$, $\Sigma_N = \emptyset$, the set of pushdown symbols $\Gamma = \{A, A'\}$, the set of accepting states $F = \{q_2\}$, and the transition functions specified as:

(1) $\delta_D(q_0, a, \bot) = (q_0, \mathtt{push}(AA'))$
(2) $\delta_D(q_0, a, A) = (q_0, \mathtt{push}(AAA))$
(3) $\delta_R(q_0, b, A) = q_1$
(4) $\delta_R(q_1, b, A) = q_1$
(5) $\delta_R(q_1, b, A') = q_2$

First, an A and a primed A are stored in the pushdown store (Transition (1)). The primed A indicates the first A pushed and is later used to determine that the correct number of b's are read. Afterwards, for every following a in the input, two A's are pushed (Transition (2)). In the second phase, the content of the pushdown store is compared with the remaining input suffix of the form b^+. The second phase starts when the first b appears in the input (Transitions (3–4)). When the A' is popped, the automaton changes to the accepting state q_2, since in this situation the number of b's read fits (Transition (5)). State q_2 is undefined for every input, thus, it either accepts, if the lengths fit, and rejects otherwise. □

By the known equality $\mathscr{L}(\text{IDPDA}) = \mathscr{L}(\text{NIDPDA})$ (see, for example, [3,13]) and Theorem 6 we immediately obtain $\mathscr{L}(\text{NIDPDA}) \subset \mathscr{L}(\text{push-IDPDA})$. Next, we climb up the next level of the automata hierarchy and compare input-driven pushdown automata that may push more than one symbol and input-driven pushdown automata that may perform stationary moves. Though input-driven pushdown automata that may perform stationary moves may push only one symbol in each move, clearly, they can simulate input-driven pushdown automata that may push more than one symbol by sequences of stationary push moves.

Theorem 7. *The family of languages accepted by stat-NIDPDAs is a proper superset of the family of languages accepted by push-NIDPDAs.*
The family of languages accepted by stat-IDPDAs is a proper superset of the family of languages accepted by push-IDPDAs.

Proof. Since any push move of a push-IDPDA that pushes more than one symbol can be simulated by some stat-IDPDA that pushes the same symbols one by one in a sequence of stationary moves that is controlled by states, we have the inclusion $\mathscr{L}(\text{push-IDPDA}) \subseteq \mathscr{L}(\text{stat-IDPDA})$, and similarly the inclusion $\mathscr{L}(\text{push-NIDPDA}) \subseteq \mathscr{L}(\text{stat-NIDPDA})$.

In order to show the properness of the inclusion we use the witness language $L = \{ a^n b^l c^m d^n \mid l, m, n \geq 0 \text{ and } l \geq m \}$. By Example 1, L is accepted by a stat-IDPDA.

By way of contradiction, assume now that L is accepted by a push-NIDPDA $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$. Applying Lemma 5 to $L \cap a^* d^*$ and to $L \cap b^* c^*$ yields that $a \in \Sigma_D$ and $b \in \Sigma_D$ must be push symbols and the symbols $c \in \Sigma_R$ and $d \in \Sigma_R$ must be pop symbols.

We consider accepting computations on words of the form $a^n b^{n+1} d^n$, for $n \geq 1$. Since there are infinitely many of those words, but $|Q| \cdot |\Gamma|$ is finite, there are two different numbers $n_1 > n_2$ such that the accepting computations on $a^{n_1} b^{n_1+1} d^{n_1}$ and $a^{n_2} b^{n_2+1} d^{n_2}$ pass through configurations in which the state and topmost pushdown symbol are identical after having processed the prefixes a^{n_1} and a^{n_2}. More precisely, we have

$$(q_0, a^{n_1} b^{n_1+1} d^{n_1}, \lambda) \vdash^+ (q_1, b^{n_1+1} d^{n_1}, z\gamma_1) \vdash^+ (q_2, d^{n_1}, \gamma_2\gamma_1) \vdash^+ (q_+, \lambda, \gamma_3\gamma_1)$$

and

$$(q_0, a^{n_2} b^{n_2+1} d^{n_2}, \lambda) \vdash^+ (q_1, b^{n_2+1} d^{n_2}, z\gamma_1') \vdash^+ (q_2', d^{n_2}, \gamma_2'\gamma_1') \vdash^+ (q_+', \lambda, \gamma_3'\gamma_1'),$$

where $q_1, q_2, q_+, q_2', q_+' \in Q$, $q_+, q_+' \in F$, $z \in \Gamma$, and $\gamma_1, \gamma_2, \gamma, \gamma_1', \gamma_2', \gamma \in \Gamma^*$. Moreover, since $b \in \Sigma_D$ we know that $|\gamma_2| \geq n_1 + 2$. So, M can perform the computation

$$(q_0, a^{n_2} b^{n_1+1} d^{n_1}, \lambda) \vdash^+ (q_1, b^{n_1+1} d^{n_1}, z\gamma_1') \vdash^+ (q_2, d^{n_1}, \gamma_2\gamma_1') \vdash^+ (q_+, \lambda, \gamma_3\gamma_1')$$

and, thus, accept $a^{n_2} b^{n_1+1} d^{n_1}$, which does not belong to L, a contradiction. □

The next level of the hierarchy is the top level formed by pushdown automata not obeying the input-driven property.

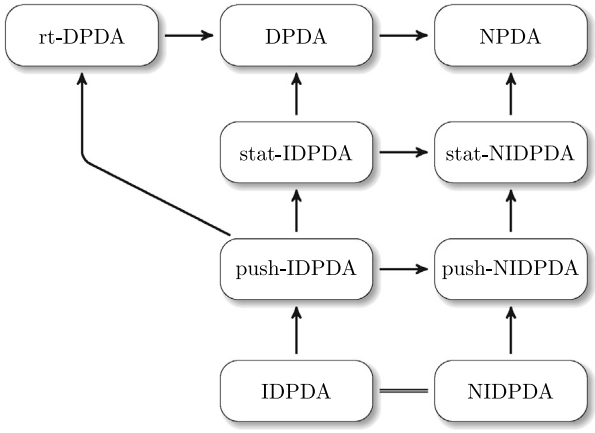

Fig. 1. Hierarchy of automata classes, where rt-DPDA denotes the class of deterministic real-time pushdown automata. An arrow indicates a proper inclusion of the induced language families. Each pair of nodes not connected by a path means that the two languages families are incomparable.

Theorem 8. *The family of languages accepted by NPDAs is a proper superset of the family of languages accepted by stat-NIDPDAs.*
The family of languages accepted by DPDAs is a proper superset of the family of languages accepted by stat-IDPDAs.
The family of languages accepted by real-time DPDAs is a proper superset of the family of languages accepted by push-IDPDAs.

Proof. By definition, only the properness of the inclusions has to be shown. To this end, we consider the language $L = \{ a^m b^m b^n a^n \mid m, n \geq 0 \}$ as witness for all three assertions. Clearly, L is accepted by some deterministic real-time pushdown automaton.

It remains to be shown that L cannot be accepted by any stat-NIDPDA. Contrarily, assume that L is accepted by some stat-NIDPDA M. Then, by applying Lemma 5 for $L \cap a^* b^*$ we obtain the result that necessarily $a \in \Sigma_D$ is a push symbol and $b \in \Sigma_R$ is a pop symbol. However, by applying Lemma 5 for $L \cap b^* a^*$ we obtain the result that necessarily $a \in \Sigma_R$ is a pop symbol and $b \in \Sigma_D$ is a push symbol. This is a contradiction that shows that L is not accepted by M. □

Next, we separate deterministic and nondeterministic classes.

Theorem 9. *The family of languages accepted by push-NIDPDAs is a proper superset of the family of languages accepted by push-IDPDAs.*
The family of languages accepted by stat-NIDPDAs is a proper superset of the family of languages accepted by stat-IDPDAs.

Proof. The language $\{ a^n b^n \mid n \geq 0 \} \cup \{ a^n b^{2n} \mid n \geq 0 \}$ is known not to be deterministic context free. But a push-NIDPDA or stat-NIDPDA accepting it can straightforwardly be constructed. □

Finally, we consider the classes depicted in Fig. 1 that are not connected by a path. Each pair of these classes describes two language families that are incomparable.

A witness language for Theorem 8 is the deterministic real-time context-free language $\{ a^m b^m b^n a^n \mid m, n \geq 0 \}$. It is not accepted by any stat-NIDPDA and, thus is not accepted by any stat-IDPDA and push-IDPDA either. On the other hand, the witness language for Theorem 9 can be accepted by a push-NIDPDA or a stat-NIDPDA, but is not deterministic context free.

Corollary 10. *The families of languages accepted by push-NIDPDA and stat-NIDPDA are both incomparable with each of the families* $\mathscr{L}$(*rt-DPDA*) *and* $\mathscr{L}$(*DPDA*).

Furthermore, from Example 1 and the proof of Theorem 7 we know that the language $\{ a^n b^l c^m d^n \mid l, m, n \geq 0 \text{ and } l \geq m \}$ is accepted by a stat-IDPDA but cannot be accepted by any push-NIDPDA. Additionally, it is not accepted by any deterministic real-time pushdown automaton.

Corollary 11. *The family of languages accepted by stat-IDPDA is incomparable with the families* $\mathscr{L}$(*push-NIDPDA*) *and* $\mathscr{L}$(*rt-DPDA*).

5. Decidability questions

In this section, we will derive some decidability questions for push-IDPDAs, stat-IDPDAs, push-NIDPDAs, and stat-NIDPDAs. Recall that a decidability problem is *semidecidable* whenever the set of all instances for which the answer is "yes" is recursively enumerable. The families $\mathscr{L}$(push-IDPDA) and $\mathscr{L}$(stat-IDPDA) are effective subsets of the deterministic context-free languages. Thus, all decidability questions that are decidable for DPDAs are decidable for push-IDPDAs and stat-IDPDAs as well. Analogously, the

families $\mathscr{L}$(push-NIDPDA) and $\mathscr{L}$(stat-NIDPDA) are effective subsets of the context-free languages and, hence, inherit their positive decidability results.

Theorem 12. *The problems of emptiness, universality, finiteness, infiniteness, equivalence, and regularity are decidable for push-IDPDAs and stat-IDPDAs. The problems of emptiness, finiteness, and infiniteness are decidable for push-NIDPDAs and stat-NIDPDAs.*

Next, we turn to undecidability results and therefore consider *valid computations* of Turing machines. Roughly speaking, these are histories of accepting Turing machine computations. It suffices to consider deterministic Turing machines with one single tape and one single read-write head. Without loss of generality and for technical reasons, we assume that the Turing machines can halt only after an odd number of moves, accept by halting, make at least three moves, and cannot print blanks. A valid computation is a string built from a sequence of configurations passed through during an accepting computation. Let Q be the state set of some Turing machine M, where q_0 is the initial state, T is the tape alphabet that contains the blank symbol and is disjoint from Q, and $\Sigma \subset T$ is the input alphabet. Then a configuration of M can be written as a word of the form T^*QT^* such that $t_1t_2\cdots t_iqt_{i+1}\cdots t_n$ is used to express that M is in state q, scanning tape symbol t_{i+1}, and $t_1t_2\cdots t_it_{i+1}\cdots t_n$ is the non-blank tape content. Then VALC(M) is the set of strings of the form $\$w_1\$w_2^R\$w_3\$w_4^R\$\cdots\$w_{2n-1}\$w_{2n}^R\$$. Here, $\$ \notin T \cup Q$, $w_i \in T^*QT^*$ are configurations of M, w_1 is an initial configuration of the form $q_0\Sigma^*$, w_{2n} is a halting, that is, accepting configuration, and w_{i+1} is the successor configuration of w_i. The set of *invalid computations* INVALC(M) is the complement of VALC(M) with respect to the alphabet $\{\$\} \cup T \cup Q$. The following facts on Turing machines are known and shown, for example, in [8].

Theorem 13. *Let M be a Turing machine.*

1. *It is not semidecidable whether or not $L(M)$ is empty, finite, or infinite.*
2. *VALC(M) is context free if and only if $L(M)$ is finite.*
3. *INVALC(M) is regular if and only if $L(M)$ is finite.*
4. *INVALC(M) is context free.*

Our first result is that the inclusion problem is not semidecidable for two push-IDPDAs with compatible signatures. This is in strong contrast to the result that the problem is decidable for IDPDAs and NIDPDAs. Hence, the ability of push-IDPDAs to push more than one symbol in one step turns the problem from decidability to non-semidecidability. As a preliminary result we show that the family $\mathscr{L}$(push-IDPDA) is closed under complementation.

Lemma 14. *Let M be a push-IDPDA. Then a push-IDPDA accepting the complement $\overline{L(M)}$ and having the same signature can effectively be constructed.*

Proof. By definition we know that an input is accepted by M only after having read it entirely. In case of rejection, either the computation of M ends in a non-accepting state after having read the input entirely or the computation of M is blocked since the transition function is undefined for the current situation. Hence, it is sufficient to switch in the latter case to a new non-accepting state in which the remaining input is read.

Thus, we have ensured that in every accepting or rejecting computation the input is completely read and we can construct a new push-IDPDA for the complement by interchanging accepting and rejecting states. Finally, we notice that the newly constructed push-IDPDA has the same signature as M. □

Theorem 15. *Let M and M' be two push-IDPDAs with compatible signatures. Then the inclusion problem $L(M) \subseteq L(M')$ is not semidecidable.*

Proof. Let M be a Turing machine. It is not difficult to show that VALC(M) is the intersection of two deterministic real-time context-free languages L_1 and L_2. A construction of two deterministic pushdown automata accepting L_1 and L_2 is given in [8]. The refinement to real-time deterministic pushdown automata is discussed in [12]. Due to Theorem 2 we know that two push-IDPDAs M_1 and M_2 having the same signature and accepting $h_r(L_1)$ and $h_r(L_2)$ can effectively be constructed. Since $\mathscr{L}$(push-IDPDA) is effectively closed under complementation, it is possible to construct a push-IDPDA M_3 having the same signature as M_1 and M_2 and accepting $\overline{h_r(L_2)}$. We know VALC(M) = ∅ if and only if $h_r(\text{VALC}(M)) = h_r(L_1 \cap L_2) = h_r(L_1) \cap h_r(L_2) = \emptyset$. Suppose that $L(M_1) \subseteq L(M_3)$ were semidecidable. Since $L(M_1) \subseteq L(M_3)$ if and only if $L(M_1) \cap \overline{L(M_3)} = h_r(L_1) \cap h_r(L_2) = \emptyset$, we could semidecide whether or not a given Turing machine accepts the empty set. This is a contradiction to Theorem 13. □

Corollary 16. *The inclusion problem as well as the inclusion problem with compatible signatures is not semidecidable for push-IDPDAs, stat-IDPDAs and their nondeterministic variants push-NIDPDAs and stat-NIDPDAs.*

Next, we are studying the universality and the regularity problem and show their non-semidecidability for push-NIDPDAs and stat-NIDPDAs.

Theorem 17. *Let M' be a push-NIDPDA over some input alphabet Σ. Then it is not semidecidable whether or not $L(M') = \Sigma^*$.*

Proof. Let M be a Turing machine. It is shown in [8] that INVALC(M) is a context-free language and it is defined over an alphabet $\Sigma_1 = \{\$\} \cup Q \cup T$. Due to Corollary 3 we know that a push-NIDPDA M' accepting $h_r(\text{INVALC}(M))$ can effectively be constructed. We consider the regular language

$$R = \{\, \#^{n_0} a_1 \#^{n_1} a_2 \#^{n_2} \cdots \#^{n_{m-1}} a_m \#^{n_m} \mid m \geq 0, n_i \geq 0, a_i \in \Sigma_1, 1 \leq i \leq m, \text{ and } (n_0 \geq 1 \text{ or } \exists 1 \leq i \leq m : \ n_i \neq 2) \,\}.$$

Since the family $\mathscr{L}$(push-NIDPDA) is clearly closed under union with regular languages, it is possible to construct a push-NIDPDA M'' accepting $h_r(\text{INVALC}(M)) \cup R$. Now, $h_r(\text{INVALC}(M)) \cup R = (\Sigma_1 \cup \{\#\})^*$ if and only if $h_r(\text{INVALC}(M)) = (\Sigma_1 \#\#)^*$ if and only if $\text{INVALC}(M) = \Sigma_1^*$. Suppose that universality were semidecidable for push-NIDPDAs. Hence, it would be semidecidable whether or not a given Turing machine accepts the empty set. This is again a contradiction to Theorem 13. □

Corollary 18. *Let M' be a stat-NIDPDA over some input alphabet Σ. Then it is not semidecidable whether or not $L(M') = \Sigma^*$.*

Since a push-NIDPDA accepting Σ^* can easily be constructed that in addition has the same signature as an arbitrarily given push-NIDPDA, it can be concluded that the non-semidecidability of the universality problem leads to the non-semidecidability of the equivalence problem even for automata with compatible signatures.

Corollary 19. *The equivalence problem with compatible signatures as well as the general equivalence problem is semidecidable neither for push-NIDPDAs nor for stat-NIDPDAs.*

Our next result shows that also the regularity problem is not semidecidable for push-NIDPDAs and stat-NIDPDAs.

Theorem 20. *Let M' be a push-NIDPDA. Then the question of whether or not M' accepts a regular language is not semidecidable.*

Proof. Let M be a Turing machine. Due to Corollary 3 it is possible to effectively construct a push-NIDPDA M' accepting $h_r(\text{INVALC}(M))$. Since the regular languages are closed under homomorphism and inverse homomorphism, we can conclude that $h_r(\text{INVALC}(M))$ is a regular language if and only if INVALC(M) is a regular language. On the other hand, it is known due to Theorem 13 that a Turing machine M accepts a finite set if and only if INVALC(M) is a regular language. Hence, if the regularity of M' would be semidecidable, then the finiteness of Turing machines would be semidecidable as well. This is a contradiction to Theorem 13. □

Corollary 21. *The regularity problem is not semidecidable for stat-NIDPDAs.*

Finally, we show that it is neither semidecidable whether a stat-NIDPDA accepts a language that can be accepted by some push-NIDPDA or IDPDA, nor whether an NPDA accepts a language that can be accepted by some stat-NIDPDA, push-NIDPDA or IDPDA, nor whether a push-NIDPDA accepts a language that can be accepted by some IDPDA. This means that it is not possible for NPDAs, stat-NIDPDAs and push-NIDPDAs to semidecide whether or not they accept languages already acceptable by weaker devices.

Theorem 22. *Let M' be a stat-NIDPDA. Then it is semidecidable neither whether or not $L(M')$ belongs to $\mathscr{L}$(push-NIDPDA) nor whether or not $L(M')$ belongs to $\mathscr{L}$(IDPDA).*

Proof. Let M be a Turing machine, $\Sigma = \{\$, \#\} \cup Q \cup T$, and $\mathfrak{c}_1, \mathfrak{c}_2, a, b$ be new symbols not in Σ. Then, we define the language

$$L_M = \{\, a^n w x b^n \mid n \geq 1, w \in h_r(\text{INVALC}(M)) \text{ and } x \in \mathfrak{c}_1 \Sigma^* \mathfrak{c}_2 \,\}.$$

First, we show that L_M can be accepted by a stat-NIDPDA M' and we sketch the construction. Let us assume for a moment that the infix x always has the form $x = \mathfrak{c}_1 \mathfrak{c}_2$. Then, M' pushes a symbol A for every a read, tests whether w belongs to $h_r(\text{INVALC}(M))$ by using the result of Corollary 3, pushes a symbol X while reading $\mathfrak{c}_1$ and pops X as well as all pushdown symbols different from A while performing stationary moves on $\mathfrak{c}_2$. Finally, the remaining b's of the input are matched against the A's on the pushdown store. If x has the form $x = \mathfrak{c}_1 x' \mathfrak{c}_2$ with $x' \neq \lambda$, then the symbols of x' might affect the height and the contents of the pushdown store and it could be difficult to match the b's at the end of the input with the a's from the beginning. To overcome this problem we will use additional stationary moves on the symbol $\mathfrak{c}_1$. When reaching the symbol $\mathfrak{c}_1$, M' guesses whether in the phase of reading x' the difference d of symbols causing a push-operation minus the symbols causing a pop-operation is positive ($d \geq 0$) or negative ($d < 0$). In the first case, $\mathfrak{c}_1$ pushes a symbol, the input x' is processed, and stationary moves on $\mathfrak{c}_2$ will pop all symbols different from A. In the second case, stationary moves on $\mathfrak{c}_1$ will push d symbols, the input x' is processed and d symbols are popped, and, finally, on $\mathfrak{c}_2$ all symbols different from A are popped. In both cases, if the guess was correct, the pushdown contents after reading $\mathfrak{c}_2$ contain only the A's pushed in the initial phase. If the guess was incorrect, then the situation may occur that a symbol A has to be popped. In this situation, the input is rejected. Hence, it is ensured that the processing of the infix x does not affect the pushdown contents and the pushdown store contains only those A's pushed in the phase of reading a's.

Second, we show that INVALC(M) has to be a regular language, if and only if $L_M \in \mathscr{L}$(push-NIDPDA). If INVALC(M) is a regular language, it is straightforward to construct a push-NIDPDA accepting L_M. It remains to be shown that $L_M \notin \mathscr{L}$(push-NIDPDA), if INVALC(M) is not a regular language. By way of contradiction, we assume that L_M is accepted by some push-NIDPDA M'' with signature $\Sigma'_D \cup \Sigma'_R \cup \Sigma'_N$. Since a, b do not belong to $\Sigma \cup \{\mathfrak{c}_1, \mathfrak{c}_2\}$, it can be shown similar to the proof of Lemma 5 that $a \in \Sigma'_D$ and $b \in \Sigma'_R$. Since INVALC(M) is context free but not regular, we know that h_r(INVALC(M)) is context free but not regular as well. Thus, there must exist at least one symbol $y \in \Sigma$ such that $y \in \Sigma'_R$. Otherwise, a finite automaton accepting the infix wx with $w \in h_r$(INVALC(M)) and $x \in \mathfrak{c}_1\Sigma^*\mathfrak{c}_2$ could be constructed. This would imply that h_r(INVALC(M)) and INVALC(M) are regular languages. This is a contradiction. Next, we fix a string $z \in h_r$(INVALC(M)) and consider accepting computations on the subset $\{ a^n z \mathfrak{c}_1 y^m \mathfrak{c}_2 b^n \mid n, m \geq 1 \}$ of L_M. In particular, since $y \in \Sigma'_R$ and m may be chosen large enough, there are at least two different words $w_1 = a^{n_1} z \mathfrak{c}_1 y^{m_1} \mathfrak{c}_2 b^{n_1}$ and $w_2 = a^{n_2} z \mathfrak{c}_1 y^{m_2} \mathfrak{c}_2 b^{n_2}$ with $n_1 \neq n_2$ such that M'' enters the same state and an empty pushdown store after processing the prefix $a^{n_1} z \mathfrak{c}_1 y^{m_1}$ of w_1 as well as the prefix $a^{n_2} z \mathfrak{c}_1 y^{m_2}$ of w_2. Hence, an input $a^{n_1} z \mathfrak{c}_1 y^{m_1} \mathfrak{c}_2 b^{n_2}$ is accepted as well. This is contradiction. Thus, we have shown that INVALC(M) has to be a regular language, if $L_M \in \mathscr{L}$(push-NIDPDA).

Now, we assume that it were semidecidable for a stat-NIDPDA whether or not its language accepted belongs to $\mathscr{L}$(push-NIDPDA). Then, we could semidecide this question in particular for the stat-NIDPDA M' accepting L_M and would obtain the semidecidability of the regularity of INVALC(M). Since we know due to Theorem 13 that INVALC(M) is regular if and only if the Turing machine M accepts a finite set, we could semidecide the finiteness problem for Turing machines which is known to be not semidecidable. Hence, we obtain a contradiction and the first claim of the theorem is proved.

For the second claim we notice that it is straightforward to construct an IDPDA accepting L_M, if INVALC(M) is a regular language. On the other hand, we already know that $L_M \notin \mathscr{L}$(push-NIDPDA), if INVALC(M) is not a regular language. It follows that $L_M \notin \mathscr{L}$(IDPDA), if INVALC(M) is not a regular language. Hence, assuming that it were semidecidable for a stat-NIDPDA whether or not its language accepted belongs to $\mathscr{L}$(IDPDA) we could analogously to the first claim derive a contradiction and obtain the second claim of the theorem. □

Theorem 23. *Let M' be an NPDA. Then it is not semidecidable whether or not $L(M')$ belongs to the families $\mathscr{L}$(stat-NIDPDA), $\mathscr{L}$(push-NIDPDA), and $\mathscr{L}$(IDPDA).*

Proof. Let M be a Turing machine, $\Sigma = \{\$\} \cup Q \cup T$, and a, b be new symbols not in Σ. Then, we define the language

$$L'_M = \{ a^n w b^n \mid n \geq 1, w \in \text{INVALC}(M) \}.$$

The language L'_M is context free and an NPDA accepting it can straightforwardly be constructed.

Next, we show that INVALC(M) has to be a regular language, if and only if $L'_M \in \mathscr{L}$(stat-NIDPDA). If INVALC(M) is a regular language, then it is straightforward to construct a stat-NIDPDA accepting L'_M.

It remains to be shown that $L'_M \notin \mathscr{L}$(stat-NIDPDA), if INVALC(M) is not a regular language. By way of contradiction, we assume that L'_M is accepted by some stat-NIDPDA M' with signature $\Sigma'_D \cup \Sigma'_R \cup \Sigma'_N$. Due to Lemma 5 we know that $a \in \Sigma'_D$ and $b \in \Sigma'_R$. In addition, since INVALC(M) is context free but not regular, there must exist at least one symbol $y \in \Sigma$ such that $y \in \Sigma'_R$. Otherwise, we could construct a finite automaton accepting the infix $w \in \text{INVALC}(M)$. Then, INVALC(M) would be a regular language which leads to a contradiction.

We consider now accepting computations on the subset $\{ a^n y^m b^n \mid n, m \geq 1 \}$ of L'_M and obtain the result that there are, in particular, at least two different words $w_1 = a^{n_1} y^{m_1} b^{n_1}$ and $w_2 = a^{n_2} y^{m_2} b^{n_2}$ with $n_1 \neq n_2$ such that M' enters the same state and an empty pushdown store after processing the prefix $a^{n_1} y^{m_1}$ of w_1 as well as the prefix $a^{n_2} y^{m_2}$ of w_2. Hence, an input $a^{n_1} y^{m_1} b^{n_2}$ is accepted as well. This is a contradiction.

Now, we show the first claim and assume, by way of contradiction, that it is semidecidable for an NPDA whether or not its language accepted belongs to $\mathscr{L}$(stat-NIDPDA). Then, we can semidecide this question for the NPDA accepting L'_M and can conclude that we can semidecide the regularity of INVALC(M). Analogously to the proof of Theorem 22 we can conclude that then the finiteness problem for Turing machines would be semidecidable. This is contradiction and proves the first claim of the theorem.

The remaining two claims can analogously be proved by first noticing that it is straightforward to construct an IDPDA or push-NIDPDA accepting L'_M, if INVALC(M) is a regular language. Second, since $L'_M \notin \mathscr{L}$(stat-NIDPDA), if INVALC(M) is not a regular language, it is clear that in this case L'_M neither belongs to $\mathscr{L}$(push-NIDPDA) nor to $\mathscr{L}$(IDPDA). □

Theorem 24. *Let M' be a push-NIDPDA. It is not semidecidable whether or not $L(M')$ belongs to $\mathscr{L}$(IDPDA).*

Proof. Let M be a Turing machine and $\Sigma = \{\$\} \cup Q \cup T$. Then, we define the language $L''_M = h_r$(INVALC(M)). Since INVALC(M) is a context-free language, we can construct a push-NIDPDA accepting L''_M due to Corollary 3.

Next, we show that INVALC(M) has to be a regular language, if and only if $L''_M \in \mathscr{L}$(IDPDA). If INVALC(M) is a regular language, then it is straightforward to construct an IDPDA accepting L''_M and it remains to be shown that $L''_M \notin \mathscr{L}$(IDPDA), if INVALC(M) is not a regular language. By way of contradiction, we assume that L''_M is accepted by some IDPDA M''. Since the language family $\mathscr{L}$(IDPDA) is closed under complementation and intersection with regular languages, we obtain the fact that $\overline{L''_M} \cap (\Sigma\#\#)^* = h_r(\text{VALC}(M))$ is accepted by some IDPDA and is, in particular, a context-free language. Since the family of context-free languages is closed under inverse homomorphism, we obtain the result that VALC(M) is a context-free language as well. Applying Theorem 13 gives that then $L(M)$ is a finite set which in turn implies that INVALC(M) is a regular language. This is a contradiction.

Table 1
Decidability questions of the language families discussed. Symbols $\subseteq_c$ and $=_c$ denote inclusion and equivalence with compatible signatures. Such questions are not defined for non-input-driven devices and are marked with '—'.

	$\emptyset$	FIN	Σ^*	$\subseteq$	$\subseteq_c$	$=$	$=_c$	REG
$\mathscr{L}$(IDPDA)	✓	✓	✓	✗	✓	✓	✓	✓
$\mathscr{L}$(push-IDPDA)	✓	✓	✓	✗	✗	✓	✓	✓
$\mathscr{L}$(stat-IDPDA)	✓	✓	✓	✗	✗	✓	✓	✓
$\mathscr{L}$(DPDA)	✓	✓	✓	✗	—	✓	—	✓
$\mathscr{L}$(push-NIDPDA)	✓	✓	✗	✗	✗	✗	✗	✗
$\mathscr{L}$(stat-NIDPDA)	✓	✓	✗	✗	✗	✗	✗	✗
$\mathscr{L}$(NPDA)	✓	✓	✗	✗	—	✗	—	✗

Analogously to the proof of Theorem 22 the assumption that it is semidecidable for a push-NIDPDA whether or not its language accepted belongs to $\mathscr{L}$(IDPDA) leads to the semidecidability of the regularity of INVALC(M). This gives the semidecidability of the finiteness of the Turing machine M which is known to be not semidecidable. □

The status of the decidability questions discussed in this section together with the results for IDPDAs, DPDAs, and NPDAs is summarized in Table 1.

CRediT authorship contribution statement

Martin Kutrib: Conceptualization, Writing – original draft, Writing – review & editing. **Andreas Malcher:** Investigation, Writing – original draft, Writing – review & editing. **Matthias Wendlandt:** Investigation, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. Alur, P. Madhusudan, Visibly pushdown languages, in: L. Babai (Ed.), Symposium on Theory of Computing (STOC 2004), ACM, 2004, pp. 202–211.
[2] R. Alur, P. Madhusudan, Adding nesting structure to words, J. ACM 56 (2009).
[3] B. von Braunmühl, R. Verbeek, Input-driven languages are recognized in log n space, in: M. Karpinski, J. van Leeuwen (Eds.), Topics in the Theory of Computation, in: Mathematics Studies, vol. 102, North-Holland, Amsterdam, 1985, pp. 1–19.
[4] D. Caucal, Synchronization of pushdown automata, in: O.H. Ibarra, Z. Dang (Eds.), Developments in Language Theory (DLT 2006), in: LNCS, vol. 4036, Springer, 2006, pp. 120–132.
[5] P. Chervet, I. Walukiewicz, Minimizing variants of visibly pushdown automata, in: L. Kucera, A. Kucera (Eds.), Mathematical Foundations of Computer Science (MFCS 2007), in: LNCS, vol. 4708, Springer, 2007, pp. 135–146.
[6] S. Crespi-Reghizzi, D. Mandrioli, Operator precedence and the visibly pushdown property, J. Comput. Syst. Sci. 78 (2012) 1837–1867.
[7] P.W. Dymond, Input-driven languages are in log n depth, Inf. Process. Lett. 26 (1988) 247–250.
[8] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Massachusetts, 1979.
[9] M. Kutrib, A. Malcher, Digging input-driven pushdown automata, RAIRO Inform. Théor. 55 (2021) 6.
[10] M. Kutrib, A. Malcher, M. Wendlandt, Tinput-driven pushdown, counter, and stack automata, Fundam. Inform. 155 (2017) 59–88.
[11] M. Kutrib, A. Malcher, M. Wendlandt, On the power of pushing or stationary moves for input-driven pushdown automata, in: P. Caron, L. Mignot (Eds.), Implementation and Application of Automata (CIAA 2022), in: LNCS, vol. 13266, Springer, 2022, pp. 140–151.
[12] A. Malcher, On the descriptional complexity of iterative arrays, IEICE Trans. Inf. Syst. E87-D (2004) 721–725.
[13] K. Mehlhorn, Pebbling mountain ranges and its application of DCFL-recognition, in: J.W. de Bakker, J. van Leeuwen (Eds.), International Colloquium on Automata, Languages and Programming (ICALP 1980), in: LNCS, vol. 85, Springer, 1980, pp. 422–435.
[14] A. Okhotin, K. Salomaa, Complexity of input-driven pushdown automata, SIGACT News 45 (2014) 47–67.