I F I G
Research
Report

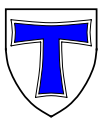
Institut für Informatik

Flip-Pushdown Automata: $k+1$ Pushdown Reversals are Better Than k

Markus Holzer Martin Kutrib

IFIG Research Report 0206
November 2002

Institut für Informatik
JLU Gießen
Arndtstraße 2
D-35392 Giessen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de



Justus-Liebig-
Universität
Giessen

IFIG Research Report
IFIG Research Report 0206, November 2002

Flip-Pushdown Automata: $k+1$ Pushdown Reversals are Better Than k

Markus Holzer[1]

Institut für Informatik, Technische Universität München
Boltzmannstraße 3, D-85748 Garching bei München, Germany

Martin Kutrib[2]

Institute für Informatik, Universität Giessen
Arndtstr. 2, D-35392 Giessen, Germany

Abstract. Flip-pushdown automata are pushdown automata with the additional power to flip or reverse its pushdown, and were recently introduced by Sarkar. We solve most of Sarkar's open problems. In particular, we show that $k+1$ pushdown reversals are better than k for both deterministic and nondeterministic flip-pushdown automata, i.e., there are languages which can be recognized by a deterministic flip-pushdown automaton with $k+1$ pushdown reversals but which cannot be recognized by a k-flip-pushdown (deterministic or nondeterministic). Furthermore, we investigate closure and non-closure properties as well as computational complexity problems such as fixed and general membership.

CR Subject Classification (1998): F.1, F.4.3

[1]E-mail: holzer@in.tum.de
[2]E-mail: mk@ifig.de

1 Introduction

A pushdown automaton is a one-way finite automaton with a separate pushdown store (PD), that is a last-in first-out (LIFO) storage structure, which is manipulated by pushing and popping. Probably, such machines are best known for capturing the family of context-free languages $\mathscr{L}$(CFL), which was independently established by Chomsky [5] and Evey [7]. The origin of the pushdown concept is not clear and is attributed by most to Burks *et al.* [4] and Newell and Shaw [14]. A little later the term LIFO storage was used explicitly in the literature, by Samelson and Bauer [16], who proposed it as an aide in the translation of ALGOL formulas into machine instructions. Pushdown automata have been extended in various ways. Examples of extensions are variants of stacks [9, 11], queues or dequeues, while restrictions are for instance counters or one-turn pushdowns [10]. The results obtained for these classes of machines hold for a large variety of formal language classes, when appropriately abstracted. This led to the rich theory of abstract families of automata (AFA), which is the equivalent of abstract families of languages (AFL) theory; for the general treatment of machines and languages we refer to Ginsburg [8].

In this paper, we consider a recently introduced extension of pushdown automata, so called flip-pushdown automata [17]. Basically, a flip-pushdown automaton is an ordinary pushdown automaton with the additional ability to flip its pushdown during the computation. This allows the machine to push and pop at both ends of the pushdown. Hence, a flip-pushdown is a form of a dequeue storage structure, and thus becomes equally powerful to Turing machines, since a dequeue automaton can simulate two pushdowns. On the other hand, if the number of pushdown flips or pushdown reversals is zero, obviously the family of context-free languages is characterized. Thus it remains to investigate the number of pushdown reversals as a natural computational resource.

By Sarkar [17] it was shown that if the number of pushdown flips is bounded by a constant, then a nonempty hierarchy of language classes is introduced, and it was conjectured that the hierarchy is strict. Obviously, since by a single pushdown reversal one can accept the non-context-free language $\{ ww \mid w \in \{a, b\}^* \}$, the base level of that hierarchy is already separated. But what about the other levels? In fact, in this paper we solve most of the open problems stated by Sarkar, especially the above mentioned one. More precisely, we show that $k + 1$ pushdown reversals are better than k for both deterministic and nondeterministic flip-pushdown automata. To this end, we develop a technique to decrease the number of pushdown reversals, which simply speaking shows that flipping the pushdown is equivalent to reverse part of the remaining input, hence calling our technique the "flip-pushdown input-reversal" theorem. An immediate consequence of this theorem is that every flip-pushdown language accepted by a flip-pushdown with a constant number of pushdown reversals obeys a semi-linear Parikh mapping.

Moreover, we also investigate closure and non-closure properties for the language families under consideration. It turns out, that the family of flip-pushdown languages share similar closure and non-closure properties as the family of context-free languages like, e.g., closure under intersection with regular sets, or the non-closure under complementation. Not surprisingly, the family of flip-pushdown languages is shown to be a full TRIO. Nevertheless, there are some interesting differences as, e.g., the non-closure under concatenation and Kleene star. Again, the flip-pushdown input-reversal theorem turns out to be very helpful in order to obtain the mentioned non-closure results.

Finally, computational complexity aspects of flip-pushdown languages with a constant number of pushdown reversals are considered. Again similarities to context-free languages are found. At first glance, we show that every flip-pushdown language accepted by a flip-pushdown automata with a constant number of pushdown reversals is context-sensitive. Moreover, it is proven that auxiliary flip-pushdown automata with exactly k pushdown reversals, i.e., a flip-pushdown automaton with a resource-bounded working-tape, capture P when their space is logarithmically bounded, and catch the important complexity class LOG(CFL) $\subseteq$ P when additionally their time is polynomially bounded. This nicely resembles the known results on auxiliary pushdown automata given by Cook [6] and Sudborough [18].

The paper is organized as follows: The next section contains preliminaries, and we show basics on flip-pushdown automata showing that the flip-pushdown languages accepted by nondeterministic flip-pushdown automata by final state are exactly those languages accepted by nondeterministic flip-pushdown automata by empty store. Then Section 3 is devoted to our main technique, the flip-pushdown input-reversal theorem and its application in the separation of the flip-pushdown hierarchy for both deterministic and nondeterministic machines. The next section deals with closure and non-closure properties and in the penultimate Section 5 we investigate computational complexity aspects of flip-pushdown languages. Finally we summarize our results and highlight the remaining open questions in Section 6.

2 Definitions

We assume the reader to be familiar with the basics of complexity theory as contained in the book of Balcázar *et al.* [2]. In particular we consider the following well-known chain of inclusions:

$$\mathrm{NC}^1 \subseteq \mathrm{LOG(CFL)} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}.$$

Here NC^1 is the class of problems accepted by uniform families of logarithmic depth, polynomial size circuits with bounded fan-in AND- and OR-gates, LOG(CFL) is the class of problems logspace many-one reducible to a context-free language, and P (NP, respectively) is the set of problems accepted by deterministic (nondeterministic, respectively) polynomially time bounded Turing

machines. Moreover, PSPACE is $\bigcup_k \mathrm{DSpace}(n^k)$. Completeness and hardness are always meant with respect to deterministic log-space many-one reducibilities.

For details on formal language we refer the reader to the book of Hopcroft and Ullman [12]. Consider the strict chain of inclusions

$$\mathscr{L}(\mathrm{REG}) \subset \mathscr{L}(\mathrm{CFL}) \subset \mathscr{L}(\mathrm{CS}) \subset \mathscr{L}(\mathrm{RE}),$$

where $\mathscr{L}(\mathrm{REG})$ denotes the family of regular languages, $\mathscr{L}(\mathrm{CFL})$ the family of context-free languages, $\mathscr{L}(\mathrm{CS})$ the family of context-senstive languages, and $\mathscr{L}(\mathrm{RE})$ the family of recursively enumerable languages.

In the following we consider pushdown automata with the ability to flip their pushdowns. These machines were recently introduced by Sarkar [17] and are defined as follows:

Definition 1 *A* flip-pushdown automaton *is a system*

$$A = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, F),$$

where Q is a finite set of states, Σ is the finite input alphabet, Γ is a finite pushdown alphabet, δ is a mapping from $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$ called the transition function, Δ is a mapping from Q to 2^Q, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is a particular pushdown symbol, called the bottom-of-pushdown symbol, which initially appears on the pushdown store, and $F \subseteq Q$ is the set of final states.

A *configuration* or *instantaneous description* of a flip-pushdown automaton is a triple (q, w, γ), where q is a state in Q, w a string of input symbols, and γ is a string of pushdown symbols. A flip-pushdown automaton A is said to be in configuration (q, w, γ) if A is in state q with w as remaining input, and γ on the pushdown store, the rightmost symbol of γ being the top symbol on the pushdown. If a is in $\Sigma \cup \{\lambda\}$, w in Σ^*, γ and β in Γ^*, and Z is in Γ, then we write $(q, aw, \gamma Z) \vdash_A (p, w, \gamma\beta)$, if the pair (p, β) is in $\delta(q, a, Z)$, for "ordinary" pushdown transitions and $(q, aw, Z_0\gamma) \vdash_A (p, aw, Z_0\gamma^R)$, if p is in $\Delta(q)$, for pushdown-flip or pushdown-reversal transitions. Whenever, there is a choice between an ordinary pushdown transition or a pushdown reversal one, then the automaton nondeterministically chooses the next move. Observe, that we do not want the flip-pushdown automaton to move the bottom-of-pushdown symbol when the pushdown is flipped. As usual, the reflexive transitive closure of $\vdash_A$ is denoted by $\vdash_A^*$. The subscript A will be dropped from $\vdash_A$ and $\vdash_A^*$ whenever the meaning remains clear.

Let k be a natural number. For a flip-pushdown automaton A we define $T_k(A)$, the language *accepted by final state and exactly k pushdown reversals**, to be

$$T_k(A) = \{\, w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_A^* (q, \lambda, \gamma) \text{ with exactly } k \text{ pushdown reversals, for any } \gamma \in \Gamma^* \text{ and } q \in F \,\}.$$

Also, we define $N_k(A)$, the language *accepted by empty pushdown and exactly k pushdown reversals*, to be

$$N_k(A) = \{\, w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_A^* (q, \lambda, \lambda) \text{ with exactly } k \text{ pushdown reversals, for any } q \in Q \,\}.$$

If the number of pushdown reversals is not limited, the language accepted by final state (empty pushdown, respectively) is analogously defined as above and denoted by $T(A)$ ($N(A)$, respectively). When accepting by empty pushdown, the set of final states is irrelevant. Thus, in this case, we usually let the set of final states be the empty set.

In order to clarify our notation we give a small example.

Example 2 Let $A = (\{q_0, q_1\}, \{a, b\}, \{A, B, Z_0\}, \delta, \Delta, q_0, Z_0, \emptyset)$ be a flip-pushdown automaton where

1. $\delta(q_0, a, Z_0) = \{(q_0, Z_0A)\}$
2. $\delta(q_0, b, Z_0) = \{(q_0, Z_0B)\}$
3. $\delta(q_0, a, A) = \{(q_0, AA)\}$
4. $\delta(q_0, b, A) = \{(q_0, AB)\}$
5. $\delta(q_0, a, B) = \{(q_0, BA)\}$
6. $\delta(q_0, b, B) = \{(q_0, BB)\}$
7. $\delta(q_1, a, A) = \{(q_1, \lambda)\}$
8. $\delta(q_1, b, B) = \{(q_1, \lambda)\}$
9. $\delta(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$

and $\Delta(q_0) = \{q_1\}$ that accepts by empty pushdown the non-context-free language $L = \{\, ww \mid w \in \{a, b\}^*\}$. This is seen as follows.

The transitions (1) through (6) allow A to store the input on the pushdown. If A decides that the middle of the input string has been reached, then the flip operation specified by $\Delta(q_0) = \{q_1\}$ is selected and A goes to state q_1 and tries to match the remaining input symbols with the reversed pushdown content. This is done with the transitions (7) and (8). Thus, if the guess of A was right, and the input is of the form ww, then the inputs will match, and A will empty its pushdown with transition (9), and therefore accept the input string (by empty pushdown).

*One may define language acceptance of flip-pushdown automata with *at most k* pushdown reversals. Since a flip-pushdown automaton can count the number of reversals performed during its computation in its finite control, it is an easy exercise to show that these to language acceptance mechanisms coincide.

The next theorem generalizes the theorem on ordinary pushdown automata, that languages accepted by nondeterministic flip-pushdown automata by final state are exactly those languages accepted by nondeterministic flip-pushdown automata by empty storage. We state the theorem without proof, since it is a simple adaption of the proof for ordinary pushdown automata.

Theorem 3 *Let k be some natural number. Then language L is accepted by some flip-pushdown automaton A_1 with empty pushdown making exactly k pushdown reversals, i.e., $L = N_k(A_1)$, if and only if language L is accepted by some flip-pushdown automaton A_2 by final state making exactly k pushdown reversals, i.e., $L = T_k(A_2)$. The statement remains valid for flip-pushdown automata with an unbounded number of pushdown reversals.* □

The family of languages accepted by flip-pushdown automata with empty pushdown or equivalently by final state making exactly k or equivalently at most k pushdown reversals is denoted by $\mathscr{L}(\mathrm{FNPDA}_k)$. Furthermore, let

$$\mathscr{L}(\mathrm{FNPDA}_{fin}) = \bigcup_{k=0}^{\infty} \mathscr{L}(\mathrm{FNPDA}_k)$$

and if the number of pushdown reversals is unbounded, the corresponding language family is referred to $\mathscr{L}(\mathrm{FNPDA})$. We recall the following theorem of Sarkar [17].

Theorem 4

$$\mathscr{L}(\mathrm{CFL}) = \mathscr{L}(\mathrm{FNPDA}_0) \subseteq \mathscr{L}(\mathrm{FNPDA}_1) \subseteq \cdots$$
$$\cdots \subseteq \mathscr{L}(\mathrm{FNPDA}_{fin}) \subseteq \mathscr{L}(\mathrm{FNPDA}) = \mathscr{L}(\mathrm{RE})$$

An immediate question that arises from the previous theorem is, whether the hierarchy on pushdown reversals is a strict hierarchy, and whether the upper bound can be improved to the family of context-sensitive languages $\mathscr{L}(\mathrm{CS})$. In the next sections we positively affirm these questions in the sense, that the hierarchy is strict and that the upper bound can be improved.

3 The Flip-Pushdown Input-Reversal Technique

In this section we prove an essential technique for flip-pushdown automata, which will be called "flip-pushdown input-reversal" since flipping the pushdown can be simulated by reversing the (remaining) input. The main theorem of this section reads as follows:

Theorem 5 *Let k be a natural number. Language L is accepted by a flip-pushdown $A_1 = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, \emptyset)$ by empty pushdown with $k+1$ pushdown reversals, i.e., $L = N_{k+1}(A_1)$, if and only if language*

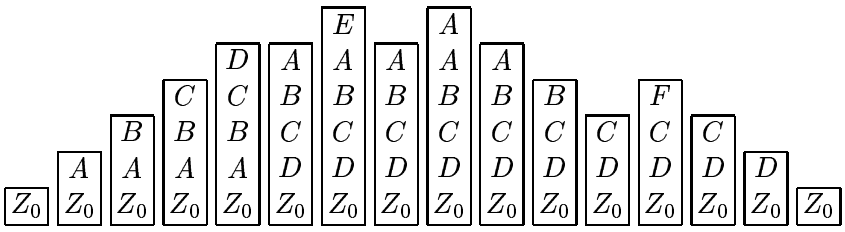
$$L_R = \{\, wv^R \mid (q_0, w, Z_0) \vdash_{A_1}^* (q_1, \lambda, Z_0\gamma) \text{ with } k \text{ reversals, } q_2 \in \Delta(q_1), \\ \text{and } (q_2, v, Z_0\gamma^R) \vdash_{A_1}^* (q_3, \lambda, \lambda) \text{ without any reversal} \,\}$$

is accepted by a flip-pushdown automaton A_2 by empty pushdown with k pushdown reversals, i.e., $L_R = N_k(A_2)$. The statement remains valid if state acceptance is considered.

Before we prove the above given statement, we want to give some insights and explanations. Consider the following sample computation on a flip-pushdown automaton:

$$\begin{aligned}
(q_0, abcdefghijklmno, Z_0) &\vdash (q_1, bcdefghijklmno, Z_0A) \\
&\vdash (q_2, cdefghijklmno, Z_0AB) \\
&\vdash (q_3, defghijklmno, Z_0ABC) \\
&\vdash (q_4, efghijklmno, Z_0ABCD) \\
&\vdash (q_5, efghijklmno, Z_0DCBA) \\
&\vdash (q_6, fghijklmno, Z_0DCBAE) \\
&\vdash (q_7, ghijklmno, Z_0DCBA) \\
&\vdash (q_8, hijklmno, Z_0DCBAA) \\
&\vdash (q_9, ijklmno, Z_0DCBA) \\
&\vdash (q_{10}, jklmno, Z_0DCB) \\
&\vdash (q_{11}, klmno, Z_0DC) \\
&\vdash (q_{12}, lmno, Z_0DCF) \\
&\vdash (q_{13}, mno, Z_0DC) \\
&\vdash (q_{14}, no, Z_0D) \vdash (q_{15}, o, Z_0) \vdash (q_{16}, \lambda, \lambda)
\end{aligned}$$

First, let us take a closer look on the pushdown actions. The behaviour of the flip-pushdown can be visualized as follows:



Now assume that we write A (A^{-1}, respectively), if we push (pop, respectively) symbol A. Then the push-pop action sequences on the given sample

computation read as

$$Z_0ABCD \quad \text{and} \quad EE^{-1}AA^{-1}A^{-1}B^{-1}FF^{-1}C^{-1}D^{-1}Z_0^{-1},$$

taking the flip-pushdown move into account. Since the push-pop action sequences must specify a valid flip-pushdown computation we find that

$$Z_0DCBA \cdot EE^{-1}AA^{-1}A^{-1}B^{-1}FF^{-1}C^{-1}D^{-1}Z_0^{-1} \quad \text{reduces to} \quad \lambda,$$

if rules of the form $XX^{-1} \to \lambda$ are applied.

Now assume that the pushdown reversal move is not done, and the sample computation is simulated backwards from the last state in the configuration sequence towards the pushdown reversal move. Then the push-pop action sequences are

$$Z_0ABCD \quad \text{and} \quad Z_0DCFF^{-1}BAAA^{-1}EE^{-1}.$$

Observe, that the latter sequence is the reverse-inverse of the above given sequence $EE^{-1}AA^{-1}A^{-1}B^{-1}FF^{-1}C^{-1}D^{-1}Z_0^{-1}$, since pushing becomes popping and *vice versa*. Moving Z_0 from the beginning of the latter sequence to its end, then we find that

$$Z_0ABCD \cdot DCFF^{-1}BAAA^{-1}EE^{-1}Z_0 \quad \text{reduces to} \quad Z_0ABCDDCBAZ_0,$$

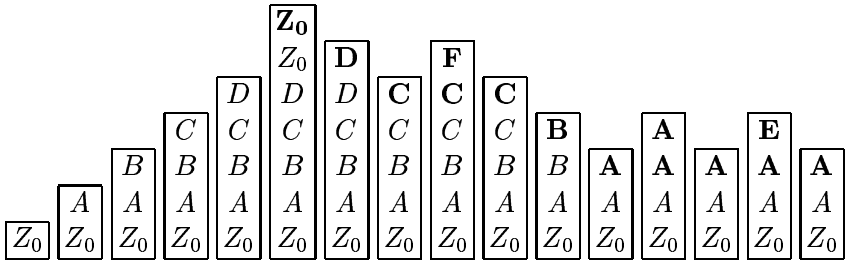
if rules of the form $XX^{-1} \to \lambda$ are applied. After this first reduction phase, sequence

$$Z_0ABCDDCBAZ_0 \quad \text{can be reduced to} \quad \lambda,$$

if rules of the form $XX \to \lambda$ are applied at the borderline, where the pushdown reversal of the original computation has appeared. In fact, these two types of reduction steps can be inter-winded. Later we will use this in the simulation of a valid (flip-)pushdown protocol. In order to distinguish between the pushdown symbols stored before the last flip and the symbols used in the ultimate phase, we use boldface symbols in the latter. In fact, the inter-winded application of rules from the two different phases can be simulated by the following rules (1) $XX^{-1} \to \lambda$, (2) $X\mathbf{XY} \to \mathbf{Y}$, (3) $\mathbf{XX}^{-1} \to \lambda$, and (4) $Z_0\mathbf{Z_0} \to \lambda$. The special form of the rules (2) will become clearer later. Then one finds that

$$Z_0ABCD \cdot \mathbf{DCFF}^{-1}\mathbf{BAAA}^{-1}\mathbf{EE}^{-1}\mathbf{Z_0} \quad \text{reduces to} \quad \lambda$$

within a single phase. These rules can be made the basis for simulating a valid computation, when the pushdown is not flipped. Such a computation is visualized next:



where the ultimate step pops $\mathbf{A}$, A, and Z_0 in order to terminate the computation and to accept the input. Observe, that the major problem in the backward simulation is to have the appropriate pushdown symbol at the top of the storage available in order to simulate a single step backwards. As the reader may verify, the visualized computation shown above has this property.

In order to simplify presentation, we introduce the notion of a generalized flip-pushdown automaton $A = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, F)$, where Q, Σ, Γ, Δ, $q_0 \in Q$, $Z_0 \in \Gamma$, and $F \subseteq Q$ are as in the case of ordinary flip-pushdown automata, and δ is a *finite domain* mapping from $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma^*$ to the finite subsets of $Q \times \Gamma^*$. With standard techniques one can construct an ordinary flip-pushdown automaton from a given generalized one, without increasing the number of pushdown-flips. Due to the ability to read words instead of symbols, the necessary checks, whether a push or pop action can be performed in the backward simulation becomes easier to describe.

Proof. [of Theorem 5] We only prove the direction from left to right. The converse implication can be shown by similar arguments.

Let $A_1 = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, \emptyset)$ be a flip-pushdown automaton satisfying $\gamma \in \{\lambda\} \cup \{\, ZX \mid X \in \Gamma \,\}$ for all $(p, \gamma) \in \delta(q, a, Z)$, where $p, q \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $Z \in \Gamma$. This normal form can be easily achieved.

Then we define a generalized flip-pushdown automaton

$$A_2 = (Q \cup \mathbf{Q} \cup \{\mathbf{q}_f\}, \Sigma, \Gamma \cup \mathbf{\Gamma} \cup Q, \delta', \Delta', q_0, Z_0, \{\mathbf{q}_f\}),$$

where $\mathbf{Q} = \{\, \mathbf{q} \mid q \in Q \,\}$, $\mathbf{\Gamma} = \{\, \mathbf{Z} \mid Z \in \Gamma \,\}$, and δ' and Δ' are specified as follows:

1. For all $q \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $Z \in \Gamma$, set $\delta'(q, a, Z)$ includes all elements of $\delta(q, a, Z)$ and
2. for all $q \in Q$, let $\Delta'(q)$ contain all elements of $\Delta(q)$.
3. For all $r \in Q$, if $\Delta(r) \neq \emptyset$, then $\delta'(r, a, Z)$ contains $(\mathbf{q}, ZZ_0r\mathbf{Z}_0)$, where $q \in Q$ satisfies $(p, \lambda) \in \delta(q, a, Z_0)$ for some $p \in Q$ and $a \in \Sigma \cup \{\lambda\}$.
4. For all $p, q \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $X, Y \in \Gamma$, let $\delta'(\mathbf{q}, a, \mathbf{XY})$ contain $(\mathbf{p}, \mathbf{X})$ if $(q, XY) \in \delta(p, a, X)$.
5. For all $p, q, r \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $X, Y \in \Gamma$, then
   (a) let $\delta'(\mathbf{q}, a, \mathbf{X})$ contain $(\mathbf{p}, \mathbf{XY})$ if $(q, \lambda) \in \delta(p, a, Y)$ and
   (b) let $\delta'(\mathbf{q}, a, Xr\mathbf{X})$ contain $(\mathbf{p}, r\mathbf{Y})$ if $(q, \lambda) \in \delta(p, a, Y)$.
6. For all $X \in \Gamma$ and $p \in \Delta(r)$, for some $r \in Q$, let $\delta'(\mathbf{p}, \lambda, Z_0Xr\mathbf{X})$ contain $(\mathbf{q}_f, \lambda)$.

Simulation			
forward		backward	
flip	$p \in \Delta(q)$	$(\mathbf{p}', ZZ_0q\mathbf{Z}_0) \in \delta'(\mathbf{q}, a, Z)$ if $(p'', \lambda) \in \delta(p', a, Z_0)$	push
push	$(p, XY) \in \delta(q, a, X)$	$(\mathbf{q}, \mathbf{X}) \in \delta'(\mathbf{p}, a, \mathbf{XY})$	pop
pop	$(p, \lambda) \in \delta(q, a, Y)$	$(\mathbf{q}, \mathbf{XY}) \in \delta'(\mathbf{p}, a, \mathbf{X})$	push (a)
		$(\mathbf{q}, r\mathbf{Y}) \in \delta'(\mathbf{p}, a, Xr\mathbf{X})$	push (b)
accept	$(p, \lambda) \in \delta(q, a, Z_0)$	—	
—		$(\mathbf{q}_f, \lambda) \in \delta'(\mathbf{p}, \lambda, Z_0Xq\mathbf{X})$ if $p \in \Delta(q)$	accept

Table 1: Transitions of A_2 for the backward simulation of A_1.

We summarize the transitions for the backward simulation of A_2 in Table 1.

Transitions from (1) and (2) cause A_2 to simulate A_1 step-by-step until the $(k+1)$st pushdown reversal done by A_1 appears. All elements described in (3), (4), (5), and (6) allow A_2 to start a backward simulation of A_1 on the reverse remaining input. To be more precise, the transitions in (3) start the backward simulation of A_2 by undoing the very last step of A_1, i.e., by pushing $Z_0r\mathbf{Z}_0$ onto the pushdown, reading symbol a, and continuing with state $\mathbf{q}$, whenever A_1 has used transition $(p, \lambda) \in \delta(q, a, Z_0)$, for some $p \in Q$, in its last computation step. Then in (4) push moves of A_1 are simulated as pop moves by A_2, always assuming to have a boldface symbol on top of the pushdown. Moreover, transitions specified in (5) simulate pop moves of A_1 by push moves of A_2. Here we have to consider two cases, namely starting a sub-computation which (a) comes back to the same pushdown height or (b) comes not back to the same pushdown height. In the latter case A_2 has to pop a compatible non-boldface symbol together with a boldface symbol in order to decrease the pushdown height. Finally, in (6) the termination of the computation is done, by checking that the pushdown contains a string of the form $Z_0Xr\mathbf{X}$ for some $X \in \Gamma$ and $r \in Q$, and has reached some state in $\Delta(r)$.

Now assume that $w \in N_{k+1}(A_1)$ such that $w = uva$ with

$$(q_0, uva, Z_0) \vdash_{A_1}^* (q_1, va, Z_0X\gamma) \vdash_{A_1} (q_2, va, Z_0\gamma^R X) \vdash_{A_1}^* (q_3, a, Z_0) \vdash_{A_1} (q_4, \lambda, \lambda),$$

where $u, v \in \Sigma^*$, $a \in \Sigma \cup \{\lambda\}$, $X \in \Gamma \cup \{\lambda\}$, $\gamma \in \Gamma^*$, $X = \lambda$ implies $\gamma = \lambda$, and the last pushdown reversal appears at $(q_1, va, Z_0X\gamma) \vdash_{A_1} (q_2, va, Z_0\gamma^R X)$. Thus, by our previous considerations we find the simulation

$$(q_0, uav^R, Z_0) \vdash_{A_2}^* (q_1, av^R, Z_0X\gamma) \vdash_{A_2} (\mathbf{q}_3, v^R, Z_0X\gamma Z_0q_1\mathbf{Z}_0) \vdash_{A_2}^* (\mathbf{q}_2, \lambda, Z_0Xq_1\mathbf{X}) \vdash_{A_2} (\mathbf{q}_f, \lambda, \lambda),$$

and therefore $uav^R = u(va)^R$ belongs to $T_k(A_2)$, since the number of reversals was decreased by one. By similar reasoning, if $u(va)^R \in T_k(A_2)$, then $uva \in N_{k+1}(A_1)$. Since state acceptance and acceptance by empty pushdown coincides for flip-pushdown automata, the claim follows. □

An immediate consequence of Theorem 5 is that *unary* languages, i.e., languages over a singleton letter alphabet, accepted by flip-pushdown automata with a constant number of pushdown reversals are regular, since L equals L_R and unary context-free languages are regular. More formally the statement reads as follows:

Corollary 6 *If L is a unary language accepted by some flip-pushdown automaton with exactly k flips, for some $k \geq 0$, then L is a regular language.* □

Another consequence of the flip-pushdown-input reversal theorem is, that we can separate the hierarchy of pushdown reversal language families for both deterministic and nondeterministic flip-pushdown automata. Another essential ingredients of the proof of the following theorem is a generalization of Ogden's lemma, which is due to Bader and Moura [1] and reads as follows: For any context-free language L, there exists a natural number n, such that for all words z in L, if d positions in z are "distinguished" and e positions are "excluded," with $d > n^{e+1}$, then there are words u, v, w, x, and y such that $z = uvwxy$ and (1) vx contains at least one distinguished position and no excluded positions, (2) if r is the number of distinguished positions and s is the number of excluded positions in vwx, then $r \leq n^{s+1}$, and (3) word uv^iwx^iy is in L for all $i \geq 0$. Now we are ready to prove the flip-pushdown hierarchy theorem.

Theorem 7

$$\mathscr{L}(\text{FDPDA}_k) \subset \mathscr{L}(\text{FDPDA}_{k+1}) \quad \textit{and} \quad \mathscr{L}(\text{FNPDA}_k) \subset \mathscr{L}(\text{FNPDA}_{k+1}),$$

for all $k \geq 0$, where $\mathscr{L}(\text{FDPDA}_k)$ denotes the family of languages accepted by deterministic *flip-pushdown automata with exactly k pushdown reversals.*

Proof. It suffices to prove that $\mathscr{L}(\text{FDPDA}_{k+1}) \setminus \mathscr{L}(\text{FNPDA}_k) \neq \emptyset$. To this end, we define, for $k \geq 1$, the language

$$L_k = \{\, \#w_1\$w_1\#w_2\$w_2\#\ldots\#w_k\$w_k\# \mid w_i \in \{a,b\}^* \text{ for } 1 \leq i \leq k \,\}.$$

Language L_{k+1} is accepted by a (deterministic) flip-pushdown automaton making exactly $k+1$ pushdown reversals. Hence $L_{k+1} \in \mathscr{L}(\text{FDPDA}_{k+1})$.

Next we prove that $L_{k+1} \notin \mathscr{L}(\text{FNPDA}_k)$. Assume to the contrary, that language L_{k+1} is accepted by some flip-pushdown automaton A with exactly k pushdown reversals. Then applying the flip-pushdown input-reversal Theorem 5 exactly k times, results in a context-free language L. Now the idea is

to pump an appropriate word from the context-free language and to undo the flip-pushdown input-reversals, in order to obtain a word that must be in L_{k+1}. If the pumping is done such that no input reversal boundaries in the word are pumped, then the flip-pushdown input-reversals can be undone. Therefore, we need the generalization of Ogden's lemma.

Let n be the constant in the generalization of Ogden's lemma for L and $z = (\#a^{n^{2k+1}}b^{n^{2k+1}}\$a^{n^{2k+1}}b^{n^{2k+1}})^{k+1}\#$ be in L_{k+1}. Consider the word z when transformed into an instance z' of the context-free language L. When applying Theorem 5 to a word wv it becomes wv^R, then we mark the last position of w and the first position of v^R as excluded. Hence, after k applications of Theorem 5 word z' in L contains at most $2k$ excluded positions e. Moreover, since only k flip-pushdown input-reversals are allowed, and $k+1$ blocks $\#a^{n^{2k+1}}b^{n^{2k+1}}\$a^{n^{2k+1}}b^{n^{2k+1}}\#$ exist, due to the pigeon-hole principle there must be at least one block, which was not cut and (its remaining input) reversed. We pick one of these intact blocks in z' and mark all its positions as distinguished. Thus, there are $d = 4 \cdot n^{2k+1} + 2$ distinguished positions in z', with $d > n^{e+1}$.

Now assume that words u, v, w, x, and y satisfy the properties of the generalization of Ogden's lemma. First, we can easily see that if either v or x contains symbols \$ or #, then we obtain a contradiction by considering word uv^2wx^2y, since every word in L (L_{k+1}, respectively) contains exactly $k+1$ symbols \$ and exactly $k+2$ symbols #. Second, we know that because vx contains at least one distinguished position, word v or x lies completely within our chosen intact block $\#a^{n^{2k+1}}b^{n^{2k+1}}\$a^{n^{2k+1}}b^{n^{2k+1}}\#$ (excluding the symbols \$ and #). Then we distinguish three cases:

1. Both words v and x are within the block under consideration. Then the number of excluded positions in vwx equals zero, and hence $|vwx| \leq n$. Then we obtain, that the block under consideration looses its "copy" form in the word $\tilde{z}' = uv^2wx^2y$, i.e., the block we are looking at is not of the form $\#w\$w\#$, for some w, anymore.

2. Word v is within the block under consideration, but x is not. Then the number of excluded positions in vwx is at most $2k$, and hence $|v| \leq n^{2k+1}$. Again, the block under consideration looses its form in the word $\tilde{z}' = uv^2wx^2y$.

3. Word v is not within the block under consideration, but x is. Then a similar reasoning as in the case above applies.

Since we know little about the context-free language L, we now transform our pumped string $\tilde{z}'$ back towards language L_{k+1}, according to Theorem 5. Now the advantage of the excluded positions comes into play. Since we have never pumped on excluded positions, the pushdown reversal move is still valid. Hence, word $\tilde{z}'$ leads us to a word $\tilde{z}$, where the original intact block considered so far is now not of the form $\#w\$w\#$, for some w anymore. Observe, that the

application of Theorem 5 is done exactly in the reverse order as above. This means, that an input reversal appears only at excluded positions (or in-between two excluded ones). In particular, the block considered so far remains untouched during this process. Therefore, word $\tilde{z}$ is not a member of language L_{k+1}. This contradicts our assumption, and thus $L_{k+1} \notin \mathscr{L}(\text{FNPDA}_k)$. □

4 Closure Properties of Flip-Pushdown Languages

In this section we consider closure properties of the family of flip-pushdown languages with a finite number of pushdown reversals. As expected, flip-pushdown languages share many closure properties with the family of context-free languages but there are also some significant differences. For the below given theorem, we need the notion of a rational a-transducer, where we refer to Berstel [3]. Since the proof is an adaption from the context-free case, we omit the proof.

Theorem 8 *The language families $\mathscr{L}(\text{FNPDA}_k)$, for $k \geq 0$, and $\mathscr{L}(\text{FNPDA}_{fin})$ are closed under rational a-transduction. Hence, the families under consideration are full TRIO's, i.e., closed under intersection with regular languages, arbitrary homomorphism, and inverse homomorphism.* □

Next we consider the boolean operations union, intersection, and complement as well as concatenation and Kleene star

Theorem 9 *The language families $\mathscr{L}(\text{FNPDA}_k)$, for $k \geq 0$, and $\mathscr{L}(\text{FNPDA}_{fin})$ are both closed under union, but both families are* not *closed under intersection and complementation. Moreover, $\mathscr{L}(\text{FNPDA}_k)$ is not closed under concatenation, while $\mathscr{L}(\text{FNPDA}_{fin})$ is closed, and both language families are not closed under Kleene star.*

Proof. The closures are immediate. The non-closure results are seen as follows: In case of intersection it suffices to show that the language

$$L = \{\, a^n b^n c^n \mid n \geq 1 \,\},$$

which is the intersection of two context-free languages is not a flip-pushdown language. Assume to the contrary, that language L belongs to $\mathscr{L}(\text{FNPDA}_k)$ for some k. Then we k times apply the flip-pushdown input-reversal Theorem 5 to L obtaining a context-free language. Since we do the input reversal from right-to-left, the block of c's remains intact in all words. Hence a word w in the context-free language reads as $w = uc^n v$, where $|uv|_a = |uv|_b = n$. Then it is an easy exercise to show that this language cannot be context-free using Ogden's lemma. This contradicts our assumption, and thus, language L doesn't belong to $\mathscr{L}(\text{FNPDA}_k)$, for any $k \geq 0$. This shows the non-closure under intersection and complementation due to DeMorgan's law.

Operation	Language family			
	$\mathscr{L}$(CFL)	$\mathscr{L}$(FNPDA$_k$) with $k \geq 1$	$\mathscr{L}$(FNPDA$_{fin}$)	$\mathscr{L}$(FNPDA)
Union	Yes	Yes	Yes	Yes
Intersection	No	No	No	Yes
Complementation	No	No	No	No
Homomorphism	Yes	Yes	Yes	Yes
Inverse hom.	Yes	Yes	Yes	Yes
Intersection with regular sets	Yes	Yes	Yes	Yes
Concatenation	Yes	No	Yes	Yes
Kleene star	Yes	No	No	Yes
Quotient with regular sets	Yes	Yes	Yes	Yes

Table 2: Closure properties of flip-pushdown languages.

For concatention and Kleene star we argue as follows: Let $k \geq 1$. Obviously, language L_{k+1}, defined in the proof of Theorem 7, satisfies

$$L_{k+1} = L_k \cdot \{\, w\$w\# \mid w \in \{a,b\}^* \,\},$$

where both languages on the right-hand side of the equation belong to the family $\mathscr{L}$(FNPDA$_k$). Since by Theorem 7 language $L_{k+1} \in \mathscr{L}$(FNPDA$_{k+1}$) $\setminus$ $\mathscr{L}$(FNPDA$_k$), the non-closure of the language family $\mathscr{L}$(FNPDA$_k$), for $k \geq 1$, immediately follows. Moreover, since $L_{k+1} = \# \cdot \{\, w\$w\# \mid w \in \{a,b\}^* \,\}^{k+1}$, the language $L_\infty = \bigcup_{k=0}^{\infty} L_k$ equals $\# \cdot \{\, w\$w\# \mid w \in \{a,b\}^* \,\}^*$. Thus, if L_∞ would belong to some family $\mathscr{L}$(FNPDA$_k$), for some $k \geq 1$, then language $L_{k+1} = L_\infty \cap \#(\{a,b\}^*\$\{a,b\}^*\#)^{k+1}$ is a member of $\mathscr{L}$(FNPDA$_k$), which contradicts the proof of Theorem 7 due to the closure of this language family under intersection with regular sets and concatenation with a regular set to the left—the latter closure property follows from the closure under TRIO operations. Hence, $\mathscr{L}$(FNPDA$_k$), for $k \geq 1$, and $\mathscr{L}$(FNPDA$_{fin}$) are both not closed under Kleene star. □

Finally, in Table 2 we summarize our results on closure and non-closure properties for flip-pushdown language families. Note, that $\mathscr{L}$(CFL) = $\mathscr{L}$(FNPDA$_0$) is the lowest level in the flip-pushdown hierarchy, while unbounded pushdown reversals are the other end, i.e., $\mathscr{L}$(RE) = $\mathscr{L}$(FNPDA).

5 Computational Complexity of Flip-Pushdown Languages

We consider some computational complexity problems of flip-pushdown languages in more detail. Firstly, we improve the upper bound on the $\mathscr{L}(\text{FNPDA}_k)$ language families given in Theorem 4.

Theorem 10 *$\mathscr{L}(\text{CFL}) \subset \mathscr{L}(\text{FNPDA}_k) \subset \mathscr{L}(\text{CS})$ for $k \geq 1$.*

Proof. The first inclusion is straight forward and its strictness follows from Example 2. The containment of $\mathscr{L}(\text{FNPDA}_k)$ in $\mathscr{L}(\text{CS})$ is seen as follows: Let A be a flip-pushdown automaton making exactly k pushdown reversals. According to Theorem 5 we construct a context-free language L. In order to check membership in $T_k(A)$ a linear bounded automaton guesses a length k sequence of flip-pushdown input-reversals and applies it to the input w to transform it into an instance of the context-free language L. Since context-free membership can be decided by a linear bounded Turing machine, the second inclusion follows. Strictness is seen by Corollary 6, because, e.g., language $\{ a^p \mid p \text{ is prim} \}$ is a context-sensitive language, which is not regular. □

Now the question arises, how complicated is it to decide membership for flip-pushdown languages.

Theorem 11 *The following problems are complete w.r.t. deterministic logspace many-one reductions: (1) The fixed membership problem for k-flip-pushdown languages is* LOG(CFL)*-complete and (2) the general membership problem for k-flip-pushdown automata languages is* P*-complete.*

Proof. In both cases, the hardness results immediately follow from the inclusion $\mathscr{L}(\text{CFL}) \subseteq \mathscr{L}(\text{FNPDA}_k)$ for any $k \geq 0$, and the LOG(CFL)-completeness of fixed membership for context-free languages [18] and the P-completeness for general membership [13]. For the upper bounds we argue as in the proof of Theorem 10. The main difference in the proof is, that we can not guess a length k sequence of flip-pushdown input-reversals. Nevertheless, a deterministic logspace machine can enumerate all possible outcomes of length k sequences of flip-pushdown input-reversals separated by \$ symbols. This suffices to prove the upper bounds—the details are left to the reader. □

The above given theorem can be restated in terms of auxiliary flip-pushdown automata. Here an auxiliary flip-pushdown automaton is a *two-way* flip-pushdown automaton equipped with a space bounded work-tape. Then Theorem 11 shows that auxiliary flip-pushdown automata with exactly k pushdown reversal and a logarithmically space bounded work-tape capture P, and when additionally their time is polynomially bounded the class LOG(CFL) $\subseteq$ P.

6 Conclusions

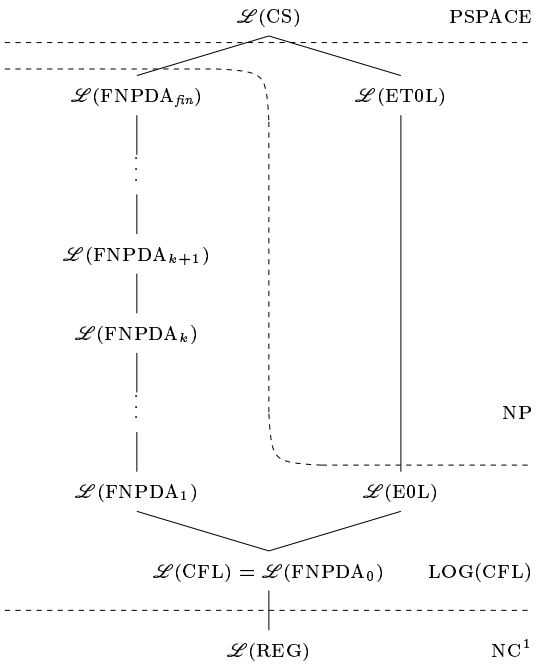


Figure 1: Inclusion structure.

We have investigated flip-pushdown automata with a constant number of pushdown reversals, which were recently introduced by Sarkar [17]. The major contribution of this paper is a positive answer to Sarkar's conjecture on the strictness of the flip-pushdown hierarchy w.r.t. the number of pushdown reversals for both deterministic and nondeterministic flip-pushdown automata. Moreover, we also considered closure and non-closure properties, as well as some computational complexity problems of these language families. In most cases, flip-pushdown languages share similar properties than context-free languages. In Figure 1 the inclusion relations among the classes considered and their computational complexities (completeness) are depicted.

The results presented imply that flip-pushdown languages accepted by flip-pushdown automata with a constant number of pushdown reversals are almost *mildly context-sensitive*, i.e., each language is semi-linear, each language has a deterministic polynomial time solvable membership problem, and the language family contains the following three non-context-free languages: Multiple agreements $L_1 = \{ a^n b^n c^n \mid n \geq 0 \}$, crossed agreements

$$L_2 = \{ a^n b^m c^n d^m \mid n, m \geq 0 \},$$

and duplication $L_3 = \{ ww \mid w \in \{a, b\}^* \}$. Except the non-containment of L_1 all properties of mildly context-sensitive languages are fulfilled.

Nevertheless, several questions for flip-pushdown languages remain unanswered. We mention two of these questions: (1) How does the deterministic and nondeterministic flip-pushdown language hierarchies w.r.t. the number of pushdown reversals relate to each other? (2) What is the relationship between these language families and other well known formal language classes? Especially, the latter question is of some interest, because we were not even able to clear the relationship between the family of flip-pushdown languages and some Lindenmayer families like, e.g. E0L or ET0L languages. For more on Lindenmayer languages we refer to Rozenberg and Salomaa [15]. We conjecture incomparability, but have no proof yet. Obviously, $\{ a^n b^n c^n \mid n \geq 0 \}$ is an E0L language which is not a member of $\mathscr{L}(\text{FNPDA}_{fin})$, but for the other way around we need a language with a semi-linear Parikh mapping which is not an ET0L language.

References

[1] Ch. Bader and A. Moura. A generalization of Ogden's lemma. *Journal of the ACM*, 29(2):404–407, 1982.

[2] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.

[3] J. Berstel. *Transductions and Context-Free Languages*, volume 38 of *Leitfäden der angewandten Mathematik und Mechanik LAMM*. Teubner, 1979.

[4] A. W. Burks, D. W. Warren, and J. B. Wright. An analysis of a logical machine using parenthesis-free notation. *Mathematical Tables and Other Aids to Computation*, 8(46):53–57, April 1954.

[5] N. Chomsky. *Handbook of Mathematic Psychology*, volume 2, chapter Formal Properties of Grammars, pages 323–418. Wiley & Sons, New York, 1962.

[6] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, January 1971.

[7] R. J. Evey. *The Theory and Applications of Pushdown Store Machines*. Ph.D thesis, Harvard University, Massachusetts, May 1963.

[8] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland, Amsterdam, 1975.

[9] S. Ginsburg, S. A. Greibach, and M. A. Harrison. One-way stack automata. *Journal of the ACM*, 14(2):389–418, April 1967.

[10] S. Ginsburg and E. H. Spanier. Finite-turn pushdown automata. *SIAM Journal on Computing*, 4(3):429–453, 1966.

[11] S. A. Greibach. An infinite hierarchy of context-free languages. *Journal of the ACM*, 16(1):91–106, January 1969.

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

[13] N. D. Jones and W. T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3:105–117, 1977.

[14] A. Newell and J. C. Shaw. Programming the logic theory machine. In *Proceedings of the 1957 Western Joint Computer Conference*, pages 230–240, Institute of Radio Engineers, New York, February 1957.

[15] G. Rozenberg and A. Salomaa. *The Mathematical Theory of L Systems*, volume 90 of *Pure and Applied Mathematics.* Academic Press, 1980.

[16] K. Samelson and F. L. Bauer. Sequential formula translation. *Communications of the ACM*, 3(2):76–83, February 1960.

[17] P. Sarkar. Pushdown automaton with the ability to flip its stack. Report TR01-081, Electronic Colloquium on Computational Complexity (ECCC), November 2001.

[18] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, July 1978.