Flip-pushdown automata: nondeterministic ε-moves can be removed*

Pavol Ďuriš and Marek Košta

Comenius University, Bratislava, Slovakia,
duris@dcs.fmph.uniba.sk
kosta1@st.fmph.uniba.sk

Abstract. *Flip-pushdown automaton is pushdown automaton which has ability to flip its pushdown throughout the computation. This model was introduced in [3] by Sarkar. Here we solve in the affirmative the following open problem posed by Holzer and Kutrib in [1]: What is the power of ε-moves for nondeterministic flip-pushdown automata – can they be removed without affecting the computational capacity? (ε denotes the empty word.) Moreover, we prove here that the family of languages recognized by the deterministic variant of the flip-pushdown automata (with k-pushdown reversals) is closed under intersection with regular sets, complement and inverse homomorphism, but it is not closed under union, intersection, (non-erasing) homomorphism, reverse, concatenation and (positive) iteration. Finally, we formulate some new questions and pose new problems.*

1 Introduction

A flip-pushdown automaton, introduced by Sarkar [3], is an ordinary one-way pushdown automaton with the ability to flip its pushdown during the computation. It is known [3] that the flip-pushdown automata without any limit on the number of flips are equally powerful to Turing machines.

Holzer and Kutrib [1, 2] have introduced the so-called "flip-pushdown input-reversal technique" and using it they have shown that $k+1$ pushdown reversals are more powerful than k for deterministic and nondeterministic flip-pushdown automata, and, nondeterminism is more powerful than determinism for flip-pushdown automata with constant number of flips. Another use of this technique led to the fact that languages accepted by flip-pushdown automata using constant number of flips can be accepted by linear bounded automata, so investigated language classes lie between context-free and extended context-sensitive language classes in Chomsky hierarchy. These papers raised also some new questions and pointed to some interesting problems.

* This work was supported by Slovak Grant Agency for Science (VEGA) under contract #1/0726/09 "Algorithmics and Complexity Aspects of Information Processing".

The problem we deal with is the power of ε-moves. Well-known and famous result by Greibach [5] is the so-called Greibach normal form for the context-free grammar. With this result one can easily prove that for every pushdown automaton ε-free pushdown automaton can be constructed. By ε-free pushdown automaton we mean automaton that does not use ε-moves. Our main result parallels this nicely: for every flip-pushdown automaton other ε-free flip-pushdown automaton accepting the same language can be found. Moreover, our proof of this fact is constructive. Of course we will use Greibach normal form quite extensively.

In section 2 we give necessary formal definitions and cite the most important Theorems. Section 3 is the central section of the paper, here we state and prove our main result – that ε-moves can be removed without affecting computational power of the nondeterministic flip-pushdown automaton model. Some closure properties of families of languages described by deterministic flip-pushdown automata are discussed in section 4. Finally, we summarize our results and pose new problems in section 5.

2 Preliminaries

The reader should be familiar with basic facts from formal language theory, where we refer to standard textbook on this subject [4]. We use ε to denote empty word, powerset of a set S by 2^S and length of w by $|w|$.

Definition 1. *A nondeterministic* flip-pushdown automaton (NFPDA) *is a system $A = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite pushdown alphabet, δ is a mapping from $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$ called the transition function, Δ is a mapping from Q to 2^Q, q_0 is the initial state, $Z_0 \in \Gamma$ is a specific 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* of this automaton is a triple (q, w, γ), where $q \in Q$, $w \in \Sigma^*$ and $\gamma \in \Gamma^*$. When flip-pushdown automaton A is in configuration (q, w, γ),

then it is in state q, remaining input is w and stack content is γ. Flip-pushdown automaton has two possibilities how to change configuration: via δ-function or via Δ-function. We write $(q, aw, \gamma Z) \vdash_A (p, w, \gamma\alpha)$ if and only if $(p, \alpha) \in \delta(q, a, Z)$ when δ-step is taken. In second case we write $(q, w, Z_0\gamma) \vdash_A (p, w, Z_0\gamma^R)$ if and only if $p \in \Delta(q)$. Intuitively, the first case is the standard pushdown computational step. Second case, however, is new possibility (in comparison to standard pushdown automaton) and in this case content of the pushdown is reversed (flipped). So the flip-pushdown automaton has two possibilities in any configuration: either apply δ-function or apply Δ-function. Nondeterministic choice is made to choose the next configuration. Note that when A is flipping pushdown content then special symbol (Z_0) has to be at the bottom of the pushdown and is not flipped. The reflexive and transitive closure of relation $\vdash_A$ just defined is denoted by $\vdash_A^*$. The subscript A will be omitted whenever the meaning is clear.

Let $k \geq 0$. Then we define the language accepted by flip-pushdown automaton A by final state and by at most k pushdown reversals as $T_{\leq k}(A) = \{w \in \Sigma^*$ such that $(q_0, w, Z_0) \vdash_A^* (q, \varepsilon, \gamma)$ with at most k pushdown reversals, for some $\gamma \in \Gamma^*$ and $q \in F\}$.

Language accepted by empty pushdown and by at most k pushdown reversals is defined as $N_{\leq k}(A) = \{w \in \Sigma^*$ such that $(q_0, w, Z_0) \vdash_A^* (q, \varepsilon, \varepsilon)$ with at most k pushdown reversals$\}$. When accepting by empty pushdown, set of the final states is usually defined to be empty. Similarly, one can also define $N_{=k}(A)$ or $T_{=k}(A)$ with apparent meaning.

Class of languages accepted by nondeterministic flip-pushdown automata by at most k flips (i.e. those languages L for which there is A, such that $T_{\leq k}(A) = L$) is denoted by $\mathscr{L}(\mathrm{NFPDA}(\leq k))$. Broader class containing all these classes is

$$\bigcup_{i=0}^{\infty} \mathscr{L}(\mathrm{NFPDA}(\leq \mathrm{k})) = \mathscr{L}(\mathrm{NFPDA}(\mathrm{fin})).$$

When dealing with nondeterministic flip-pushdown automata the mode of acceptance does not change the computational power of the model. In deterministic case, however, situation is a little bit more complicated. For more information we refer to [1].

Just for completeness, let us define Greibach normal form.

Definition 2. *Context free grammar $G = (N, T, P, S)$ is in Greibach normal form if all rules are of the form $A \to a\beta$ where $A \in N$, $a \in T$ and $\beta \in N^*$.*

Already mentioned "flip-pushdown input-reversal" technique is very powerful tool we will use extensively in the following.

Theorem 1 ([2]). *Let $k \geq 0$. Then language L is accepted by a nondeterministic flip-pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, F)$ by final state with at most $k+1$ pushdown reversals, i.e. $L = T_{\leq k+1}(M)$,* **if and only if** *language $L_R = \{wv^R$ such that $(q_0, w, Z_0) \vdash_M^* (q_1, \varepsilon, Z_0\gamma)$ with k reversals, $q_2 \in \Delta(q_1)$ and $(q_2, v, Z_0\gamma^R) \vdash_M^* (q_3, \varepsilon, \varepsilon)$ without any reversal$\}$ is accepted by a nondeterministic flip-pushdown automaton M' by final state with at most k pushdown reversals, i.e. $L_R = T_{\leq k}(M')$. The same holds when acceptance by empty pushdown is considered. Moreover, automaton M' can be effectively constructed from automaton M and vice versa.*

This Theorem provides trade-off between number of flips and number of reversals. The number of flips can be reduced by one when correct suffixes of words in L are reversed. But the real power of this Theorem is revealed when it is applied k times on language L accepted by some k-flip pushdown automaton. Then we get context-free language L' that has strong connection with language L. Simply speaking, we get words in L' first by reversing some correct suffixes of words in L and then applying this process inductively.

3 Main result

In this section we will prove that for every k-flip pushdown automaton M, a k-flip pushdown automaton $\bar{M}$ can be constructed in such a way that $\bar{M}$ does **not** use ε-steps. This means that $\delta(q, \varepsilon, Z) = \emptyset$ holds for every q, Z and δ-function of $\bar{M}$ and $\bar{M}$ accepts the same language as M.

3.1 Main idea

Suppose we have a language L that is accepted by some k-flip pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, \emptyset)$ by empty pushdown. In previously defined notation this means that $L = N_{=k}(M)$. Let L_k be the language containing the words of the form $v_0\$v_1\$\ldots\$v_k\$$ such that $w = v_0v_1\ldots v_k \in L$, and for w there is an accepting computation of M on w during which M makes a flip of the pushdown content at the end of each v_i for $0 \leq i \leq k-1$. New automaton M' accepting L_k can be easily constructed: just simulate M and after each flip symbol \$ must be read or computation will stuck. Symbol \$ must be also read at the end of the word.

So we have marked places in words accepted by M where flip of pushdown occurred. We will also construct languages $L_{k-1}, \ldots, L_0$ in this way: language L_i is constructed from language L_{i+1} $(0 \leq i \leq k-1)$ by applying "flip-pushdown input-reversal technique",

see Theorem 1. So L_i is accepted by some i-flip pushdown automaton which can be constructed according to Theorem 1. These facts are formally described in two following Lemmas.

Lemma 1. *Let L_k be the previously defined language. Let $L_k, \ldots, L_0$ be the languages, where the language L_{i-1} is obtained from language L_i by one application of Theorem 1 for $1 \leq i \leq k$. Then word $w =$*

$$v_0\$v_1\$\ldots\$v_{i-1}\$v_i\$v_{i+1}\$\ldots\$v_{k-1}\$v_k\$$$

is in language L_i if and only if word $w' =$

$$v_0\$v_1\$\ldots\$v_{i-1}\$v_k^R\$v_{k-1}^R\$\ldots\$v_{i+1}^R\$v_i^R\$$$

is in language L_{i-1} for $1 \leq i \leq k$.

Proof. Quite obvious. The last flip in computation on word w occurs after subword v_i. Because symbol \$ is new and also from formulation of Theorem 1 stated correspondence between words from L_i and L_{i-1} holds. □

Lemma 2. *When $k = 2l$ then word $w_0 =$*

$$v_0\$v_2\$\ldots\$v_{2l}\$v_{2l-1}^R\$\ldots\$v_3^R\$v_1^R\$$$

belongs to language L_0 if and only if word $w_k = v_0\$v_1\$ \ldots \$v_k\$$ belongs to L_k. In case $k = 2l+1$ words in L_0 have form

$$v_0\$v_2\$\ldots\$v_{2l}\$v_{2l+1}^R\$v_{2l-1}^R\$\ldots\$v_3^R\$v_1^R\$$$

Proof. Just apply Lemma 1 inductively k times. □

In the following we assume for simplicity that $k = 2l$, case when $k = 2k + 1$ is proved similarly. We will exploit correspondence between words in L_0 and those in L_k stated in Lemma 2. Language L_0 is by definition context-free so there is Greibach normal form context free grammar generating it. We want to simulate leftmost derivations in this grammar (and others constructed from this one) with flip-pushdown automaton.

Let us have context-free grammar $G_0 = (N_0, \Sigma \cup \{\$\}, P_0, S_0)$, such that $L(G_0) = L_0$. Leftmost derivation of word $w_0 \in L_0$ in grammar G_0 looks like this:

$$\begin{aligned}
S_0 &\Rightarrow^* v_0\alpha_0 \Rightarrow^* \\
&\Rightarrow^* v_0\$v_2\alpha_2 \Rightarrow^* v_0\$v_2\$v_4\alpha_4 \Rightarrow^* \\
&\Rightarrow^* v_0\$v_2\$v_4\ldots\$v_{2l}\alpha_{2l} \Rightarrow^* \\
&\Rightarrow^* v_0\$v_2\$v_4\ldots\$v_{2l}\$v_{2l-1}^R\alpha_{2l-1} \Rightarrow^* \\
&\Rightarrow^* v_0\$v_2\$v_4\ldots\$v_{2l}\$v_{2l-1}^R\$v_{2l-3}^R\alpha_{2l-3} \Rightarrow^* \\
&\Rightarrow^* \cdots \Rightarrow^* \\
&\Rightarrow^* v_0\$v_2\$v_4\ldots\$v_{2l}\$v_{2l-1}^R\$\ldots\$v_3^R\$v_1^R\$ = w_0
\end{aligned} \tag{1}$$

Here α_i is non-empty string of nonterminals, other symbols are terminal.

If we were able to reverse and regulate this derivation in right way we could obtain "derivation" of word w_k instead of w_0 (w_0 and w_k are from Lemma 2). This means that if we could after (1) reverse string α_0 and also reverse terminal words derived from these nonterminals (i.e. derive in grammar G_1 obtained from G_0 by reversing right-hand side of each production), we could obtain something like this:

$$\begin{aligned}
S_0 \Rightarrow^*_{G_0} v_0\alpha_0 &\quad \text{reverse nonterminal string} \\
v_0\alpha_0^R &\quad \text{now continue in } G_1 \\
\Rightarrow^*_{G_1} v_0\$v_1\beta &\quad \beta \text{ is nonterminal string in } G_1
\end{aligned}$$

The main idea of the previous process is this. We want to simulate the leftmost derivation (in grammar G_0) by pushdown automaton, storing nonterminal string on the stack. When reversal takes place, automaton will simulate leftmost derivation in G_1 (we will call it reverse grammar) and so on. By continuing this simulation in suitable grammars making reversals along the way we can obtain valid accepting computation on word w_k. Deeper analysis will follow but first we need some notation based on (1).

$$\begin{aligned}
\alpha_0 &= A_1\ldots A_n \\
A_i &\Rightarrow^*_{G_0} x_i \in T^* \\
u &= x_1\ldots x_n \\
u &= \$v_2\$v_4\ldots\$v_{2l}\$v_{2l-1}^R\$\ldots\$v_3^R\$v_1^R\$ \\
u^R &= \$v_1\$v_3\$\ldots\$v_{2l-1}\$v_{2l}^R\$\ldots\$v_4^R\$v_2^R\$
\end{aligned} \tag{2}$$

We will use the concept of reverse grammar extensively in the following so here is formal definition.

Definition 3. *We say that context-free grammar $H = (N_H, T, P_H, S_H)$ is reverse grammar to context-free grammar $I = (N_I, T, P_I, S_I)$ if the following conditions hold:*

1. *H is in Greibach normal form*
2. *$N_H \cap N_I = N_I$*
3. *$(\forall \xi \in N_I)(\forall z \in T^*)\ \xi \Rightarrow^*_H z^R \iff \xi \Rightarrow^*_I z$*

Lemma 3. *Assume that H is reverse grammar to I. Then for every α (non-empty nonterminal string of grammar I) and v (terminal string) we have:*

$$\alpha \Rightarrow^*_I v \quad \text{if and only if} \quad \alpha^R \Rightarrow^*_H v^R \tag{3}$$

Proof. Easy induction on the length of α. From definition 3 we see that the statement holds when $|\alpha| = 1$. Assume that the statement hold for all α of length k. Now let $\alpha A \Rightarrow^*_I v_\alpha v_A$. By induction hypothesis and definition 3 this is equivalent to $A\alpha^R \Rightarrow^*_H v_A^R v_\alpha^R$. So the statement of the Lemma follows by induction. □

We want to construct flip-pushdown automaton $\widetilde{M}$ without ε-moves, which will simulate leftmost derivations in $G_0, \ldots, G_k$ (here G_{i+1} is reverse to G_i) as follows. On input w_k, $\widetilde{M}$ is going to verify if w_k belongs to L_k. Initial configuration of $\widetilde{M}$ is (q_0, w_k, S_0). After simulating the leftmost derivation (in G_0) the configuration will be $(q, \$v_1\$ \ldots \$v_k\$, A_n \ldots A_1)$ [1]. Flip now takes place and $\widetilde{M}$ will continue simulating G_1 – reverse grammar to grammar G_0. From Lemma 3 and (2) we have:

$$\begin{aligned} A_i &\Rightarrow^*_{G_1} x_i^R \\ u^R &= x_n^R \ldots x_1^R \\ u^R &= \$v_1\$v_3\$ \ldots \$v_{2l-1}\$v_{2l}^R\$ \ldots \$v_4^R\$v_2^R\$ \quad (4) \end{aligned}$$

From these facts it is apparent that

$$A_n \ldots A_1 \Rightarrow^*_{G_1} \$v_1 B_1 B_2 \ldots B_m$$

where B_i are nonterminals of grammar G_1. So $\widetilde{M}$ will continue from configuration $(p, \$v_1\$ \ldots \$v_k\$, A_1 \ldots A_n)$, simulating G_1, to configuration $(r, \$v_2\$ \ldots \$v_k\$, B_m \ldots B_1)$.

We believe that the main idea is now clear. The most important point here is that we are able to simulate leftmost derivation in G_0 by standard δ-steps of flip-pushdown automaton with sentential form on the stack. Then flip can be done and we can continue by simulating derivation in G_1 which is reverse grammar to G_0. This process of simulating and flipping will continue up to grammar G_k. Crucial here is that all these grammars used along the way are in Greibach normal form so $\widetilde{M}$ will not use ε-moves when simulating these grammars.

3.2 Proof

So we have M' accepting language L_k (see the beginning of subsection 3.1) by empty pushdown and by exactly k flips. By Lemma 2, L_0 can be generated by some context-free grammar $G_0 = (N_0, \Sigma, P_0, S_0)$. We want to construct automaton $\widetilde{M}$ without ε-moves which will accept L_k by final state and by exactly k reversals. However, this acceptation mode will be more special than this: from construction it will be apparent that $\widetilde{M}$ will accept only in configuration in which final state is reached, pushdown is empty and exactly k flips took place.

According to previously stated ideas, automaton $\widetilde{M}$ will simulate leftmost derivations consequently in grammar G_0 then G_1 and so on up to G_k. Now we sketch how to construct these grammars.

[1] we remind that the top of the stack is on the right

Construction of Reverse Grammars. Initially, we have grammar G_0. We show construction of G_i when G_{i-1} is already constructed. So assume that we have grammar $G_{i-1} = (N_i, \Sigma, P_i, S_i)$. We want to construct grammar G_i which satisfy all conditions in definition 3. From these the most important is the third condition. Consider language $L_A = \{u^R \mid A \Rightarrow^*_{G_{i-1}} u\}$ for every nonterminal A of G_{i-1}. Greibach normal form grammar G_A for this language can be constructed by well-known construction. From these grammars one can get grammar G_i satisfying all three conditions in definition 3 easily – just by union of rules and nonterminal. Some renaming, however, must be done to ensure the second condition of definition 3.

Technical Consideration. In definition 1 we wanted to be coherent with established formalism. But this definition is a little bit cumbersome. Our idea is to represent sentential form on the stack and do computation with this representation (doing flips and simulate grammars G_i). So it is inconvenient to bother with special Z_0 symbol and convention that it is always at the bottom and never flipped. For the sake of simplicity, we will formally do our construction without this restriction. But we must note that this in no way spoils the result or prevents construction in accordance with definition 1. This construction is straightforward but technical – some simulation of bottom of pushdown, some information stored in states, new marked pushdown symbols and so on. The most important point is that all these things can be done *without* any use of ε-steps. So in the following construction Δ-steps cause reversal of the whole pushdown word and no special symbol must be at the bottom of the stack. So sentential form is represented "as is" on the stack.

Let $\widetilde{M} = (Q, \Sigma \cup \{\$\}, \Gamma, \delta, \Delta, q_0, S_0, \{q_F\})$, where

$Q = \{q_0, q_1, \ldots, q_k\} \cup \{\overline{q_1}, \ldots, \overline{q_k}\} \cup \{q_F\}$,

Σ is alphabet of M' and \$ is new symbol,

$\Gamma = \{A \mid$ where A is nontermial of G_i, $0 \le i \le k\}$,

$\Delta(q_i) = \{\overline{q_{i+1}}\} \wedge 0 \le i \le k-1$.

Standard pushdown moves (i.e. δ-steps) will ensure the simulation of the leftmost derivation in grammars G_i. This can be seen from next lines:

$$\begin{aligned} \delta(q_0, a, S_0) \ni (q_0, \beta^R) &\quad\Leftrightarrow\quad a \in \Sigma \ \wedge\ S_0 \to a\beta \in P_0 \\ \delta(\overline{q_i}, \$, Z) \ni (q_i, \beta^R) &\quad\Leftrightarrow\quad Z \to \$\beta \in P_i \\ \delta(q_i, a, Z) \ni (q_i, \beta^R) &\quad\Leftrightarrow\quad a \in \Sigma \ \wedge\ Z \to a\beta \in P_i \\ \delta(q_k, \$, Z) \ni (q_F, \varepsilon) &\quad\Leftrightarrow\quad Z \to \$ \in P_k \end{aligned}$$

For every $i \in \{1, \ldots, k\}$ we have according lines in the δ-function. No other lines except of these are in δ-function (i.e. for all other combinations of state, input and pushdown symbol δ-function is defined to be empty set).

First line is used for start of the simulation in G_0. Next two lines ensure that after each flip (i.e. Δ-step) symbol \$ is read from the input and simulation of grammar G_i takes place. Last line ensures accepting end of the whole computation in successful cases. Essence of simulation and derivation is described in the following Lemma. This key Lemma is obvious right from the construction of $\widetilde{M}$. Simulation of context free grammar within pushdown automaton is idea that was mentioned already a few times but we refer unfamiliar reader (once more) to [4].

Lemma 4. *Let $\widetilde{M}$ be constructed from grammars $G_0, \ldots, G_k$ in the way just described. Then the following equivalence holds for every $v \in \Sigma^*$ and $\alpha, \beta \in \Gamma^*$.*

$$(q_0, v, S_0) \vdash^* (q_0, \varepsilon, \alpha^R) \Leftrightarrow S_0 \Rightarrow^*_{G_0} v\alpha$$
$$(\overline{q_i}, \$v, \alpha^R) \vdash^* (q_i, \varepsilon, \beta^R) \Leftrightarrow \alpha \Rightarrow^*_{G_i} \$v\beta$$
$$(\overline{q_k}, \$v\$, \alpha^R) \vdash^* (q_F, \varepsilon, \varepsilon) \Leftrightarrow \alpha \Rightarrow^*_{G_k} \$v\$$$

Here $i \in \{1, \ldots, k-1\}$ and all $\vdash$ transitions are by means of δ-function (i.e. without any pushdown reversal).

Proof. Is really straightforward because δ-function contains all transitions needed for proper simulation of the leftmost derivation in grammar G_i. Formally, everything would be done by induction on the length of the leftmost derivation (or number of $\vdash$ steps). □

Theorem 2. *Automaton $\widetilde{M}$ does not use ε-moves and accepts language L_k by final state and by exactly k flips.*

Proof. It is immediate from construction that $\widetilde{M}$ does *not* use ε-moves because all grammars $\widetilde{M}$ simulates are in Greibach normal form. So every production is of the form $N_i \to \Sigma N_i^*$ for some i. Simulating this production $\widetilde{M}$ reads symbol from input.

Now we have to prove two inclusions. We prove the more apparent one first. We assume for simplicity that $k = 2l$. The proof is similar for $k = 2l + 1$.

$\boxed{L_k \subseteq T_{=k}(\widetilde{M})}$:

Assume that w_k is in L_k. By Lemma 2 we know that w_0 belongs to L_0, generated by grammar G_0. This means that there exists some leftmost derivation of w_0 in grammar G_0. According to previously defined notation we have:

$$S_0 \Rightarrow^*_{G_0} v_0\$v_2\$\ldots\$v_{2l}\$v_{2l-1}^R\$\ldots\$v_1^R\$ = w_0$$
$$S_0 \Rightarrow^*_{G_0} v_0\alpha \quad (5)$$
$$\alpha \Rightarrow^*_{G_0} \$v_2\$\ldots\$v_{2l}\$v_{2l-1}^R\$\ldots\$v_1^R\$ \quad (6)$$

Following facts also hold.

$$(q_0, v_0, S_0) \vdash^* (q_0, \varepsilon, \alpha^R) \quad (7)$$
$$(q_0, \varepsilon, \alpha^R) \vdash (\overline{q_1}, \varepsilon, \alpha) \quad (8)$$
$$\alpha^R \Rightarrow^*_{G_1} \$v_1\beta \quad (9)$$
$$\beta \Rightarrow^*_{G_1} \$v_3\ldots\$v_{2l-1}\$v_{2l}^R\$\ldots\$v_2^R\$ \quad (10)$$

Reasons for this are as follows. Fact (7) is due to (5) and Lemma 4. Fact (8) is just Δ-step in automaton $\widetilde{M}$ – construction of $\widetilde{M}$ allows this transition. Facts (9) and (10) are implied by (6) and Lemma 3.

By (10) and Lemma 3, $\beta^R \Rightarrow^*_{G_2} \$v_2\gamma$ and $\gamma \Rightarrow^*_{G_2} \$v_4\$\ldots\$v_{2l}\$v_{2l-1}^R\$\ldots\$v_3^R\$$ for some nonterminal word γ. This means together with (5), (9) and Lemma 4 that $(q_0, v_0, S_0) \vdash^* (q_0, \varepsilon, \alpha^R)$, $(\overline{q_1}, \$v_1, \alpha) \vdash^* (q_1, \varepsilon, \beta^R)$ and $(\overline{q_2}, \$v_2, \beta) \vdash^* (q_2, \varepsilon, \gamma^R)$. Since $\widetilde{M}$ can flip the stack just before reading \$ (see construction of $\widetilde{M}$ above), then we get $(q_0, v_0\$v_1\$v_2, S_0) \vdash^* (q_2, \varepsilon, \gamma^R)$. Generalizing this approach – using the idea that the derivation of w_0 in G_0 yields some (sub)derivations of v_0 in G_0, $\$v_1$ in G_1, $\$v_2$ in G_2 and so on up to $\$v_k$ in G_k (see (5), (9), see above, ...), we can construct (by Lemma 4) an accepting computation of $\widetilde{M}$ on $w_k = v_0\$v_1\$v_2\ldots\$v_k\$$. Hence $L_k \subseteq T_{=k}(\widetilde{M})$ (recall the assumption $w_k \in L_k$, see above).

$\boxed{T_{=k}(\widetilde{M}) \subseteq L_k}$:

Lemmas 4 and 3 will be now used in reverse direction: from existence of accepting computation we will conclude that some derivations in grammars $G_k, G_{k-1}, \ldots, G_0$ exist. We will also use induction.

Assume that $w_k = v_0\$v_1\ldots\$v_{k-1}\$v_k\$$ belongs to $T_{=k}(\widetilde{M})$. From construction of $\widetilde{M}$ we know that w_k *has to* be of this form – after each flip symbol \$ must be read, exactly k flips have to be done and this kind of argumentation (based only on construction of $\widetilde{M}$) imply these statements:

$$(q_0, v_0\$v_1\$\ldots\$v_k\$, S_0) \vdash^* (q_0, u_1, \alpha_1^R)$$
$$(q_i, u_{i+1}, \alpha_{i+1}^R) \vdash (\overline{q_{i+1}}, u_{i+1}, \alpha_{i+1}) \quad (11)$$
$$(\overline{q_i}, u_i, \alpha_i) \vdash^* (q_i, u_{i+1}, \alpha_{i+1}^R) \quad (12)$$
$$(\overline{q_k}, u_k, \alpha_k) \vdash^* (q_F, \varepsilon, \varepsilon) \quad (13)$$

Here $u_i = \$v_i\$\ \ldots\ \$v_k\$$, $1 \le i \le k$. This is just computation of $\widetilde{M}$ written in convenient way, assuming that $w_k \in T_{=k}(\widetilde{M})$. Fact (11) is just pushdown reversal, (12) is part of computation between i-th and $i+1$-th flip and (13) is the final part of accepting computation. We want to show that some derivation of word $w_0 = v_0\ \$v_2\$\ \ldots\ \$v_k\$\ v_{k-1}^R\$\ \ldots\ \$v_1^R\$$ in grammar G_0 exists, i.e. w_0 is in L_0. This and Lemma 2 will imply that w_k belongs to L_k.

From assumption (13) and Lemma 4 we can infer that

$$\alpha_k^R \Rightarrow_{G_k}^* u_k = \$v_k\$ \tag{14}$$

Fact (12) for $i = k-1$ and second equivalence of Lemma 4 lead to

$$\alpha_{k-1}^R \Rightarrow_{G_{k-1}}^* \$v_{k-1}\alpha_k \tag{15}$$

Facts (14) and (15) with Lemma 3 imply that

$$\alpha_{k-1}^R \Rightarrow_{G_{k-1}}^* \$v_{k-1}\$v_k^R\$ \tag{16}$$

Ideas contained in these statments can be transformed into formal inductive proof of the fact that in grammar G_0 word w_0 can be derived. Main idea is that from facts like (14) and (15) we can infer (16). So from existence of particular part of computation one can infer that corresponding partial derivation exists. These partial derivations are then joined inductively to form derivation of word w_0 in grammar G_0. Some "descent" from grammar G_k to grammar G_{k-1} is also important in the whole argument. We will not elaborate statements in fully formal way but provide one specific example to demonstrate these ideas.

Example for $k = 2$. In this case $w_2 = v_0\$v_1\$v_2\$$. Parts of accepting computation look like this:

$$\begin{aligned}(q_0, v_0, S_0) &\vdash^* (q_0, \varepsilon, \alpha_1^R)\\ (\overline{q_1}, \$v_1\$, \alpha_1) &\vdash^* (q_1, \varepsilon, \alpha_2^R)\\ (\overline{q_2}, \$v_2\$, \alpha_2) &\vdash^* (q_F, \varepsilon, \varepsilon)\end{aligned}$$

These parts of computation with Lemma 4 give us:

$$\begin{aligned}S_0 &\Rightarrow_{G_0}^* v_0\alpha_1\\ \alpha_1^R &\Rightarrow_{G_1}^* \$v_1\alpha_2\\ \alpha_2^R &\Rightarrow_{G_2}^* \$v_2\$\end{aligned}$$

These facts with Lemma 3 lead to:

$$\begin{aligned}\alpha_2 &\Rightarrow_{G_1}^* \$v_2^R\$\\ \alpha_1^R &\Rightarrow_{G_1}^* \$v_1\$v_2^R\$\\ \alpha_1 &\Rightarrow_{G_0}^* \$v_2\$v_1^R\$\\ S_0 &\Rightarrow_{G_0}^* v_0\$v_2\$v_1^R\$ = w_0\end{aligned}$$

□

So now we have automaton $\widetilde{M}$, accepting language L_k which does not use ε-moves. The final step in our proof is to delete marks (i.e. symbols \$) from L_k, thus obtaining language L. Deletion of marks must be done without using ε-steps. This is easy in principle, but technical. One symbol can be easily deleted without using ε-steps. Idea is quite easy: join two steps into one. This is standard well-known construction. Constant number of symbols can be deleted by iterating this construction. So by applying this construction on automaton $\widetilde{M}$ repeatedly k-times we get the desired automaton $\bar{M}$ that accepts language L.

3.3 Summary of the proof

Our proof consists of five main steps, we review them here.

1. Language L is given, accepted by automaton M, ε-steps are allowed.
2. Marks are inserted into L to denote that flip took place in computation. This gives us language L_k accepted by M'.
3. By means of flip-pushdown input-reversal technique languages $L_{k-1}, \ldots, L_1, L_0$ are constructed. Correspondence between words in L_k and L_0 is proved.
4. Automaton $\widetilde{M}$ is constructed. It simulates suitable Greibach normal form grammars. $\widetilde{M}$ doesn't use ε-steps and accepts language L_k.
5. Marks are deleted from language L_k yielding language L accepted by $\bar{M}$. Well-known trick of joining two steps into one is used to achieve this.

4 Closure properties

In this section we establish some closure properties of deterministic flip-pushdown automata. A deterministic flip-pushdown automaton (DFPDA) is nondeterministic flip-pushdown automaton which has at most one choice of action for any possible configuration. This means that there must never be a choice of using an input symbol, using ε-step or Δ-step. Formally, a flip-pushdown automaton $A = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, F)$ is *deterministic* if these conditions are satisfied for all $a \in \Sigma \cup \{\varepsilon\}, q \in Q, Z \in \Gamma$

1. $|\delta(q, a, Z)| \leq 1$
2. If $\delta(q, \varepsilon, Z) \neq \emptyset$ then $\delta(q, a, Z) = \emptyset$.
3. $|\Delta(q)| \leq 1$
4. If $\Delta(q) \neq \emptyset$ then $\delta(q, a, Z) = \emptyset$.

For deterministic flip-pushdown automaton we can naturally define acceptance by final state, empty pushdown, by exactly k flips or at most k flips. We will use notation as in previous parts, i.e. $T_{\leq k}(A)$ or $N_{=k}(A)$ with obvious meaning. Classes of languages are defined without confusion. For example, $\mathscr{L}(\mathrm{DFPDA}(\leq k))$ is class of languages accepted by deterministic flip-pushdown automata by final state and by at most k pushdown reversals. Note that some nuances are associated with deterministic variant of the model. Acceptance by final state is not equivalent to acceptance by empty pushdown for example. We refer to [1] where results concerning these questions can be found. We will explicitly state which mode of acceptance we are considering.

The following result will be of great importance for us.

Theorem 3 ([2]). *Language*

$$L_{abc} = \{a^n b^n c^n \mid n \geq 1\}$$

is not accepted by any nondeterministic flip-pushdown automaton. This means that $L_{abc} \notin \mathscr{L}(\mathrm{NFPDA(fin)})$.

Separation of hierachy is also one of the interesting results which was achieved.

Theorem 4 ([2, 1]). *Let* k *be natural number greater than zero. Consider language*

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

Then

$$J_k \in \mathscr{L}(\mathrm{DFPDA(= k)}) \subset \subset \mathscr{L}(\mathrm{DFPDA(\leq k)}) \subset \mathscr{L}(\mathrm{NFPDA(= k)}),$$
$$J_k \notin \mathscr{L}(\mathrm{NFPDA(= k-1)}).$$

Our next result is quite interesting. In principle it says that one nondeterministic step of standard pushdown automaton cannot be simulated by any finite number of pushdown reversals with deterministic flip-pushdown automaton.

Theorem 5. $\mathscr{L}(\mathrm{DFPDA}(\leq k))$ *is not closed under union for any* $k \geq 0$.

Proof. Consider languages

$$L_1 = \{a^n b^n \mid n \geq 1\} \in \mathscr{L}(\mathrm{DFPDA(\leq k)}),$$
$$L_2 = \{a^n b^{2n} \mid n \geq 1\} \in \mathscr{L}(\mathrm{DFPDA(\leq k)}).$$

These are simple deterministic pushdown languages. We will prove by contradiction that $L_1 \cup L_2$ does not belong to $\mathscr{L}(\mathrm{DFPDA}(\leq k))$ for any k. Suppose that this is not the case. So deterministic flip-pushdown automaton $A = (Q, \{a,b\}, \Gamma, \delta, \Delta, q_0, Z_0, F)$ exists such that $T_{\leq k}(A) = L_1 \cup L_2$. From A we will construct another nondeterministic flip-pushdown automaton B which will accept the language $T_{\leq 2k}(B)$ such that $T_{\leq 2k}(B) \cap a^+b^+c^+ = L_{abc}$. This contradiction with Theorem 3 will conclude the proof.

We construct $B = (Q \cup \overline{Q}, \{a,b,c\}, \Gamma, \delta_B, \Delta_B, q_0, Z_0, F \cup \overline{F})$ from A as follows.

$$\overline{Q} = \{\overline{q} \mid q \in Q\}$$
$$\overline{F} = \{\overline{q} \mid q \in F\}$$
$$\delta_B(q,a,Z) \ni (p,\beta) \Leftrightarrow \delta(q,a,Z) \ni (p,\beta)$$
$$\delta_B(q,b,Z) \ni (p,\beta) \Leftrightarrow \delta(q,b,Z) \ni (p,\beta)$$
$$\delta_B(\overline{q},a,Z) \ni (\overline{p},\beta) \Leftrightarrow \delta(q,a,Z) \ni (p,\beta)$$
$$\delta_B(\overline{q},c,Z) \ni (\overline{p},\beta) \Leftrightarrow \delta(q,b,Z) \ni (p,\beta)$$
$$\delta_B(q,\varepsilon,Z) \ni (\overline{q},Z) \Leftrightarrow q \in F$$
$$\Delta_B(q) = \Delta(q)$$
$$\Delta_B(\overline{q}) = \{\overline{p} \mid p \in \Delta(q)\}$$

Informally, our construction did this. Take two copies of automaton A, in second copy use symbol c instead of symbol b when reading input. Then there is an ε-step from accepting states of first copy into corresponding accepting states of second copy. Initial state is in first copy (identical with that of A).

How does the language $T_{\leq k}(B)$ look like? Automaton B could end accepting computation in some state q_F in first copy, or in $\overline{q_F}$ in second copy. First case is not really interesting: from construction of B it is obvious that words which caused this computation are exactly those in $T_{\leq k}(A) = L_1 \cup L_2$.

Second case is the interesting one. B ended in $\overline{q_F}$, so one ε-step from first to second copy must have taken place. Consider time when B did this step. By construction, B is at this time in a configuration in which A accepts, so input word already read at this time must be one of $a^n b^n$ or $a^n b^{2n}$ for some n. When the input word already read is $a^n b^n$, then B is capable to accept $a^n b^n c^n$, since $a^n b^n$ is a prefix of $a^n b^{2n}$ accepted by A, and after ε-step, B works as the second copy of A, where the symbol b is replaced by c. Similarly, when the input word already read is $a^n b^{2n}$ then B cannot accept any word of the form $a^n b^{2n} v$ for any $v \neq \varepsilon$, since otherwise A has to accept $a^n b^{2n} v'$, where $v' \neq \varepsilon$ is obtained from v by replacing symbols c by b, but A (recognizing $L_1 \cup L_2$) cannot accept any such word $a^n b^{2n} v'$.

Automaton A accepts $L_1 \cup L_2$ by at most k-flips so B accepts mentioned words by at most $2k$-flips (in each copy at most k flips took place).

Consequently, we have that

$$T_{\leq 2k}(B) = \{a^n b^n \mid n \geq 1\} \cup \cup \{a^n b^{2n} \mid n \geq 1\} \cup \{a^n b^n c^n \mid n \geq 1\}.$$

Intersection with regular language $a^+b^+c^+$ leads to language L_{abc}. This is the desired contradiction with Theorem 3, since one can easily observe that $\mathscr{L}(\mathrm{NFPDA}(\leq l))$ is closed under intersection with regular set for every l. □

Proposition 1. $\mathscr{L}(\mathrm{DFPDA}(\leq k))$ *is closed under intersection with regular set, complement and inverse homomorphism for every* $k \geq 0$.

Proof. Trivial simulation of finite automaton will give us closure under intersection with regular set.

Closure under inverse homomorphism is done via standard construction. Automaton reads input, on this homomorphism is applied and simulation takes place from buffer.

Complement is more peculiar. We give here just sketch of the proof. Idea is the same as with standard deterministic pushdown automata [4]. First we must ensure that automaton reads its whole input.

Here there are two main problems – automaton stucks during computation because of undefined transition or in infinite sequence of ε-moves. The first problem is handled easily. Problem of looping is the main part of the proof. Crucial here is that deterministic flip-pushdown automaton uses only constant number of flips. After each flip detection of looping on empty input can be done by mentioned technique. From these two facts it can be inferred that detection of infinite loops can be done effectively.

When this normal form is achieved, construction is quite simple. We repeatedly refer reader to [4]. □

Proposition 2. *$\mathscr{L}(\mathrm{DFPDA}(\leq k))$ is not closed under intersection, homomorphism (even non-erasing), reverse, contatenation and (positive) iteration for any $k \geq 0$.*

Proof. For intersection it suffices to use De Morgan's laws, Theorem 5 and Proposition 1.

For concatenation and iteration it suffices to consider language J_k and Theorem 4.

For homomorphism just take language

$$L = \{ca^{n-1}b^n \mid n \geq 1\} \cup \\ \cup \{da^{n-1}b^{2n} \mid n \geq 1\}$$

and homomorphism $h(a) = h(c) = h(d) = a$, $h(b) = b$. Apply h and use Theorem 5.

For reverse just apply construction from Theorem 5 on language L^R where

$$L = d\{b^{2n}a^n \mid n \geq 1\} \cup \{b^n a^n \mid n \geq 1\}.$$

This construction will give us nondeterministic flip-pushdown automaton accepting language

$$L' = \{a^n b^n \mid n \geq 1\} \cup \\ \cup \{a^n b^{2n} \mid n \geq 1\}d \cup \{a^n b^n c^n d \mid n \geq 1\}.$$

But this also leads to contradiction with Theorem 3. □

5 Conclusions

We discussed flip-pushdown automaton model. Our main contribution to this area of theoretical research is solution of problem of ε-moves. We proved that ε-moves can be removed and normal form effectively achieved. Some (non)closure properties of deterministic model were also investigated. Our results answer some open questions formulated in [1]. Finally, we list some interesting questions which wait for answer.

1. Is $\mathscr{L}(\mathrm{DFPDA}(\leq k))$ closed under right quotient with regular set? From our results it can be easily shown that this is not the case when left quotient is considered but we were unable to investigate right quotient deeper. We tried to generalize the idea of predicting machine from deterministic pushdown automata, but without success.
2. What do we get when we apply flip-pushdown input-reversal technique on deterministic k-flip pushdown automaton? Do we get deterministic $k-1$-flip pushdown language? Or does there exist some other similar technique which can be used for deterministic variant?
3. Can number of states in nondeterministic k-flip pushdown automaton be bounded without affecting the computational power? In pushdown automata normal form with one state can be achieved. Our construction from section 3 yields normal form with at most $O(k)$ states, where k is number of flips. Is this boundary tight?
4. What properties "pushdown language" L_p has?

   $$L_p = \{\alpha \in \Gamma^* \mid (q_0, w, \alpha) \vdash^* (q, \varepsilon, \varepsilon)\}$$

   for some $w \in \Sigma^*$ and $q \in Q$. This language of words which can be erased from pushdown by some input word is regular when one considers standard pushdown automaton. What we can say about it here?
5. Which properties are decidable? This is only interesting when deterministic variant is considered because for nondeterministic variant these results can be derived from previous work easily. We highlight two non-trivial properties about deterministic flip-pushdown automata which are not easily implied by previous results. These are: regularity problem and equality problem. Are these decidable?

References

1. M. Holzer and M. Kutrib: *Flip-pushdown automata: nondeterminism is better than determinism.* LNCS 2710, 2003, 361–372.
2. M. Holzer and M. Kutrib: *Flip-pushdown automata: $k+1$ pushdown reversals are better than k.* LNCS 2719, 2003, 490–501.
3. P. Sarkar: *Pushdown automaton with the ability to flip its stack.* ECCC Report No. 81, 2001.
4. J.E. Hopcroft and J.D. Ullman: *Introduction to automata theory, languages and computation.* Addison-Wesley, 1979.
5. S.A. Greibach: *A new normal-form theorem for context-free phrase structure grammars.* Journal of ACM 12, 1965, 42–52.