Two Grammatical Equivalents of Flip-Pushdown Automata*

Peter Kostolányi

Department of Computer Science, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Mlynská dolina, 842 48 Bratislava, Slovakia

Abstract. Flip-pushdown automata, introduced by Sarkar [7], are pushdown automata with an additional ability to reverse the contents of their pushdown, and with the most interesting setting arising when the number of such flips is limited by a constant. Two characterizations of flip-pushdown automata (with a limited number of flips) in terms of grammars are presented in this paper. First, the model is characterized by context-free grammars with an extra ability to generate reversals, which are called *reversal-generating context-free grammars* (RGCFG). Next, a model of parallel word production called *parallel interleaving grammar system* (PIGS) is introduced, for which the equivalence with flip-pushdown automata is proved, linking flip-pushdown automata to parallelism. The characterization in terms of PIGS is used to prove that flip-pushdown automata (with a limited number of flips) are weaker than ET0L systems, which solves an open problem of Holzer and Kutrib [2].

Keywords: Flip-Pushdown Automaton, Reversal-Generating Context-Free Grammar, RGCFG, Parallel Interleaving Grammar System, PIGS

1 Introduction

Nondeterministic flip-pushdown automata (NFPDA) are an extension of ordinary (nondeterministic) pushdown automata, given an additional ability to flip the pushdown store, i.e., to reverse its contents. This flipping operation allows to read the pushdown store at both of its ends and, in particular, to use it as a dequeue.

Since dequeue automata can be used to simulate Turing machines, it is obvious that when the number of pushdown flips is unlimited, flip-pushdown automata and Turing machines are equal in their computational power [7]. Thus, the research focus has been mainly on flip-pushdown automata with the number of flips limited by a constant. Flip-pushdown automata, limited in this way, are known to be more powerful than ordinary pushdown automata, while retaining virtually all of their pleasant properties, which makes the study of flip-pushdown automata particularly interesting [2].

* This work has been supported by the grants VEGA 1/0979/12 and UK/322/2014.

Flip-pushdown automata have been introduced by Sarkar in [7]. Since then, many of their properties have been resolved. Most of these results have been obtained by Holzer and Kutrib in [2] and [3].

However, up to now, there has been no type of grammar proved to be equivalent to NFPDA with a limited number of flips. In this paper, we introduce such grammars. First, we show that (limited) NFPDA are equivalent to an extension of context-free grammars that we call *reversal-generating context-free grammars* (RGCFG). Such grammars have an ability to generate reversals along with ordinary symbols, and are related to the flip-pushdown input-reversal technique introduced by Holzer and Kutrib [2]. Next, we show that (limited) NFPDA are equivalent to systems of parallel grammars that we call *parallel interleaving grammar systems* (PIGS). PIGS appear to be a natural way of describing parallel word productions, so the equivalence of NFPDA and PIGS establishes a strong link between NFPDA and parallelism.

The viewpoint of grammars can make the reasoning about NFPDA-languages much easier, as we demonstrate by providing an example application. In particular, Holzer and Kutrib have posed in [2] an open problem of the relation between (limited) NFPDA and E0L systems and between (limited) NFPDA and ET0L systems, and in both cases they have conjectured incomparability. The former problem has been solved by Ďuriš and Košta in [1], where they have proved that NFPDA and E0L are in fact incomparable. In this paper, we solve the latter problem by showing that the strict inclusion holds, i.e., NFPDA with a limited number of flips are strictly weaker than ET0L systems.

2 Definitions

In this section, we shall review the formal definition of NFPDA and the corresponding families of languages, and briefly survey some of the basic results obtained so far.

Definition 1. *A* (nondeterministic) flip-pushdown automaton (NFPDA) *is a tuple* $A = (K, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, F)$*, where* K *is a finite set of states,* Σ *is an input alphabet,* Γ *is a pushdown alphabet,* δ *is an ordinary transition function from* $K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ *to finite subsets of* $K \times \Gamma^*$*,* Δ *is a flip transition function from* K *to subsets of* K*,* q_0 *in* K *is an initial state,* Z_0 *in* Γ *is a bottom-of-pushdown symbol, and* $F \subseteq K$ *is a set of accepting states.*

A *configuration* of the NFPDA A is defined similarly as for ordinary PDA, i.e., it is a triple (q, w, s), where q in K is a state, w in Σ^* is an unread part of the input word, and s in Γ^* is a content of the pushdown (written from the bottom of the pushdown). A *computation step* of the NFPDA A is a relation $\vdash_A$ on its configurations defined as follows: for p, q in K, a in $\Sigma \cup \{\varepsilon\}$, u in Σ^*, s, t in Γ^*, and Z in Γ, we define $(p, au, sZ) \vdash_A (q, u, st)$ if (q, t) is in $\delta(p, a, Z)$ (ordinary transitions) and $(p, u, Z_0 s) \vdash_A (q, u, Z_0 s^R)$ if q is in $\Delta(p)$ (flip transitions). Observe that the flip transition function does not depend neither on the input, nor on the contents of the pushdown. However, it may be easily seen

that this definition is equivalent to the definition with a flip transition function Δ going from $K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to K. If A is understood from the context, we shall write $\vdash$ instead of $\vdash_A$. The *language* $L(A)$ accepted by A *by a final state* and the language $N(A)$ accepted by A *by empty pushdown* are defined as usual: $L(A) = \{w \in \Sigma^* \mid \exists q \in F, s \in \Gamma^* : (q_0, w, Z_0) \vdash^* (q, \varepsilon, s)\}$, and $N(A) = \{w \in \Sigma^* \mid \exists q \in K : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$. We say that the NFPDA A *operates in at most (exactly) k flips*, if in every its computation, the pushdown store is reversed at most (exactly) k times. This can be easily turned into a syntactic definition as well.

In [2], it is proved that NFPDA accepting by a final state and NFPDA accepting by empty pushdown are equivalent, while the simulations involved do not change the number of flips performed. Moreover, it is proved there that NFPDA with at most k flips are equivalent to NFPDA with *exactly* k flips. In [7], it is observed that NFPDA with unrestricted number of pushdown flips are equivalent to Turing machines in their computational power.

Definition 2. *We denote the family of languages accepted (either by a final state or by empty pushdown) by some NFPDA operating in at most k flips by $\mathscr{L}(NFPDA_k)$. Further, we define*

$$\mathscr{L}(NFPDA_{fin}) = \bigcup_{k=0}^{\infty} \mathscr{L}(NFPDA_k),$$

and denote the family of languages accepted by arbitrary NFPDA by $\mathscr{L}(NFPDA)$ (it is already shown that $\mathscr{L}(NFPDA) = \mathscr{L}(RE)$).

In the previous works [7], [2], and [3], these families of languages have been defined in a slightly different manner. However, both definitions can be clearly seen to be equivalent. In [2], it has been proved that the families $\mathscr{L}(\mathrm{NFPDA}_k)$ form an infinite hierarchy, i.e.,

$$\mathscr{L}(\mathrm{CF}) = \mathscr{L}(\mathrm{NFPDA}_0) \subsetneq \mathscr{L}(\mathrm{NFPDA}_1) \subsetneq \mathscr{L}(\mathrm{NFPDA}_2) \subsetneq \ldots$$

Finally, let us state the important *Flip-pushdown input-reversal theorem*, introduced by Holzer and Kutrib in [2] (in a slightly different form).

Theorem 1 (Holzer, Kutrib [2]). *Let k be in $\mathbb{N}$. A language L is accepted by empty pushdown by a NFPDA $A_1 = (K, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, \emptyset)$ operating in $k+1$ pushdown flips iff the language*

$$L_R = \{uv^R \mid (q_0, u, Z_0) \vdash_{A_1}^* (q_1, \varepsilon, Z_0 s) \text{ with } k \text{ flips}, q_2 \in \Delta(q_1), \\ \text{and } (q_2, v, Z_0 s^R) \vdash_{A_1}^* (q_3, \varepsilon, \varepsilon) \text{ without any flip}\}$$

is accepted by empty pushdown by some NFPDA A_2 operating in k pushdown flips. The same statement holds for NFPDA accepting by a final state.

3 Reversal-Generating Context-Free Grammars

In this section, we define an extension of context-free grammars that has an additional ability to generate reversals. We shall call such context-free grammars *reversal-generating* and we shall prove that they are equivalent to NFPDA. Our definition of these grammars will be closely related to the *flip-pushdown input-reversal technique* of Holzer and Kutrib [2]. In fact, word production by reversal-generating grammars can be viewed as an inverse of this technique. While in the flip-pushdown input-reversal technique one gets a word from a NFPDA-language and transforms this word by a series of reversals into a word from a certain context-free language, a reversal-generating grammar generates a context-free language with special reversal symbols. By interpreting these reversal symbols in an appropriate order, one actually performs an inverse of the flip-pushdown input-reversal technique, and obtains a word from some NFPDA-language.

Definition 3. *A* reversal-generating context-free grammar (RGCFG) *is a five-tuple* $G = (N, T, P, \sigma, ®)$*, where* (N, T, P, σ) *is a context-free grammar, and* ® *is a special reversal symbol belonging to* T*.*

A *derivation step* of G is defined as for the context-free grammar $G' = (N, T, P, \sigma)$. The only difference is in the definition of the generated language. We define $L(G) = \{\varrho(w) \mid w \in L(G')\}$, where $\varrho : T^* \to (T - \{®\})^*$ is the reversal-interpreting function defined inductively by

$$\varrho(w) = \begin{cases} w & \text{for } w \text{ without an occurrence of } ®, \\ u\varrho(v^R) & \text{for } w = u®v,\ u \text{ without an occurrence of } ®, \text{ and } v \text{ in } T^*. \end{cases}$$

That is, reversal symbols are interpreted in the left-to-right order. We illustrate this by the following example.

Example 1. Let us consider a RGCFG $G = (N, T, P, \sigma, ®)$, such that (N, T, P, σ) generates a language consisting of words having the form $u®v®x®y®z$. Then, for every such word, the language $L(G)$ contains the word

$$\begin{aligned} \varrho(u®v®x®y®z) &= u\varrho(z^R®y^R®x^R®v^R) = uz^R\varrho(v®x®y) = \\ &= uz^Rv\varrho(y^R®x^R) = uz^Rvy^R\varrho(x) = uz^Rvy^Rx. \end{aligned}$$

The following proposition can be proved easily by induction, so we omit its proof.

Proposition 1. *Let* $G = (N, T, P, \sigma, ®)$ *be a RGCFG and let us denote the context-free grammar* (N, T, P, σ) *by* G'*. Then,*

$$\begin{aligned} L(G) = \{&w_1w_{2n}^Rw_2w_{2n-1}^R \dots w_nw_{n+1}^R \mid w_1®w_2® \dots ®w_{2n} \in L(G')\} \cup \\ \cup\ \{&w_1w_{2n+1}^Rw_2w_{2n}^R \dots w_nw_{n+2}^Rw_{n+1} \mid w_1®w_2® \dots ®w_{2n+1} \in L(G')\}. \end{aligned}$$

Similarly as in the case of NFPDA, we shall be interested mainly in reversal-generating grammars generating a limited number of reversal symbols. We shall call RGCFG generating at most k reversal symbols *k-reversal-generating*. It can be observed that this can be easily turned into a syntactic constraint, and that the condition of generating at most k reversal symbols is equivalent to the condition of generating *exactly* k reversal symbols. Further, a *reversal-aware normal form* of k-reversal generating CFGs can be considered, such that every nonterminal is aware of which reversals it produces in a terminal word. This can be formalized by taking a set of nonterminal symbols of the form $N = N' \times 2^{\{1,\ldots,k\}}$, where the second projection of each nonterminal is a (possibly empty) set of indices, corresponding to reversal symbols it generates in a terminal word. Obviously, these sets always consist of contiguous numbers. A proof that this is indeed a normal form is easy, and left to the reader.

Definition 4. *We denote the family of languages generated by k-reversal generating context-free grammars by $\mathscr{L}(RGCFG_k)$. Furthermore, we define*

$$\mathscr{L}(RGCFG_{fin}) = \bigcup_{k=0}^{\infty} \mathscr{L}(RGCFG_k).$$

We denote the family of languages generated by unrestricted reversal-generating context-free grammars by $\mathscr{L}(RGCFG)$.

Finally, we may proceed to the main theorem of this section, asserting that NFPDA and RGCFG are equivalent.

Theorem 2. *For all k in $\mathbb{N}$, the identity $\mathscr{L}(NFPDA_k) = \mathscr{L}(RGCFG_k)$ holds.*

Proof. First, let G be a RGCFG producing k reversal symbols. An equivalent NFPDA A performing k pushdown flips can be defined similarly as in the standard simulation of context-free grammars on pushdown automata (see, e.g., [4]). Let us initialize P_{sim} to be the set of production rules of G. The automaton A first makes its pushdown store contain the word $Z_0\sigma$, where Z_0 is a fixed bottom-of-pushdown symbol (different from the terminals and nonterminals of G), and σ is an initial nonterminal of G. Next, the following procedure is repeated: If a nonterminal of G is on the top of the pushdown, read nothing from the input and rewrite the nonterminal on the pushdown using some production rule in P_{sim}. If ® is on the top of the pushdown, erase it, perform a flip, and reverse the right-hand sides of production rules in P_{sim}. If any other terminal is on the top of the pushdown, erase it, and read the same symbol from the input. If it is not possible, A gets stuck. The automaton accepts if Z_0 is on the top of the pushdown (and the whole input is read). Clearly, $L(A) = L(G)$.

Now, let us prove the remaining inclusion. Let A be a NFPDA performing k flips. Obviously, by a minor change of its transition function, it is possible to obtain a NFPDA A', which behaves exactly like A, except that before every pushdown flip, it is forced to read some new special symbol # (and which cannot accept the input if some #-transition is not followed by a pushdown flip). Clearly,

$L(A')$ consists of words $u = u_1\#u_2\#\ldots\#u_{k+1}$, such that $u_1u_2\ldots u_{k+1}$ can be accepted by A with k flips performed at the positions marked in u by symbols $\#$. Now, by applying the flip-pushdown input-reversal technique [2] (see also Theorem 1 of the present paper) k times, one obtains the language

$$L' = \{u_1\#u_3\#\ldots\#u_k\#u_{k+1}^R\#u_{k-1}^R\#\ldots\#u_2^R \mid u_1\#u_2\#\ldots\#u_{k+1} \in L(A')\},$$

if k is odd, and the language

$$L' = \{u_1\#u_3\#\ldots\#u_{k+1}\#u_k^R\#u_{k-2}^R\#\ldots\#u_2^R \mid u_1\#u_2\#\ldots\#u_{k+1} \in L(A')\},$$

if k is even (this can be easily proved by induction). The Flip-pushdown input-reversal theorem implies that the language L' is context-free. Consider a context-free grammar G' generating L', and define a RGCFG G to be the same as G', but generating ® instead of $\#$. Then, it follows by Proposition 1 that

$$L(G) = \{u_1u_2\ldots u_{k+1} \mid u_1\#u_2\#\ldots\#u_{k+1} \in L(A')\} = L(A).$$

The theorem is proved. □

Corollary 1. $\mathscr{L}(NFPDA_{fin}) = \mathscr{L}(RGCFG_{fin})$.

4 Parallel Interleaving Grammar Systems

In this section, we shall introduce systems of parallel grammars, which we shall call *parallel interleaving grammar systems* (PIGS). Roughly said, languages generated by PIGS consist of words $u_1v_1u_2v_2\ldots u_mv_m$, where $u_1\#u_2\#\ldots\#u_m$ is generated by one context-free grammar, and $v_1\#v_2\#\ldots\#v_m$ is generated by another one. Here, $\#$ is a special *switch symbol* marking points in which the grammars interleave. The number of switch symbols generated need not be the same for both grammars, but the principle sketched above can be clearly generalized to hold in this setting as well.

Thus, we have two context-free grammars interleaving each other. However, these two grammars need not generate their words from scratch, but some sequential precomputation may take place. This means that first, an initial sentential form is precomputed sequentially, and this is then used by both grammars to start the generative process from. We shall prove that if the language of sequentially precomputed sentential forms is regular, then PIGS (with some restriction on the number of switch symbols) are equivalent to NFPDA (with a restriction on the number of pushdown flips).

To sum up, the generative process of PIGS can be divided into three stages. First, a sequential precomputation takes place, resulting in some sentential form. Next, this sentential form is used as an axiom by two context-free grammars, both of which (asynchronously) produce a terminal word with special switch symbols $\#$. Finally, these two words are combined into the final output by the following procedure: start by copying the word generated by the first grammar. When a switch symbol is encountered, do not copy it to the output, but continue by copying the word generated by the second grammar, etc.

Definition 5. *A* parallel interleaving grammar system *with* two context-free grammars *and with a* regular set of axioms (PIGS(2,CF,Reg)) *is a six-tuple* $G = (N, T, P_1, P_2, \#, I)$*, where* N *is a finite set of nonterminals,* T *is a finite set of terminals,* $N \cap T = \emptyset$*,* $P_1, P_2 \subseteq N \times (N \cup T)^*$ *are two finite sets of context-free production rules,* $\#$ *in* T *is a switch symbol, and* $I \subseteq (N \cup T)^*$ *is a regular language of initial sentential forms (axioms).*

Remark 1. In order to make the above definition strictly finitary, the regular set I may be replaced, e.g., by a regular grammar generating I.

Remark 2. The notation PIGS(2,CF,Reg) has been chosen in regard to possible future generalizations: PIGS consisting of more than two grammars can be studied, and some other family of grammars, and/or axiom sets can be considered. We believe that these generalizations are worth of research interest.

A *derivation step of the first grammar* of G is a binary relation $\Rightarrow_{G,1}$ (or simply $\Rightarrow_1$, if G is understood) on $(N \cup T)^*$ defined as follows: $u \Rightarrow_{G,1} v$ iff there are words u_1, u_2, x in $(N \cup T)^*$ and a nonterminal ξ in N, such that $u = u_1 \xi u_2$, $v = u_1 x u_2$, and $\xi \to x$ is in P_1. A *derivation step of the second grammar* of G, $\Rightarrow_{G,2}$ (or $\Rightarrow_2$, if G is understood), is defined similarly. The *language generated by the first grammar* of G from the axiom x is defined by

$$L(G, 1, x) = \{w \in T^* \mid x \Rightarrow_1^* w\},$$

and we make an analogous definition for the second grammar of G as well. Let us define a *locally regulated shuffle* $w_1 \sqcup\!\sqcup_{\#} w_2$ of words w_1, w_2 to be the output $\varphi(w_1, w_2)$ of the word combining function φ defined by

$$\varphi(u_1 \# u_2 \# \ldots \# u_i, v_1 \# v_2 \# \ldots \# v_j) = u_1 v_1 u_2 v_2 \ldots u_j v_j u_{j+1} \ldots u_i$$

for $i \geq j$, and by

$$\varphi(u_1 \# u_2 \# \ldots \# u_i, v_1 \# v_2 \# \ldots \# v_j) = u_1 v_1 u_2 v_2 \ldots u_i v_i v_{i+1} \ldots v_j$$

for $i \leq j$. For languages L_1, L_2, let us define

$$L_1 \sqcup\!\sqcup_{\#} L_2 = \{w_1 \sqcup\!\sqcup_{\#} w_2 \mid w_1 \in L_1, w_2 \in L_2\}.$$

Then, we define the *language generated* by G by

$$L(G) = \bigcup_{x \in I} L(G, 1, x) \sqcup\!\sqcup_{\#} L(G, 2, x).$$

Remark 3. Every language L, generated by some PIGS(2,CF,Reg), can be expressed by $L = \bigcup_{w \in R} \tau_1(w) \sqcup\!\sqcup_{\#} \tau_2(w)$, where R is a regular language, and τ_1, τ_2 are context-free substitutions.

Remark 4. The operation $\sqcup\!\sqcup_{\#}$ can be regarded as an analogy of *shuffles on trajectories*, studied in [5], with a local control instead of a global one. Thus, we believe it is worth of research attention.

Similarly as in the case of NFPDA and RGCFG, we shall be interested mostly in PIGS generating a limited number of switch symbols. We shall call a PIGS (i,j)-*switch-generating*, if its first grammar generates i switch symbols, and its second grammar generates j switch symbols (it can be easily seen that this is equivalent to the condition of generating *at most* i resp. j switch symbols). By a direct analogy with reversal-aware RGCFGs, a *switch-aware normal form* of (i,j)-switch-generating PIGS can be defined (cf. Section 3). However, two sets of indices – one for each grammar – have to be remembered here for every nonterminal.

Definition 6. *We denote the family of languages generated by (i,j)-switch generating PIGS(2,CF,Reg) by $\mathscr{L}(PIGS_{(i,j)}(2, CF, Reg))$. Further, we define*

$$\mathscr{L}(PIGS_{fin}(2, CF, Reg)) = \bigcup_{i,j\geq 0} \mathscr{L}(PIGS_{(i,j)}(2, CF, Reg)),$$

and we denote the family of languages generated by unrestricted PIGS(2,CF,Reg) by $\mathscr{L}(PIGS(2, CF, Reg))$.

Now, we may present the main result of this paper, characterizing languages accepted by NFPDA in terms of PIGS.

Theorem 3. *For all $k \geq 1$, $\mathscr{L}(NFPDA_k) = \mathscr{L}(PIGS_{\lceil \frac{k-1}{2} \rceil, \lfloor \frac{k-1}{2} \rfloor}(2, CF, Reg))$.*

Proof. We shall use Theorem 2, i.e., we shall prove our statement by showing that the identity

$$\mathscr{L}(\mathrm{RGCFG}_k) = \mathscr{L}(\mathrm{PIGS}_{\lceil \frac{k-1}{2} \rceil, \lfloor \frac{k-1}{2} \rfloor}(2, \mathrm{CF}, \mathrm{Reg}))$$

holds for all $k \geq 1$. In the rest of the proof, we shall assume that k is odd. The proof for the case when k is even is analogous.

First, let $G = (N, T, P, \sigma, \circledR)$ be a k-reversal-generating RGCFG in the reversal-aware normal form. We shall construct a $((k-1)/2, (k-1)/2)$-switch-generating PIGS(2,CF,Reg) $G' = (N', T', P'_1, P'_2, \#, I')$, such that $L(G') = L(G)$. We shall call nonterminals, from which the middle (i.e., the $(k+1)/2$-st) reversal symbol is generated, *middle nonterminals*. Obviously, the initial nonterminal σ is middle, and there is at most one middle nonterminal in each sentential form.

Now, for each word w generated by G, consider a derivation such that production rules from middle nonterminals are used first, followed by all other rules. The first stage of the derivation has a form $\sigma \Rightarrow^* u\circledR v$, where u, v are in $(N \cup T)^*$, and it follows by Proposition 1 that $u \Rightarrow^* w_1\circledR \dots \circledR w_{(k+1)/2} =: x$, $v \Rightarrow^* w_{(k+3)/2}\circledR \dots \circledR w_{k+1} =: y$, where $w = w_1 w_{k+1}^R \dots w_{(k+1)/2} w_{(k+3)/2}^R$. Thus, it is obviously sufficient to construct G' so that its first grammar generates x, and its second grammar generates y^R (in both cases with $\circledR$ replaced by $\#$), for all words x, y as above (and nothing else is generated by G').

In order to do this, let us first observe that a regular language R and homomorphisms h_1, h_2 do exist, such that

$$\{(h_1(z), h_2(z)) \mid z \in R\}$$

is exactly the set of all pairs (u, v^R), such that G generates $u®v$ in the first stage of some derivation, where the reversal symbol between u and v is middle. The language R may be, for instance, defined to be a language over the alphabet of production rules of G, consisting of all valid chains of production rules from middle nonterminals, together forming a complete first stage of some derivation.

Suppose that the alphabet Σ_R of R and $N \cup T$ are disjoint. Now, set $I' = R$, and define P'_1 to contain rules simulating h_1, i.e., rewriting all symbols c from Σ_R by $h_1(c)$, and, furthermore, all production rules of G with ® replaced by #. Formally, $P'_1 = \{c \to h(h_1(c)) \mid c \in \Sigma_R\} \cup \{\xi \to h(x) \mid \xi \to x \in P\}$, where h is a homomorphism such that $h(®) = \#$ and $h(c) = c$ for c in $(N \cup T) - \{®\}$. Furthermore, define P'_2 to contain rules simulating h_2 and rules of G with their right side reversed (and with ® replaced by #). This can be formally written as $P'_2 = \{c \to h(h_2(x)) \mid c \in \Sigma_R\} \cup \{\xi \to h(x)^R \mid \xi \to x \in P\}$. Finally, let us set $N' = N \cup \Sigma_R$, $T' = (T \cup \{\#\}) - \{®\}$, and the PIGS G' is completely defined. By what has been noted above, it is clear that $L(G') = L(G)$.

Now, we shall prove the remaining inclusion. Suppose that we are given a $((k-1)/2, (k-1)/2)$-switch-generating PIGS(2,CF,Reg) $G = (N, T, P_1, P_2, \#, I)$. We shall construct a k-reversal-generating RGCFG $G' = (N', T', P', \sigma', ®)$, such that $L(G') = L(G)$. Let $G_I = (N_I, T_I, P_I, \sigma_I)$ be a regular grammar generating I, such that N_I and $N \times \{1, 2\} \cup T \cup \{®\}$ are disjoint. Let us define G' as follows: $N' = N_I \cup N \times \{1, 2\}$, $T' = (T \cup \{®\}) - \{\#\}$, $\sigma' = \sigma_I$, and

$$\begin{aligned} P' = &\{\xi \to h_1(x)\eta h_2(x)^R \mid x \in T_I^*, \eta \in N_I, \xi \to x\eta \in P_I\} \ \cup \\ &\cup \ \{\xi \to h_1(x)®h_2(x)^R \mid x \in T_I^*, \xi \to x \in P_I\} \ \cup \\ &\cup \ \{(\xi, 1) \to h_1(x) \mid \xi \to x \in P_1\} \cup \{(\xi, 2) \to h_2(x)^R \mid \xi \to x \in P_2\}, \end{aligned}$$

where $h_1, h_2 : (N \cup T)^* \to (N' \cup T')^*$ are homomorphisms defined by $h_1(\xi) = (\xi, 1), h_2(\xi) = (\xi, 2)$ for ξ in N, $h_1(\#) = h_2(\#) = ®$ and $h_1(c) = h_2(c) = c$ for c in $T - \{\#\}$. It is obvious that the context-free grammar (N', T', P', σ') generates words $w_1®\ldots®w_{(k+1)/2}®w^R_{(k+3)/2}®\ldots®w^R_{k+1}$ such that the first grammar of G generates $w_1\#\ldots\#w_{(k+1)/2}$, and the second grammar generates $w_{k+1}\#\ldots\#w_{(k+3)/2}$. Then, $L(G') = L(G)$ follows directly by Proposition 1. □

Corollary 2. $\mathscr{L}(NFPDA_{fin}) = \mathscr{L}(PIGS_{fin}(2, CF, Reg))$.

Proof. The inclusion $\mathscr{L}(\mathrm{NFPDA}_{fin}) \subseteq \mathscr{L}(\mathrm{PIGS}_{fin}(2, \mathrm{CF}, \mathrm{Reg}))$ follows directly by Theorem 3. The remaining inclusion holds, since for all i, j in $\mathbb{N}$, we have $\mathscr{L}(\mathrm{PIGS}_{i,j}(2, \mathrm{CF}, \mathrm{Reg})) \subseteq \mathscr{L}(\mathrm{PIGS}_{\max\{i,j\},\max\{i,j\}}(2, \mathrm{CF}, \mathrm{Reg}))$, which, by Theorem 3, equals $\mathscr{L}(\mathrm{NFPDA}_{2\max\{i,j\}+1})$. □

5 A Relation to ET0L Systems

In this section, we shall present an example application of the characterization of NFPDA in terms of PIGS: We shall prove that flip-pushdown automata (with a constant number of flips) are strictly weaker than ET0L systems (for the definition, see, e.g., [6]).

The problem of the relation between flip-pushdown automata and ET0L systems has been posed by Holzer and Kutrib in [2]. There, they have conjectured that both NFPDA and E0L, and NFPDA and ET0L are incomparable. Ďuriš and Košta have already confirmed the former conjecture [1]. In this section, we shall prove that the latter conjecture does not hold, i.e., NFPDA are strictly weaker than ET0L systems.

Theorem 4. $\mathscr{L}(NFPDA_{fin}) \subsetneq \mathscr{L}(ET0L)$.

Proof. It has been already known that $\mathscr{L}(\text{NFPDA}_{fin}) \not\supseteq \mathscr{L}(\text{ET0L})$. This can be proved, e.g., by showing that the language $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is in $\mathscr{L}(\text{ET0L})$, but not in $\mathscr{L}(\text{NFPDA}_{fin})$. For more information, see [2].

Thus, it remains to prove that $\mathscr{L}(\text{NFPDA}_{fin}) \subseteq \mathscr{L}(\text{ET0L})$. We shall show this by applying Theorem 3, i.e., we shall prove that for all nonnegative integers i, j and every (i, j)-switch-generating PIGS(2,CF,Reg) $G = (N, T, P_1, P_2, \#, I)$, there is an ET0L system $\mathscr{S} = (V, \mathscr{P}, x, \Sigma)$, such that $L(\mathscr{S}) = L(G)$. Without loss of generality, we may suppose that $i = j$, that G is in the switch-aware normal form, and that $I \subseteq N^*$. Further, let $G_I = (N_I, T_I, P_I, \sigma_I)$ be a regular grammar generating I, such that $P_I \subseteq N_I \times (T_I N_I \cup \{\varepsilon\})$, $N_I \cap (N \cup T) = \emptyset$, and $T_I \subseteq N$. Finally, we shall assume that G generates switch symbols only by rules of the form $\alpha \to \#$, with α in N.

Since $\mathscr{L}(\text{CF}) \subseteq \mathscr{L}(\text{ET0L})$, it is possible to simulate both context-free grammars of G by an ET0L-system. Thus, the only problem is with interleaving. To overcome this, we have to generate regular axioms of G already appropriately interleaved. The only serious problem here is when a nonterminal from an axiom generates a switch symbol (in one or both of the grammars of G), since then it can generate symbols in two different contiguous parts of the final word, and it may produce some other nonterminals that generate symbols in even other contiguous parts of the final word (between the two contiguous parts it generates symbols in directly). In that case, we shall split the nonterminal into two parts that will be rewritten always in parallel. These parts may be split even further.

Besides this, it is important to mark each symbol with the number (1 or 2) of the grammar of G it corresponds to. Then, after generating the interleaved and marked axioms, it is possible to simulate both grammars on these axioms (with the presence of split nonterminals described above).

Now, we shall describe the formal construction. In what follows, h_1 and h_2 are homomorphisms, such that $h_1(\xi) = (\xi, 1)$ and $h_2(\xi) = (\xi, 2)$ for ξ in $N \cup N_I$, and $h_1(c) = h_2(c) = c$ for c in T. Next, for ξ in N, we define $\zeta_1(\xi)$ to be (s, t) if ξ generates switch symbols s to t in the first grammar of G, and to be $(0, 0)$ if it generates none. The notation $\zeta_2(\xi)$ has a similar meaning for the second grammar of G. The derivation of $\mathscr{S}$ will proceed in four phases, with the right order enforced by symbols Π_1, Π_2, Π_3, and Π_4 in $V - \Sigma$. In the s-th phase, the symbol Π_s is present in the sentential form. Moreover, every table of production rules designated for the s-th phase contains rules $\Pi_s \to \Pi_s$ and $\Pi_t \to F$ for $t \neq s$, where F in $V - \Sigma$ is a special *fail* symbol. The only rule from F in each table is $F \to F$. Thus, the sentential form containing F cannot be terminated.

The axiom x of $\mathscr{S}$ will be $h_1(\sigma_I)\$_1 h_2(\sigma_I)\$_{i+2}\$_2\$_{i+3}\ldots\$_{i+1}\$_{2i+2}\Pi_1$, where $\$_s$ is a symbol in $V-\Sigma$ denoting the end of the s-th contiguous part of the word being generated. These symbols will be erased in the end.

In the first phase, appropriately interleaved axioms of G are generated. For each rule $\alpha \to x\beta$ of G_I, there is one separate table $P_{\alpha\to x\beta}$ in $\mathscr{P}$. If $\zeta_1(x) = (0,0)$, then there is a rule $h_1(\alpha) \to h_1(x)h_1(\beta)$ in $P_{\alpha\to x\beta}$, and similarly for $\zeta_2(x) = (0,0)$. If $\zeta_1(x) = (s,t) \neq (0,0)$, then there are rules $h_1(\alpha) \to (h_1(x), left)$, $\$_k \to L_k R_k \$_k$ for $s < k \leq t$ and $\$_{t+1} \to (h_1(x), right)h_1(\beta)\$_{t+1}$ in $P_{\alpha\to x\beta}$, and similarly for the second grammar (these rules correspond to the splitting of nonterminals). Symbols L_k and R_k are placeholders for the further subdivision of split nonterminals. It follows from our assumption of all switch-producing rules having the form $\alpha \to \#$ that all these symbols will be rewritten sometimes. In addition, there are rules $h_1(\xi) \to F$ and $h_2(\xi) \to F$ for all ξ in N_I, $\xi \neq \alpha$ in this table. This is in order to assure that in the first phase, only the tables of rules corresponding to the nonterminal of G_I present in the sentential form are used. Moreover, there are rules $\Pi_1 \to \Pi_1$ and $\Pi_s \to F$ for $s > 1$ in this table. For all other symbols y, there is a rule $y \to y$.

For each rule $\alpha \to \varepsilon$ of G_I, there is one separate table $P_{\alpha\to\varepsilon}$ in $\mathscr{P}$. This contains rules $h_1(\alpha) \to \varepsilon$, $h_2(\alpha) \to \varepsilon$. Further, for all ξ in N_I, $\xi \neq \alpha$, there are rules $h_1(\xi) \to F$ and $h_2(\xi) \to F$. Since this table is used at the end of the first phase, there is a rule $\Pi_1 \to \Pi_2$. Similarly as above, rules $\Pi_s \to F$ for $s > 1$ are present, and for all other symbols y, there is a rule $y \to y$.

In the second phase, split (switch-producing) nonterminals are rewritten. We shall only describe the tables corresponding to the first grammar of G, since the tables for the second grammar are analogous. Let ξ in N be a nonterminal, such that $\zeta_1(\xi) \neq (0,0)$. Then, for each rule $\xi \to u\eta v$ in P_1 with u, v in $(N \cup T)^*$ and $\zeta_1(\eta) = \zeta_1(\xi)$ (note that the only switch-generating nonterminal on the right side is η), there is a table $P^{(1)}_{\xi\to u\eta v}$ in $\mathscr{P}$ containing the rules $(h_1(\xi), left) \to h_1(u)(h_1(\eta), left)$, $(h_1(\xi), right) \to (h_1(\eta), right)h_1(v)$, $\Pi_2 \to \Pi_2$, $\Pi_s \to F$ for $s \neq 2$, and $y \to y$ for all other symbols y. This table can be used also if (split) ξ is not present – in that case, the sentential form is not changed.

Now, let us consider rules with two switch-producing nonterminals on the right side, i.e., $\xi \to u\alpha v\beta w$ in P_1, where $\zeta_1(\xi) = (s,t) \neq (0,0)$, $\zeta_1(\alpha) = (s,k)$, $\zeta_1(\beta) = (k+1,t)$ for some $s \leq k < t$ and u, v, w in $(N \cup T)^*$. To every such rule, there is a table $P^{(1)}_{\xi\to u\alpha v\beta w}$ in $\mathscr{P}$ containing the following production rules: $(h_1(\xi), left) \to h_1(u)(h_1(\alpha), left)$, $L_{k+1} \to (h_1(\alpha), right)h_1(v)$, $R_{k+1} \to (h_1(\beta), left)$, and $(h_1(\xi), right) \to (h_1(\beta), right)h_1(w)$. Further, for every nonterminal $\eta \neq \xi$ generating the k-th and the $(k+1)$-st switch symbol, this table contains the rules $(h_1(\eta), left) \to F$ and $(h_1(\eta), right) \to F$. This is in order to assure that if in some terminable derivation, L_{k+1} or R_{k+1} is rewritten using this table, then this derivation step is valid, i.e., $(h_1(\xi), left)$ and $(h_1(\xi), right)$ were present. However, when neither L_{k+1} nor R_{k+1} is present in the sentential form, this table can be used (with no effect). Finally, the table contains rules $\Pi_2 \to \Pi_2$, $\Pi_s \to F$ for $s \neq 2$, and $y \to y$ for all other symbols y. The case of more than two switch-producing nonterminals on the right side is analogous.

For all rules of the type $\alpha \to \#$ in P_1, there is a table $P^{(1)}_{\alpha\to\#}$ in $\mathscr{P}$, with rules $(h_1(\alpha), left) \to \varepsilon$, $(h_1(\alpha), right) \to \varepsilon$, $\Pi_2 \to \Pi_2$, $\Pi_s \to F$ for $s \neq 2$, and $y \to y$ for all other symbols y.

A table used for finalizing the second phase has rules $\Pi_2 \to \Pi_3$, $\Pi_s \to F$ for $s \neq 2$, and $y \to y$ for all other symbols y. This table may be used also if the second phase is not finished yet, however, as we shall see, the sentential form cannot be terminated in that case (F will be produced in the third phase).

In the third phase, the rest of nonterminals is rewritten. For each rule $\xi \to u$ in P_1, where $\zeta_1(\xi) = (0, 0)$ and u is in $(N \cup T)^*$, there is a table $P^{(1)}_{\xi\to u}$ in $\mathscr{P}$, with rules $h_1(\xi) \to h_1(u)$, and $h_1(\xi) \to h_1(\xi)$. Further, for all (split) switch-producing nonterminals η and all symbols R_k and L_k, it contains rules that rewrite them to F. Finally, it contains rules $\Pi_3 \to \Pi_3$, $\Pi_s \to F$ for $s \neq 3$, and $y \to y$ for all other symbols y. Similarly for rules $\xi \to u$ in P_2.

A table for finalizing the third phase is similar to the table for finalizing the second phase.

Finally, there is only one table designated to the fourth phase. For each nonterminal, it contains rules rewriting it to F (that is, the third phase has to be finished). Moreover, it contains rules $\$_k \to \varepsilon$ for all k, $1 \leq k \leq 2i+2$, $\Pi_4 \to \varepsilon$, $\Pi_s \to F$ for $s \neq 4$, and $y \to y$ for all other symbols y. □

6 Acknowledgements

Many thanks go to Branislav Rovan and Pavel Labath, for the insightful comments they have made on the preliminary version of this paper.

References

1. Ďuriš, P., Košta, M.: Flip-Pushdown Automata with k Pushdown Reversals and E0L Systems are Incomparable. Inform. Process. Lett. 114, 417–420 (2014)
2. Holzer, M., Kutrib, M.: Flip-Pushdown Automata: $k + 1$ Pushdown Reversals Are Better than k. In: Baeten, J.C.M. et al. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 490–501. Springer-Verlag, Berlin Heidelberg (2003)
3. Holzer, M., Kutrib, M.: Flip-Pushdown Automata: Nondeterminism is Better than Determinism. In: Ésik, Z., Fülöp, Z. (eds.): DLT 2003. LNCS, vol. 2710, pp. 361–372. Springer-Verlag, Berlin Heidelberg (2003)
4. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation 2nd ed. Addison-Wesley, Reading (2001)
5. Mateescu, A., Rozenberg, G., Salomaa, A.: Shuffle on trajectories: Syntactic constraints. Theor. Comput. Sci. 197, 1–56 (1998)
6. Rozenberg, G., Salomaa, A.: The Mathematical Theory of L Systems. Academic Press, London (1980)
7. Sarkar, P.: Pushdown automaton with the ability to flip its stack. Report No. 81, Electronic Colloquium on Computational Complexity (2001)