# REGULATED NONDETERMINISM IN PDAS: THE NON-REGULAR CASE

Tomáš Masopust
Faculty of Information Technology, Brno University of Technology
Božetěchova 2, Brno 61266, Czech Republic
Email: masopust@fit.vutbr.cz

***Abstract***
*In this paper, we discuss pushdown automata which can make a nondeterministic decision only if the pushdown content forms a string that belongs to a given control language. We prove that if the control language is linear and non-regular, then the computational power of pushdown automata regulated in this way is increased to the power of Turing machines. Naturally, from the practical point of view, checking the pushdown content in each computational step is not efficient. Therefore, we prove that two checks of the form of the pushdown content during any computation are sufficient enough for these automata to be computationally complete. Based on this observation, a new model is discussed. Finally, some descriptional complexity results are presented.*

## 1. Introduction

While finite automata are of great interest in the theory and applications of regular expressions and languages, pushdown automata (PDAs) play an important role in the analysis of programming and natural languages. However, it is well-known that both programming and natural languages have some features that are not context-free, which means that they cannot be recognized by ordinary pushdown automata. For that reason, there are some attempts to introduce regulating mechanisms to increase the computational power of pushdown automata so that they are able to handle these features without loss of the practical efficiency.

Motivated by some types of regulations of grammars in the field of regulated rewriting (see [3, 4] for more information), so-called regulated pushdown automata have been introduced and studied in [7]. These automata have an additional control language over the alphabet of transitions regulating the applications of transitions so that an input string is accepted whenever it is accepted by the pushdown automaton by a sequence of transitions that forms a string belonging to the given control language. It has been shown that while regulated pushdown automata with regular control languages are equivalent to ordinary pushdown automata, regulated pushdown automata with non-regular, linear control languages are computationally complete (the reader is referred to [11] for more results).

Another variant of pushdown automata with some type of regulation is discussed in [8]. Instead of a control language over the alphabet of transitions, these automata are given a control

language over the alphabet of pushdown symbols. Then, an input string is accepted whenever it is accepted by the pushdown automaton by a computation where the pushdown content of each computational step forms a string that belongs to the given control language. It is proved in [8] that if the control language is regular, then the computational power of pushdown automata regulated in this way is the same as the computational power of ordinary pushdown automata. On the other hand, an example showing that non-regular, linear control languages increase the computational power of these automata is presented as well. Nevertheless, the question concerning the precise computational power of these automata with non-regular, linear control languages is left open.

Recently, considering the nondeterminism in pushdown automata, the previous modification of pushdown automata has been generalized, and so-called $R$-PDAs have been introduced and studied in [9, 10]. Specifically, given a control language $R$, an $R$-PDA is a pushdown automaton which makes a nondeterministic step whenever the pushdown content forms a string that belongs to the control language $R$, and makes a deterministic step whenever the pushdown content forms a string that does not belong to the control language $R$. Thus, $R$-PDAs behave nondeterministically if and only if their pushdown content forms a string that belongs to the control language $R$. It has been shown (see [9, 10]) that regular control languages do not change the computational power of pushdown automata with this type of regulation, while non-regular, linear control languages increase their computational power. For further results and properties concerning $R$-PDAs, where $R$ is a regular control language, the reader is referred to [9]; there, the case of the precise computational power of $R$-PDAs with non-regular control languages is formulated as an open problem.

In this paper, we answer this question. More specifically, we prove that $R$-PDAs are computationally complete even if the control language $R$ is a very simple non-regular language, i.e. a linear language. In addition, we demonstrate that it is sufficient to check the form of the pushdown content no more than twice during any computation, and that the number of states and pushdown symbols can be bounded.

Naturally, from the practical point of view, to check the form of the pushdown content in each computational step is not very effective. Therefore, we introduce and discuss so-called *state-controlled* $R$-PDAs ($R$-sPDAs). Specifically, given a control language $R$, an $R$-sPDA is a pushdown automaton which has a special set of distinguished states in which it makes a computational step according to its transition function only if the pushdown content forms a string that belongs to the control language $R$; in all other states, the automaton behaves as an ordinary pushdown automaton. As a result, we prove that two checks of the form of the pushdown content make $R$-sPDAs computationally complete. On the other hand, $R$-sPDAs with only one pushdown content check are shown to be more powerful than pushdown automata. However, their precise computational power is an open problem.

Finally, in the conclusion of the paper, we discuss $R$-PDAs and state-controlled $R$-PDAs, where the core pushdown automata are *deterministic*. In addition, we also formulate some open problems.

## 2. Preliminaries and Definitions

In this paper, we assume that the reader is familiar with automata and formal language theory (see [13, 14]). For a set $A$, $|A|$ denotes the cardinality of $A$. For an alphabet (finite nonempty set) $V$, $V^*$ represents the free monoid generated by $V$, where the unit of $V^*$ is denoted by $\lambda$. Set $V^+ = V^* \setminus \{\lambda\}$. For a string $w \in V^*$, $|w|$ denotes the length of $w$, and $w^R$ denotes the mirror image of $w$. For a language $L \subseteq V^*$, $L^R = \{w^R : w \in L\}$ denotes the mirror image of $L$.

A (phrase structure) *grammar* is a quadruple $G = (N, T, P, S)$, where $N$ is the alphabet of nonterminals, $T$ is the alphabet of terminals such that $N \cap T = \emptyset$, $V = N \cup T$ is the total alphabet, $S \in N$ is the start symbol, and $P$ is a finite set of productions of the form $u \to v$, where $u \in V^*NV^*$ and $v \in V^*$. For any two strings $x, y \in V^*$ and a production $u \to v \in P$, define the relation $xuy \Rightarrow xvy$. The language of $G$ is defined as $L(G) = \{w \in T^* : S \Rightarrow^* w\}$, where $\Rightarrow^*$ is the reflexive and transitive closure of the relation $\Rightarrow$. In addition, $G$ is said to be *linear* if each production $u \to v \in P$ satisfies the conditions (i) $u \in N$ is a nonterminal symbol and (ii) $v \in T^* \cup T^*NT^*$ is a string of symbols containing no more than one nonterminal symbol. A language $L$ is *linear* if there is a linear grammar $G$ such that $L = L(G)$.

A *pushdown automaton* (PDA) is a septuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where $Q$ is a finite set of states, $\Sigma$ is the input alphabet, $\Gamma$ is the pushdown alphabet, $\delta$ is a transition function from $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is the initial pushdown symbol, and $F \subseteq Q$ is the set of accepting states. A *configuration* of a pushdown automaton $\mathcal{M}$ is a triple $(q, w, \gamma)$, where $q$ is the current state, $w$ is the unread part of the input, and $\gamma$ is the current content of the pushdown (the leftmost symbol of $\gamma$ is the top pushdown symbol). If $p, q \in Q$, $a \in \Sigma \cup \{\lambda\}$, $w \in \Sigma^*$, $\gamma, \beta \in \Gamma^*$, $Z \in \Gamma$, and $(p, \beta) \in \delta(q, a, Z)$, then $\mathcal{M}$ makes a move from $(q, aw, Z\gamma)$ to $(p, w, \beta\gamma)$, formally $(q, aw, Z\gamma) \vdash_{\mathcal{M}} (p, w, \beta\gamma)$. For simplicity, the initial pushdown symbol appears only at the bottom of the pushdown during any computation, i.e., if $(p, \beta) \in \delta(q, a, Z)$, then either $\beta$ does not contain $Z_0$, or $\beta = \beta' Z_0$, where $\beta'$ does not contain $Z_0$ and $Z = Z_0$. As usual, the reflexive and transitive closure of the relation $\vdash_{\mathcal{M}}$ is denoted by $\vdash^*_{\mathcal{M}}$. The subscript $\mathcal{M}$ is removed whenever the meaning is clear. The *language accepted* by $\mathcal{M}$ is defined as

$$T(\mathcal{M}) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash^*_{\mathcal{M}} (q, \lambda, \gamma) \text{ for some } q \in F \text{ and } \gamma \in \Gamma^*\}.$$

A pushdown automaton $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is said to be *deterministic* (DPDA) if there is no more than one move the automaton can make from any configuration, i.e., the following two conditions hold:

1. $|\delta(q, a, Z)| \leq 1$, for all $a \in \Sigma \cup \{\lambda\}$, $q \in Q$, and $Z \in \Gamma$, and
2. for all $q \in Q$ and $Z \in \Gamma$, if $\delta(q, \lambda, Z) \neq \emptyset$, then $\delta(q, a, Z) = \emptyset$, for all $a \in \Sigma$.

In this case, $\delta(q, a, Z) = (p, \gamma)$ is written instead of $\delta(q, a, Z) = \{(p, \gamma)\}$.

The family of languages accepted by automata of type $X$ is denoted by $\mathscr{L}(X)$. It is well-known that $\mathscr{L}(\text{DPDA}) \subset \mathscr{L}(\text{PDA})$.

### 2.1. Pushdown Automata with Regulated Nondeterminism

In comparison with ordinary nondeterministic pushdown automata, a control language $R$ is given so that nondeterministic steps are allowed only if the current content of the pushdown forms a string that belongs to $R$.

Formally, let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA and $R \subseteq (\Gamma \setminus Z_0)^*$ be a control language. Then, $\mathcal{M}$ is called an $R$-PDA if the following two conditions hold:

1. for all $q \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $Z \in \Gamma$, $\delta$ can be written as
$$\delta(q, a, Z) = \delta_d(q, a, Z) \cup \delta_{nd}(q, a, Z)\,,$$
where $(Q, \Sigma, \Gamma, \delta_d, q_0, Z_0, F)$ is a DPDA and $(Q, \Sigma, \Gamma, \delta_{nd}, q_0, Z_0, F)$ is a (nondeterministic) PDA, and
2. for all $q, q' \in Q$, $a \in \Sigma \cup \{\lambda\}$, $w \in \Sigma^*$, $Z \in \Gamma$, and $\gamma \in \Gamma^*$,
$$(q, aw, Z\gamma) \vdash_{\mathcal{M}} (q', w, \gamma'\gamma) \text{ if}$$
   (a) either $(q', \gamma') \in \delta_{nd}(q, a, Z)$, $Z\gamma = \gamma''Z_0$, and $(\gamma'')^R \in R$,
   (b) or $\delta_d(q, a, Z) = (q', \gamma')$, $Z\gamma = \gamma''Z_0$, and $(\gamma'')^R \notin R$.

Condition 2 says that whenever the pushdown content forms a string that does not belong to the control language $R$, the automaton operates deterministically.

## 3. Results

This section presents the main results of this paper. The following theorem, proved in [9, 10], shows that if $R$ is a regular language, then each $R$-PDA can effectively be transformed to an equivalent PDA.

**Theorem 1** ([9, 10])**.** *Let $R$ be a regular language and $\mathcal{M}$ be an $R$-PDA. Then, an equivalent PDA $\mathcal{M}'$ can effectively be constructed.*

Note that an analogous result is proved in [8] for a special variant of $R$-PDAs, where the control language $R$ is a regular language and the deterministic transition function $\delta_d$ is empty, i.e., the pushdown content has to form a string that belongs to $R$ in each computational step. Moreover, it has been demonstrated (see [8, 9, 10]) that if $R$ is a linear, deterministic context-free control language, then there exists an $R$-PDA accepting a non-context-free language. This is also demonstrated in the following example, where an $\{a^nb^n : n \geq 1\}$-PDA accepting the language $\{a^nb^nc^nd^n : n \geq 1\}$ is given.

**Example 2.** Let $R = \{a^n b^n : n \geq 1\}$ be a control language. Clearly, $R$ is linear and deterministic context-free. Let $\mathcal{M} = (\{q_a, q_b, q_c, q_d, q_f\}, \{a, b, c, d\}, \{a, b, Z_0\}, \delta, q_a, Z_0, \{q_f\})$ be an $R$-PDA operating as follows:

- starting in $q_a$, $\mathcal{M}$ deterministically reads $a$ from the input and pushes $a$ to the pushdown;
- reading the first $b$, $\mathcal{M}$ deterministically goes to state $q_b$ and pushes $b$ to the pushdown, i.e., the pushdown contains $ba^nZ_0$;
- being in $q_b$, $\mathcal{M}$ deterministically reads $b$ from the input and pushes $b$ to the pushdown;
- reading the first $c$, $\mathcal{M}$ nondeterministically goes to state $q_c$, checking that the content of the pushdown is of the form $b^na^nZ_0$, and removes $b$ from the top of the pushdown;
- being in $q_c$, $\mathcal{M}$ deterministically reads $c$ from the input and removes $b$ from the pushdown;
- being in $q_c$ and having $a$ on the top of the pushdown, $\mathcal{M}$ deterministically goes to state $q_d$, reads $d$ from the input, and removes $a$ from the pushdown, i.e., $c^n$ has been read;
- being in $q_d$, $\mathcal{M}$ deterministically reads $d$ from the input and removes $a$ from the pushdown;
- finally, being in $q_d$ and having $Z_0$ on the top of the pushdown, $\mathcal{M}$ deterministically goes to the only final state $q_f$ from which no other symbol can be read; moreover, nothing is read from the input, and $Z_0$ is removed from the pushdown.

It is not hard to see that $T(\mathcal{M}) = \{a^nb^nc^nd^n : n \geq 1\}$, which is a non-context-free language.

In what follows, we prove that every recursively enumerable language is accepted by an $R$-PDA $\mathcal{M}$, for some convenient non-regular, linear control language $R$.

**Theorem 3.** *Let $L$ be a recursively enumerable language. Then, there exist a linear control language $R$ and an $R$-PDA $\mathcal{M}$ such that $L = T(\mathcal{M})$.*

To prove this, let $L \subseteq T^*$ be a recursively enumerable language. Then, by [5, 6], it is known that there is a phrase structure grammar $G = (\{S, A, B, C, D\}, T, P \cup \{AB \to \lambda, CD \to \lambda\}, S)$ in Geffert normal form such that $L = L(G)$ and $P$ contains only context-free productions of the following three forms:

- $S \to uSa$,
- $S \to uSv$,
- $S \to uv$,

where $u \in \{A, C\}^*$, $v \in \{B, D\}^*$, and $a \in T$. In addition, any successful derivation of $G$ is divided into the following two parts: the first part is of the form

$$S \Rightarrow_G^* w_1' S w_2' w \Rightarrow_G w_1 w_2 w \,,$$

generated only by context-free productions from $P$, where $w_1 \in \{A, C\}^*$, $w_2 \in \{B, D\}^*$, and $w \in T^*$, and the other part is of the form

$$w_1 w_2 w \Rightarrow_G^* w \,,$$

generated only by erasing productions $AB \to \lambda$ and $CD \to \lambda$. This immediately implies that the second part of the successful derivation verifies that $w_1 = h(w_2)^R$, for a homomorphism $h : \{B, D\}^* \to \{A, C\}^*$ defined as $h(B) = A$ and $h(D) = C$.

*Proof of Theorem 3.* Let $L \subseteq T^*$ be a recursively enumerable language. Then, there is a phrase structure grammar $G = (\{S, A, B, C, D\}, T, P \cup \{AB \to \lambda, CD \to \lambda\}, S)$ in Geffert normal form such that $L = L(G)$. Let

$$G_1 = (N_1, T_1, P_1, S_1)$$

be a linear grammar, where $N_1 = \{S_1, S\}$, $T_1 = T \cup \{A, B, C, D, \$\}$, for $\$$ being a new symbol, and $P_1 = P \cup \{S_1 \to \$S\}$. Then, it is not hard to see that the language generated by $G_1$ satisfies

$$L(G_1) \cap \{\$, A, B, C, D\}^* T^* = \{\$ w_1 w_2 w : S \Rightarrow_G^* w_1' S w_2' w \Rightarrow_G w_1 w_2 w\} \,.$$

Let

$$G_2 = (N_2, T_2, P_2, S_2)$$

be a linear grammar, where $N_2 = \{S_2, Y, Z\}$, $T_2 = T \cup \{A, B, C, D, @\}$, for @ being a new symbol, and $P_2 = \{S_2 \to @Y, Y \to Z, Z \to AZB, Z \to CZD, Z \to \lambda\} \cup \{Y \to Ya : a \in T\}$. Then, the language generated by $G_2$ is

$$L(G_2) = \{@ w_1 w_2 w : w_1 \in \{A, C\}^*, w_2 \in \{B, D\}^*, w \in T^*, w_1 = h(w_2)^R\} \,,$$

where $h$ is the homomorphism from $\{B, D\}^*$ to $\{A, C\}^*$ defined before this proof, i.e., $h(B) = A$ and $h(D) = C$.

Let

$$R = L(G_1)^R \cup L(G_2)^R \cup (\{A, B, C, D\} \cup T)^*$$

be the linear control language, and define the $R$-PDA $\mathcal{M} = (Q, T, \Gamma, \delta, q_0, Z_0, F)$ so that

- $Q = \{q_0, q_1, q_f\}$,
- $\Gamma = T \cup \{A, B, C, D, \$, @, Z_0\}$,
- $F = \{q_f\}$,

and $\delta$ is defined as follows:

$$\begin{aligned}
\delta_{nd}(q_0, \lambda, X) &= \{(q_0, aX) : a \in T \cup \{A, B, C, D\}\},\ X \in \{A, B, C, D, Z_0\} \cup T, \\
\delta_{nd}(q_0, \lambda, X) &= \{(q_0, \$X)\},\ X \in \{A, B, C, D, Z_0\} \cup T, \\
\delta_{nd}(q_0, \lambda, \$) &= \{(q_0, @)\}, \\
\delta_{nd}(q_0, \lambda, @) &= \{(q_1, \lambda)\}, \\
\delta_{nd}(q_1, \lambda, X) &= \{(q_1, \lambda)\},\ X \in \{A, B, C, D\}, \\
\delta_{nd}(q_1, a, a) &= \{(q_1, \lambda)\},\ a \in T, \\
\delta_{nd}(q_1, \lambda, Z_0) &= \{(q_f, \lambda)\}.
\end{aligned}$$

Finally, $\delta_d$ is empty.

Informally, $\mathcal{M}$ operates so that it first nondeterministically pushes symbols from the alphabet $T \cup \{A, B, C, D\}$ onto its pushdown. This is possible because the pushdown content is of the from $(\{A, B, C, D\} \cup T)^*$. Then, when \$ is pushed onto the pushdown, i.e., when the configuration is of the form $(q_0, w, \$\gamma Z_0)$, for some $\gamma \in \Gamma^*$, $\mathcal{M}$ verifies that $\$\gamma$ belongs to $L(G_1)$. This means that $\gamma = w_1 w_2 w$ such that there is a derivation $S \Rightarrow_G^* w_1' S w_2' w \Rightarrow_G w_1 w_2 w$ in $G$. If so, \$ is replaced with @, i.e., $(q_0, w, \$\gamma Z_0) \vdash_{\mathcal{M}} (q_0, w, @\gamma Z_0)$, and $\mathcal{M}$ verifies that $@\gamma$ belongs to $L(G_2)$. If so, then we have that $\gamma = w_1 w_2 w$, where

- $S \Rightarrow^* w_1' S w_2' w \Rightarrow w_1 w_2 w$ in $G$,
- $w_1 \in \{A, C\}^*$, $w_2 \in \{B, D\}^*$, $w \in T^*$, and
- $w_1 = h(w_2)^R$,

i.e., there is a derivation

$$w_1 w_2 w \Rightarrow_G^* w$$

in $G$. The automaton then finishes the computation by the following sequence of transitions: $(q_0, w, @\gamma Z_0) \vdash_{\mathcal{M}} (q_1, w, w_1 w_2 w Z_0) \vdash_{\mathcal{M}}^* (q_1, w, w Z_0) \vdash_{\mathcal{M}}^* (q_1, \lambda, Z_0) \vdash_{\mathcal{M}} (q_f, \lambda, \lambda)$.

Formally, to prove that $L(G) \subseteq T(\mathcal{M})$, let $S \Rightarrow^* w_1' S w_2' w \Rightarrow w_1 w_2 w \Rightarrow^* w$ be a successful derivation of $G$. Then, it is not hard to see that the corresponding computation of $\mathcal{M}$ accepting $w$ is as follows:

$$\begin{aligned}
(q_0, w, Z_0) &\vdash^* (q_0, w, w Z_0) \\
&\vdash^* (q_0, w, w_2 w Z_0) \\
&\vdash^* (q_0, w, w_1 w_2 w Z_0) \\
&\vdash (q_0, w, \$ w_1 w_2 w Z_0) \\
&\vdash (q_0, w, @ w_1 w_2 w Z_0) \\
&\vdash (q_1, w, w_1 w_2 w Z_0) \\
&\vdash^* (q_1, w, w Z_0) \\
&\vdash^* (q_1, \lambda, Z_0) \\
&\vdash (q_f, \lambda, \lambda).
\end{aligned}$$

Thus, we have that $w \in T(\mathcal{M})$.

On the other hand, to prove the other inclusion, $T(\mathcal{M}) \subseteq L(G)$, consider a computation of $\mathcal{M}$ accepting $w$. Such a computation is of the form

$$\begin{aligned}
(q_0, w, Z_0) &\vdash^* (q_0, w, \gamma Z_0) \\
&\vdash (q_0, w, \$\gamma Z_0) && (1)\\
&\vdash (q_0, w, @\gamma Z_0) && (2)\\
&\vdash (q_1, w, \gamma Z_0) \\
&\vdash^* (q_1, \lambda, Z_0) \\
&\vdash (q_f, \lambda, \lambda)
\end{aligned}$$

for some $\gamma \in \Gamma^*$. From (2), it follows that $@\gamma \in L(G_2)$, which means that

$$\gamma = w_1 w_2 w' \,,$$

where $w_1 \in \{A, C\}^*$, $w_2 \in \{B, D\}^*$, $w' \in T^*$, and $w_1 = h(w_2)^R$. Moreover, from (1), it follows that there is a derivation

$$S \Rightarrow^* w_1' S w_2' w' \Rightarrow w_1 w_2 w'$$

in $G$. As $w_1 = h(w_2)^R$, there is also a derivation

$$w_1 w_2 w' \Rightarrow^* w'$$

in $G$ by productions $AB \to \lambda$ and $CD \to \lambda$. It remains to prove that $w' = w$. However, examining the following part of the computation,

$$\begin{aligned}
(q_0, w, @w_1 w_2 w' Z_0) &\vdash (q_1, w, w_1 w_2 w' Z_0) \\
&\vdash^* (q_1, w, w_2 w' Z_0) \\
&\vdash^* (q_1, w, w' Z_0) \\
&\vdash^* (q_1, \lambda, Z_0) \\
&\vdash (q_f, \lambda, \lambda)
\end{aligned}$$

it immediately follows that $w' = w$ because the only input-reading transitions are of the form $\delta_{nd}(q_1, a, a) = \{(q_1, \lambda)\}$, for $a \in T$. Thus, we have that $w \in L(G)$. □

As an immediate corollary of the previous theorem, we have the following descriptional complexity result.

**Corollary 4.** *Let $L$ be a recursively enumerable language. Then, there exist a linear language $R$ and an $R$-PDA $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ such that $|Q| \leq 3$, $|\Gamma| \leq |\Sigma| + 7$, and $L = T(\mathcal{M})$.*

In general, the proof of Theorem 3 is based on the fact that for any recursively enumerable language $L$, there exist a homomorphism $h$ and two linear languages $L_1$ and $L_2$ such that

$L^R = h(L_1 \cap L_2)$. The automaton accepting $L$ operates so that it first nondeterministically pushes symbols onto its pushdown. Then, it checks that the form of its pushdown content, $\gamma$, forms a string that belongs to $L_1$ and $L_2$, i.e., $\gamma \in L_1 \cap L_2$. To do this, two checks of the form of the pushdown content are sufficient. After that, the automaton reads $X$ from the top of the pushdown and $h(X)^R$ from the input tape. It is known that the languages $L_1$ and $L_2$ can be of some special minimal forms that belong to some proper subfamilies of the family of linear languages. For an overview of these forms, the reader is referred to Table 1 in [12].

In addition, the following result can be achieved by simple modifications of grammars $G_1$ and $G_2$ from Theorem 3 so that each production $S \to \alpha \in P_1$ of $G_1$ is replaced with $S \to \alpha c_i$, where $c_i$ is a new symbol, for each $1 \leq i \leq |P_1|$; $G_2$ is modified in a corresponding way. Then, it is not hard to see that $L(G_1)^R$ is deterministic context-free, since for $S \to (c_i v_i S u_i)^R \in P_1$, $c_i$ says that $v_i^R$ is read from the input, and $u_i^R$ is pushed to the pushdown.

**Corollary 5.** *Let $L$ be a recursively enumerable language. Then, there exist two linear, deterministic context-free languages $L_1$, $L_2$, a regular language $R$, and an $(L_1 \cup L_2 \cup R)$-PDA $\mathcal{M}$ such that $L = T(\mathcal{M})$.*

Using the definitions and results of [12], we immediately have the following corollary. First, however, recall that a linear language $L \subseteq T^*$ is said to be *minimal linear* if it is generated by a linear grammar $G = (N, T, P, S)$, where $N = \{S\}$ is a singleton set and $G$ has a unique terminal production $S \to c$, where $c \in T$ appears only in this production. In addition, $G = (\{S\}, T, P, S)$ is said to be $(1,1)$-*minimal linear* if it is minimal linear and for each production $S \to \alpha S \beta \in P$, where $\alpha, \beta \in T^*$, we have that $|\alpha| = |\beta| = 1$. A language is said to be $(1,1)$-minimal linear if it is generated by a $(1,1)$-minimal linear grammar $G$.

**Corollary 6.** *Let $L$ be a recursively enumerable language. Then, there exist a minimal linear language $L_1 \subseteq \Sigma^*$, a $(1,1)$-minimal linear language $L_2 \subseteq \Sigma^*$, a regular language $R \subseteq \Sigma^*$, and an $(c_1 L_1 \cup c_2 L_2 \cup R)$-PDA $\mathcal{M}$, where $c_1 \neq c_2$, $c_1, c_2 \notin \Sigma$, such that $L = T(\mathcal{M})$.*

*Proof.* It is proved in [12] that for every recursively enumerable language $L$, there exist a minimal linear language $L_1 \subseteq \Sigma^*$, a $(1,1)$-minimal linear language $L_2 \subseteq \Sigma^*$, and a homomorphism $h : \Sigma^* \to \Sigma^*$ such that $L = h(L_1 \cap L_2)$. Let $R = \Sigma^*$ be a regular language, let $c_1, c_2 \notin \Sigma$ be two different symbols, and let $(c_1 L_1 \cup c_2 L_2 \cup R)$-PDA $\mathcal{M}$ be constructed by the method discussed above. Then, $L = T(\mathcal{M})$. □

Note that it is an open problem whether the two languages $L_1$ and $L_2$ from Corollary 6 can, in addition, be deterministic context-free.

## 4. State-Controlled $R$-PDAs

Naturally, from the practical point of view, to check the form of the pushdown content in each computational step is not very effective. Fortunately, taking a careful look at the proof of

Theorem 3, the reader can see that only two checks of the form of the pushdown content are needed; the first check of the pushdown content is made when \$ is pushed to the pushdown, and the other check is made when \$ is replaced with @ on the top of the pushdown. This observation leads to the following definition of state-controlled $R$-PDAs.

Let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Q_c, Z_0, F)$ be a PDA, where $Q_c \subseteq Q$ is a set of *checking states*, and all other symbols are as in an ordinary pushdown automaton. Let $R \subseteq (\Gamma \setminus Z_0)^*$ be a control language. Then, $\mathcal{M}$ is called a *state-controlled* $R$-PDA ($R$-sPDA) if for all $q, q' \in Q$, $a \in \Sigma \cup \{\lambda\}$, $w \in \Sigma^*$, $Z \in \Gamma$, and $\gamma \in \Gamma^*$,

$$(q, aw, Z\gamma) \vdash_{\mathcal{M}} (q', w, \gamma'\gamma)$$

if $(q', \gamma') \in \delta(q, a, Z)$ and

1. either $q \in Q \setminus Q_c$,
2. or $q \in Q_c$, $Z\gamma = \gamma'' Z_0$, and $(\gamma'')^R \in R$.

The reader can imagine $R$-sPDAs as pushdown automata with an oracle, which is able to answer the questions of whether the current content of the pushdown forms a string that belongs to the given control language $R$.

The following theorem can be proved by the same technique used in case of $R$-PDAs, see [9, 10].

**Theorem 7.** *Let $R$ be a regular language and $\mathcal{M}$ be an $R$-sPDA. Then, an equivalent PDA $\mathcal{M}'$ can effectively be constructed.*

Now, we can prove the following result.

**Theorem 8.** *Let $L$ be a recursively enumerable language. Then, there exist a linear language $R$ and an $R$-sPDA $\mathcal{M}$ such that $L = T(\mathcal{M})$. In addition, $\mathcal{M}$ checks the form of its pushdown content no more than twice during any computation.*

*Proof.* Consider the construction from the proof of Theorem 3. Let $R = L(G_1)^R \cup L(G_2)^R$ be the linear control language, and let the transition function $\delta$ be modified so that for all $X \in \{A, B, C, D, Z_0\} \cup T$, $Y \in \{A, B, C, D\}$, and $a \in T$,

- $\delta(q_0, \lambda, X) = \{(q_0, aX) : a \in T \cup \{A, B, C, D\}\}$,
- $\delta(q_0, \lambda, X) = \{(q_c, \$X)\}$,
- $\delta(q_c, \lambda, \$) = \{(q_c, @)\}$,
- $\delta(q_c, \lambda, @) = \{(q_1, \lambda)\}$,

- $\delta(q_1, \lambda, Y) = \{(q_1, \lambda)\}$,
- $\delta(q_1, a, a) = \{(q_1, \lambda)\}$,
- $\delta(q_1, \lambda, Z_0) = \{(q_f, \lambda)\}$,

where $q_c$ is the only state in which $\mathcal{M}$ checks the form of the pushdown content. The proof now follows from the proof of Theorem 3. □

As an immediate corollary, we have the following descriptional complexity result.

**Corollary 9.** *Let $L$ be a recursively enumerable language. Then, there exist a linear language $R$ and an $R$-sPDA $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Q_c, Z_0, F)$ which checks the form of the pushdown content no more than twice during any computation, such that $|Q| \leq 4$, $|Q_c| = 1$, $|\Gamma| \leq |\Sigma| + 6$, and $L = T(\mathcal{M})$.*

*Proof.* Clearly, only one of the pushdown symbols \$ and @ is necessary to distinguish between the two linear languages $L(G_1)^R$ and $L(G_2)^R$. Thus, @ can be removed. As the checks are made only in state $q_c$, $Q = \{q_0, q_c, q_1, q_f\}$ and $Q_c = \{q_c\}$. □

The following result follows from Corollary 5.

**Corollary 10.** *Let $L$ be a recursively enumerable language. Then, there exist two linear, deterministic context-free languages $L_1$, $L_2$, and an $(L_1 \cup L_2)$-sPDA $\mathcal{M}$ which checks the form of the pushdown content no more than twice during any computation, such that $L = T(\mathcal{M})$.*

Using the results of [12], we have the following consequence.

**Corollary 11.** *Let $L$ be a recursively enumerable language. Then, there exist a minimal linear language $L_1 \subseteq \Sigma^*$, a $(1,1)$-minimal linear language $L_2 \subseteq \Sigma^*$, $c \notin \Sigma$, and an $(cL_1 \cup L_2)$-sPDA $\mathcal{M}$ which checks the form of the pushdown content no more than twice during any computation, such that $L = T(\mathcal{M})$.*

*Proof.* It is proved in [12] that for every recursively enumerable language $L$, there exist a minimal linear language $L_1 \subseteq \Sigma^*$, a $(1,1)$-minimal linear language $L_2 \subseteq \Sigma^*$, and a homomorphism $h : \Sigma^* \to \Sigma^*$ such that $L = h(L_1 \cap L_2)$. Let $c \notin \Sigma$ be a new symbol, and let $(cL_1 \cup L_2)$-sPDA $\mathcal{M}$ be constructed by the method discussed above Corollary 5. Then, $L = T(\mathcal{M})$. □

Note that it is an open problem whether the two languages $L_1$ and $L_2$ from Corollary 11 can, in addition, be deterministic context-free.

By a simple modification of the definition of $R$-sPDAs, we can obtain automata which are able to check, according to the current checking state, whether the pushdown content forms a string that belongs to either $L_1$ or $L_2$. The advantage of this modification is a question of the further research and/or practical applications.

Furthermore, Example 1 demonstrates that there is an $R$-sPDA $\mathcal{M}$, where $R$ is a simple linear, deterministic context-free control language, which recognizes a non-context-free language $\{a^nb^nc^nd^n : n \geq 1\}$ with only one check of the form of the pushdown content. However, the question of the computational power of $R$-sPDAs making no more than one check of the form of their pushdown content during any computation is an open problem.

## 5. Conclusion

In this paper, we have shown that every recursively enumerable language can be accepted by an $R$-PDA, where $R$ is a non-regular, linear control language. In addition, no more than two checks of the form of the pushdown content are needed during any computation. Then, a new type of $R$-PDAs has been introduced and discussed, so-called state-controlled $R$-PDAs. As an immediate consequence of the previous results, it follows that every recursively enumerable language can be accepted by an $R$-sPDA which makes no more than two checks of the form of the pushdown content during any computation. On the other hand, it has been shown that $R$-sPDAs with no more than one check of the form of the pushdown content during any computation are able to accept non-context-free languages. However, their precise computational power is an open problem.

Furthermore, from the practical point of view, it seems to be of some interest to study the deterministic variant of $R$-sPDAs, so-called $R$-sDPDAs. Clearly, Example 1 illustrates that there are non-context-free languages that can be accepted by $R$-sDPDAs with only one check of the form of the pushdown content. Again, the precise computational power as well as all other properties are open.

Of some interest is also the above mentioned modification of the definition of $(L_1 \cup L_2)$-sPDAs using two (or more) control languages, whereby we can obtain automata which are able to check, according to the current checking state, whether the pushdown content forms a string that belongs to either $L_1$ or $L_2$. Clearly, this modification makes it possible to use deterministic PDAs to check the form of the pushdown content even if both $L_1$ and $L_2$ are deterministic context-free, but their union is not. However, this modification is a question of the further research and/or practical applications.

Another interesting variant seems to be so-called visibly (also called input-driven) $R$-sPDAs, where the pushdown operations are driven by the input symbols (see [1, 2] for more information).

Finally, in [9], an infinite hierarchy of language families is obtained by using some specific regular control languages. Can an analogous infinite hierarchy be also achieved using linear, non-regular control languages?

In addition, another open problem is the following question: fixing the control language $R$ and considering two $R$-PDAs $\mathcal{M}_1$ and $\mathcal{M}_2$, are the languages $T(\mathcal{M}_1) \cup T(\mathcal{M}_2)$, $T(\mathcal{M}_1) \cap T(\mathcal{M}_2)$, $T(\mathcal{M}_1) \cdot T(\mathcal{M}_2)$, $T(\mathcal{M}_1)^*$, etc. also accepted by an $R$-PDA? Are the problems that are decidable for $R$-PDAs, where $R$ is a regular control language, decidable also in case $R$ is a linear, non-regular control language (for the automata discussed in this section)?

## Acknowledgements

The author gratefully acknowledges useful suggestions and comments of the anonymous referees. This work was supported by the Czech Ministry of Education under the Research Plan No. MSM 0021630528.

## References

[1] R. Alur and P. Madhusudan. Visibly pushdown languages. In L. Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 202–211. ACM, 2004.

[2] B. Bollig. On the expressive power of 2-stack visibly pushdown automata. *Logical Methods in Computer Science*, 4(4):1–35, 2008.

[3] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*. Springer, Berlin, 1989.

[4] J. Dassow, Gh. Păun, and A. Salomaa. Grammars with controlled derivations. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 2, pages 101–154, Berlin, 1997. Springer.

[5] V. Geffert. Context-free-like forms for the phrase-structure grammars. In M. Chytil, L. Janiga, and V. Koubek, editors, *Proceedings of the 13th International Conference on Mathematical Foundations of Computer Science*, volume 324 of *Lecture Notes in Computer Science*, pages 309–317. Springer, 1988.

[6] V. Geffert. Normal forms for phrase-structure grammars. *RAIRO – Theoretical Informatics and Applications*, 25(5):473–496, 1991.

[7] D. Kolář and A. Meduna. Regulated pushdown automata. *Acta Cybernetica*, 4:653–664, 2000.

[8] Z. Křivka. *Rewriting Systems with Restricted Configurations*. PhD thesis, Faculty of Information Technology, Brno University of Technology, Brno, 2008.

[9] M. Kutrib, A. Malcher, and L. Werlein. Regulated nondeterminism in pushdown automata. doi:10.1016/j.tcs.2009.06.002.

[10] M. Kutrib, A. Malcher, and L. Werlein. Regulated nondeterminism in pushdown automata. In J. Holub and J. Zdárek, editors, *Proceedings of the 12th International Conference on Implementation and Application of Automata*, volume 4783 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2007.

[11] A. Meduna and D. Kolář. One-turn regulated pushdown automata and their reduction. *Fundamenta Informaticae*, 51(4):399–405, 2002.

[12] S. Okawa and S. Hirose. Homomorphic characterizations of recursively enumerable languages with very small language classes. *Theoretical Computer Science*, 250(1-2):55–69, 2001.

[13] A. Salomaa. *Formal Languages*. Academic Press, New York, 1973.

[14] A. Salomaa. *Computation and Automata*. Cambridge University Press, Cambridge, 1985.