

## On State-Synchronized Automata Systems

ALEXANDER MEDUNA, JIŘÍ KUČERA

Formal Language Research Group, Department of Information Systems  
Faculty of Information Technology, Brno University of Technology  
Božetěchova 2, 612 66 Brno, Czech Republic  
e-mail: {meduna, ikucera}@fit.vutbr.cz

**Abstract.** In this paper, we introduce a new kind of automata systems, called *state-synchronized automata systems* of degree  $n$ . In general, they consists of  $n$  pushdown automata, referred to as their components. These systems can perform a computation step provided that the concatenation of the current states of all their components belongs to a prescribed control language. As its main result, the paper demonstrates that these systems characterize the family of recursively enumerable languages. In fact, this characterization is demonstrated in both deterministic and nondeterministic versions of these systems. Restricting their components, these systems provides less computational power.

**Keywords:** state-synchronized automata systems, automata systems, push-down automata, determinism, recursively enumerable languages.

### 1. Introduction

At present, processing information in a largely discontinuous and concurrent way represents a common computational phenomenon. Indeed, consider a process  $p$  that deals with information  $i$ . During a single computational step,  $p$  can simultaneously perform several concurrent subprocesses, each of which works with a different piece of information in  $i$ . As obvious, computation like this necessitates a highly sophisticated communication and synchronization between the subprocesses. As a result, to explore computation of this kind systematically and rigorously, computer science needs formal models that adequately reflect this way of computation.

Traditionally, formal language theory has always provided computer science with language-defining models to explore various information processors mathematically, so it should do so for the purpose sketched above as well. However, the classical versions of these models, such as automata and grammars, work in a completely single and isolated way, so they can hardly serve as appropriate models of the computation sketched above. Therefore, a proper formalization of processors that work in the way described above necessitates an adaptation of classical automata and grammars so they work on words in a multiple and synchronised way. At the same time, any adaptation of this kind should conceptually maintain the original structure of these models as much as possible so computer science can quite naturally base its investigation upon these newly adapted models by analogy with the standard approach based upon their classical versions. Simply put, these new models should work on words in a communicating and synchronized way while keeping their structural conceptualization unchanged. That is why, over the past two decades or so, formal language theory has introduced several versions of these models that work in the above-mentioned way to some extent. More specifically, this theory has conceptualized systems whose components consist of several grammars or automata, whose communication is based on some prespecified strategy, usually referred to as communication protocols. Let us give a broad overview of the most important systems of this kind.

The early work on grammar systems, inspired by problem solving theory, was focused to model an agent systems, where agents use for cooperation so called ‘blackboard model’. Such a systems, called *cooperating distributed grammar systems* [3], consist of finite number of Chomsky grammars [2] (‘agents’). The ‘blackboard’ here is represented by a sentential form shared by all grammars. The communication in these systems is performed in the way that when some grammar ends its derivation on the shared sentential form, another grammar continues. This task switching between grammars is driven by *derivation modes* (see [3]).

The *parallel communicating grammar systems* [3] use a different strategy. Each grammar works independently on its own sentential form. When some grammar generates a *query symbol*, all derivations are paused and this query symbol is substituted by corresponding sentential form from requested grammar. After that, the derivation process may continue. A similar strategy is used in *parallel communicating finite automata systems* [7], where the communication between finite automata components is realized by query states. If some finite automaton performs a move to a query state, then this state is replaced by the current state of corresponding requested component. The *parallel communicating pushdown automata systems* [4] use the query symbols in the same manner as in parallel communicating grammar systems, except that all operations are performed on pushdowns.

Another ways of communication between components can be found in [1] and [10]. In *multigenerative grammar systems* [10], the cooperation between components is synchronized either by rules or by nonterminal symbols. In these systems, all components work simultaneously on their own sentential forms. If some component is unable to do its derivation, remaining components are blocked. The direct derivation in these systems is performed if and only if all components can perform their direct derivations and the rules used during these derivations form a tuple from a prescribed control set (for nonterminal symbols, the situation is similar).

In *n-accepting state-restricted pushdown automata systems* [1], the states of the components determine which components can do their moves and which components leave their configurations unchanged. This activity of components is changed during each computation step depending on the fact if an  $n$ -tuple formed from states is the left-hand side of some *switch rule* from the prescribed control set.

The present paper contributes to this vivid trend of formal language theory by introducing new automata systems whose behavior is elegantly synchronized by state-based restrictions. More specifically, these systems, called *state-synchronized automata systems* of degree  $n$ , have pushdown automata as their components. The synchronization is performed by finite control language containing words formed from states of particular components. In these systems, only first component can read from input tape. A computation step is performed if and only if all components can simultaneously do their moves and their states form a word from the control language.

The organization of this paper is as follows. After the next section with preliminary definitions, we give the definitions of state-synchronized automata systems, deterministic and nondeterministic. In the section with theoretical results, we study the accepting power of these systems with various types of components. We show that state-synchronized automata systems are able to accept every recursively enumerable language. It remains an open problem whether or not a deterministic state-synchronized automata systems with two or more one-turn pushdown automata can describe the whole family of recursively enumerable languages.

## 2. Preliminaries

We assume that the reader is familiar with the basic notions of formal language theory (see [5] and [8]). Let  $S$  be a set. Then, the cardinality of  $S$  is denoted by  $\text{card}(S)$ . Let  $\Sigma$  be an alphabet. Then,  $\Sigma^*$  represents the free monoid generated by  $\Sigma$  under the operation of concatenation, with  $\varepsilon$  as the unit of  $\Sigma^*$ . Let  $\varrho$  be a (binary) relation. Then,  $\varrho^*$  denotes the reflexive and transitive closure of  $\varrho$ , and  $\varrho^i$  denotes the  $i$ th power of  $\varrho$ ,  $i \geq 0$ . Let  $w$  be a word over  $\Sigma$ . Then, the length of  $w$  is denoted by  $|w|$ , and the set of all subwords contained in  $w$  is denoted by  $\text{subword}(w)$ . **LIN**, **CF**, and **RE** denotes the family of linear languages, the family of context-free languages, and the family of recursively enumerable languages, respectively.

A *finite automaton* (FA) is a quintuple  $M = (Q, \Sigma, R, s, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  is an input alphabet,  $Q \cap \Sigma = \emptyset$ ,  $R \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$  is a finite set of rules,  $s \in Q$  is the initial state, and  $F \subseteq Q$  is a set of final states. Instead of  $(p, a, q) \in R$ , we write  $pa \rightarrow q \in R$  throughout. A *configuration* of  $M$  is a word from  $Q\Sigma^*$ . The relation of *direct move*  $\vdash_M \subseteq Q\Sigma^* \times Q\Sigma^*$  is defined as follows: if  $w \in \Sigma^*$  and  $pa \rightarrow q \in R$ , then  $paw \vdash_M qw$ . The *language accepted by M*,  $L(M)$ , is defined as  $L(M) = \{w \in \Sigma^* \mid sw \vdash_M^* f, f \in F\}$ .

A *pushdown automaton* (PDA), is a septuple  $M = (Q, \Sigma, \Gamma, R, s, S, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  is an input alphabet,  $\Gamma$  is a pushdown alphabet,  $Q$ ,  $\Sigma$ , and  $\Gamma$  are pairwise disjoint,  $R \subseteq \Gamma \times Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \times Q$  is a finite set

of rules,  $s \in Q$  is the initial state,  $S \in \Gamma$  is the initial symbol on pushdown, and  $F \subseteq Q$  is a set of final states. Instead of  $(A, p, a, x, q) \in R$ , we write  $Apa \rightarrow xq \in R$  throughout. A *configuration* of  $M$  is a word from  $\Gamma^*Q\Sigma^*$ . The relation of *direct move*  $\vdash_M \subseteq \Gamma^*Q\Sigma^* \times \Gamma^*Q\Sigma^*$  is defined as follows: if  $u \in \Gamma^*$ ,  $w \in \Sigma^*$ , and  $Apa \rightarrow xq \in R$ , then  $uApaw \vdash_M uxqw$ . The *language accepted by  $M$  by final state*,  $L(M)_f$ , is defined as  $L(M)_f = \{w \in \Sigma^* \mid Ssw \vdash_M^* \gamma f, \gamma \in \Gamma^*, f \in F\}$ . The *language accepted by  $M$  by empty pushdown*,  $L(M)_\varepsilon$ , is defined as  $L(M)_\varepsilon = \{w \in \Sigma^* \mid Ssw \vdash_M^* q, q \in Q\}$ . The *language accepted by  $M$  by final state and empty pushdown*,  $L(M)_{f\varepsilon}$ , is defined as  $L(M)_{f\varepsilon} = \{w \in \Sigma^* \mid Ssw \vdash_M^* f, f \in F\}$ . A pushdown automaton  $M$  is said to be *deterministic* (dPDA) if for every rule  $Apa \rightarrow xq \in R$  it holds  $\text{card}(\{\alpha \rightarrow \gamma \in R \mid \alpha \in \text{subword}(Apa)\}) = 1$ .

The family of languages accepted by  $X$  by *final state*, *empty pushdown*, and *final state and empty pushdown*, where  $X$  is PDA or dPDA, is denoted by  $\mathcal{L}(X)_f$ ,  $\mathcal{L}(X)_\varepsilon$ , and  $\mathcal{L}(X)_{f\varepsilon}$ , respectively.

Let  $M = (Q, \Sigma, \Gamma, R, s, S, F)$  be a pushdown automaton, and let  $xoabw \vdash_M ypbw \vdash_M zqw$ , where  $x, y, z \in \Gamma^*$ ,  $o, p, q \in Q$ ,  $a, b \in \Sigma \cup \{\varepsilon\}$ , and  $w \in \Sigma^*$ . If  $|y| \geq |x|$  and  $|y| > |z|$ , then during the move  $ypbw \vdash_M zqw$ ,  $M$  makes a *turn* in its pushdown. We say that  $M$  is a *one-turn pushdown automaton* (one-turn PDA) if it makes no more than one turn in its pushdown during any computation.

A *two-pushdown automaton* is an 8-tuple  $M = (Q, \Sigma, \Gamma, R, s, S_1, S_2, F)$ , where  $Q$ ,  $\Sigma$ ,  $\Gamma$ ,  $s$ , and  $F$  are defined as in PDA,  $Q$ ,  $\Sigma$ , and  $\Gamma$  are pairwise disjoint,  $S_1 \in \Gamma$  is the initial symbol on pushdown 1,  $S_2 \in \Gamma$  is the initial symbol on pushdown 2, and  $R \subseteq \Gamma \times \Gamma \times Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \times \Gamma^* \times Q$  is a finite set of rules. Instead of  $(A, B, p, a, x, y, q) \in R$ , we write  $A\#Bpa \rightarrow x\#yq \in R$  throughout, where  $\# \notin \Sigma \cup \Gamma \cup Q$ . A *configuration* of  $M$  is a word from  $\Gamma^*\{\#\}\Gamma^*Q\Sigma^*$ . The relation of *direct move*  $\vdash_M \subseteq \Gamma^*\{\#\}\Gamma^*Q\Sigma^* \times \Gamma^*\{\#\}\Gamma^*Q\Sigma^*$  is defined as follows: if  $u, v \in \Gamma^*$ ,  $w \in \Sigma^*$ , and  $A\#Bpa \rightarrow x\#yq \in R$ , then  $uA\#vBpaw \vdash_M ux\#vyqw$ . The *language accepted by  $M$  by final state*,  $L(M)_f$ , is defined as  $L(M)_f = \{w \in \Sigma^* \mid S_1\#S_2sw \vdash_M^* \gamma_1\#\gamma_2f, \gamma_1, \gamma_2 \in \Gamma^*, f \in F\}$ . The *language accepted by  $M$  by empty pushdown*,  $L(M)_\varepsilon$ , is defined as  $L(M)_\varepsilon = \{w \in \Sigma^* \mid S_1\#S_2sw \vdash_M^* \#q, q \in Q\}$ . The *language accepted by  $M$  by final state and empty pushdown*,  $L(M)_{f\varepsilon}$ , is defined as  $L(M)_{f\varepsilon} = \{w \in \Sigma^* \mid S_1\#S_2sw \vdash_M^* \#f, f \in F\}$ . A two-pushdown automaton  $M$  is said to be *deterministic* if for every rule  $A\#Bpa \rightarrow x\#yq \in R$  it holds  $\text{card}(\{\alpha \rightarrow \gamma \in R \mid \alpha \in \text{subword}(A\#Bpa)\}) = 1$ .

Let  $M = (Q, \Sigma, \Gamma, R, s, S_1, S_2, F)$  be a two-pushdown automaton. Let

$$x_1\#x_2oabw \vdash_M y_1\#y_2pbw \vdash_M z_1\#z_2qw$$

in  $M$ , where  $o, p, q \in Q$ ,  $a, b \in \Sigma \cup \{\varepsilon\}$ ,  $w \in \Sigma^*$ , and  $x_1, x_2, y_1, y_2, z_1, z_2 \in \Gamma^*$ . If  $|y_i| \geq |x_i|$  and  $|y_i| > |z_i|$  for some  $i \in \{1, 2\}$ , then during the move  $y_1\#y_2pbw \vdash_M z_1\#z_2qw$ ,  $M$  makes a *turn* in pushdown  $i$ . If  $|y_i| \geq |x_i|$  and  $|y_i| > |z_i|$  for both  $i = 1, 2$ , then during move  $y_1\#y_2pbw \vdash_M z_1\#z_2qw$ ,  $M$  makes a *simultaneous turn* in both pushdowns. We say that  $M$  is *one-turn* if it makes no more than one turn in either of its pushdowns during any computation. We say that  $M$  is *simultaneously one-turn* if it makes either no turn or one simultaneous turn in both pushdowns during any computation.

### 3. Definitions and Examples

This section recalls the definition of state-synchronized automata system from [6].

**Definition 1** Let  $n$  be a positive integer. A state-synchronized automata system of degree  $n$  (SCAS $_n$  for short) is an  $(n+1)$ -tuple  $\Gamma = (M_1, M_2, \dots, M_n, \Psi)$ , where  $M_i$  is an FA or a PDA, and it is referred to as the  $i$ th component of  $\Gamma$ , for all  $1 \leq i \leq n$ .  $\Psi \subseteq Q_1 Q_2 \dots Q_n$  is a control language of  $\Gamma$ , where  $Q_i$  is the set of states in  $M_i$ ,  $1 \leq i \leq n$ . Furthermore,  $\Sigma$ ,  $\Gamma_i$ ,  $s_i$ ,  $S_i$ , and  $F_i$  denote the input alphabet of  $\Gamma$ , the pushdown alphabet of  $M_i$ , the initial state of  $M_i$ , the initial symbol on  $M_i$ 's pushdown, and the set of final states in  $M_i$ , respectively, for all  $1 \leq i \leq n$ . If  $M_i$  is an FA, we set  $\Gamma_i = \emptyset$  and  $S_i = \varepsilon$ , for all  $1 \leq i \leq n$ .

A configuration of  $\Gamma$  is an  $n$ -tuple  $(\chi_1, \chi_2, \dots, \chi_n)$ , where  $\chi_i$  is a configuration of  $M_i$ , for all  $1 \leq i \leq n$ .

Let  $\pi_i$  be the mapping from  $\Gamma_i^* Q_i \Sigma^*$  to  $Q_i$  such that  $\pi_i(x_i q_i w) = q_i$ ,  $x_i \in \Gamma_i^*$ ,  $q_i \in Q_i$ ,  $w \in \Sigma^*$ , for all  $1 \leq i \leq n$ . Furthermore, let  $\alpha = (\chi_1, \chi_2, \dots, \chi_n)$  and  $\alpha' = (\chi'_1, \chi'_2, \dots, \chi'_n)$  be two configurations of  $\Gamma$ . The relation of direct move in  $\Gamma$ ,  $\vdash_\Gamma$ , is defined as follows: if for every  $1 \leq i \leq n$  it holds  $\chi_i \vdash_{M_i} \chi'_i$ , and  $\pi_1(\chi_1)\pi_2(\chi_2)\dots\pi_n(\chi_n) \in \Psi$ , then  $\alpha \vdash_\Gamma \alpha'$ .

Analogously to PDA, define three types of languages accepted by  $\Gamma$  as follows:

$$\begin{aligned} L(\Gamma)_f &= \{w \in \Sigma^* \mid (S_1 s_1 w, S_2 s_2, \dots, S_n s_n) \vdash_\Gamma^* (\gamma_1 f_1, \gamma_2 f_2, \dots, \gamma_n f_n), \\ &\quad \gamma_i \in \Gamma_i^*, f_i \in F_i, 1 \leq i \leq n\}; \\ L(\Gamma)_\varepsilon &= \{w \in \Sigma^* \mid (S_1 s_1 w, S_2 s_2, \dots, S_n s_n) \vdash_\Gamma^* (q_1, q_2, \dots, q_n), \\ &\quad q_i \in Q_i, q_i \in F_i \text{ if } M_i \text{ is an FA}, 1 \leq i \leq n\}; \\ L(\Gamma)_{f\varepsilon} &= \{w \in \Sigma^* \mid (S_1 s_1 w, S_2 s_2, \dots, S_n s_n) \vdash_\Gamma^* (f_1, f_2, \dots, f_n), \\ &\quad f_i \in F_i, 1 \leq i \leq n\}. \end{aligned}$$

The following example demonstrates the capability of SCAS $_n$  to accept a language which is not context-free.

**Example 1** Let  $\Gamma = (M, M', \Psi)$  be an SCAS $_2$ , where  $M, M'$  are PDAs defined as

- $M = (\{s, q_a, q_b, q_c, f\}, \{a, b, c\}, \{S, A, B, C\}, R, s, S, \{f\})$ ;
- $M' = (\{s', q'_a, q'_b, q'_c, f'\}, \{a, b, c\}, \{S, A, B, C\}, R', s', S, \{f'\})$ ;
- $R = \{Ssa \rightarrow SAq_a, Aq_a a \rightarrow AAq_a, Aq_a b \rightarrow Aq_b, Aq_b b \rightarrow Aq_b, Aq_b c \rightarrow q_c, Aq_c c \rightarrow q_c, Sq_c \rightarrow f\}$ ;
- $R' = \{Ss' \rightarrow Sq'_a, Sq'_a \rightarrow Sq'_a, Sq'_a \rightarrow SBq'_b, Bq'_b \rightarrow BBq'_b, Bq'_b \rightarrow q'_c, Bq'_c \rightarrow q'_c, Sq'_c \rightarrow f'\}$ .

Finally,  $\Psi$  is a control language of  $\Gamma$  defined as  $\Psi = \{ss', q_a q'_a, q_b q'_b, q_c q'_c\}$ . The word

$aaabbbccc$  is accepted by  $\Gamma$  in this way

$$\begin{array}{l}
(Ssaaabbbccc, Ss') \vdash_{\Gamma} (SAq_a aabbbccc, Sq'_a) \\
\vdash_{\Gamma} (SAAq_a abbbccc, Sq'_a) \\
\vdash_{\Gamma} (SAAAq_a bbbccc, Sq'_a) \\
\vdash_{\Gamma} (SAAAq_b bbccc, SBq'_b) \\
\vdash_{\Gamma} (SAAAq_b bccc, SBBq'_b) \\
\vdash_{\Gamma} (SAAAq_b ccc, SBBBq'_b) \\
\vdash_{\Gamma} (SAAq_c cc, SBBq'_c) \\
\vdash_{\Gamma} (SAq_c c, SBq'_c) \\
\vdash_{\Gamma} (Sq_c, Sq'_c) \\
\vdash_{\Gamma} (f, f')
\end{array}$$

Clearly,  $L(\Gamma)_f = L(\Gamma)_{\varepsilon} = L(\Gamma)_{f\varepsilon} = \{a^n b^n c^n \mid n \geq 1\}$ .

Observe that  $\text{SCAS}_n \Gamma$ ,  $n \geq 1$ , to make a successful final computation step, do not need to contain a word in  $\Psi$  which is formed from states from a final configuration. On the other hand, as will be shown later in Definition 2, it would be useful to introduce  $\Psi_f$  as

$$\Psi_f = \Psi \cup \{q_1 q_2 \dots q_n \mid (\gamma_1 q_1, \gamma_2 q_2, \dots, \gamma_n q_n) \text{ is a final configuration of } \Gamma, \\
\gamma_i \in \Gamma_i^*, q_i \in Q_i, 1 \leq i \leq n\}.$$

Let  $\Gamma$  be an  $\text{SCAS}_n$ , for some  $n \geq 1$ . By  $\mathcal{R}_{\Gamma}$ , we denote the Cartesian product  $\mathcal{R}_{\Gamma} = R_1 \times R_2 \times \dots \times R_n$ , where  $R_i$  is the set of rules of the  $i$ th component of  $\Gamma$ , for all  $1 \leq i \leq n$ . If  $\alpha \in \mathcal{R}_{\Gamma}$ , then by  $\alpha(i)$ , we denote the  $i$ th element of  $\alpha$ , and clearly  $\alpha(i) \in R_i$ , for all  $1 \leq i \leq n$ . For a rule  $r = u \rightarrow v$ , we use  $\text{lhs}(r)$  and  $\text{rhs}(r)$  as an abbreviation for the left-hand side and right-hand side of  $r$ , respectively, so  $\text{lhs}(r) = u$  and  $\text{rhs}(r) = v$ .

Define the mapping  $\pi_l$  from  $\mathcal{R}_{\Gamma}$  to  $Q_1 Q_2 \dots Q_n$  as follows:

$$\pi_l(\alpha) = \pi_1(\text{lhs}(\alpha(1)))\pi_2(\text{lhs}(\alpha(2))) \dots \pi_n(\text{lhs}(\alpha(n))),$$

that is  $\pi_l$  maps an  $n$ -tuple of rules

$$(A_1 p_1 a_1 \rightarrow x_1 q_1, A_2 p_2 \rightarrow x_2 q_2, \dots, A_n p_n \rightarrow x_n q_n)$$

to the word  $p_1 p_2 \dots p_n$ . The mapping  $\pi_r$  from  $\mathcal{R}_{\Gamma}$  to  $Q_1 Q_2 \dots Q_n$  is defined analogously as  $\pi_r(\alpha) = \pi_1(\text{rhs}(\alpha(1)))\pi_2(\text{rhs}(\alpha(2))) \dots \pi_n(\text{rhs}(\alpha(n)))$ .

We are now at position to introduce the definition of determinism in  $\text{SCAS}_n$ , which is given formally in Definition 2.

**Definition 2** *Let  $\Gamma$  be an  $\text{SCAS}_n$ , for some  $n \geq 1$ . Then  $\Gamma$  is said to be deterministic ( $\text{dSCAS}_n$ ) if for every  $\alpha \in \mathcal{R}_{\Gamma}$ , such that  $\pi_l(\alpha) \in \Psi$  and  $\pi_r(\alpha) \in \Psi_f$ , it holds*

$$\text{card}(\{\alpha' \in \mathcal{R}_{\Gamma} \mid \pi_r(\alpha') \in \Psi_f, \text{lhs}(\alpha'(i)) \in \text{subword}(\text{lhs}(\alpha(i))), 1 \leq i \leq n\}) = 1.$$

Clearly, the  $\text{SCAS}_2$  from Example 1 is deterministic. The following example shows the difference between deterministic and nondeterministic  $\text{SCAS}_n$  according to Definition 2.

**Example 2** Consider we have  $\text{SCAS}_2 \Gamma = (M_1, M_2, \Psi)$ , where  $M_1, M_2$  are PDAs defined as

- $M_1 = (\{s, q_1, q_2, f\}, \{a, b\}, \{S, A, B\}, \{r_1, r_2, r_3, r_4\}, s, S, \{f\})$  and
- $M_2 = (\{s', q'_1, q'_2, f'\}, \{a, b\}, \{S, A, B\}, \{r'_1, r'_2, r'_3, r'_4\}, s', S, \{f'\})$ , where

$$\begin{array}{ll} r_1 = Ssa \rightarrow Sq_1; & r'_1 = Ss' \rightarrow Sq'_1; \\ r_2 = Ssb \rightarrow Sq_2; & r'_2 = Ss' \rightarrow Sq'_2; \\ r_3 = Sq_1 \rightarrow Sf; & r'_3 = Sq'_1 \rightarrow SAf'; \\ r_4 = Sq_2 \rightarrow Sf; & r'_4 = Sq'_2 \rightarrow SBf'. \end{array}$$

Let  $\Psi = \{ss', q_1q'_1, q_2q'_2\}$ . This  $\text{SCAS}_2$ , depending on its input, modify either its first or second pushdown. According to Definition 2, the allowed combinations of rules are:

$$\begin{array}{l} (r_1, r'_1): \text{card}(\{(r_1, r'_1)\}) = 1; \\ (r_2, r'_2): \text{card}(\{(r_2, r'_2)\}) = 1; \\ (r_3, r'_3): \text{card}(\{(r_3, r'_3)\}) = 1; \\ (r_4, r'_4): \text{card}(\{(r_4, r'_4)\}) = 1; \end{array}$$

so  $\Gamma$  is a  $\text{dSCAS}_2$ . Observe that condition  $\pi_r(\alpha') \in \Psi_f$  is necessary because its omission leads to

$$\begin{array}{l} (r_1, r'_1): \text{card}(\{(r_1, r'_1), (r_1, r'_2)\}) = 2; \\ (r_2, r'_2): \text{card}(\{(r_2, r'_2), (r_2, r'_1)\}) = 2; \\ (r_3, r'_3): \text{card}(\{(r_3, r'_3)\}) = 1; \\ (r_4, r'_4): \text{card}(\{(r_4, r'_4)\}) = 1; \end{array}$$

which is in contradiction with the fact that  $\Gamma$  is deterministic.

#### 4. Theoretical results

Since deterministic  $\text{SCAS}_n$ s are special cases of ordinary  $\text{SCAS}_n$ s, we show firstly that every recursively enumerable language can be accepted by some deterministic  $\text{SCAS}_n$ .

**Theorem 1** For every recursively enumerable language  $L$  over an alphabet  $\Sigma$ , there exists a deterministic  $\text{SCAS}_2$ ,  $\Gamma = (M_1, M_2, \Psi)$ , where

$$M_i = (Q_i, \Sigma, \Gamma_i, R_i, s_i, S_i, F_i)$$

is a PDA,  $1 \leq i \leq 2$ , such that  $L(\Gamma)_f = L$ .

**PROOF.** Let  $L$  be a recursively enumerable language over an alphabet  $\Sigma$ . Then there exists a deterministic two-pushdown automaton,  $M$ , such that  $L = L(M)_f$  (see Theorem 8.2.3.3 in [8]). Let  $M = (Q, \Sigma, \hat{\Gamma}, R, s, S_1, S_2, F)$  be a deterministic two-pushdown automaton such that  $L = L(M)_f$ . From  $M$ , we construct a  $\text{dSCAS}_2$   $\Gamma$  in the following way.

1. Set  $Q_1 = \emptyset$ ,  $Q_2 = \emptyset$ , and  $\Psi = \emptyset$ .
2. Set  $\Gamma_1 = \Gamma_2 = \hat{\Gamma}$ .
3. For every rule  $r = A\#Bpa \rightarrow x\#yq$  from  $R$ :
  - add states  $\bar{p}$ ,  $\langle \bar{r} \rangle$ , and  $\bar{q}$  to  $Q_1$ ;
  - add states  $\hat{p}$ ,  $\langle \hat{r} \rangle$ , and  $\hat{q}$  to  $Q_2$ ;
  - add rules  $A\bar{p}a \rightarrow A\langle \bar{r} \rangle$  and  $A\langle \bar{r} \rangle \rightarrow x\bar{q}$  to  $R_1$ ;
  - add rules  $B\hat{p} \rightarrow B\langle \hat{r} \rangle$  and  $B\langle \hat{r} \rangle \rightarrow y\hat{q}$  to  $R_2$ ;
  - add words  $\bar{p}\bar{p}$ ,  $\bar{q}\bar{q}$ , and  $\langle \bar{r} \rangle \langle \bar{r} \rangle$  to  $\Psi$ .
4. Set  $F_1 = \{\bar{f} \mid f \in F\}$  and  $F_2 = \{\hat{f} \mid f \in F\}$ .

From the construction above follows that if  $M$  is deterministic, then  $\Gamma$  must be also deterministic. Next, we prove that  $L(M)_f = L(\Gamma)_f$ .

**Claim 2** *If  $u_0\#v_0q_0w_0 \vdash_M^i u_i\#v_iq_iw_i$ , then*

$$(u_0\bar{q}_0w_0, v_0\hat{q}_0) \vdash_{\Gamma}^{2i} (u_i\bar{q}_iw_i, v_i\hat{q}_i),$$

where  $u_i \in \Gamma_1^*$ ,  $v_i \in \Gamma_2^*$ ,  $q_i \in Q$ ,  $\bar{q}_i \in Q_1$ ,  $\hat{q}_i \in Q_2$ , and  $w_i \in \Sigma^*$ , for all  $i \geq 0$ .

PROOF. The proof is established by induction on  $i \geq 0$ .

*Basis:* For  $i = 0$ ,  $u_0\#v_0q_0w_0 \vdash_M^0 u_0\#v_0q_0w_0$  implies that

$$(u_0\bar{q}_0w_0, v_0\hat{q}_0) \vdash_{\Gamma}^0 (u_0\bar{q}_0w_0, v_0\hat{q}_0).$$

Thus, the basis holds.

*Induction Hypothesis:* Suppose that the claim holds for all  $0 \leq i \leq k$ , for some  $k \geq 0$ .

*Induction Step:* If

$$u_0\#v_0q_0w_0 \vdash_M^k u_k\#v_kq_kw_k \vdash_M u_{k+1}\#v_{k+1}q_{k+1}w_{k+1},$$

then there exists a rule  $r = A\#Bq_ka \rightarrow x\#yq_{k+1}$  in  $R$  such that  $u_k = \gamma A$ ,  $v_k = \delta B$ ,  $u_{k+1} = \gamma x$ ,  $v_{k+1} = \delta y$ , and  $w_k = aw_{k+1}$ , where  $\gamma \in \Gamma_1^*$ ,  $\delta \in \Gamma_2^*$ . Then, there also exist rules  $A\bar{q}_ka \rightarrow A\langle \bar{r} \rangle \in R_1$ ,  $A\langle \bar{r} \rangle \rightarrow x\bar{q}_{k+1} \in R_1$ ,  $B\hat{q}_k \rightarrow B\langle \hat{r} \rangle \in R_2$ ,  $B\langle \hat{r} \rangle \rightarrow y\hat{q}_{k+1} \in R_2$ , and words  $\bar{q}_k\hat{q}_k$ ,  $\bar{q}_{k+1}\hat{q}_{k+1}$ ,  $\langle \bar{r} \rangle \langle \hat{r} \rangle \in \Psi$ , which implies that

$$\begin{aligned} (u_0\bar{q}_0w_0, v_0\hat{q}_0) &\vdash_{\Gamma}^{2k} (u_k\bar{q}_kw_k, v_k\hat{q}_k) \\ &\vdash_{\Gamma} (u_k\langle \bar{r} \rangle w_{k+1}, v_k\langle \hat{r} \rangle) \\ &\vdash_{\Gamma} (u_{k+1}\bar{q}_{k+1}w_{k+1}, v_{k+1}\hat{q}_{k+1}), \end{aligned}$$

and the claim holds for  $k + 1$ , too. Therefore, Claim 2 holds.  $\square$

**Claim 3** *If*

$$(u_0\bar{q}_0w_0, v_0\hat{q}_0) \vdash_{\Gamma}^{2i} (u_i\bar{q}_iw_i, v_i\hat{q}_i),$$

then  $u_0\#v_0q_0w_0 \vdash_M^i u_i\#v_iq_iw_i$ , where  $u_i \in \Gamma_1^*$ ,  $v_i \in \Gamma_2^*$ ,  $q_i \in Q$ ,  $\bar{q}_i \in Q_1$ ,  $\hat{q}_i \in Q_2$ , and  $w_i \in \Sigma^*$ , for all  $i \geq 0$ .



PROOF. The proof is established by induction on  $i \geq 0$ . Observe that only even number of computation steps are possible in  $\Gamma$  (see the construction of  $\Gamma$  from  $M$  above).

*Basis:* For  $i = 0$ ,

$$(u_0\bar{q}_0w_0, v_0\hat{q}_0) \vdash_{\Gamma}^0 (u_0\bar{q}_0w_0, v_0\hat{q}_0)$$

implies that  $u_0\#v_0q_0w_0 \vdash_M^0 u_0\#v_0q_0w_0$ . Thus, the basis holds.

*Induction Hypothesis:* Suppose that the claim holds for all  $0 \leq i \leq k$ , for some  $k \geq 0$ .

*Induction Step:* If

$$\begin{aligned} (u_0\bar{q}_0w_0, v_0\hat{q}_0) &\vdash_{\Gamma}^{2k} (u_k\bar{q}_k w_k, v_k\hat{q}_k) \\ &\vdash_{\Gamma} (u_k\langle\bar{r}\rangle w_{k+1}, v_k\langle\hat{r}\rangle) \\ &\vdash_{\Gamma} (u_{k+1}\bar{q}_{k+1} w_{k+1}, v_{k+1}\hat{q}_{k+1}), \end{aligned}$$

then there exist rules  $A\bar{q}_k a \rightarrow A\langle\bar{r}\rangle \in R_1$ ,  $A\langle\bar{r}\rangle \rightarrow x\bar{q}_{k+1} \in R_1$ ,  $B\hat{q}_k \rightarrow B\langle\hat{r}\rangle \in R_2$ ,  $B\langle\hat{r}\rangle \rightarrow y\hat{q}_{k+1} \in R_2$ , and words  $\bar{q}_k\hat{q}_k, \bar{q}_{k+1}\hat{q}_{k+1}, \langle\bar{r}\rangle\langle\hat{r}\rangle \in \Psi$ , such that  $u_k = \gamma A$ ,  $v_k = \delta B$ ,  $u_{k+1} = \gamma x$ ,  $v_{k+1} = \delta y$ ,  $\gamma \in \Gamma_1^*$ ,  $\delta \in \Gamma_2^*$ ,  $w_k = aw_{k+1}$ , and  $r = A\#Bq_k a \rightarrow x\#yq_{k+1} \in R$ . This immediately implies that

$$u_0\#v_0q_0w_0 \vdash_M^k u_k\#v_kq_kw_k \vdash_M u_{k+1}\#v_{k+1}q_{k+1}w_{k+1},$$

which proves that the claim holds for  $k + 1$ , too. Therefore, Claim 3 holds.  $\square$

From Claim 2 and Claim 3, it immediately follows that for every  $w \in \Sigma^*$ ,

$$S_1\#S_2sw \vdash_M^* \gamma_1\#\gamma_2f \quad \text{iff} \quad (S_1s_1w, S_2s_2) \vdash_{\Gamma}^* (\gamma_1f_1, \gamma_2f_2),$$

where  $\gamma_1 \in \Gamma_1^*$ ,  $\gamma_2 \in \Gamma_2^*$ . Therefore,  $L(M)_f = L(\Gamma)_f$ , and Theorem 1 holds.  $\square$

**Theorem 4** *For every recursively enumerable language  $L$  over an alphabet  $\Sigma$ , there exists a deterministic SCAS<sub>2</sub>,  $\Gamma = (M_1, M_2, \Psi)$ , where*

$$M_i = (Q_i, \Sigma, \Gamma_i, R_i, s_i, S_i, F_i)$$

*is a PDA,  $1 \leq i \leq 2$ , such that  $L(\Gamma)_{\varepsilon} = L$ .*

PROOF. Let  $L$  be a recursively enumerable language over an alphabet  $\Sigma$ . Then there exists a deterministic two-pushdown automaton,  $M$ , such that  $L = L(M)_{\varepsilon}$  (see Chapter 8 in [8]). The rest of proof is analogous as in Theorem 1.  $\square$

**Theorem 5** *For every recursively enumerable language  $L$  over an alphabet  $\Sigma$ , there exists a deterministic SCAS<sub>2</sub>,  $\Gamma = (M_1, M_2, \Psi)$ , where*

$$M_i = (Q_i, \Sigma, \Gamma_i, R_i, s_i, S_i, F_i)$$

*is a PDA,  $1 \leq i \leq 2$ , such that  $L(\Gamma)_{f\varepsilon} = L$ .*

PROOF. The proof is analogous to the proof of Theorem 4.  $\square$

**Theorem 6** *Let  $k \geq 3$ . For every recursively enumerable language  $L$  over an alphabet  $\Sigma$ , there exists a deterministic SCAS $_k$ ,*

$$\Gamma = (M_1, M_2, \dots, M_k, \Psi),$$

where  $M_i$  is a PDA, for all  $1 \leq i \leq k$ , such that  $L = L(\Gamma)_f$ .

PROOF. Let  $L$  be a recursively enumerable language over an alphabet  $\Sigma$ . Then, by Theorem 1, there exists a deterministic SCAS $_2$ ,  $\Gamma'$ , such that all its components are PDAs and  $L = L(\Gamma')_f$ . Let  $\Gamma' = (M'_1, M'_2, \Psi')$  be a deterministic SCAS $_2$ , where  $M'_i$  is a PDA,  $1 \leq i \leq 2$ , such that  $L = L(\Gamma')_f$ . Let  $k \geq 3$ . From  $\Gamma'$ , we construct a deterministic SCAS $_k$ ,

$$\Gamma = (M_1, M_2, \dots, M_k, \Psi),$$

where  $M_i$  is a PDA, for all  $1 \leq i \leq k$ , in the following way.

1.  $M_1 = M'_1$ ,  $M_2 = M'_2$ ;
2.  $M_i = (\{s_i\}, \Sigma, \{S_i\}, \{S_i s_i \rightarrow S_i s_i\}, s_i, S_i, \{s_i\})$ , for all  $3 \leq i \leq k$ ;
3.  $\Psi = \Psi' \{s_3 s_4 \dots s_k\}$ .

It is obvious that components  $M_3$  to  $M_k$  are redundant in  $\Gamma$ . Therefore,  $L(\Gamma)_f = L(\Gamma')_f$ , and Theorem 6 holds.  $\square$

**Theorem 7** *Let  $k \geq 3$ . For every recursively enumerable language  $L$  over an alphabet  $\Sigma$ , there exists a deterministic SCAS $_k$ ,*

$$\Gamma = (M_1, M_2, \dots, M_k, \Psi),$$

where  $M_i$  is a PDA, for all  $1 \leq i \leq k$ , such that  $L = L(\Gamma)_\varepsilon$ .

PROOF. Let  $L$  be a recursively enumerable language over an alphabet  $\Sigma$ . Then, by Theorem 4, there exists a deterministic SCAS $_2$ ,  $\Gamma'$ , such that all its components are PDAs and  $L = L(\Gamma')_\varepsilon$ . Let  $\Gamma' = (M'_1, M'_2, \Psi')$  be a deterministic SCAS $_2$ , where  $M'_i$  is a PDA,  $1 \leq i \leq 2$ , such that  $L = L(\Gamma')_\varepsilon$ . Let  $k \geq 3$ . From  $\Gamma'$ , we construct a deterministic SCAS $_k$ ,

$$\Gamma = (M_1, M_2, \dots, M_k, \Psi),$$

where  $M_i$  is a PDA, for all  $1 \leq i \leq k$ , in the following way.

1. Let  $M'_1 = (Q'_1, \Sigma, \Gamma'_1, R'_1, s'_1, S'_1, F'_1)$ . Then  $M_1 = (Q_1, \Sigma, \Gamma_1, R_1, s_1, S_1, F'_1)$ , where
  - $Q_1 = Q'_1 \cup \{s_1, q_1\}$ , where  $s_1, q_1 \notin Q'_1$ ;
  - $\Gamma_1 = \Gamma'_1 \cup \{S_1\}$ , where  $S_1 \notin \Gamma'_1$ ;
  - $R_1 = R'_1 \cup \{S_1 s_1 \rightarrow S_1 S'_1 s'_1, S_1 p \rightarrow q_1 \mid p \in Q'_1\}$ .
2. Analogously, we construct  $M_2$  from  $M'_2$ .
3. For all  $3 \leq i \leq k$ , set  $M_i = (\{s_i, p_i, q_i\}, \Sigma, \{S_i\}, \{S_i s_i \rightarrow S_i p_i, S_i p_i \rightarrow S_i p_i, S_i p_i \rightarrow q_i\}, s_i, S_i, \emptyset)$ .

4. Set  $\Psi = \Psi' \{p_3 p_4 \dots p_k\} \cup \{s_1 s_2 \dots s_k, q_1 q_2 \dots q_k\}$ .

$\Gamma$  works in this way:

1. During its move from  $s_i$  to  $s'_i$ ,  $M_i$  pushes  $S'_i$  on its pushdown,  $1 \leq i \leq 2$ . Simultaneously,  $M_j$  moves from  $s_j$  to  $p_j$ , for all  $3 \leq j \leq k$ .
2.  $\Gamma$  accepts (or rejects) its input word. During this phase,  $M_1$  and  $M_2$  perform their moves by using the same sequences of rules like  $M'_1$  and  $M'_2$ , respectively.  $M_3$  through  $M_k$  do loops over states  $p_3$  through  $p_k$ , respectively.
3. When  $M'_1$  and  $M'_2$  in  $\Gamma'$  empty their pushdowns, the pushdowns of  $M_1$  and  $M_2$  in  $\Gamma$  have  $S_1$  and  $S_2$  on their tops, respectively. At this point,  $M_1$  through  $M_k$  deterministically empty their pushdowns by moving to states  $q_1$  through  $q_k$ , respectively.

Thus,  $\Gamma$  accepts its input word if and only if  $\Gamma'$  accepts its input word, which completes the proof of Theorem 7.  $\square$

**Theorem 8** *Let  $k \geq 3$ . For every recursively enumerable language  $L$  over an alphabet  $\Sigma$ , there exists a deterministic SCAS $_k$ ,*

$$\Gamma = (M_1, M_2, \dots, M_k, \Psi),$$

where  $M_i$  is a PDA, for all  $1 \leq i \leq k$ , such that  $L = L(\Gamma)_{f\varepsilon}$ .

PROOF. Prove this by analogy with Theorem 7 except that states  $q_1$  through  $q_k$  are final.  $\square$

**Corollary 9** *Let  $n \geq 2$ . For every recursively enumerable language  $L$ , there exists a deterministic SCAS $_n$ ,  $\Gamma$ , such that all its components are PDAs and  $L = L(\Gamma)_f$ .*

PROOF. This follows from Theorem 1 and Theorem 6.  $\square$

**Corollary 10** *Let  $n \geq 2$ . For every recursively enumerable language  $L$ , there exists a deterministic SCAS $_n$ ,  $\Gamma$ , such that all its components are PDAs and  $L = L(\Gamma)_\varepsilon$ .*

PROOF. This follows from Theorem 4 and Theorem 7.  $\square$

**Corollary 11** *Let  $n \geq 2$ . For every recursively enumerable language  $L$ , there exists a deterministic SCAS $_n$ ,  $\Gamma$ , such that all its components are PDAs and  $L = L(\Gamma)_{f\varepsilon}$ .*

PROOF. This follows from Theorem 5 and Theorem 8.  $\square$

The family of recursively enumerable languages can be also characterized by SCAS $_n$ , where  $n \geq 2$ , such that all its components are one-turn PDAs.

**Theorem 12** *Let  $x \in \{f, \varepsilon, f\varepsilon\}$ . For every recursively enumerable language  $L$  over an alphabet  $\Sigma$ , there exists an  $\text{SCAS}_2$ ,  $\Gamma = (M_1, M_2, \Psi)$ , where both  $M_1$  and  $M_2$  are one-turn pushdown automata, such that  $L = L(\Gamma)_x$ .*

PROOF. Let  $x \in \{f, \varepsilon, f\varepsilon\}$ . Let  $L$  be a recursively enumerable language over an alphabet  $\Sigma$ . In [9], it was shown that there exists a simultaneously one-turn two-pushdown automaton,  $M$ , such that  $L = L(M)_x$ . We can construct  $\Gamma$  from  $M$  in the exactly same way as demonstrated in Theorem 1, and we can use the same proof to show the accepted language identity of  $\Gamma$  and  $M$ . From the construction of  $\Gamma$  and from the definition of one-turn two pushdown automaton follows that  $M_1$  and  $M_2$  are both one-turn pushdown automata.  $\square$

**Theorem 13** *Let  $x \in \{f, \varepsilon, f\varepsilon\}$  and  $k \geq 3$ . For every recursively enumerable language  $L$  over an alphabet  $\Sigma$ , there exists an  $\text{SCAS}_k$ ,*

$$\Gamma = (M_1, M_2, \dots, M_k, \Psi),$$

where  $M_i$  is a one-turn PDA, for all  $1 \leq i \leq k$ , such that  $L = L(\Gamma)_x$ .

PROOF. Let  $x \in \{f, \varepsilon, f\varepsilon\}$  and  $k \geq 3$ . Let  $L$  be a recursively enumerable language over an alphabet  $\Sigma$ . Then, by Theorem 12, there exists an  $\text{SCAS}_2$ ,  $\Gamma' = (M'_1, M'_2, \Psi')$ , where  $M'_1$  and  $M'_2$  are both one-turn PDAs, such that  $L = L(\Gamma')_x$ . From  $\Gamma'$ , we can construct  $\Gamma$ , and then proof the identity of languages accepted by  $\Gamma'$  and  $\Gamma$ , in the same way as demonstrated in proofs of Theorem 6, Theorem 7, and Theorem 8. Observe that components  $M_3$  through  $M_k$  of  $\Gamma$  are all one-turn PDAs.  $\square$

**Corollary 14** *Let  $x \in \{f, \varepsilon, f\varepsilon\}$  and  $n \geq 2$ . For every recursively enumerable language  $L$ , there exists an  $\text{SCAS}_n$ ,  $\Gamma$ , such that all its components are one-turn PDAs and  $L = L(\Gamma)_x$ .*

PROOF. This follows from Theorem 12 and Theorem 13.  $\square$

The previous results are summarized in the following theorem.

**Theorem 15** *For every  $L \in \mathbf{RE}$ , there are*

- a) *an  $\text{SCAS}_n$   $\Gamma$  such that all its components are PDAs and  $L = L(\Gamma)_x$ , where  $n \geq 2$  and  $x \in \{f, \varepsilon, f\varepsilon\}$ ;*
- b) *an  $\text{dSCAS}_n$   $\Gamma$  such that all its components are PDAs and  $L = L(\Gamma)_x$ , where  $n \geq 2$  and  $x \in \{f, \varepsilon, f\varepsilon\}$ ;*
- c) *an  $\text{SCAS}_n$   $\Gamma$  such that all its components are one-turn PDAs and  $L = L(\Gamma)_x$ , where  $n \geq 2$  and  $x \in \{f, \varepsilon, f\varepsilon\}$ .*

PROOF. This theorem follows from Corollary 9, Corollary 10, Corollary 11, Corollary 14, and from the obvious observation that deterministic  $\text{SCAS}_n$ s are no more powerful than nondeterministic  $\text{SCAS}_n$ s.  $\square$

Next, we study the case when no more than one pushdown component is permitted in  $\text{SCAS}_n$ .

**Lemma 16** *Let  $x \in \{f, \varepsilon, f\varepsilon\}$  and  $n \geq 1$ . For every SCAS $_n$ ,*

$$\Gamma = (M_1, M_2, \dots, M_n, \Psi),$$

*where for some  $1 \leq i \leq n$ ,  $M_i$  is a PDA, and for all  $1 \leq j \leq n$ ,  $i \neq j$ ,  $M_j$  is an FA, there exists a PDA,  $M$ , such that  $L(M)_x = L(\Gamma)_x$ .*

PROOF. Let  $x \in \{f, \varepsilon, f\varepsilon\}$ . For  $n = 1$ ,  $\Gamma$  has only one component, which is a PDA, and therefore Lemma 16 holds immediately. Further is shown that the lemma holds also for  $n \geq 2$ , where the case that the only first component of  $\Gamma$  can be a PDA is considered (the other situations can be proved analogously).

Let  $n \geq 2$  and  $\Gamma = (M_1, M_2, \dots, M_n, \Psi)$  be an SCAS $_n$ , where  $M_1$  is a PDA, and  $M_2$  through  $M_n$  are FAs. From  $\Gamma$ , we construct a PDA,  $M$ , such that  $L(M)_x = L(\Gamma)_x$ , in the following way.

1. Let  $M_1 = (Q_1, \Sigma, \Gamma_1, R_1, s_1, S_1, F_1)$  be a PDA from  $\Gamma$ , and let

$$M_i = (Q_i, \Sigma, R_i, s_i, F_i)$$

be an FA from  $\Gamma$ , for all  $2 \leq i \leq n$ .

2. Set  $M = (Q, \Sigma, \Gamma_1, R, \langle s_1 s_2 \dots s_n \rangle, S_1, F)$ , where

- $Q = \{\langle \omega \rangle \mid \omega \in \Psi \cup F_1 F_2 \dots F_n\}$ ;
- $R = \{A \langle \omega_1 \rangle a \rightarrow x \langle \omega_2 \rangle \mid$   
 $Ap_1 a \rightarrow xq_1 \in R_1, p_2 \rightarrow q_2 \in R_2, \dots, p_n \rightarrow q_n \in R_n$   
 $\omega_1 = p_1 p_2 \dots p_n, \omega_2 = q_1 q_2 \dots q_n$   
 $\langle \omega_1 \rangle, \langle \omega_2 \rangle \in Q$   
 $\}$ ;
- $F = \{\langle \omega \rangle \mid \omega \in F_1 F_2 \dots F_n\}$ .

To prove that  $L(\Gamma)_x = L(M)_x$ , we first establish the following two claims.

**Claim 17** *If*

$$(up_1 w, p_2, \dots, p_n) \vdash_{\Gamma}^i (u' q_1 w', q_2, \dots, q_n),$$

*then  $u \langle p_1 p_2 \dots p_n \rangle w \vdash_M^i u' \langle q_1 q_2 \dots q_n \rangle w'$ , where  $u, u' \in \Gamma_1^*$ ,  $w, w' \in \Sigma^*$ ,  $p_i, q_i \in Q_i$ ,  $1 \leq i \leq n$ , and  $\langle p_1 p_2 \dots p_n \rangle, \langle q_1 q_2 \dots q_n \rangle \in Q$ .*

PROOF. The proof is made by induction on  $i \geq 0$ .

*Basis:* For  $i = 0$ , we have

$$(up_1 w, p_2, \dots, p_n) \vdash_{\Gamma}^0 (up_1 w, p_2, \dots, p_n)$$

implies  $u \langle p_1 p_2 \dots p_n \rangle w \vdash_M^0 u \langle p_1 p_2 \dots p_n \rangle w$ , so the claim holds for  $i = 0$ .

*Induction Hypothesis:* Suppose that the claim holds for all  $0 \leq i \leq k$ , for some  $k \geq 0$ .

*Induction Step:* If

$$(uAp_1 aw, p_2, \dots, p_n) \vdash_{\Gamma} (uxo_1 w, o_2, \dots, o_n) \vdash_{\Gamma}^k (u' q_1 w', q_2, \dots, q_n),$$

then there exist rules  $Ap_1a \rightarrow x o_1 \in R_1$ ,  $p_j \rightarrow o_j \in R_j$ ,  $2 \leq j \leq n$ , and words  $p_1 p_2 \dots p_n, o_1 o_2 \dots o_n \in \Psi \cup F_1 F_2 \dots F_n$ . According to the construction of  $M$  from  $\Gamma$  above, this implies that there also exists a rule  $A\langle p_1 p_2 \dots p_n \rangle a \rightarrow x \langle o_1 o_2 \dots o_n \rangle \in R$ , so

$$uA\langle p_1 p_2 \dots p_n \rangle aw \vdash_M ux \langle o_1 o_2 \dots o_n \rangle w \vdash_M^k u' \langle q_1 q_2 \dots q_n \rangle w',$$

and the claim holds for  $k + 1$ , too. Therefore, Claim 17 holds.  $\square$

**Claim 18** *If*

$$u\langle p_1 p_2 \dots p_n \rangle w \vdash_M^i u' \langle q_1 q_2 \dots q_n \rangle w',$$

then  $(up_1 w, p_2, \dots, p_n) \vdash_\Gamma^i (u' q_1 w', q_2, \dots, q_n)$ , where  $u, u' \in \Gamma_1^*$ ,  $w, w' \in \Sigma^*$ ,  $p_i, q_i \in Q_i$ ,  $1 \leq i \leq n$ , and  $\langle p_1 p_2 \dots p_n \rangle, \langle q_1 q_2 \dots q_n \rangle \in Q$ .

PROOF. The proof is made by induction on  $i \geq 0$ .

*Basis:* For  $i = 0$ , we have that

$$u\langle p_1 p_2 \dots p_n \rangle w \vdash_M^0 u\langle p_1 p_2 \dots p_n \rangle w$$

implies  $(up_1 w, p_2, \dots, p_n) \vdash_\Gamma^0 (up_1 w, p_2, \dots, p_n)$ , so the claim holds for  $i = 0$ .

*Induction Hypothesis:* Suppose that the claim holds for all  $0 \leq i \leq k$ , for some  $k \geq 0$ .

*Induction Step:* If

$$uA\langle p_1 p_2 \dots p_n \rangle aw \vdash_M ux \langle o_1 o_2 \dots o_n \rangle w \vdash_M^k u' \langle q_1 q_2 \dots q_n \rangle w',$$

then there exists a rule  $A\langle p_1 p_2 \dots p_n \rangle a \rightarrow x \langle o_1 o_2 \dots o_n \rangle \in R$ , which implies that there must exist words  $p_1 p_2 \dots p_n, o_1 o_2 \dots o_n \in \Psi \cup F_1 F_2 \dots F_n$ , and rules  $Ap_1 a \rightarrow x o_1 \in R_1$ ,  $p_j \rightarrow o_j \in R_j$ ,  $2 \leq j \leq n$ , in  $\Gamma$ , so

$$(uAp_1 aw, p_2, \dots, p_n) \vdash_\Gamma (ux o_1 w, o_2, \dots, o_n) \vdash_\Gamma^k (u' q_1 w', q_2, \dots, q_n),$$

and the claim holds for  $k + 1$ , too. Therefore, Claim 18 holds.  $\square$

From Claim 17 and Claim 18, it immediately follows that for every  $w \in \Sigma^*$ ,

$$\begin{aligned} (S_1 s_1 w, s_2, \dots, s_n) \vdash_\Gamma^* (\gamma f_1, f_2, \dots, f_n) & \text{ iff } S_1 \langle s_1 s_2 \dots s_n \rangle w \vdash_M^* \gamma \langle f_1 f_2 \dots f_n \rangle; \\ (S_1 s_1 w, s_2, \dots, s_n) \vdash_\Gamma^* (q_1, q_2, \dots, q_n) & \text{ iff } S_1 \langle s_1 s_2 \dots s_n \rangle w \vdash_M^* \langle q_1 q_2 \dots q_n \rangle; \\ (S_1 s_1 w, s_2, \dots, s_n) \vdash_\Gamma^* (f_1, f_2, \dots, f_n) & \text{ iff } S_1 \langle s_1 s_2 \dots s_n \rangle w \vdash_M^* \langle f_1 f_2 \dots f_n \rangle; \end{aligned}$$

where  $\gamma \in \Gamma_1^*$ ,  $q_i \in Q_i$ ,  $f_i \in F_i$ ,  $1 \leq i \leq n$ . Therefore,  $L(M)_x = L(\Gamma)_x$ , and Lemma 16 holds.  $\square$

**Lemma 19** *Let  $x \in \{f, \varepsilon, f\varepsilon\}$  and  $n \geq 1$ . For every PDA,  $M$ , there exists an  $\text{SCAS}_n$ ,  $\Gamma = (M_1, M_2, \dots, M_n, \Psi)$ , where  $M_1$  is a PDA, and  $M_2$  through  $M_n$  are FAs, such that  $L(\Gamma)_x = L(M)_x$ .*

PROOF. Let  $x \in \{f, \varepsilon, f\varepsilon\}$ . Let  $M = (Q, \Sigma, \Gamma_M, R, s, S, F)$  be a PDA. For  $n = 1$ ,  $\Gamma = (M, Q)$ , and the lemma holds immediately. For some  $n \geq 2$ , we construct an  $\text{SCAS}_n$ ,

$$\Gamma = (M_1, M_2, \dots, M_n, \Psi),$$

where  $M_1$  is a PDA, and  $M_i$  is an FA, for all  $2 \leq i \leq n$ , such that  $L(\Gamma)_x = L(M)_x$ , in the following way.

1. Set  $M_1 = M$ .
2. For every  $2 \leq i \leq n$ , set  $M_i = (\{s_i\}, \Sigma, \{s_i \rightarrow s_i\}, s_i, \{s_i\})$ .
3. Set  $\Psi = Q\{s_2 s_3 \dots s_n\}$ .

Thus, for every  $w \in \Sigma^*$ ,

$$\begin{aligned} (Ssw, s_2, \dots, s_n) \vdash_{\Gamma}^* (\gamma f, s_2, \dots, s_n) & \text{ iff } Ssw \vdash_M^* \gamma f; \\ (Ssw, s_2, \dots, s_n) \vdash_{\Gamma}^* (q, s_2, \dots, s_n) & \text{ iff } Ssw \vdash_M^* q; \\ (Ssw, s_2, \dots, s_n) \vdash_{\Gamma}^* (f, s_2, \dots, s_n) & \text{ iff } Ssw \vdash_M^* f; \end{aligned}$$

where  $\gamma \in \Gamma_M^*$ ,  $q \in Q$ ,  $f \in F$ . Therefore,  $L(\Gamma)_x = L(M)_x$ , and Lemma 19 holds.  $\square$

**Theorem 20** *For every  $L \in \mathbf{CF}$ , there is an  $\text{SCAS}_n \Gamma$  containing a PDA and  $n - 1$  FAs as its components such that  $L = L(\Gamma)_x$ , where  $n \geq 1$  and  $x \in \{f, \varepsilon, f\varepsilon\}$ .*

PROOF. This theorem follows from Lemma 16 and Lemma 19.  $\square$

**Theorem 21** *For every  $L \in \mathbf{LIN}$ , there is an  $\text{SCAS}_n \Gamma$  containing a one-turn PDA and  $n - 1$  FAs as its components such that  $L = L(\Gamma)_x$ , where  $n \geq 1$  and  $x \in \{f, \varepsilon, f\varepsilon\}$ .*

PROOF. Let  $L \in \mathbf{LIN}$  and  $n \geq 1$ . Let  $x \in \{f, \varepsilon, f\varepsilon\}$ . Then, there exists a one-turn PDA,  $M$ , such that  $L$  is accepted by  $M$ . From  $M$ , as is demonstrated in Lemma 19, construct an  $\text{SCAS}_n \Gamma$ , such that  $L(M)_x = L(\Gamma)_x$ . Thus,  $\Gamma$  has only one PDA as its component, which must be a one-turn PDA.

Conversely, let  $\Gamma$  be an  $\text{SCAS}_n$  with at most one one-turn PDA and arbitrary number of FAs as its components. Let  $x \in \{f, \varepsilon, f\varepsilon\}$ . By Lemma 16, there exists a PDA,  $M$ , such that  $L(\Gamma)_x = L(M)_x$ . By Definition 1, if only one turn is possible to perform in some PDA component of  $\Gamma$ , then no more than one turn is possible to perform in  $\Gamma$ . Therefore, from the construction of  $M$  from  $\Gamma$  in Lemma 16, it follows that  $M$  must be a one-turn PDA, which implies that  $L(M)_x \in \mathbf{LIN}$ .  $\square$

**Corollary 22** *For every  $L \in \mathcal{L}(\text{dPDA})_x$ , where  $x \in \{f, \varepsilon, f\varepsilon\}$ , there is an  $\text{dSCAS}_n \Gamma$  containing a PDA and  $n - 1$  FAs as its components such that  $L = L(\Gamma)_y$ , where  $n \geq 1$  and  $y \in \{f, \varepsilon, f\varepsilon\}$ .*

PROOF. Observe that both Lemma 16 and Lemma 19 also hold for deterministic variants of  $\text{SCAS}_n$ s and PDAs (see their proofs).  $\square$

## 5. Concluding remarks

The result stating that state-synchronized automata systems are computational complete is expectable, because Theorem 1 says that they can be simply transformed into well-known computational complete model. Theorem 12 demonstrates that even

nondeterministic state-synchronized automata systems with one-turn pushdown automata are computational complete. For the deterministic case, we state the following conjecture, which we have not been able to verify rigorously, however.

**Conjecture 1** *Let  $n \geq 2$ . Then, there exists a recursively enumerable language that cannot be accepted by any  $\text{dSCAS}_n$  such that all its components are one-turn PDAs.*  $\square$

## Acknowledgements

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), the TAČR grant TE01020415, and the BUT grant FIT-S-14-2299.

## 6. References

- [1] Čermák M., Meduna A., *n-Accepting restricted pushdown automata systems*. In: *13th International Conference on Automata and Formal Languages*. Computer and Automation Research Institute, Hungarian Academy of Sciences, Nyíregyháza, 2011, pp. 168–183.
- [2] Chomsky N., *Three models for the description of language*. IRE Transactions on Information Theory, 1956, 2(3), pp. 113–124.
- [3] Csuhaaj-Varjú E., Dassow J., Kelemen J., Păun Gh., *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach, London, 1994.
- [4] Csuhaaj-Varjú E., Martín-Vide C., Mitrana V., Vaszil G., *Parallel communicating pushdown automata systems*. International Journal of Foundations of Computer Science, 2000, 11(4), pp. 631–650.
- [5] Harrison M.A., *Introduction to Formal Language Theory*. Addison-Wesley, Boston, 1978.
- [6] Kučera J., *On state-synchronized automata systems*. In: Dražanský M., Orság F. (eds.), *Proceedings of the 19th Conference STUDENT EEICT 2013*, Faculty of Information Technology Brno University of Technology, Brno, 2013, pp. 216–218.
- [7] Martín-Vide C., Mateescu A., Mitrana V., *Parallel finite automata systems communicating by states*. International Journal of Foundations of Computer Science, 2002, 13(5), pp. 733–749.



- [8] Meduna A., *Automata and Languages: Theory and Applications*. Springer, London, 2000.
- [9] Meduna A., *Simultaneously one-turn two-pushdown automata*. International Journal of Computer Mathematics, 2003(80), 2003, pp. 679–687.
- [10] Meduna A., Lukáš R., *Multigenerative grammar systems*. Schedae Informaticae, 2006, pp. 175–188.