International Journal of Foundations of Computer Science
Vol. 26, No. 5 (2015) 643–666

DOI: 10.1142/S0129054115500367

Asynchronous Parallel Communicating Systems of Pushdown Automata*

Friedrich Otto
Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

Received 16 December 2014
Accepted 9 March 2015
Communicated by Erzsébet Csuhaj-Varjú

We introduce *asynchronous* variants of the parallel communicating systems of pushdown automata of Csuhaj-Varjú *et al.* These are obtained by using a *response symbol* in addition to the usual *query symbols*. Our main result states that centralized asynchronous parallel communicating systems of pushdown automata of degree n that work in returning mode have exactly the same expressive power as n-head pushdown automata. This holds in the nondeterministic as well as in the deterministic case. In addition, it is shown that the class of binary relations that is computed by centralized asynchronous parallel communicating systems of pushdown automata of degree two that are working in returning mode coincides with the class of pushdown relations.

Keywords: Parallel communicating system of pushdown automata; asynchronous PC system; multi-head pushdown automaton; pushdown relation.

1. Introduction

Parallel communicating grammar systems, or *PC grammar systems* for short, have been invented to realize the so-called *class room model* of cooperation [10]. Here a group of experts, modelled by grammars, work together in order to produce a document, that is, an output word. These experts work on their own, but synchronously, and they exchange information on request.

In the literature many different types and variants of PC grammar systems have been studied (see, e.g., [10, 12]). The notion of PC system has also been carried over to various types of automata. For example, *PC systems of finite automata* that communicate by states have been introduced in [17] and various modes of operation have been studied for these systems (see, e.g., [4, 7]). Here we are interested in the

*Some of the results presented here have been announced at the 7th International Conference on Language and Automata Theory and Applications, LATA 2013, that took place in Bilbao, Spain, in April 2013. An extended abstract appeared in the proceedings of that conference (see below).

PC systems of pushdown automata introduced in [11], which we will modify into *asynchronous PC systems of pushdown automata*. In a PC system of pushdown automata, a finite number n of pushdown automata, say $A_1, \ldots, A_n$, work in parallel in a synchronous way, where the number n of components is called the *degree* of the PC system. If one of these pushdown automata, say A_i, encounters a special *query symbol* as the topmost symbol on its pushdown store, say K_j, then a *communication step* takes place: the symbol K_j on the top of the pushdown of A_i is replaced by the complete pushdown contents of the pushdown automaton A_j, provided that the topmost symbol of it is not itself a query symbol. The PC system is said to work in *returning mode*, if by this communication step the contents of the pushdown of A_j is reset to its initial symbol Z_j; otherwise, it is said to work in *nonreturning mode*. If there is only one component, called the *master* of the system, that can use query symbols, then the PC system is called *centralized*.

A *multi-head pushdown automaton* is a pushdown automaton with a finite-state control, a pushdown store, and an input tape on which a fixed finite number of one-way read-only input heads operate [9, 14]. These automata are quite powerful, e.g., they accept all finite intersections of context-free languages, and accordingly the *emptiness problem* for them is undecidable. Nevertheless, they have received renewed attention only recently in the area of formal verification, as it turned out that the class of languages accepted by multi-head pushdown automata is *perfect modulo bounded languages* [13], that is, this class of languages is closed under Boolean operations relative to every bounded language, and the emptiness problem is decidable relative to every bounded language.

It is known that a centralized PC system of pushdown automata of degree n that is working in returning mode can simulate an n-head pushdown automaton [11]. In [3], it was claimed that also conversely, centralized PC systems of pushdown automata of degree n working in returning mode can be simulated by n-head pushdown automata, but it was shown in [18] that the proof given in [3] is incorrect. In fact, it was established by Petersen that every recursively enumerable language is accepted by a centralized PC system of pushdown automata of degree two that is working in returning mode [20]. It follows that multi-head pushdown automata are strictly weaker than centralized PC systems of pushdown automata that work in returning mode.

As observed in [18] it is the inherent synchronization of the various components of a PC system of pushdown automata that makes these systems so powerful. This is further exemplified in [20], as the proofs given in that paper make use of this synchronous behaviour in essential ways. Here we do away with this synchronous behaviour by defining *asynchronous PC systems of pushdown automata*, abbreviated as APCPDA. These systems are obtained by introducing an additional *response symbol*. Assume that $\mathcal{A}$ is an APCPDA of degree n with components $A_1, \ldots, A_n$. If one of these pushdown automata, say A_i, encounters a special *query symbol* as the topmost symbol on its pushdown store, say K_j, then it wishes to perform a communication (see above). However, this is possible only if the pushdown automaton A_j has the special *response symbol* R as the topmost symbol on its pushdown. If this is the case, then the reponse symbol R is first removed from the top of the pushdown of A_j, and then the symbol K_j on the top of the pushdown of A_i is replaced by the pushdown contents of A_j (without the response symbol R). In addition, when working in *returning mode*, then the contents of the pushdown of A_j is reset to its initial symbol Z_j. However, if the topmost symbol on the pushdown of A_j is not the special response symbol, then A_i will have to wait until the communication is enabled. Symmetrically, if A_j has the special response symbol R as the topmost symbol on its pushdown, then it has to wait until A_i is ready for the corresponding communication. Thus, in this model the component, the pushdown contents of which is sent to a requesting component, is fully aware of this fact and of the time of this communication. Further, as the sending (the receiving) component has to wait until the receiving (the sending) component is ready for the communication, we see that the various components do not work in complete synchronization anymore. That's why we choose to call this model *asynchronous*. Actually, this idea of introducing asynchronicity has already been used before for PC systems of finite automata [22, 23] and for PC systems of restarting automata [24, 25].

As our main result we will show that the centralized asynchronous PC systems of pushdown automata of degree n that work in returning mode correspond in expressive power exactly to the multi-head pushdown automata of degree n. Actually, from a centralized asynchronous PC system $\mathcal{A}$ of pushdown automata of degree n, one can effectively construct an n-head pushdown automaton B of size $O(\text{size}(\mathcal{A})^{n+1})$ that accepts the language that $\mathcal{A}$ accepts in returning mode, and conversely, from an n-head pushdown automaton B, one can effectively construct a centralized asynchronous PC system $\mathcal{A}$ of pushdown automata of degree n that, when working in returning mode, accepts the language of B, and $\text{size}(\mathcal{A}) = O(\text{size}(B))$. This result also holds for the deterministic case.

Naturally, an asynchronous PC system of pushdown automata of degree n can be used to compute (or rather, accept) an n-ary relation. From our proofs given for the characterization result above it follows easily that the class of relations that are computed by centralized asynchronous PC systems of (deterministic) pushdown automata of degree n that are working in returning mode coincides with the class of relations that are accepted by n-tape (deterministic) pushdown automata. In particular, this implies that the class of binary relations that are computed by centralized asynchronous PC systems of pushdown automata of degree two that are working in returning mode coincides with the well-known class of *pushdown relations* [2, 6].

The paper is structured as follows. In Sec. 2 we restate the definition of PC systems of pushdown automata from [11] in short and recall some of their properties, and we define the asynchronous PC systems of pushdown automata. We also present a detailed example, and we prove that centralized asynchronous PC systems of pushdown automata working in returning mode can be simulated by PC systems of the same type and degree working in nonreturning mode. In the next section we recall the definition of the multi-head pushdown automaton and derive our main result. Then in Sec. 4, we discuss centralized asynchronous PC systems of pushdown automata working in nonreturning mode, and in Sec. 5 we consider non-centralized asynchronous PC systems of pushdown automata. Finally, in Sec. 6 we study the relations that are computed by centralized asynchronous PC systems of pushdown automata that are working in returning mode. The paper closes with a short summary and some open problems.

Notation. For a finite alphabet Σ, we use Σ^* to denote the set of all words of finite length over Σ, which includes the *empty word* ε. The *length* of a word $w \in \Sigma^*$ is denoted by $|w|$. Further, for a set S, we use the notation $|S|$ to denote the *cardinality* of S and 2^S to denote the *power set* of S.

2. PC Systems of Pushdown Automata

First we restate the definition of the PC system of pushdown automata from [11]. However, we slightly modify the presentation of the pushdown automaton itself, as we will use a special *end marker* for its input. Strictly speaking, for a nondeterministic pushdown automaton this end marker is not really required, but it makes the presentation somewhat easier, while for deterministic pushdown automata, it is essential, as it allows such a pushdown automaton to recognize and to communicate the situation that it has read its input completely.

Definition 1. *A* PC system of pushdown automata *is given through a tuple* $\mathcal{A} = (\Sigma, \Gamma, A_1, \dots, A_n, K)$, *where*

- Σ *is a finite input alphabet, and* Γ *is a finite pushdown alphabet,*
- *for each* $1 \le i \le n$, $A_i = (Q_i, \Sigma, \Gamma, ¢, \delta_i, q_i, Z_i, F_i)$ *is a nondeterministic pushdown automaton with a finite set* Q_i *of internal states, an initial state* $q_i \in Q_i$, *and a finite set* F_i *of final states,* $F_i \subseteq Q_i$, *input alphabet* Σ, *pushdown alphabet* Γ, *an end marker* $¢ \notin \Sigma$, *an initial pushdown symbol* $Z_i \in \Gamma$, *and a transition relation* $\delta_i : Q_i \times (\Sigma \cup \{¢, \varepsilon\}) \times \Gamma \to 2^{Q_i \times \Gamma^*}$, *where, for each* $s \in Q_i$, $a \in \Sigma \cup \{¢, \varepsilon\}$ *and* $A \in \Gamma$, $\delta_i(s, a, A)$ *is a finite subset of* $Q_i \times \Gamma^*$,
- *and* $K \subseteq \{K_1, K_2, \dots, K_n\} \subseteq \Gamma$ *is a set of* query symbols.

Here the pushdown automata $A_1, \dots, A_n$ are the *components* of the system $\mathcal{A}$, and the integer n is called the *degree* of this PC system.

Definition 2. *A* configuration *of* $\mathcal{A}$ *is described by a* $3n$*-tuple*

$$(s_1, x_1¢, \alpha_1, s_2, x_2¢, \alpha_2, \dots, s_n, x_n¢, \alpha_n),$$

where, for $1 \le i \le n$,

- $s_i \in Q_i$ *is the current state of component* A_i,
- $x_i \in \Sigma^*$ *is the remaining part of the input which has not yet been read by component* A_i, *and*
- $\alpha_i \in \Gamma^*$ *is the current contents of the pushdown of* A_i, *where the first symbol of* α_i *is the topmost symbol on the pushdown.*

On the set of configurations $\mathcal{A}$ induces a *computation relation* $\vdash^*_{\mathcal{A},r}$ that is the reflexive and transitive closure of the following relation $\vdash_{\mathcal{A},r}$.

Definition 3. *For two configurations*

$$(s_1, x_1¢, c_1\alpha_1, \ldots, s_n, x_n¢, c_n\alpha_n) \text{ and } (p_1, y_1¢, \beta_1, \ldots, p_n, y_n¢, \beta_n),$$

where $c_1, \ldots, c_n \in \Gamma$, *we have*

$$(s_1, x_1¢, c_1\alpha_1, \ldots, s_n, x_n¢, c_n\alpha_n) \vdash_{\mathcal{A},r} (p_1, y_1¢, \beta_1, \ldots, p_n, y_n¢, \beta_n)$$

if one of the following two conditions is satisfied:

(1) $K \cap \{c_1, \ldots, c_n\} = \emptyset$, *and for all* $1 \leq i \leq n$, $x_i = a_i y_i$ *for some* $a_i \in \Sigma \cup \{\varepsilon\}$, $(p_i, \gamma_i) \in \delta_i(s_i, a_i, c_i)$, *and* $\beta_i = \gamma_i\alpha_i$, *or* $x_i = \varepsilon = y_i$, $(p_i, \gamma_i) \in \delta_i(s_i, ¢, c_i)$, *and* $\beta_i = \gamma_i\alpha_i$, *or*

(2)
- $K \cap \{c_1, \ldots, c_n\} \neq \emptyset$,
- *for all* $i \in \{1, \ldots, n\}$ *such that* $c_i = K_{j_i}$ *and* $c_{j_i} \notin K$, $\beta_i = c_{j_i}\alpha_{j_i}\alpha_i$ *and* $\beta_{j_i} = Z_{j_i}$,
- $\beta_r = c_r\alpha_r$ *for all other values of* $r \in \{1, \ldots, n\}$,
- $y_t = x_t$ *and* $p_t = s_t$ *for all* $t \in \{1, \ldots, n\}$.

The steps of form (1) are called *local steps*, as in them each component A_i $(1 \leq i \leq n)$ performs a local step, concurrently, but otherwise independently. The steps of form (2) are called *communication steps*, as in them the topmost symbol K_{j_i} on the pushdown of component A_i is replaced by the complete contents $c_{j_i}\alpha_{j_i}$ of the pushdown of component A_{j_i}, provided that the topmost symbol c_{j_i} is itself not a query symbol. Observe that there could be two (or more) components A_i and $A_{i'}$ such that $c_i = K_{j_i} = c_{i'}$. In this case c_i and $c_{i'}$ are both replaced by the word $c_{j_i}\alpha_{j_i}$. At this time also the pushdown of A_{j_i} is reset to its initial symbol Z_{j_i}. Accordingly, $\mathcal{A}$ is said to work in *returning mode*. If the contents of the pushdown of A_{j_i} remains unchanged by the communication step, then $\mathcal{A}$ is said to work in *nonreturning mode*. The computation relation for this mode is denoted by $\vdash^*_{\mathcal{A}}$. As defined in [11], a PC system of pushdown automata accepts in the following way.

Definition 4. (a) *The* language $L_r(\mathcal{A})$ *that is accepted by* $\mathcal{A}$ *working in returning mode is defined by*

$$L_r(\mathcal{A}) = \{\, w \in \Sigma^* \mid \text{For all } i = 1, \ldots, n, \text{ there are } s_i \in F_i \text{ and } \alpha_i \in \Gamma^* : (q_1, w¢, Z_1, \ldots, q_n, w¢, Z_n) \vdash^*_{\mathcal{A},r} (s_1, ¢, \alpha_1, \ldots, s_n, ¢, \alpha_n) \,\},$$

and the language $L(\mathcal{A})$ that is accepted by $\mathcal{A}$ working in nonreturning mode is defined by

$$L(\mathcal{A}) = \{\, w \in \Sigma^* \mid \textit{For all } i = 1, \ldots, n, \textit{ there are } s_i \in F_i \textit{ and } \alpha_i \in \Gamma^* : \\ (q_1, w¢, Z_1, \ldots, q_n, w¢, Z_n) \vdash^*_{\mathcal{A}} (s_1, ¢, \alpha_1, \ldots, s_n, ¢, \alpha_n) \,\}.$$

(b) *By $\mathcal{L}_r(\mathsf{PCPDA}(n))$ we denote the class of languages that are accepted by PC systems of pushdown automata of degree n working in returning mode, and by $\mathcal{L}(\mathsf{PCPDA}(n))$ we denote the class of languages that are accepted by PC systems of pushdown automata of degree n working in nonreturning mode.*

(c) *A PC system of pushdown automata $\mathcal{A} = (\Sigma, \Gamma, A_1, \ldots, A_n, K)$ is* centralized *if there is only a single component, say A_1, that can use query symbols. In this case, A_1 is called the* master *of the system $\mathcal{A}$. By $\mathcal{L}_r(\mathsf{CPCPDA}(n))$ we denote the class of languages that are accepted by centralized PC systems of pushdown automata of degree n working in returning mode, and by $\mathcal{L}(\mathsf{CPCPDA}(n))$ we denote the class of languages that are accepted by centralized PC systems of pushdown automata of degree n working in nonreturning mode.*

The following result is known on the expressive power of PC systems of pushdown automata.

Theorem 5. [11] *The classes $\mathcal{L}(\mathsf{PCPDA}(2))$ and $\mathcal{L}_r(\mathsf{PCPDA}(3))$ coincide with the class of all recursively enumerable languages.*

Recently the result on PC systems of pushdown automata working in returning mode has been improved considerably by Petersen.

Theorem 6. [20] *The class $\mathcal{L}_r(\mathsf{CPCPDA}(2))$ coincides with the class of all recursively enumerable languages.*

Let $\mathcal{A} = (\Sigma, \Gamma, A_1, \ldots, A_n, K)$ be a centralized PC system of pushdown automata, and let

$$(s_1, x_1¢, K_j\alpha_1, \ldots, s_j, x_j¢, \alpha_j, \ldots) \vdash_{\mathcal{A},r} (s_1, x_1¢, \alpha_j\alpha_1, \ldots, s_j, x_j¢, Z_j, \ldots)$$

be a communication step of $\mathcal{A}$. The component A_1 is *actively* involved in this communication step, while component A_j is only *passively* involved in it, that is, it does not really know about its involvement in this step. It can at best realize its involvement *after* the communication has taken place. On the other hand, A_1 knows exactly how many local steps A_j has executed before the communication step, as A_1 and A_j perform their local steps strictly synchronously.

We now define a new variant of PC systems of pushdown automata, in which we use a *response symbol* in addition to the query symbols. In this way

- we will enable the component A_j to become an active participant in the communication above, and

- we will break the strict synchronicity of local steps between the various components of a PC system.

Definition 7. *An* asynchronous PC system of pushdown automata *is given through a tuple* $\mathcal{A} = (\Sigma, \Gamma, A_1, \ldots, A_n, K, R)$, *where*

- Σ, Γ, $A_1, \ldots, A_n$, *and* K *are defined as in Definition* 1, *and*
- $R \in \Gamma \setminus K$ *is a special* response symbol *such that* $R \neq Z_i$ *for all* $i = 1, \ldots, n$.

Configurations of these systems are defined in the same way as for PC systems of pushdown automata. On the set of configurations $\mathcal{A}$ induces a *computation relation* $\vdash_{\mathcal{A},r}^*$ that is the reflexive and transitive closure of the following relation $\vdash_{\mathcal{A},r}$.

Definition 8. *For two configurations*

$$(s_1, x_1¢, c_1\alpha_1, \ldots, s_n, x_n¢, c_n\alpha_n) \text{ and } (p_1, y_1¢, \beta_1, \ldots, p_n, y_n¢, \beta_n),$$

where $c_1, \ldots, c_n \in \Gamma$, *we have*

$$(s_1, x_1¢, c_1\alpha_1, \ldots, s_n, x_n¢, c_n\alpha_n) \vdash_{\mathcal{A},r} (p_1, y_1¢, \beta_1, \ldots, p_n, y_n¢, \beta_n)$$

if one of the following two conditions is satisfied:

(1) *if there are indices* $i, j \in \{1, \ldots, n\}$ *such that* $c_i = K_j$ *and* $c_j = R$, *then a* communication step *takes place:*

- *for all* $i \in \{1, \ldots, n\}$ *and all* $j \in \{1, \ldots, n\}$ *such that* $c_i = K_j$ *and* $c_j = R$, $\beta_i = \alpha_j\alpha_i$ *and* $\beta_j = Z_j$,
- $\beta_r = c_r\alpha_r$ *for all other values of* $r \in \{1, \ldots, n\}$, *and*
- $y_t = x_t$ *and* $p_t = s_t$ *for all* $t \in \{1, \ldots, n\}$, *or*

(2) *if there are no such indices* i *and* j, *then a* local step *takes place:*

- *for all* $i \in \{1, \ldots, n\}$ *such that* $c_i \in K \cup \{R\}$, $\beta_i = c_i\alpha_i$, $y_i = x_i$ *and* $p_i = s_i$,
- *for all other values of* $i \in \{1, \ldots, n\}$, $x_i = a_i y_i$ *for some* $a_i \in \Sigma \cup \{\varepsilon\}$, $(p_i, \gamma_i) \in \delta_i(s_i, a_i, c_i)$, *and* $\beta_i = \gamma_i\alpha_i$, *or* $x_i = \varepsilon = y_i$, $(p_i, \gamma_i) \in \delta_i(s_i, ¢, c_i)$, *and* $\beta_i = \gamma_i\alpha_i$.

Observe that a *communication step* is executed as soon as there are two components, say A_i and A_j, such that the topmost symbol on the pushdown of A_i is the query symbol K_j, and the topmost symbol on the pushdown of A_j is the response symbol R. In this case, the symbol K_j is replaced by the pushdown contents of A_j without the symbol R, and the pushdown contents of A_j is reset to its initial symbol Z_j. In fact, such communications are carried out for all components that satisfy the above requirements. In particular, it is possible that $c_i = K_j = c_{i'}$ for some $i \neq i'$ and $c_j = R$. Then both, c_i and $c_{i'}$, are replaced by the word α_j. If no communication is possible, then all components that have neither a query symbol nor a response symbol on the top of their pushdowns execute a single step of a local computation, while all those components that have a query symbol or a response symbol at the top of their pushdowns just wait.

Observe that all those components that perform a local step do so synchronously, but as a communication can only take place when both parties (the 'sender' and the 'receiver') are ready for it (that is, each of them has the corresponding symbol on the top of its pushdown), communicating components have no way of knowing the exact number of steps that they have been executing since the previous communication step. In this sense, these types of PC systems of pushdown automata are *asynchronous*.

As in a communication between A_i and A_j as above, the pushdown contents of A_j is reset to the initial symbol Z_j, we say that the above definition describes the *returning mode* of operation for $\mathcal{A}$. If we require that in the above communication, just the response symbol R is deleted from the pushdown of A_j, then we say that $\mathcal{A}$ works in the *nonreturning mode*, which is denoted by the relation $\vdash^*_{\mathcal{A}}$.

The *language* $L_r(\mathcal{A})$ that is accepted by $\mathcal{A}$ working in returning mode and the language $L(\mathcal{A})$ that is accepted by $\mathcal{A}$ working in nonreturning mode are defined as in Definition 4. By $\mathcal{L}_r(\mathsf{APCPDA}(n))$ we denote the class of languages that are accepted by asynchronous PC systems of pushdown automata of degree n working in returning mode, and by $\mathcal{L}(\mathsf{APCPDA}(n))$ we denote the class of languages that are accepted by asynchronous PC systems of pushdown automata of degree n working in nonreturning mode. An asynchronous PC system of pushdown automata $\mathcal{A} = (\Sigma, \Gamma, A_1, \ldots, A_n, K, R)$ is called *centralized* if there is only a single component, say A_1, that can use query symbols. By $\mathcal{L}_r(\mathsf{CAPCPDA}(n))$ we denote the class of languages that are accepted by centralized asynchronous PC systems of pushdown automata of degree n working in returning mode, and by $\mathcal{L}(\mathsf{CAPCPDA}(n))$ we denote the class of languages that are accepted by centralized asynchronous PC systems of pushdown automata of degree n working in nonreturning mode. Finally, an asynchronous PC system of pushdown automata $\mathcal{A} = (\Sigma, \Gamma, A_1, \ldots, A_n, K, R)$ is called *deterministic*, if all its components are deterministic pushdown automata. By $\mathcal{L}_r(\mathsf{APCDPDA}(n))$ we denote the class of languages that are accepted by asynchronous PC systems of deterministic pushdown automata of degree n working in returning mode, and by $\mathcal{L}(\mathsf{APCDPDA}(n))$ we denote the class of languages that are accepted by asynchronous PC systems of deterministic pushdown automata of degree n working in nonreturning mode, and analogously for centralized systems.

Next we present a simple example of an asynchronous PC system of pushdown automata.

Example 9. *Let $f : \{a, b\}^* \to \{a, b\}^*$ be the morphism that is defined by $a \mapsto aa$ and $b \mapsto bb$, and let $L_{\mathrm{mcopy}} = \{ ucf(u) \mid u \in \{a, b\}^* \}$, that is, L_{mcopy} is a modified version of the copy language, and it is easily seen that this language is not even growing context-sensitive (see, e.g., [5]). We now present a centralized asynchronous PC system of pushdown automata that accepts this language in returning mode. Our system will consist of two components, A_1 and A_2. Given an input word of the form $w = ucv$, where $u, v \in \{a, b\}^*$, component A_2 will read the syllable u letter by letter, and for each letter, it will put a corresponding symbol onto its pushdown together* *with the response symbol R. Component A_1 will skip across the prefix uc and will then request the pushdown content from A_2. It will compare the next two letters from the input to the symbol that was read by A_1. In case of success, it will again request the pushdown content from A_2. In this way A_1 checks whether $v = f(u)$ holds.*

So let $\mathcal{A} = (\Sigma, \Gamma, A_1, A_2, \{K_2\}, R)$ be the centralized asynchronous PC system of degree 2 that contains the components $A_1 = (Q_1, \Sigma, \Gamma, ¢, \delta_1, p_1, Z_1, F_1)$ and $A_2 = (Q_2, \Sigma, \Gamma, ¢, \delta_2, q_1, Z_2, F_2)$, where

- $Q_1 = \{p_1, p_2, p_a, p_b, p_3\}$ *and* $Q_2 = \{q_1, q_2, q_3\}$,
- $F_1 = \{p_3\}$ *and* $F_2 = \{q_3\}$,
- $\Sigma = \{a, b, c\}$ *and* $\Gamma = \{Z_1, Z_2, A, B, C, K_2, R\}$, *and*
- *the transition relations are defined as follows:*

(1) $\delta_1(p_1, a, Z_1) = \{(p_1, Z_1)\}$,	(10) $\delta_2(q_1, a, Z_2) = \{(q_1, RAZ_2)\}$,
(2) $\delta_1(p_1, b, Z_1) = \{(p_1, Z_1)\}$,	(11) $\delta_2(q_1, b, Z_2) = \{(q_1, RBZ_2)\}$,
(3) $\delta_1(p_1, c, Z_1) = \{(p_2, K_2 Z_1)\}$,	(12) $\delta_2(q_1, c, Z_2) = \{(q_2, RCZ_2)\}$,
(4) $\delta_1(p_2, a, A) = \{(p_a, \varepsilon)\}$,	(13) $\delta_2(q_2, a, Z_2) = \{(q_2, Z_2)\}$,
(5) $\delta_1(p_2, b, B) = \{(p_b, \varepsilon)\}$,	(14) $\delta_2(q_2, b, Z_2) = \{(q_2, Z_2)\}$,
(6) $\delta_1(p_a, a, Z_2) = \{(p_2, K_2)\}$,	(15) $\delta_2(q_2, ¢, Z_2) = \{(q_3, Z_2)\}$,
(7) $\delta_1(p_b, b, Z_2) = \{(p_2, K_2)\}$,	(16) $\delta_2(q_3, ¢, Z_2) = \{(q_3, Z_2)\}$.
(8) $\delta_1(p_2, ¢, C) = \{(p_3, \varepsilon)\}$,	
(9) $\delta_1(p_3, ¢, Z_2) = \{(p_3, Z_2)\}$,	

On input abcaabb, the system $\mathcal{A}$ executes the following computation:

$$
\begin{array}{llll}
(p_1, abcaabb¢, Z_1, q_1, abcaabb¢, Z_2) & \vdash_{\mathcal{A},r} & (p_1, bcaabb¢, Z_1, q_1, bcaabb¢, RAZ_2) & \vdash_{\mathcal{A},r} \\
(p_1, caabb¢, Z_1, q_1, bcaabb¢, RAZ_2) & \vdash_{\mathcal{A},r} & (p_2, aabb¢, K_2 Z_1, q_1, bcaabb¢, RAZ_2) & \vdash_{\mathcal{A},r} \\
(p_2, aabb¢, AZ_2 Z_1, q_1, bcaabb¢, Z_2) & \vdash_{\mathcal{A},r} & (p_a, abb¢, Z_2 Z_1, q_1, caabb¢, RBZ_2) & \vdash_{\mathcal{A},r} \\
(p_2, bb¢, K_2 Z_1, q_1, caabb¢, RBZ_2) & \vdash_{\mathcal{A},r} & (p_2, bb¢, BZ_2 Z_1, q_1, caabb¢, Z_2) & \vdash_{\mathcal{A},r} \\
(p_b, b¢, Z_2 Z_1, q_2, aabb¢, RCZ_2) & \vdash_{\mathcal{A},r} & (p_2, ¢, K_2 Z_1, q_2, aabb¢, RCZ_2) & \vdash_{\mathcal{A},r} \\
(p_2, ¢, CZ_2 Z_1, q_2, aabb¢, Z_2) & \vdash_{\mathcal{A},r} & (p_3, ¢, Z_2 Z_1, q_2, abb¢, Z_2) & \vdash_{\mathcal{A},r} \\
(p_3, ¢, Z_2 Z_1, q_2, bb¢, Z_2) & \vdash_{\mathcal{A},r} & (p_3, ¢, Z_2 Z_1, q_2, b¢, Z_2) & \vdash_{\mathcal{A},r} \\
(p_3, ¢, Z_2 Z_1, q_2, ¢, Z_2) & \vdash_{\mathcal{A},r} & (p_3, ¢, Z_2 Z_1, q_3, ¢, Z_2), &
\end{array}
$$

that is, $abcaabb \in L_r(\mathcal{A})$. Notice how, after pushing the response symbol onto its pushdown, A_2 must wait until A_1 pushes the corresponding query symbol K_2 onto its pushdown. It is now easily checked that $L_r(\mathcal{A}) = L_{\mathrm{mcopy}}$ holds. Observe that $\mathcal{A}$ is in fact a deterministic system.

Finally, the *size* of an asynchronous PC system $\mathcal{A} = (\Sigma, \Gamma, A_1, \ldots, A_n, K, R)$ of pushdown automata $A_i = (Q_i, \Sigma, \Gamma, ¢, \delta_i, q_i, Z_i, F_i)$, $1 \leq i \leq n$, is defined by taking

$$\operatorname{size}(\mathcal{A}) = \sum_{i=1}^{n} |Q_i| + |\Sigma| + |\Gamma| + \sum_{i=1}^{n} |\delta_i|, \tag{1}$$

where

$$|\delta_i| = \sum_{\substack{(s',\alpha) \in \delta_i(s,a,A), \text{ where} \\ s \in Q_i, a \in \Sigma \cup \{¢, \varepsilon\}, A \in \Gamma}} (4 + |\alpha|).$$

Thus, size($\mathcal{A}$) corresponds to the length of the description of $\mathcal{A}$. For example, the centralized asynchronous PC system $\mathcal{A}$ of Example 9 has size 102, as $|Q_1| = 5$, $|Q_2| = 3$, $|\Sigma| = 3$, $|\Gamma| = 7$, $|\delta_1| = 43$, and $|\delta_2| = 41$.

In a computation of an asynchronous PC system of pushdown automata, each component realizes its involvement in a communication step. If the system works in nonreturning mode, then in a communication step in which the pushdown contents of A_j is copied to A_i, the response symbol R is removed from the top of the pushdown of the sending component A_j. This can be used by A_j to reset its pushdown to its bottom marker Z_j before it continues with its computation. By exploiting this idea we obtain the following result.

Theorem 10. *Let $n \geq 2$, and let $\mathcal{A} \in \mathsf{CAPCPDA}(n)$. Then one can effectively construct a system $\mathcal{A}' \in \mathsf{CAPCPDA}(n)$ of size $O(\text{size}(\mathcal{A}))$ such that $L(\mathcal{A}') = L_r(\mathcal{A})$. In addition, if $\mathcal{A}$ is deterministic, then so is $\mathcal{A}'$.*

Proof. Let $\mathcal{A} = (\Sigma, \Gamma, A_1, \ldots, A_n, K, R)$ be a centralized asynchronous PC system of pushdown automata of degree $n \geq 2$ that is working in returning mode, where $A_i = (Q_i, \Sigma, \Gamma, ¢, \delta_i, q_i, Z_i, F_i)$ for $1 \leq i \leq n$. Without loss of generality we may assume that none of the components A_i ever empties its pushdown store completely. This means in particular that the response symbol R cannot occur as the bottommost symbol on any of the pushdown stores. Further, we may assume that, for all $2 \leq i \leq n$, the bottom marker Z_i can only occur as the bottommost symbol on the pushdown of A_i.

The system $\mathcal{A}' = (\Sigma, \Gamma', A'_1, \ldots, A'_n, K, R)$ is obtained from $\mathcal{A}$ by taking

- $\Gamma' = \Gamma \cup (\Gamma \times \{2, \ldots, n\}) \cup \{R'\}$, where R' is a new symbol,
- $\rho : \Gamma^* \to \Gamma'^*$ is the morphism that is defined by $A \mapsto A$ for all $A \neq R$ and $R \mapsto R'$,
- $A'_1 = (Q_1, \Sigma, \Gamma', ¢, \delta'_1, q_1, Z_1, F_1)$, where δ'_1 is obtained from δ_1 by simply interpreting each pushdown symbol of the form (A, i), where $A \in \Gamma$ and $i \in \{2, \ldots, n\}$, as the symbol A,
- and $A'_i = (Q'_i, \Sigma, \Gamma', ¢, \delta'_i, q_i, Z_i, F_i)$, $2 \leq i \leq n$, is obtained from A_i by taking $Q'_i = Q_i \cup \{ (q, r) \mid q \in Q_i \}$, where r is a new symbol, and by defining δ'_i as follows, where $q \in Q_i$, $a \in \Sigma \cup \{¢, \varepsilon\}$, and $A \in \Gamma \smallsetminus \{R', Z_i\}$:

  – $\delta'_i(q, a, A) = \{ (p, \rho(\alpha)) \mid (p, \alpha) \in \delta_i(q, a, A) \}$,
  – $\delta'_i(q, a, Z_i) = \{ (p, \rho(\alpha Z_i)) \mid (p, \alpha Z_i) \in \delta_i(q, a, Z_i) \} \cup$
  $\{ (p, \rho(\alpha)(B, i)) \mid (p, \alpha B) \in \delta_i(q, a, Z_i), B \neq Z_i \}$,

– $\delta'_i(q, \varepsilon, R') = \{((q, r), R)\}$,
– $\delta'_i((q, r), \varepsilon, A) = \{((q, r), \varepsilon)\}$,
– $\delta'_i((q, r), \varepsilon, Z_i) = \{(q, Z_i)\}$,
– $\delta'_i((q, r), \varepsilon, (A, i)) = \{(q, Z_i)\}$.

The system $\mathcal{A}'$, which is to work in nonreturning mode, simulates $\mathcal{A}$ step by step. However, instead of using the response symbol R in their pushdown operations, the components A'_2 to A'_n use the replacement R'. Also A'_i uses the symbol (B, i) to mark the bottommost position on its pushdown store, if A_i replaces its bottom marker Z_i by the symbol B. When A'_i encounters the symbol R' as the topmost symbol on its pushdown, then it realizes that a communication is called for. Accordingly, it changes from state $q \in Q_i$ into state (q, r), and it replaces the symbol R' by the response symbol R. Now it has to wait for a communication with the master A_1, and it realizes that the communication has taken place by observing that the symbol R has been removed from the top of its pushdown. Then, using state (q, r), it empties its pushdown until it detects the bottommost symbol, which it replaces by the bottom marker Z_i. At that point it changes back to the original state q and continues with the simulation of A_i. It follows that $L(\mathcal{A}')$ coincides with the language $L_r(\mathcal{A})$. Further, we see from its definition that $\mathcal{A}'$ is a deterministic system, if $\mathcal{A}$ is.

Finally, the system $\mathcal{A}'$ is obtained from the system $\mathcal{A}$ by introducing $(n-1)\cdot|\Gamma|+1$ additional pushdown symbols, by replacing the symbol R by R' in all transitions of A_2 to A_n, and by introducing $O(|Q_i| \cdot \Gamma)$ additional transitions for A'_i that replace the current pushdown contents by the bottom marker Z_i. Hence, we see that $\text{size}(\mathcal{A}') \in O(\text{size}(\mathcal{A}))$ holds. □

Thus, we have the following inclusions.

Corollary 11. *For all $n \geq 2$,* (a) $\mathcal{L}_r(\mathsf{CAPCDPDA}(n)) \subseteq \mathcal{L}(\mathsf{CAPCDPDA}(n))$.
(b) $\mathcal{L}_r(\mathsf{CAPCPDA}(n)) \subseteq \mathcal{L}(\mathsf{CAPCPDA}(n))$.

3. Multi-Head Pushdown Automata

Next we repeat in short the definition of the multi-head pushdown automaton, where we follow the presentation in [11] (see also [8, 9]).

Definition 12. *For $n \geq 1$, an n-*head pushdown automaton *is given through a 9-tuple $B = (n, Q, \Sigma, \Gamma, ¢, \delta, q_0, Z_0, F)$, where Q is a finite set of internal states, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is a set of final states, Σ is a finite input alphabet and Γ is a finite pushdown alphabet with initial pushdown symbol $Z_0 \in \Gamma$, the symbol $¢ \notin \Sigma$ is a special end marker, and*

$$\delta : Q \times (\Sigma \cup \{¢, \varepsilon\})^n \times \Gamma \to 2^{Q \times \Gamma^*}$$

is a transition relation such that $\delta(q, a_1, \ldots, a_n, X)$ is a finite subset of $Q \times \Gamma^*$ for all $q \in Q$, $a_1, \ldots, a_n \in \Sigma \cup \{¢, \varepsilon\}$ and $X \in \Gamma$.

If $(q', \alpha) \in \delta(q, a_1, \ldots, a_n, X)$, *then this means that* B, *when in state* q *with* X *as the topmost symbol on its pushdown and reading* a_i *with its* i*-th head* $(1 \leq i \leq n)$, *can change to state* q' *and replace the symbol* X *by the string* α *on the top of the pushdown. In addition, if* $a_i \in \Sigma$, *then head* i *moves one step to the right, and if* $a_i = ¢$ *or* $a_i = \varepsilon$, *then head* i *remains stationary. The* size *of the* n*-head pushdown automaton* B *is defined as*

$$\mathrm{size}(B) = |Q| + |\Sigma| + |\Gamma| + \sum_{\substack{(q', \alpha) \in \delta(q, a_1, \ldots, a_n, X),\ \text{where} \\ q \in Q, a_1, \ldots, a_n \in \Sigma \cup \{¢, \varepsilon\}, X \in \Gamma}} (n + 3 + |\alpha|), \qquad (2)$$

that is, $\mathrm{size}(B)$ *is essentially the length of a description of* B.

A configuration *of* B *is described by an* $(n+2)$*-tuple*

$$(q, x_1¢, \ldots, x_n¢, \alpha) \in Q \times (\Sigma^* \cdot \{¢\})^n \times \Gamma^*,$$

where q *is the current internal state,* x_i *is the remaining part of the input still unread by head* i $(1 \leq i \leq n)$, *and* α *is the current contents of the pushdown, where the first symbol of* α *is the topmost symbol. By* $\vdash_B$ *we denote the single-step computation relation that* B *induces on its set of configurations, and* $\vdash_B^*$ *is its reflexive and transitive closure. If* δ *is a partial function* $\delta : Q \times (\Sigma \cup \{¢, \varepsilon\})^n \times \Gamma \to Q \times \Gamma^*$ *such that the induced transition relation on configurations of* B *is a function, then* B *is a* deterministic n*-head pushdown automaton.*

The language $L(B)$ *accepted by* B *is now defined as*

$$L(B) = \{\, w \in \Sigma^* \mid \text{There exist } s \in F \text{ and } \alpha \in \Gamma^* \text{ such that } (q_0, w¢, \ldots, w¢, Z_0) \vdash_B^* (s, ¢, \ldots, ¢, \alpha) \,\}.$$

In [11] it is shown that each language that is accepted by an n-head pushdown automaton is also accepted by some centralized PC system of pushdown automata of degree n that is working in returning mode. Here we carry this result over to asynchronous PC systems of pushdown automata.

Theorem 13. *Let* $n \geq 2$, *and let* B *be an* n*-head pushdown automaton. Then one can effectively construct a system* $\mathcal{A} \in \mathsf{CAPCPDA}(n)$ *of size* $O(\mathrm{size}(B))$ *such that* $L_r(\mathcal{A}) = L(B)$. *In addition, if* B *is deterministic, then so is* $\mathcal{A}$.

Proof. Let $B = (n, Q, \Sigma, \Gamma, ¢, \delta, q_0, Z_1, F)$ be an n-head (deterministic) pushdown automaton for some $n \geq 2$. Without loss of generality we may assume that Σ and Γ are disjoint alphabets. From B we now construct a centralized asynchronous PC system of (deterministic) pushdown automata $\mathcal{A} = (\Sigma, \Gamma_1, A_1, A_2, \ldots, A_n, K, R)$ of degree n that simulates B. This simulation will proceed as follows: first the master requests the symbols from all the other components that are currently under their input heads, and it stores this information within its finite-state control. Then based on this information, it can determine (deterministically) the next step of B that it must simulate. In this way we will be able to simulate B by a centralized asynchronous PC system of (deterministic) pushdown automata. Accordingly, the PC system $\mathcal{A}$ is defined as follows:

- $K = \{K_2, \ldots, K_n\}$ and $\Gamma_1 = \Sigma \cup \Gamma \cup K \cup \{R\} \cup \{Z_2, \ldots, Z_n\}$, where $K_2, \ldots, K_n, R, Z_2, \ldots, Z_n$ are $2n-1$ new symbols,
- $A_1 = (Q_1, \Sigma, \Gamma_1, ¢, \delta_1, q_0^{(1)}, Z_1, F_1)$ and
- $A_i = (Q_i, \Sigma, \Sigma \cup \{Z_i, R\}, ¢, \delta_i, q_0^{(i)}, Z_i, F_i)$, $i = 2, \ldots, n$. Here
  - $Q_1 = \{q_0\} \cup \{\, q_{[p,\mu_1,\ldots,\mu_n]} \mid p \in Q, \mu_1, \ldots, \mu_n \in \Sigma \cup \{¢, \perp, \perp'\}\,\}$, where $\perp$ and $\perp'$ are two new symbols, and
  - $Q_i = \{q_0^{(i)}\}$ for $i = 2, \ldots, n$,
  - $q_0^{(1)} = q_0$, $F_1 = \{\, q_{[p,¢,\ldots,¢]} \mid p \in F\,\}$, and $F_i = \{q_0^{(i)}\}$, $i = 2, \ldots, n$,
  - the transition relation δ_1 is given by the following description, where $p \in Q$, $a_1, \ldots, a_n \in \Sigma \cup \{¢\}$, $A \in \Gamma$, $i \geq 2$, and $b_1, \ldots, b_n \in \Sigma \cup \{¢, \perp\}$:

$$
\begin{array}{lrl}
(1) & \delta_1(q_0, a_1, Z_1) = & \{(q_{[q_0,a_1,\perp,\ldots,\perp]}, Z_1)\}, \\
(2) & \delta_1(q_{[p,\perp,b_2,\ldots,b_n]}, a_1, A) = & \{(q_{[p,a_1,b_2,\ldots,b_n]}, A)\}, \\
(3) & \delta_1(q_{[p,a_1,\ldots,a_{i-1},\perp,b_{i+1},\ldots]}, \varepsilon, A) = & \{(q_{[p,a_1,\ldots,a_{i-1},\perp',b_{i+1},\ldots]}, K_i A)\}, \\
(4) & \delta_1(q_{[p,a_1,\ldots,a_{i-1},\perp',b_{i+1},\ldots]}, \varepsilon, a_i) = & \{(q_{[p,a_1,\ldots,a_{i-1},a_i,b_{i+1},\ldots]}, \varepsilon)\}, \\
(5) & \delta_1(q_{[p,a_1,\ldots,a_n]}, \varepsilon, A) = & \{\, (q_{[p',b_1,\ldots,b_n]}, \alpha) \mid (p', \alpha) \\
 & & \in \delta(p, c_1, \ldots, c_n, A), \quad c_j = a_j \text{ and } b_j = \perp, \\
 & & \text{or} \quad c_j = \varepsilon \quad \text{and} \quad b_j = a_j, 1 \leq j \leq n\,\},
\end{array}
$$

  and the other transition relations are defined by

  (6) $\delta_i(q_0^{(i)}, a, Z_i) = \{(q_0^{(i)}, Ra)\}$ for all $a \in \Sigma \cup \{¢\}$, $2 \leq i \leq n$.

In its finite-state control, A_1 remembers the actual state of B and the symbols that are currently under the heads of B. Based on this information and the symbol on the top of its pushdown, A_1 can determine the next step of B, which it then simulates (see (5)). In this step B may consume only some of the symbols $a_1, \ldots, a_n$, as some of its heads may just perform ε-steps. Accordingly, A_1 consumes only those symbols a_i that are consumed (read) by the corresponding heads of B. These symbols are replaced by the symbol $\perp$ within the finite-state control of A_1. Then using the rules (2) to (4), A_1 requests the next symbols these heads will read. After obtaining all the required symbols, A_1 can simulate the next step of B. It now follows easily that $L_r(\mathcal{A}) = L(B)$ holds. In addition, $\mathcal{A}$ is deterministic, if B is.

Finally, the n-head pushdown automaton B is of size

$$\text{size}(B) = |Q| + |\Sigma| + |\Gamma| + \sum_{\substack{(q', \alpha) \in \delta(q, a_1, \ldots, a_n, X), \text{ where} \\ q \in Q, a_1, \ldots, a_n \in \Sigma \cup \{¢, \varepsilon\}, X \in \Gamma}} (n + 3 + |\alpha|),$$

while

$$
\begin{aligned}
\text{size}(\mathcal{A}) &= \textstyle\sum_{i=1}^{n} |Q_i| + |\Sigma| + |\Gamma_1| + \sum_{i=1}^{n} |\delta_i| \\
&= n + (|Q| \cdot (|\Sigma| + 3)^n) + 2|\Sigma| + |\Gamma| + 2n - 1 + |\delta_1| + (n-1) \cdot 6 \cdot (|\Sigma| + 1).
\end{aligned}
$$

As $|Q| \cdot (|\Sigma| + 3)^n \in O(\text{size}(B))$ and $|\delta_1| \in O(\text{size}(B))$, we see that $\text{size}(\mathcal{A}) \in O(\text{size}(B))$ follows. This completes the proof of Theorem 13. $\square$

Also the converse of Theorem 13 holds.

Theorem 14. *Let $n \geq 2$, and let $\mathcal{A}$ be a centralized asynchronous PC system of pushdown automata of degree n that is working in returning mode. Then one can effectively construct an n-head pushdown automaton B of size $O(\text{size}(\mathcal{A})^{n+1})$ such that $L(B) = L_r(\mathcal{A})$. In addition, if $\mathcal{A}$ is deterministic, then so is B.*

Proof. Here we can follow the proof idea of Balan [3], which works correctly now that centralized PC systems of pushdown automata are considered that are asynchronous.

This simulation works as follows. Each of the n heads of the n-head pushdown automaton B will simulate the input head of one of the components of the centralized asynchronous PC system $\mathcal{A}$. First, using head 1, B simulates the master A_1 of the system $\mathcal{A}$ up to the point, where a query symbol K_j occurs as the topmost symbol on the pushdown. Then using head j, B simulates the component A_j using the pushdown of B. Thus, at the time of the communication step, the pushdown contents of B will correspond exactly to the pushdown contents of the master A_1 *after* execution of the communication step. Now for the PC systems of [11] the problem was that B cannot recognize this moment in time. However, as we consider an asynchronous PC system, the component A_j puts the response symbol R onto its pushdown, when it is ready for the communication to take place. Thus, at this moment B can switch back to simulating the master component A_1. Once the simulation of A_1 has been completed successfully, B can simulate the other components, one by one, as no more communication steps will occur.

For describing this construction in detail, let $\mathcal{A} = (\Sigma, \Gamma, A_1, \ldots, A_n, K, R)$ be a centralized asynchronous PC system of pushdown automata $A_i = (Q_i, \Sigma, \Gamma, ¢, \delta_i, q_i, Z_i, F_i)$, $1 \leq i \leq n$, for some $n \geq 2$. Without loss of generality we may assume that none of the components A_i will ever empty its pushdown store completely. From $\mathcal{A}$ we construct an n-head pushdown automaton $B = (n, Q, \Sigma, \Gamma_1, ¢, \delta, q_0, Z_0, F)$ by taking $Q = \{1, \ldots, n\} \times Q_1 \times \cdots \times Q_n$, $q_0 = (1, q_1, q_2, \ldots, q_n)$, and $F = \{n\} \times F_1 \times \cdots \times F_n$, $\Gamma_1 = \Gamma \cup \{\perp\}$, where $\perp$ is a new symbol, and by defining δ as follows:

$$
\begin{array}{lrl}
(1) & \delta((i, p_1, \ldots, p_i, \ldots, p_n), \varepsilon, \ldots, a_i, \ldots, \varepsilon, A) = & \{ ((i, p_1, \ldots, p_i', \ldots, p_n), \alpha) \\
 & & \mid (p_i', \alpha) \in \delta_i(p_i, a_i, A) \}, \\
(2) & \delta((1, p_1, \ldots, p_j, \ldots, p_n), \varepsilon, \ldots, \varepsilon, \perp) = & \{((1, p_1, \ldots, p_j, \ldots, p_n), \varepsilon)\}, \\
(3) & \delta((1, p_1, \ldots, p_j, \ldots, p_n), \varepsilon, \ldots, \varepsilon, K_j) = & \{((j, p_1, \ldots, p_j, \ldots, p_n), Z_j\perp)\}, \\
(4) & \delta((j, p_1, \ldots, p_j, \ldots, p_n), \varepsilon, \ldots, \varepsilon, R) = & \{((1, p_1, \ldots, p_j, \ldots, p_n), \varepsilon)\}, \\
(5) & \delta((1, s_1, p_2, \ldots, p_n), ¢, \varepsilon, \ldots, \varepsilon, X) = & \{((2, s_1, p_2, \ldots, p_n), Z_2\perp)\}, \\
(6) & \delta((2, s_1, s_2, p_3, \ldots, p_n), ¢, ¢, \varepsilon, \ldots, \varepsilon, X) = & \{((3, s_1, s_2, p_3, \ldots, p_n), Z_3\perp)\}, \\
 & \vdots & \\
(7) & \delta((n-1, s_1, \ldots, s_{n-1}, p_n), ¢, \ldots, ¢, \varepsilon, X) = & \{((n, s_1, \ldots, s_{n-1}, p_n), Z_n\perp)\},
\end{array}
$$

where $p_i \in Q_i$, $s_i \in F_i$, and $a_i \in \Sigma \cup \{¢, \varepsilon\}$, $1 \leq i \leq n$, $A \in \Gamma \setminus (K \cup \{R\})$, $X \in \Gamma$, and $j \geq 2$.

Using the instructions of type (1), B simulates a particular component A_i of $\mathcal{A}$. When a query symbol K_j occurs as the topmost symbol on the pushdown while simulating A_1, then B switches to simulating A_j using the corresponding instruction of type (3). During this step the query symbol is replaced by the corresponding bottom marker Z_j of A_j, and a copy of the new symbol $\perp$ is inserted below Z_j to prevent the simulation of A_j to access the pushdown content below Z_j. When the response symbol R occurs as the topmost symbol on the pushdown while B simulates a component A_j for some $j \geq 2$, then it switches back to simulating A_1 using the corresponding instruction of type (4). During this step the symbol R is removed from the top of the pushdown. When the simulation of A_1 encounters an occurrence of the special symbol $\perp$ on the pushdown, then it simply pops this symbol using an instruction of type (2). Finally, when the simulation of A_1 reaches a final state $s_1 \in F_1$ while the corresponding head reads the end marker ¢, then B starts with simulating the final computation of A_2 using instruction (5). Observe that A_2 has not been used before, or that it has been used in order to fullfill a request by A_1. In either case, the simulation starts with the pushdown of A_2 containing only its bottom marker Z_2. Accordingly, in instruction (5), the topmost symbol on the pushdown of B is simply replaced by $Z_2\perp$. Once A_2 reaches a final state $s_2 \in F_2$ while the corresponding head reads the end marker ¢, then A_3 is simulated, and so forth up to A_n. If all components reach final states while reading the end marker ¢, then B reaches a final state $(n, s_1, \ldots, s_n) \in F$ and accepts. It follows that $L(B) = L_r(\mathcal{A})$ holds. In addition, if $\mathcal{A}$ is deterministic, then B can be designed in such a way that it is deterministic, too.

The centralized asynchronous PC system $\mathcal{A}$ is of size

$$\text{size}(\mathcal{A}) = \sum_{i=1}^{n} |Q_i| + |\Sigma| + |\Gamma| + \sum_{i=1}^{n} |\delta_i|,$$

while

$$\begin{aligned}\text{size}(B) &= |Q| + |\Sigma| + |\Gamma_1| + \textstyle\sum_{(q',\alpha)\in\delta(q,a_1,\ldots,a_n,A)} (n+3+|\alpha|)\\ &\leq n \cdot \textstyle\prod_{i=1}^{n} |Q_i| + |\Sigma| + |\Gamma_1| + (\sum_{i=1}^{n} |\delta_i|) \cdot \prod_{i=1}^{n} |Q_i|\\ &\in O(\text{size}(\mathcal{A})^{n+1}).\end{aligned}$$

This completes the proof of Theorem 14. □

In summary we obtain the following characterization concerning the language classes accepted by centralized asynchronous PC systems of pushdown automata working in returning mode.

Corollary 15. *For all* $n \geq 1$,

$$\mathcal{L}_r(\mathsf{CAPCPDA}(n)) = \mathcal{L}(n\text{-}\mathsf{PDA}) \textit{ and } \mathcal{L}_r(\mathsf{CAPCDPDA}(n)) = \mathcal{L}(n\text{-}\mathsf{DPDA}),$$

that is, a language is accepted by a centralized asynchronous PC system of (*deterministic*) *pushdown automata of degree n that is working in returning mode if and only if it is accepted by a* (*deterministic*) *n-head pushdown automaton.*

4. On Centralized Systems Working in Nonreturning Mode

We have seen in Corollary 11 that centralized asynchronous PC systems of pushdown automata working in returning mode can be simulated by centralized asynchronous PC systems of the same type and the same number of components that work in nonreturning mode. However, in nonreturning mode, centralized APCPDA systems are actually much more expressive than in returning mode. Here the following result holds.

Theorem 16. *Each recursively enumerable language is accepted by a centralized asynchronous PC system of pushdown automata of degree two that is working in nonreturning mode, that is,* $\mathcal{L}(\mathsf{CAPCPDA}(2)) = \mathsf{RE}$.

To prove this result we make use of the so-called *two-pushdown automaton* (see, e.g., [15]). Such a two-pushdown automaton has a finite-state control, an input tape with a single head, and two pushdown stores. Essentially it works just like a (standard) pushdown automaton, just that it has two pushdowns that it can use. Since each single-tape Turing machine can be simulated by a two-pushdown automaton, we see that the class of languages that are accepted by the latter coincides with the class RE of all recursively enumerable languages.

Proof of Theorem 16. Let $L \subseteq \Sigma^*$ be a recursively enumerable language. Then there exists a two-pushdown automaton $B = (Q, \Sigma, \Delta, \delta, q_0, Z_0^{(1)}, Z_0^{(2)}, F)$ such that $L(B) = L$ holds. We now construct a centralized asynchronous PC system of pushdown automata $\mathcal{A}$ of degree two that will simulate B in nonreturning mode. Here $\mathcal{A} = (\Sigma, \Gamma, A_1, A_2, \{K_2\}, R)$ works as follows. The master component A_1 will simulate the internal state control of B, its input, and its first pushdown, while the component A_2 will simulate the internal state control of B, its input, and its second pushdown. However, while in each step, B sees the topmost symbols on both its pushdowns, A_1 and A_2 each only see the topmost symbol of one of these pushdowns. Accordingly, a single step of B is simulated as follows by several steps of $\mathcal{A}$:

(1) A_2 guesses a transition of B that is compatible with the state of B stored by A_2 and the topmost symbol on the second pushdown. A_2 stores this guess together with the response symbol R on the top of its pushdown.
(2) A_1 requests the pushdown contents from A_2. It stores the guess made by A_2 in its finite-state control, and then it deletes the other symbols of the pushdown contents of A_2 from its own pushdown. This can easily be realized by taking different versions of the pushdown alphabet for A_1 and A_2.
(3) Now A_1 checks whether the guess made by A_2 is compatible with the current input symbol (if indeed a symbol is to be read) and with its topmost pushdown symbol. If this is the case, then A_1 applies the corresponding transition, in this

way simulating the effect of the guessed step of B on the internal state of B, on the input, and on the first pushdown, otherwise, A_1 just halts and rejects.

(4) After the communication has taken place, A_2 applies the transition of B it chose above, in this way simulating the effect of this step on the internal state of B, on the input, and on the second pushdown. Thereafter, it goes back to (1).

It should be clear that the system $\mathcal{A}$ does indeed simulate B when working in nonreturning mode. Thus, it follows that $L(\mathcal{A}) = L(B) = L$ holds, which completes the proof of Theorem 16. □

Obviously, this result has the following consequence.

Corollary 17. *For all* $n \geq 2$*,* $\mathcal{L}(\mathsf{CAPCPDA}(n)) = \mathsf{RE}$.

It remains to determine the expressive power of centralized asynchronous PC systems of deterministic pushdown automata that work in nonreturning mode. We have no characterization for these systems, but we have the following result.

Theorem 18. *For all* $n \geq 2$*,* $\mathcal{L}_r(\mathsf{CAPCDPDA}(n)) \subsetneq \mathcal{L}(\mathsf{CAPCDPDA}(n))$.

Proof. It remains to show that the inclusions above are proper. For $n \geq 2$, let L_n denote the following language on $\Sigma = \{0, 1, \#, \$\}$:

$$L_n = \{\, w_1\#\ldots\#w_n\$w_n\#\ldots\#w_1 \mid w_1, \ldots, w_n \in \{0,1\}^* \,\}.$$

It is known that the language L_n is accepted by a k-head pushdown automaton if and only if $n \leq \binom{k}{2}$ [8, 9]. Because of Corollary 15, this means that L_n is accepted by a centralized asynchronous PC system of pushdown automata of degree k that is working in returning mode if and only if $n \leq \binom{k}{2}$. This implies that the language

$$L_\infty = \{\, w_1\#\ldots\#w_n\$w_n\#\ldots\#w_1 \mid n \geq 1, w_1, \ldots, w_n \in \{0,1\}^* \,\}$$

is not accepted by any centralized asynchronous PC system of pushdown automata that is working in returning mode. The proof of the theorem can now be completed by establishing the following claim.

Claim. L_∞ is accepted by a centralized asynchronous PC system of deterministic pushdown automata of degree 2 that is working in nonreturning mode.

Proof. A centralized asynchronous PC system $\mathcal{A} = (\Sigma, \Gamma, A_1, A_2, \{K_2\}, R)$ can accept the language L_∞ working in nonreturning mode as follows:

(1) Let $w_1\#\ldots\#w_n\$y_n\#\ldots\#y_1$ be the input given, where $n \geq 1$ and $w_1, \ldots, w_n, y_n, \ldots, y_1 \in \{0,1\}^*$. First A_1 pushes the prefix $w_1\#\ldots\#w_n$ onto its pushdown, which yields the pushdown contents $w_n^R\#\ldots\#w_1^R Z_1$, where w^R denotes the *reversal* of the word w, and on reading the middle marker \$, it puts the query symbol K_2 onto its pushdown.

(2) A_2 skips the prefix $w_1\#\ldots\#w_n\$$. Then it pushes the syllable y_n onto its pushdown, and on reading the separator symbol $\#$, it puts the response symbol R onto its pushdown. This yields the pushdown contents $Ry_n^R Z_2$.
(3) Now a communication in nonreturning mode takes place, which causes the contents $y_n^R Z_2$ of the pushdown of A_2 to be copied onto the pushdown of A_1, where it replaces the query symbol K_2, and the response symbol is deleted from the top of the pushdown of A_2.
(4) After the communication step, A_1 just remembers the last symbol y_{n,m_n} of y_n, deletes the string $y_n^R Z_2$ from its pushdown, and compares the symbol y_{n,m_n} remembered to the symbol that is now topmost on its pushdown. If these symbols do not coincide, then A_1 halts immediately, which means that $\mathcal{A}$ rejects, but if the symbols coincide, then A_1 replaces the topmost symbol on its pushdown by the query symbol K_2. Meanwhile, A_2 has simply replaced the topmost symbol on its pushdown, that is, the symbol y_{n,m_n}, by the response symbol R.
(5) Now again a communication takes place. Proceeding as above, A_1 and A_2 compare the syllables w_n and y_n letter by letter, from right to left. As soon as a mismatch is found, A_1 halts, and therewith, $\mathcal{A}$ halts and rejects. If, however, no mismatch occurs, then both syllables are eventually deleted completely from the two pushdowns. Thereafter, A_1 replaces the symbol $\#$ on top of its pushdown by the query symbol K_2, A_2 pushes the syllable y_{n-1} onto its pushdown, and on reading the separator symbol $\#$, it puts the response symbol R onto its pushdown. Now the syllables w_{n-1} and y_{n-1} are compared to each other, and this continues until all syllables have been compared. As soon as a mismatch is found, A_1 halts, which causes $\mathcal{A}$ to reject. If, however, all syllables can be checked without detecting a mismatch, then A_1 skips the rest of the input, and both, A_1 and A_2 accept on reading the end marker.

From the above description it is clear that $L_\infty \in \mathcal{L}(\mathsf{CAPCDPDA}(2))$ holds. $\square$

5. On Non-Centralized Asynchronous PC Systems of Pushdown Automata

Each recursively enumerable language is accepted by a deterministic two-pushdown automaton B. Now the proof idea of Theorem 16 can be used to simulate B by an asynchronous PC system $\mathcal{A}$ of deterministic pushdown automata of degree two that works in nonreturning mode. Here A_1 first obtains the complete pushdown contents of A_2, in this way determining the topmost symbol on the pushdown of A_2, and then A_2 receives the complete pushdown contents of A_1 to determine its topmost symbol. Thereafter, both components have the complete knowledge of the current configuration of B, and so they can both simulate the next step of B. Observe that here we make essential use of the fact that the system $\mathcal{A}$ is not centralized. Thus, we have the following result.

Theorem 19. *Each recursively enumerable language is accepted by an asynchronous PC system of deterministic pushdown automata of degree two that is working in nonreturning mode, that is,* $\mathcal{L}(\mathsf{APCDPDA}(2)) = \mathsf{RE}$.

Obviously, this result has the following consequences.

Corollary 20. *For all* $n \geq 2$, $\mathcal{L}(\mathsf{APCDPDA}(n)) = \mathcal{L}(\mathsf{APCPDA}(n)) = \mathsf{RE}$.

Concerning non-centralized asynchronous PC systems of pushdown automata that work in returning mode, it was stated in [19] that each recursively enumerable language is accepted by an asynchronous PC system of deterministic pushdown automata of degree three that is working in returning mode. This can be shown by replacing the APCDPDA-system of degree two that works in nonreturning mode in the simulation of a deterministic two-pushdown automaton outlined above by an APCDPDA-system of degee three that works in returning mode. However, meanwhile this result has become obsolete as, recently, Petersen has obtained the following stronger result by using the characterization of recursively enumerable languages by Post machines,

Theorem 21. [21] *Each recursively enumerable language is accepted by an asynchronous PC system of deterministic pushdown automata of degree two that is working in returning mode,*

Obviously, this result has the following consequences.

Corollary 22. *For all* $n \geq 2$, $\mathcal{L}_r(\mathsf{APCDPDA}(n)) = \mathcal{L}_r(\mathsf{APCPDA}(n)) = \mathsf{RE}$.

The diagram in Fig. 1 summarizes our results on the classes of languages that are accepted by the various types of asynchronous PC system of pushdown automata.

$$
\begin{tikzcd}
\mathcal{L}(\mathsf{CAPCPDA}(2)) \arrow[r, equal] & \mathcal{L}(\mathsf{APC(D)PDA}(2)) \arrow[r, equal] & \mathcal{L}_r(\mathsf{APC(D)PDA}(2)) = \mathsf{RE} \\
\mathcal{L}(\mathsf{CAPCDPDA}(n)) \arrow[u, "?"'] & \mathcal{L}(n\text{-}\mathsf{DPDA}) = \mathcal{L}_r(\mathsf{CAPCDPDA}(n)) \arrow[l] \arrow[r] \arrow[u] & \mathcal{L}_r(\mathsf{CAPCPDA}(n)) = \mathcal{L}(n\text{-}\mathsf{PDA}) \arrow[u] \\
\vdots \arrow[u, "?"'] & \vdots \arrow[u] & \vdots \arrow[u] \\
\mathcal{L}(\mathsf{CAPCDPDA}(2)) \arrow[u, "?"'] & \mathcal{L}(2\text{-}\mathsf{DPDA}) = \mathcal{L}_r(\mathsf{CAPCDPDA}(2)) \arrow[l] \arrow[r] \arrow[u] & \mathcal{L}_r(\mathsf{CAPCPDA}(2) = \mathcal{L}(2\text{-}\mathsf{PDA}) \arrow[u] \\
 & \mathsf{DCFL} = \mathcal{L}(\mathsf{DPDA}) \arrow[u] \arrow[r] & \mathcal{L}(\mathsf{PDA}) = \mathsf{CFL} \arrow[u]
\end{tikzcd}
$$

Fig. 1. Hierarchy of classes of languages accepted by various types of asynchronous PC systems of pushdown automata. Arrows denote proper inclusions, while question marks indicate inclusions not known to be proper.

6. Relations Computed by CAPCPDA-Systems Working in Returning Mode

In an asynchronous PC system $\mathcal{A} = (\Sigma, \Gamma, A_1, \ldots, A_n, K, R)$ of degree n, n pushdown automata work in parallel, communicating by exchanging their pushdown contents on request. To accept a language $L \subseteq \Sigma^*$, all components are given the same input word. However, it is quite natural to use $\mathcal{A}$ to accept an n-ary relation $R \subseteq (\Sigma^*)^n$. Here we study the special case that $\mathcal{A}$ is a centralized asynchronous PC system of pushdown automata of degree n that is working in returning mode, which appears to be the msot interesting case due to Corollary 15. As we want to accept n-ary relations of the form $R \subseteq \Sigma_1^* \times \cdots \times \Sigma_n^*$, where Σ_i is the alphabet for the i-th component $(1 \leq i \leq n)$, we have to adjust Definition 7 slightly.

Definition 23. *A* centralized asynchronous PC system of pushdown automata for computing an n-ary relation *is given through a* $(2n+3)$*-tuple* $\mathcal{A} = (\Sigma_1, \ldots, \Sigma_n, \Gamma, A_1, \ldots, A_n, K, R)$, *where*

- Σ_i, $1 \leq i \leq n$, *is a finite* input alphabet, *and* Γ *is a finite* pushdown alphabet,
- $A_1 = (Q_1, \Sigma_1, \Gamma, ¢, \delta_1, q_1, Z_1, F_1)$ *is a pushdown automaton, called the* master *of the system* $\mathcal{A}$,
- $A_i = (Q_i, \Sigma_i, \Gamma, ¢, \delta_i, q_i, Z_i, F_i)$, $2 \leq i \leq n$, *is a pushdown automaton,*
- $K = \{K_2, \ldots, K_n\}$ *is the set of* query symbols *to be used by* A_1 *to request the pushdown contents from* $A_2, \ldots, A_n$,
- *and* R *is the* response symbol *to be used by* $A_2, \ldots, A_n$ *to signal that they are ready for a communication.*

The configurations and the computation relation $\vdash^*_{\mathcal{A},r}$ are defined as in Definition 8. Finally, the *relation* $R_r(\mathcal{A})$ *computed by* $\mathcal{A}$ in returning mode is defined as follows:

$$\begin{aligned} R_r(\mathcal{A}) = \{\, &(u_1, \ldots, u_n) \in \Sigma_1^* \times \cdots \times \Sigma_n^* \mid \forall i = 1, \ldots, n\, \exists s_i \in F_i\, \exists \alpha_i \in \Gamma^* : \\ &(q_1, u_1¢, Z_1, \ldots, q_n, u_n¢, Z_n) \vdash^*_{\mathcal{A},r} (s_1, ¢, \alpha_1, \ldots, s_n, ¢, \alpha_n) \,\}. \end{aligned}$$

By $\mathsf{Rel}_r(\mathsf{CAPCPDA}(n))$ we denote the class of relations that can be computed by centralized asynchronous PC systems of pushdown automata of degree n working in returning mode, and by $\mathsf{Rel}_r(\mathsf{CAPCDPDA}(n))$ we denote the class of relations that can be computed by centralized asynchronous PC systems of deterministic pushdown automata of degree n working in returning mode. We will see below that $\mathsf{Rel}_r(\mathsf{CAPCPDA}(2))$ coincides with the class of *pushdown relations*.

Our first result concerning relations states that all n-ary relations that are accepted by n-tape (deterministic) pushdown automata can be computed by centralized asynchronous PC systems of (deterministic) pushdown automata of degree n that work in returning mode. Here an *n-tape* (*deterministic*) *pushdown automaton* B is defined similar to an n-head (deterministic) pushdown automaton (see Definition 12). However, instead of having n heads that all read from the same input tape, B has n input tapes, each equipped with a single one-way read-only input head [14]. Thus, B accepts an n-ary relation $R \subseteq \Sigma_1^* \times \cdots \times \Sigma_n^*$, where Σ_i is the input alphabet used on the i-th tape ($1 \leq i \leq n$). From the proof of Theorem 13 we immediately obtain the following result.

Corollary 24. *Let $n \geq 2$, and let B be an n-tape pushdown automaton. Then one can effectively construct a system $\mathcal{A} \in \mathsf{CAPCPDA}(n)$ of size $O(\text{size}(B))$ such that $R_r(\mathcal{A})$ coincides with the n-ary relation accepted by B. In addition, if B is deterministic, then so is $\mathcal{A}$.*

Also the converse statement holds, as is easily seen from the proof of Theorem 14.

Corollary 25. *Let $n \geq 2$, and let $\mathcal{A}$ be a centralized asynchronous PC system of pushdown automata of degree n that is working in returning mode and that computes an n-ary relation. Then one can effectively construct an n-tape pushdown automaton B of size $O(\text{size}(\mathcal{A})^{n+1})$ such that B accepts the relation $R_r(\mathcal{A})$. In addition, if $\mathcal{A}$ is deterministic, then so is B.*

Thus, we have the following characterization.

Corollary 26. *For all $n \geq 2$, an n-ary relation is computed by a centralized asynchronous PC system of* (*deterministic*) *pushdown automata of degree n that is working in returning mode if and only if it is accepted by a* (*deterministic*) *n-tape pushdown automaton.*

Finally, a binary relation $R \subseteq \Sigma_1^* \times \Sigma_2^*$ is called a *pushdown relation* if it is accepted by a nondeterministic 2-tape pushdown automaton. By PDR we denote the class of all pushdown relations. Actually, pushdown relations are usually defined using the notion of *pushdown transducer*, which is a nondeterministic pushdown automaton that outputs a word from Σ_2^* in each step (see, e.g., [2, 6]). However, it is easily seen that instead of reading an input word $u \in \Sigma_1^*$ and producing an output word $v \in \Sigma_2^*$ by a pushdown transducer, we can use a nondeterministic 2-tape pushdown automaton that reads u from its first and v from its second input tape, and conversely, each binary relation accepted by a nondeterministic 2-tape pushdown automaton can also be computed by a pushdown transducer [1]. Thus, as a special case of the above characterization, we obtain the following result.

Corollary 27. *A binary relation can be computed by a centralized asynchronous PC system of pushdown automata of degree two working in returning mode if and only if it is a pushdown relation, that is,* $\mathsf{Rel}_r(\mathsf{CAPCPDA}(2)) = \mathsf{PDR}$.

7. Concluding Remarks

We have introduced asynchronous variants of the PC systems of pushdown automata of [11] by using a special response symbol. In this way we obtained a characterization of the language classes defined by n-head (deterministic and nondeterministic) pushdown automata in terms of centralized asynchronous PC systems of pushdown automata that work in returning mode. In fact, we obtained effective transformations from n-head pushdown automata to centralized asynchronous PC systems of pushdown automata and back that increase the size of the system by at most a polynomial function. This may be of interest for bounded verification as explained by Esparza *et al.* in [13]. In that paper it is shown that the *control reachability problem* modulo a bounded expression for *recursive counter machines* is NP-complete by using an encoding by multi-head pushdown automata. However, this encoding can be realized most naturally by centralized asynchronous PC systems of pushdown automata.

In the nondeterministic case, the centralized asynchronous PC systems of pushdown automata accept all recursively enumerable languages, when they work in nonreturning mode. In addition, we could show that also in the deterministic case, centralized asynchronous PC systems of pushdown automata that work in nonreturning mode are more powerful than the systems of the same type that work in returning mode. However, we did not succeed in obtaining a characterization for the class of languages that are accepted by centralized asynchronous PC systems of deterministic pushdown automata that work in nonreturning mode. Given sufficiently many components, do these PC systems accept all recursively enumerable languages?

In our definition a component of a PC system of pushdown automata sends its pushdown contents to each and every component that has the corresponding query symbol as the topmost symbol on its pushdown. It is, however, conceivable that a component may want to choose to which other component it is willing to send its pushdown contents. This could be achieved, for example, by using a set of response symbols $\{R_1, \ldots, R_n\}$ similar to the way in which communication is realized in PC systems of restarting automata as defined in [16, 24, 25]: a communication that sends the pushdown contents of component A_j to A_i can be executed only if the topmost symbol on the pushdown of A_i is the query symbol K_j, and the topmost symbol on the pushdown of A_j is the response symbol R_i. It appears, however, that all our results extend to this type of PC systems.

References

[1] A.V. Aho and J.D. Ullman. Properties of syntax directed translations. *J. Comp. Syst. Sci.*, 3:319–334, 1969.

[2] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume I: Parsing. Prentice-Hall, Englewood Cliffs, N.J., 1972.

[3] M. Balan. Serializing the parallelism in parallel communicating pushdown automata systems. In J. Dassow, G. Pighizzini, and B. Truthe, editors, *DCFS* 2009, volume 3 of *Electronic Proceedings in Theoretical Computer Science*, pages 59–68. 2009.

[4] H. Bordihn, M. Kutrib, and A. Malcher. On the computational capacity of parallel communicating finite automata. *Intern. J. Found. Comput. Sci.*, 23:713–732, 2012.

[5] G. Buntrock and F. Otto. Growing context-sensitive languages and Church-Rosser languages. *Inform. and Comput.*, 141:1–36, 1998.

[6] C. Choffrut and K. Culik. Properties of finite and pushdown transducers. *SIAM J. Comput.*, 12:300–315, 1983.

[7] A. Choudhary, K. Krithivasan, and V. Mitrana. Returning and non-returning parallel communicating finite automata are equivalent. *RAIRO Theor. Inform. and Appl.*, 41:137–145, 2007.

[8] M. Chrobak. Hierarchies of one-way multihead automata languages. *Theor. Comput. Sci.*, 48:153–181, 1986.

[9] M. Chrobak and M. Li. $k+1$ heads are better than k for PDAs. *J. Comp. Syst. Sci.*, 37:144–155, 1988.

[10] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, and G. Păun. *Grammar Systems - A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach, London, 1994.

[11] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana, and G. Vaszil. Parallel communicating pushdown automata systems. *Intern. J. Found. Comput. Sci.*, 11:631–650, 2000.

[12] J. Dassow, G. Păun, and G. Rozenberg. Grammar systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages. Vol.* 2*, Linear Modelling: Background and Applications*, pages 101–154. Springer, Berlin - Heidelberg, 1997.

[13] J. Esparza, P. Ganty, and R. Majumdar. A perfect model for bounded verification. In *LICS* 2012*, Proc.*, pages 285–294. IEEE, 2012.

[14] M.A. Harrison and O.H. Ibarra. Multi-tape and multi-head pushdown automata. *Inform. and Control*, 13:433–470, 1968.

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

[16] N. Hundeshagen, F. Otto, and M. Vollweiler. Transductions computed by PC-systems of monotone deterministic restarting automata. In M. Domaratzki and K. Salomaa, editors, *CIAA* 2010*, Proc.*, Lecture Notes in Computer Science 6482, pages 163–172. Springer, Berlin, 2011.

[17] C. Martín-Vide, A. Mateescu, and V. Mitrana. Parallel finite automata systems communicating by states. *Intern. J. Found. Comput. Sci.*, 13:733–749, 2002.

[18] F. Otto. Centralized PC systems of pushdown automata versus multi-head pushdown automata. In M. Kutrib, N. Moreira, and R. Reis, editors, *DCFS* 2012, Lecture Notes in Computer Science 7386, pages 244–251. Springer, Berlin, 2012.

[19] F. Otto. Asynchronous PC systems of pushdown automata. In A.-H. Dediu, C. Martin-Vide, and B. Truthe, editors, *Language and Automata Theory and Applications, LATA* 2013*, Proc.*, Lecture Notes in Computer Science 7810, pages 456–467. Springer, Berlin, 2013.

[20] H. Petersen. The power of centralized PC systems of pushdown automata. In H. Jürgensen and R. Reis, editors, *DCFS* 2013*, Proc.*, Lecture Notes in Computer Science 8031, pages 241–252. Springer, Heidelberg, 2013.

[21] H. Petersen. A note on pushdown automata systems. In H. Jürgensen, J. Karhumäki, and A. Okhotin, editors, *DCFS* 2014*, Proc.*, Lecture Notes in Computer Science 8614, pages 342–351. Springer, Heidelberg, 2014.

[22] M. Vollweiler. Asynchronous systems of parallel communicating finite automata. In S. Bensch, F. Drewes, R. Freund, and F. Otto, editors, *Fifth Workshop on Non-Classical Models of Automata and Applications* (*NCMA* 2013)*, Proc.*, volume 294 of *books@ocg.at*, pages 243–257. Oesterreichische Computer Gesellschaft, Wien, 2013.

[23] M. Vollweiler. Asynchronous systems of parallel communicating finite automata. *Fund. Inform.*, 136:177–197, 2015.

[24] M. Vollweiler and F. Otto. Systems of parallel communicating restarting automata. In R. Freund, M. Holzer, B. Truthe, and U. Ultes-Nitsche, editors, *Fourth Workshop on*

Non-Classical Models for Automata and Applications (*NCMA* 2012)*, Proc.*, volume 290 of *books@ocg.at*, pages 197–212. Oesterreichische Computer Gesellschaft, Wien, 2012.

[25] M. Vollweiler and F. Otto. Systems of parallel communicating restarting automata. *RAIRO Theor. Inform. and Appl.*, 48:3–22, 2014.