International Journal of Computer Mathematics
Vol. 81, No. 2, February 2004, pp. 141–156

Taylor & Francis
Taylor & Francis Group

TEAMS OF PUSHDOWN AUTOMATA

MAURICE H. TER BEEK[a,*], ERZSÉBET CSUHAJ-VARJÚ[b,†] and VICTOR MITRANA[c,‡]

[a]*Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Via G. Moruzzi 1, 56124 Pisa, Italy;*
[b]*Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende utca 13-17, 1111 Budapest, Hungary;*
[c]*Faculty of Mathematics, University of Bucharest, Str. Academiei 14, 70109 Bucharest, Romania*

(Received 30 May 2003)

We introduce team pushdown automata (PDAs) as a theoretical framework capable of modelling various communication and cooperation strategies in complex, distributed systems. Team PDAs are obtained by augmenting distributed PDAs with the notion of team cooperation or, alternatively, by augmenting team automata with pushdown memory. In a team PDA, several PDAs work as a team on the input word placed on a common one-way input tape. At any moment in time one team of PDAs, each with the same symbol on top of its stack, is active: each PDA in the active team replaces the topmost symbol of its stack and changes state, while the current input symbol is read from the input tape by a common reading head. The teams are formed according to the team cooperation strategy of the team PDA and may vary from one moment to the other. Based on the notion of competence, we introduce a variety of team cooperation strategies. If all stacks are empty when the input word has been completely read, then this word is part of the language accepted by the team PDA. Here we focus on the accepting capacity of team PDA.

Keywords: Pushdown automata; Team cooperation; Competence; Accepting capacity; Team automata

C.R. Categories: F.1.1; F.4.2

1 INTRODUCTION

During the last decade, communication and cooperation strategies in complex, distributed systems have received considerable attention in a number of research areas. Examples include formal languages, parallel computing, distributed computing, collaborative computing, DNA computing, natural language processing, problem solving, human computer interaction, multi-agent systems, cooperative information systems, expert systems, groupware systems, and computer supported cooperative work (CSCW). Along this line of development, numerous attempts were made to provide solid theoretical models for such systems. These models often abstract from concrete data, configurations, and actions, but describe the system solely in terms of *grammars* or *(pushdown) automata*. The *team pushdown automata (PDAs)* we introduce in this article are in many ways a continuation of two such attempts, viz. the *distributed pushdown automaton* model [see, *e.g.*, Refs.1–5] and the *team automaton* model [see, *e.g.*, Refs. 6–8].

* Corresponding author. Fax:+39-050-3152810; E-mail: maurice.terbeek@isti.cnr.it
† E-mail: csuhaj@sztaki.hu
‡ E-mail: mitrana@funinf.cs.unibuc.ro

ISSN 0020-7160 print; ISSN 1029-0265 online
DOI: 10.1080/00207160310001650099

However, team PDAs also borrow ideas from other such attempts, in particular from models introduced in the theory of *grammar systems* [see, *e.g.*, Refs. 9, 10].

Grammar systems consist of a set of grammars which, by interacting according to some protocol of communication and/or cooperation, generate one language. Best known are the sequential *cooperating distributed (CD) grammar systems* and the *parallel communicating grammar systems*. In CD grammar systems [see, *e.g.*, Ref. 11], a set of grammars work together, in turn, by rewriting a common string. The moment in which control is transferred from one grammar to another is determined by the cooperation strategy according to which the CD grammar system operates. The five classic cooperation strategies are $*$, t, $\leq k$, $=k$, and $\geq k$, for some $k \geq 1$, according to which a grammar rewrites the string any number of times, as long as it can, less than k times, exactly k times, and at least k times, respectively. We refer to these five classic cooperation strategies as the *CD strategies*. In Ref. [12] CD grammar systems were recognized as the formal language-theoretic counterpart of blackboard systems, multi-agent systems used within the blackboard model of problem solving [see, *e.g.*, Refs. 13, 14]. The common string models the blackboard containing the current state of the problem solving, the grammars represent the knowledge sources contributing to the problem solving by changing the contents of the blackboard according to their competence, and the CD strategies regulate the control mechanism directing the problem solving by allowing the knowledge sources to access the blackboard.

The concept of work being done in *teams* was introduced in CD grammar systems [see, *e.g.*, Refs. 15–17] by grouping several grammars and using them, in turn, to rewrite a common string in parallel. Such teams are formed automatically or are prescribed, and control is transferred from one team to another according to one of the CD strategies. Since well-structured groups are thought to outperform individuals in a variety of tasks [see, *e.g.*, Ref. 18] the concept of teams has a clear practical motivation. The concept of work being done according to a level of *competence* was introduced in CD grammar systems [see, *e.g.*, Refs. 19–23] by defining the 'cleverness' of a grammar with respect to a particular string as the number of nonterminals of this string it is able to rewrite. This recognizes the practical reality of members of a group possessing different skills.

In Refs. [1, 4] the grammars in CD grammar systems were replaced by (pushdown) automata, leading to *distributed (pushdown) automata*. Their pushdown memories (also known as stacks) can be seen as modelling the memory, or a notebook containing sketches, of the knowledge sources of blackboard systems. A similar operation was carried out in the context of parallel communicating grammar systems in Ref. [24]. Independently from the distributed PDAs of Refs. [1, 4], *multistack PDAs* with CD strategies controlling the use of the stacks were introduced in Ref [2]. These two models thus differ conceptually: in the latter case there is one PDA with n stacks, for some $n \geq 1$, whereas in the former case the n stacks are distributed over n PDAs. Furthermore, the distributed PDAs of Ref. [4] communicate with each other by allowing transitions from states of one PDA to states of another PDA. For convenience, we nevertheless refer to both models as a *CD PDAs*, which is the first model that team PDAs are related to. For the sake of completeness we mention also the *cooperating PDAs* introduced in Ref. [25], which consist of n PDAs – the transitions of each of which can depend on the current states of the others – and but one stack shared by all.

The CD PDAs accept languages similar to the way ordinary PDAs do. Given a number of stacks and a tape with an input word on it, some central unit has a state and two reading heads. One head is fixed and scans the tape from left to right, one symbol at a time. The other head is mobile and inspects the topmost symbol of each stack. Based on some predefined transitions, the central unit reads the current symbol on the tape, replaces the topmost symbol of one of the stacks by a string, and changes state. This procedure is repeated until in accordance with the CD strategy used, the mobile head is moved to another stack. The other stack is

chosen among those for which there exists a predefined transition allowing the central unit to read the current symbol on the tape, replace the topmost symbol of this stack, and change state. The word on the tape is part of the language accepted by a CD PDA accepting by final state if it can be completely read in this way and the central unit has reached a final state. If, on the other hand, the word on the tape can be completely read in this way and all stacks are empty, then this word is part of the language accepted by a CD PDA accepting by empty stack.

Consequently, deterministic and nondeterministic CD PDAs are distinguished. A CD PDA is defined to be nondeterministic if all its constituting PDAs are nondeterministic. Deterministic CD PDAs are defined in different ways in Refs. [1, 2, 4]. In all cases, the constituting PDAs of a deterministic CD PDA obviously are required to be deterministic. This, however, leaves the possiblility of a choice popping up the moment in which the mobile head is moved from one stack to another, since it can be the case that more than one transition exists that can read the current symbol on the tape, replace the topmost symbol of a stack, and change state. In Refs. [1, 4], this nondeterminism is allowed in deterministic CD PDAs. In Ref. [4], consequently, so-called completely deterministic CD PDAs do exclude this nondeterminism by requiring their constituting PDAs to be deterministic and to have pairwise disjoint sets of states. In Ref. [2], finally, this nondeterminism is excluded in deterministic CD PDAs by requiring them to have sets of transitions for which the above choice cannot occur for any input word.

In Ref. [2, 4], the chosen CD strategy has no influence on the accepting capacity of CD PDAs, regardless of the fact whether these CD PDAs are (completely) deterministic or nondeterministic and whether they accept by final state or by empty stack. In Ref. [1] this is not the case, but the exact influence is yet to be settled. In Refs. [2, 4], the accepting capacity of nondeterministic CD PDAs equals that of Turing machines, regardless of the fact whether these CD PDAs accept by final state or by empty stack. In Ref. [4], the accepting capacity of deterministic CD PDAs equals that of Turing machines as well, whereas that of completely deterministic CD PDAs equals that of deterministic PDAs only, regardless of the fact whether these (completely) distributed CD PDAs accept by final state or by empty stack. In Ref. [2], finally, the accepting capacity of deterministic CD PDAs is quite large (for any recursively enumerable language L, $\{\$\}L\{\$\}$ is accepted by a deterministic CD PDA, where \$ is a new symbol) but yet to be settled, while also the exact accepting capacity of nondeterministic CD PDAs is yet to be settled for most of the CD strategies.

We now turn to a description of team automata, which is the second model that team PDAs are related to. Team automata were introduced in Ref. [8] as a formal framework for modelling groupware systems, multi-user software systems used within CSCW [see, *e.g.*, Refs. 26, 27]. The model was inspired by Ref. [18], which conjectures that well-structured groups (called teams) outperform individuals in certain tasks but at the same time calls for models capturing concepts of group behaviour. A more technical source of inspiration was I/O automata [see, *e.g.*, Refs. 28, 29] which in fact are incorporated in the team automata framework [see, *e.g.*, Refs. 6, 7]. Team automata consist of a set of component automata, which interact in a coordinated way by synchronizations of common actions. Component automata differ from ordinary automata by their lack of final states and by the partition of their sets of actions into input, output, and internal actions. Internal actions have strictly local visibility and thus cannot be observed by other components, whereas external actions are observable by other components. These external actions are used for communication among components and comprise both input actions and output actions. Through the partition of their sets of actions and the synchronizations of shared actions, a range of intriguing coordination protocols can be modelled by team automata [see, *e.g.*, Refs. 6, 7]. This shows their usefulness within CSCW [see, *e.g.*, Refs. 6, 30, 31]. The focus of the research on team automata is on their modelling

capacity rather than their accepting capacity. Nevertheless, it was noted in Ref. [6] that their accepting capacity equals the class of prefix-closed regular languages.

We are now ready to sketch our model of team PDAs and discuss its relation with CD PDAs and team automata. Like CD PDAs and team automata, when seen as language accepting devices, team PDAs are comprised of a specific type of automata – PDAs in this case – that by means of a certain strategy accept one language. If we abstract from the alphabet partition, then we thus augment the component automata of team automata with pushdown memories. If we again think of these stacks as memories of the components or as notebooks in which the components can make sketches, then we hereby enhance the modelling capacity of team automata. A team PDA accepts languages similar to the way CD PDAs do. Whereas a CD PDA reads the current symbol on the tape, replaces the topmost symbol of *one* stack, and changes state, a team PDA replaces the topmost symbol of a *team* of stacks. We thus augment CD PDAs with team behaviour. The size of a team is based on the so-called *mode of competence* in which a team PDA accepts languages. These modes of competence are inspired by the CD strategies and result in the topmost symbol of an arbitrary one, less than k, exactly k, or at least k, for some $k \geq 1$, stacks to be replaced. With the added condition that the topmost symbol of each stack of a team must be equal, we model that the components forming a team are equally competent, *i.e.* have the same skills.

In this article, we focus on team PDAs with nondeterministic PDAs and with acceptance by empty stack. Under these assumptions, we show that team PDAs consisting of only two PDAs suffice to generate non-context-free languages, while already team PDAs consisting of three PDAs are able to accept all recursively enumerable languages. It remains an open problem, though, whether these statements hold for all modes of competence. An extended abstract of this article – in which team PDAs are introduced, but no results are proved – will be published as Ref. [32].

The organization of this article is as follows. After the Preliminaries, we define PDAs in Section 3. In Section 4, we define how to form teams of PDAs, followed by the definition of several modes of competence according to which these teams of PDAs can work in Section 5. Their accepting capacity is studied in Section 6, after which this article is concluded with some directions for future work in Section 7.

2 PRELIMINARIES

We assume the reader to be familiar with the basic notions from formal language theory and automata theory, in particular concerning PDAs. For any unexplained notions, we refer the reader to Ref. [33].

We have the following conventions. Set inclusion is denoted by $\subseteq$, whereas $\subset$ denotes a strict inclusion. The set difference of sets V and W is denoted by $V \setminus W$. For a finite set V, its cardinality is denoted by $\#V$. The powerset of a set V, formed by finite parts of V only, is denoted by $\mathcal{P}_f(V)$ and the empty set is denoted by $\emptyset$. For convenience, we sometimes denote the set $\{1, 2, \ldots, n\}$ by $[n]$. Then $[0] = \emptyset$. The empty word is denoted by λ, while the empty word vector $(\lambda, \lambda, \ldots, \lambda)$, its dimension being clear from the context, is denoted by Λ. We consider two languages L_1 and L_2 to be equal if and only if $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$, in which case we simply write $L_1 = L_2$.

The classes of context-free and recursively enumerable languages are denoted by $\mathcal{L}(\mathrm{CF})$ and $\mathcal{L}(\mathrm{RE})$, respectively. The family of context-free languages that do not contain the empty word λ is denoted by $\mathcal{L}(CF - \lambda)$.

A *shuffle* of two words is an arbitrary interleaving of subwords of the original words such that it contains all symbols of both words, like the shuffling of two decks of cards.

Formally, the shuffle of two words $u, v \in \Sigma^*$, denoted by $u \| v$, is defined as $u \| v = \{u_1 v_1 u_2 v_2 \cdots u_n v_n | u_i, v_i \in \Sigma^*, i \in [n], u_1 u_2 \cdots u_n = u, v_1 v_2 \cdots v_n = v\}$. The shuffle of two languages $L_1, L_2 \subseteq \Sigma^*$, denoted by $L_1 \| L_2$, is defined as the union of all shuffles of a word from L_1 and a word from L_2. Thus $L_1 \| L_2 = \{w \in u \| v | u \in L_1, v \in L_2\} = \cup_{u \in L_1, v \in L_2} u \| v$.

It is not difficult to prove that the shuffle operation is commutative and associative, *i.e.* $u \| v = v \| u$ and $(u \| v) \| w = u \| (v \| w)$, for all words $u, v, w \in \Sigma^*$, and likewise for languages.

Finally, the family of shuffles of n languages from language family $\mathcal{L}(F)$, denoted by n-Shuf(F), is defined as n-Shuf$(F) = \{L_1 \| L_2 \| \cdots \| L_n | L_i \in \mathcal{L}(F), i \in [n]\}$. It is well known that the shuffle of n context-free languages need not be a context-free language, for any $n \geq 2$.

3 PUSHDOWN AUTOMATA

A PDA is a sixtuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q^0, Z^0)$ with set Q of states, input alphabet Σ, pushdown alphabet Γ, transition mapping δ: $Q \times \Sigma \cup \{\lambda\} \times \Gamma \to \mathcal{P}_f(Q \times \Gamma^*)$, initial state $q^0 \in Q$, and initial stack content $Z^0 \in \Gamma$. A configuration of a PDA $\mathcal{A}$ is a triple (p, w, β) with $p \in Q$, $w \in \Sigma^*$, and $\beta \in \Gamma^*$, indicating the state that $\mathcal{A}$ is in, the part of the input word that $\mathcal{A}$ still has to read, and the current stack contents. A PDA $\mathcal{A}$ can move from a configuration $(p, aw, z\beta)$ to a configuration $(q, w, \alpha\beta)$ by changing from state p to state q, reading the first letter a (possibly λ) from the remaining part of the input word to be read, removing the topmost symbol Z from its stack, and append the (possibly empty) word α to its stack, whenever $(q, \alpha) \in \delta(p, a, Z)$. Formally, we write $(p, aw, z\beta) \vdash_{\mathcal{A}} (q, w, \alpha\beta)$ whenever $(q, \alpha) \in \delta(p, a, Z)$. If $a = \lambda$, then such a move is called a λ-move, and a PDA without λ-moves is called λ-free. If $\mathcal{A}$ is clear from the context, then we also write $\vdash$ rather than $\vdash_{\mathcal{A}}$. The language accepted by a PDA $\mathcal{A}$, denoted by $L(\mathcal{A})$, is defined as

$$L(\mathcal{A}) = \{w \in \Sigma^* | (q^0, w, Z^0) \vdash^* (q, \lambda, \lambda) \text{ for some } q \in Q\},$$

where $\vdash^*$ is the reflexive and transitive closure of $\vdash$. Two PDAs are said to be equivalent if they recognize the same language. It is known that for each PDA with λ-moves one can effectively construct an equivalent PDA without λ-moves. The family of languages recognized by [λ-free] PDAs is denoted by $\mathcal{L}$(PDA[$-\lambda$]) and it is known that $\mathcal{L}$(PDA[$-\lambda$]) = $\mathcal{L}$(CF[$-\lambda$]).

A 2-stack PDA is a septuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q^0, Z_1^0, Z_2^0)$, in which Q, Σ, Γ, and $q^0 \in Q$ have the same meaning as in a usual PDA. Now, however, there are two stacks, which initially contain Z_1^0 and Z_2^0. Consequently, the transition mapping becomes δ: $Q \times \Sigma \cup \{\lambda\} \times \Gamma^2 \to \mathcal{P}_f(Q \times \Gamma^* \times \Gamma^*)$. A configuration of a 2-stack PDA $\mathcal{A}$ is a quadruple (p, w, β_1, β_2) with $p \in Q$, $w \in \Sigma^*$, and $\beta_1, \beta_2 \in \Gamma^*$. A 2-stack PDA $\mathcal{A}$ can make a move $(p, aw, Z_1\beta_1, Z_2\beta_2) \vdash_{\mathcal{A}} (q, w, \alpha_1\beta_1, \alpha_2\beta_2)$ whenever $(q, \alpha_1, \alpha_2) \in \delta(p, a, Z_1, Z_2)$. If $a = \lambda$, then such a move is called a λ-move, and a 2-stack PDA without λ-moves is called λ-free. If $\mathcal{A}$ is clear from the context, then we also write $\vdash$ rather than $\vdash_{\mathcal{A}}$. The language accepted by a 2-stack PDA $\mathcal{A}$, denoted by $L(\mathcal{A})$, is defined as

$$L(\mathcal{A}) = \{w \in \Sigma^* | (q^0, w, Z_1^0, Z_2^0) \vdash^* (q, \lambda, \lambda, \lambda) \text{ for some } q \in Q\},$$

where $\vdash^*$ is the reflexive and transitive closure of $\vdash$. The family of languages recognized by [λ-free] 2-stack PDAs is denoted by $\mathcal{L}$(PDA2[$-\lambda$]) and it is known that $\mathcal{L}$(PDA2) = $\mathcal{L}$(RE).

4 TEAMS OF PUSHDOWN AUTOMATA

In the sequel we assume that $n \geq 2$ unless otherwise stated.

Let $\mathcal{S}$ be a set $\{\mathcal{A}_i | i \in [n]\}$ of PDAs $\mathcal{A}_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q_i^0, Z_i^0)$. Then the team PDA of degree n (n-team PDA) composed over $\mathcal{S}$ is the quintuple

$$\mathcal{T}_\mathcal{S} = (Q, \Sigma, \Gamma, q^0, Z^0),$$

where

$$Q = Q_1 \times Q_2 \times \cdots \times Q_n, \quad \Sigma = \bigcup_{i \in [n]} \Sigma_i, \quad \Gamma = \Gamma_1^* \times \Gamma_2^* \times \cdots \times \Gamma_n^*,$$
$$q^0 = (q_1^0, q_2^0, \ldots, q_n^0), \quad \text{and} \quad Z^0 = (Z_1^0, Z_2^0, \ldots, Z_n^0).$$

We refer to Q as its set of *states*, to Σ as its *input alphabet*, to Γ as its *pushdown alphabet*, to q^0 as its *initial state*, and to Z^0 as its *initial stack contents*. The PDAs constituting $\mathcal{S}$ are called the *component PDAs* of $\mathcal{T}_\mathcal{S}$. We discard the index $\mathcal{S}$ whenever it is understood from the context. Note that we assumed $n \geq 2$ since a 1-team PDA would be simply a PDA.

A *configuration* of $\mathcal{T}$ consists of its current global state (defined by the local states the component PDAs reside in), the remaining part of the input word to be read, and the current contents of its n stacks. Configurations are thus triples of the form

$$((p_1, p_2, \ldots, p_n), aw, (Z_1\beta_1, Z_2\beta_2, \ldots, Z_n\beta_n)),$$

with $(p_1, p_2, \ldots, p_n) \in Q$, $a \in \Sigma \cup \{\lambda\}$, $w \in \Sigma^*$, and $(Z_1\beta_1, Z_2\beta_2, \ldots, Z_n\beta_n) \in \Gamma$. A component PDA $\mathcal{A}_j$, $j \in [n]$, of $\mathcal{T}$ is *competent* in such a configuration if $\delta_j(p_j, a, Z_j)$ is defined.

An n-team PDA $\mathcal{T}$ as above can make a *move* from a configuration τ to a configuration τ', denoted by $\tau \vdash_\mathcal{T} \tau'$, whenever an arbitrary one of its component PDAs is competent in τ. If we assume that component PDA $\mathcal{A}_j$, with $j \in [n]$, is competent, then $\mathcal{A}_j$ can cause $\mathcal{T}$ to change state by changing state locally, reading the first letter from the remaining part of the input word to be read, and replacing the topmost symbol from the jth stack. Formally,

$$\begin{aligned}&((p_1, p_2, \ldots, p_n), aw, (Z_1\beta_1, Z_2\beta_2, \ldots, Z_n\beta_n))\\ &\quad \vdash_\mathcal{T} ((q_1, q_2, \ldots, q_n), w, (\alpha_1\beta_1, \alpha_2\beta_2, \ldots, \alpha_n\beta_n))\end{aligned}$$

if there exists a $j \in [n]$ for which $(q_j, \alpha_j) \in \delta_j(p_j, a, Z_j)$ and for all $i \in [n]\backslash\{j\}$, $q_i = p_i$ and $\alpha_i = Z_i$. If $a = \lambda$, then such a move is called a λ-*move*, and an n-team PDA without λ-moves is called λ-*free*. If $\mathcal{T}$ is clear from the context, then we also write $\vdash$ rather than $\vdash_\mathcal{T}$.

The n-team PDA starts from an *initial configuration* (q^0, w, Z^0) consisting of its initial state q^0, a word w to be read, and its initial stack content Z^0. This word w is *accepted* by $\mathcal{T}$ if its stack is the empty word vector when the word has been completely read by $\mathcal{T}$. All words that can be accepted in this way together form the language of $\mathcal{T}$. Consequently, the language accepted by $\mathcal{T}$ is denoted by $L_*(\mathcal{T})$ and is thus defined as

$$L_*(\mathcal{T}) = \{w \in \Sigma^* | (q^0, w, Z^0) \vdash^* (q, \lambda, \Lambda) \text{ for some } q \in Q\},$$

where $\vdash^*$ is the reflexive and transitive closure of $\vdash$. The family of languages recognized by [λ-free] n-team PDAs is denoted by $\mathcal{L}$ (n-PDA[$-\lambda$], $*$-comp).

In order to clarify our definitions we now present an example.

Example 1 Consider the PDAs

$$\mathcal{A}_1 = (Q_1, \{a, b, c\}, \{\mathcal{A}\}, \delta_1, p^0, A) \quad \text{and} \quad \mathcal{A}_2 = (Q_2, \{a, b, c\}, \{A\}, \delta_2, q^0, A),$$

where

$$Q_1 = \{p^0\} \cup \{p_i | i \in [3]\} \quad \text{and} \quad Q_2 = \{q^0\} \cup \{q_i | i \in [3]\},$$

and whose transition mappings δ_1 and δ_2 are defined by

$$\begin{aligned}
&\delta_1(p^0, a, A) = \{(p_1, AA)\}, && \delta_2(q^0, a, A) = \{(q_1, AA)\},\\
&\delta_1(p_1, a, A) = \{(p_1, AA)\}, && \delta_2(q_1, a, A) = \{(q_1, AA)\},\\
&\delta_1(p_1, b, A) = \{(p_2, \lambda)\}, && \delta_2(q_1, b, A) = \{(q_2, \lambda)\},\\
&\delta_1(p_2, b, A) = \{(p_2, A)\}, && \delta_2(q_2, b, A) = \{(q_2, \lambda)\},\\
&\delta_1(p_2, c, A) = \{(p_3, \lambda)\}, && \delta_2(q_2, c, A) = \{(q_2, A), (q_3, \lambda)\}, \text{ and}\\
&\delta_1(p_3, c, A) = \{(p_3, \lambda)\}. &&
\end{aligned}$$

It is not difficult to see that the languages accepted by these PDAs are

$$L(\mathcal{A}_1) = \{a^m b^n c^m | m, n \geq 1\} \text{ and } L(\mathcal{A}_2) = \{a^k b^{k+1} | k \geq 1\} \cup \{a^k b^k c^m \mid k, m \geq 1\}.$$

Let us now consider the 2-team PDA

$$\mathcal{T}_1 = (Q_1 \times Q_2, \{a, b, c\}, \{A\}^* \times \{A\}^*, (p^0, q^0), (A, A))$$

over $\{\mathcal{A}_1, \mathcal{A}_2\}$. It is easy to verify that

$$L_*(\mathcal{T}_1) = L(\mathcal{A}_1) \| L(\mathcal{A}_2)$$

is the language accepted by $\mathcal{T}_1$.

It is clear that no matter how many component PDAs a team PDA consists of, no team feature is actually used: each (global) move of a team PDA is simply brought about by a (local) move of an arbitrary one of its component PDAs. Hence any n-team PDA recognizes the shuffle of n context-free languages. The converse is also true, viz.

THEOREM 1 $\mathcal{L}(n\text{-PDA}[-\lambda], *\text{-comp}) = n\text{-Shuf}(\text{CF}[-\lambda])$.

Proof

($\subseteq$) Let $\mathcal{T}$ be the n-team PDA composed over a set $\{\mathcal{A}_i | i \in [n]\}$ of PDAs. Consequently, it suffices to take $L_i = L(\mathcal{A}_i)$, for all $i \in [n]$, and $L_*(\mathcal{T}) = L_1 \| L_2 \| \cdots \| L_n$ follows immediately.

($\supseteq$) Let L_i, for all $i \in [n]$, be a context-free language. Consequently it suffices to construct a PDA $\mathcal{A}_i$ with $L(\mathcal{A}_i) = L_i$ for all $i \in [n]$, and to compose the n-team PDA $\mathcal{T}$ over $\{\mathcal{A}_i \mid i \in [n]\}$. ■

Before we extend the definition of n-team PDAs with so-called modes of competence we make the following two observations. First, we note that any λ-free n-team PDA can be simulated by an n-team PDA with λ-moves.

Secondly, we note that any n-team PDA can be simulated by an $(n+1)$-team PDA: if the n-team PDA is composed over a set $\mathcal{S}$, then it suffices to compose the $(n+1)$-team PDA over $\mathcal{S}$ augmented with a PDA with an empty transition mapping.

THEOREM 2 $\mathcal{L}(n\text{-PDA}[-\lambda], *\text{-comp}) \subset \mathcal{L}((n+1)\text{-PDA}[-\lambda], *\text{-comp})$.

Proof By Theorem 1 it suffices to prove that n-Shuf(CF) $\subset$ $(n+1)$-Shuf(CF). To this aim, we consider the linear languages $L_i = \{a_i^k b_i^k | k \geq 1\}$, $i \in [n+1]$, and $E = L_1 \| L_2 \| \cdots \| L_{n+1}$. Let us assume that $E = F_1 \| F_2 \| \cdots \| F_n$, for some context-free languages F_j, $j \in [n]$. Clearly, for any $p \in [n+1]$, if a_p occurs in a word $x \in F_j$, for some $j \in [n]$, then b_p must occur in x as well. Moreover, all occurrences of b_p in x must be at the right of the rightmost occurrence of a_p in x. Since in no word of E an occurrence of b_p is followed by an occurrence of a_p, for any $p \in [n+1]$, it follows that for each $p \in [n+1]$ there exists a unique $j \in [n]$ such that a_p appears in some words of F_j. By the pigeon-hole principle, there exist $r \in [n]$ and $p, s \in [n+1]$, $p \neq s$, such that both a_p and a_s occur in some words of F_r. Because F_r is the only language among $F_1, F_2, \ldots, F_n$ with this property with respect to p and s, we infer that

$$\text{pres}_{\{a_p, b_p, a_s, b_s\}}(F_r) = \{a_p^k b_p^k | k \geq 1\} \| \{a_s^k b_s^k | k \geq 1\},$$

where $\text{pres}_\Gamma \colon \Sigma \to \Gamma^*$ is a homomorphism defined by $\text{pres}_\Gamma(a) = a$ if $a \in \Gamma$ and $\text{pres}_\Gamma(a) = \lambda$ otherwise.

This however contradicts our supposition and therefore $E \notin n$-Shuf(CF). ■

5 COMPETENCE IN TEAMS OF PUSHDOWN AUTOMATA

Until now an n-team PDA can move from a configuration τ to a configuration τ' if at least one of its component PDAs is competent in τ. Consequently, a global move is brought about by the local move of an arbitrary one of these competent PDAs. In other words, the component PDAs of an n-team PDA thus do not cooperate in any way.

In this section, we define several ways of cooperation between the PDAs constituting an n-team PDA. This is achieved by requiring a precise number of the component PDAs of an n-team PDA $\mathcal{T}$ to be competent in the current configuration before $\mathcal{T}$ can move. These competent component PDAs in the current configuration are furthermore required to be equally competent among each other, in the sense that they must have the same symbol on top of their stacks and all together read either the same symbol or λ from the input tape. We distinguish the modes of competence $\leq k$, $=k$, and $\geq k$, for some $k \geq 1$, which require at most k, exactly k, or at least k component PDAs, respectively, to be equally competent among each other and moreover allow no other component PDAs to be equally competent with any of them. This thus resembles the maximal competence as defined in Ref. [19]. From now on we ignore the ≤ 1-mode of competence since it equals the $=1$-mode of competence.

Let us say that in the previous section n-team PDAs accepted languages in the $*$-mode of competence. Note that this is not the same as the $=1$-mode of competence, as in the $*$-mode of competence an arbitrary one of the possibly many equally competent component PDAs is chosen to bring about the global move of $\mathcal{T}$, thus completely disregarding whether any other component PDAs are equally competent with it. An example showing the differences will be presented shortly, after we have formally defined the modes of competence intuitively presented above.

For the sequel, we let $\mathcal{T} = (Q, \Sigma, \Gamma, q^0, Z^0)$ be the n-team PDA composed over a set $\mathcal{S} = \{\mathcal{A}_i | i \in [n]\}$ of component PDAs $\mathcal{A}_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q_i^0, Z_i^0)$, we let $f \in \{\leq k | k \geq 2\} \cup \{= k, \geq k | k \geq 1\}$ be the *modes of competence*, whose functioning is explained next, and we say that a natural number ℓ *satisfies* f if and only if ℓf (*e.g.*, 4 satisfies ≤ 7 because $4 \leq 7$).

$\mathcal{T}$ can *move* from a configuration $((p_1, p_2, \ldots, p_n), aw, (Z_1\beta_1, Z_2\beta_2, \ldots, Z_n\beta_n))$ to a configuration $((q_1, q_2, \ldots, q_n), w, (\alpha_1\beta_1, \alpha_2\beta_2, \ldots, \alpha_n\beta_n))$ in the f-comp-*mode*, denoted by

$$((p_1, p_2, \ldots, p_n), aw, (Z_1\beta_1, Z_2\beta_2, \ldots, Z_n\beta_n))$$
$$\vdash_{\mathcal{T}}^{f} ((q_1, q_2, \ldots, q_n), w, (\alpha_1\beta_1, \alpha_2\beta_2, \ldots, \alpha_n\beta_n))$$

if there exists a $J \subseteq [n]$, where $\#J$ satisfies f, such that for all $j \in J$,

(1) $Z_j = X$, for some $X \in \cap_{t\in J}\Gamma_t$,
(2) $(q_j, \alpha_j) \in \delta_j(p_j, a, X)$, *i.e.* all $\mathcal{A}_j$ are equally competent in the configuration $((p_1, p_2, \ldots, p_n), aw, (Z_1\beta_1, Z_2\beta_2, \ldots, Z_n\beta_n))$,
and for all $i \in [n]\backslash J$,
(3) either $Z_i \neq X$ or $\delta_i(p_i, a, X)$ is undefined, *i.e.* $\mathcal{A}_i$ is not equally competent with the above PDAs $\mathcal{A}_j$ in the configuration $((p_1, p_2, \ldots, p_n), aw, (Z_1\beta_1, Z_2\beta_2, \ldots, Z_n\beta_n))$.

If $a = \lambda$, then such a move is called a λ-*move*, and an n-team PDA without λ-moves is called λ-*free*. If $\mathcal{T}$ is clear from the context, then we also write $\vdash^{f}$ rather than $\vdash_{\mathcal{T}}^{f}$.

An n-team PDA in the f-comp-mode can thus make a move in the f-comp-mode from a configuration τ to a configuration τ', whenever f of its component PDAs are equally competent among each other in τ. Such a set of f competent component PDAs is called f-competent. A global state change of $\mathcal{T}$ is now brought about by all of these competent component PDAs changing their local state, all reading the first letter from the remaining part of the input word to be read, and all replacing the unique symbol on top of their stacks. Note that all the equally competent component PDAs need to participate in such a move, which thus indeed resembles the maximal competence as defined in Ref. [19].

The *language* accepted by $\mathcal{T}$ in the f-comp-mode is denoted by $L_f(\mathcal{T})$ and is defined as

$$L_f(\mathcal{T}) = \{w \in \Sigma^* | (q^0, w, Z^0) \vdash^{*f} (q, \lambda, \Lambda) \text{ for some } q \in Q\},$$

where $\vdash^{*f}$ is the reflexive and transitive closure of $\vdash^{f}$. The family of languages recognized by [λ-free] n-team PDAs in the f-comp-mode is denoted by $\mathcal{L}(n\text{-PDA}[-\lambda], f\text{-comp})$.

In order to clarify our definitions, we now continue Example 1.

Example 2 As there exists no initial configuration of $\mathcal{T}_1$ such that only one of the component PDAs is competent, it follows immediately that the language accepted by $\mathcal{T}_1$ in the $=$1-comp-mode is

$$L_{=1}(\mathcal{T}_1) = \emptyset,$$

which indeed does not equal $L_*(\mathcal{T}_1)$. The language accepted by $\mathcal{T}_1$ in the $=$2-comp-mode or the $\geq$2-comp-mode is the non-context-free language

$$L_{=2}(\mathcal{T}_1) = L_{\geq 2}(\mathcal{T}_1) = \{a^n b^n c^n \mid n \geq 1\}.$$

It is obvious that $L_{=2}(\mathcal{T}_1) \subseteq \{a^m b^n c^p | m, n, p \geq 1\}$. Starting from an initial configuration $((p^0, q^0), a^m b^n c^p, (A, A))$, for some $m, n, p \geq 1$, in which both component PDAs are equally competent, by m moves in the $=$2-comp-mode we arrive in configuration $((p_1, q_1), b^n c^p, (A^{m+1}, A^{m+1}))$. Note that if $n > m$, then by $m + 1$ moves in the $=$2-comp-mode we arrive in the configuration $((p_2, q_2), b^{n-m-1} c^p, (A^m, \lambda))$ and the accepting process is blocked since both components are not equally competent anymore. If $n < m$, then by n moves in the $=$2-comp-mode we arrive in the configuration $((p_2, q_2), c^p, (A^n, A^{m+1-n}))$

which cannot lead to a final configuration because any further sequence of moves in the $=2$-comp-mode cannot remove more than one A from the second stack. Therefore, $m = n$ and the accepting process continues with the current configuration $((p_2, q_2), c^p, (A^m, A))$. Now, the first stack becomes empty if and only if p equals m, whereas the second stack becomes empty after the last move.

By considerations similar to the ones above, the reader may easily verify that the non-context-free language

$$L_{\geq 1}(\mathcal{T}_1) = L_{\leq 2}(\mathcal{T}_1) = \{a^m b^n c^m \mid 1 \leq m \leq n\} \cup \{a^k b^k c^m \mid 1 \leq k \leq m\}$$

is the language accepted by $\mathcal{T}_1$ in the ≥ 1-comp-mode or the ≤ 2-comp-mode.

Similar to the case of n-team PDAs accepting languages in the $*$-mode of competence, we make the following two observations. First, we note that any λ-free n-team PDA accepting languages in the f-comp-mode can be simulated by an n-team PDA with λ-moves and accepting languages in the same mode of competence.

Secondly, we note that any n-team PDA accepting languages in the f-comp-mode can be simulated by an $(n + 1)$-team PDA accepting languages in the same mode of competence.

Before we investigate the accepting capacity of n-team PDAs in one of the modes of competence introduced in this section, we note

PROPOSITION 1 *Let $\mathcal{T}$ be the n-team PDA composed over a set $\{\mathcal{A}_i \mid i \in [n]\}$ of component PDAs $\mathcal{A}_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q_i^0, Z_i^0)$. Then*

(1) $L_{\geq 1}(\mathcal{T}) = L_{\leq k}(\mathcal{T})$, *for all* $k \geq n$,
(2) $L_{=k}(\mathcal{T}) = L_{\geq k}(\mathcal{T}) = \emptyset$, *for all* $k > n$,
(3) $L_{=n}(\mathcal{T}) = L_{\geq n}(\mathcal{T})$, *and*
(4) *if for all* $J \subseteq [n]$ *with* $\#J = k + 1$, $\cap_{j \in J} \Sigma_j = \emptyset$ *or* $\cap_{j \in J} \Gamma_j = \emptyset$, *then* $L_{=k}(\mathcal{T}) = L_{\geq k}(\mathcal{T})$.

The condition of Proposition 1(4) guarantees that no configuration of the n-team PDA $\mathcal{T}$ exists in which $k + 1$ of its component PDAs are competent, for some $k \in [n]$. Note that this condition is not implied by simply requiring $k + 1$ initial contents of the n stacks of $\mathcal{T}$ to be pairwise disjoint. Naturally, this condition is trivially met in case $k = n$, as stated in Proposition 1(3).

6 POWER OF COMPETENCE IN TEAMS OF PUSHDOWN AUTOMATA

In this section, we study the accepting capacity of n-team PDAs in the modes of competence introduced in the previous section.

THEOREM 3 *Let $g_1 \in \{=k, \geq k \mid k \geq 2\}$ and let $g_2 \in \{=1, \geq 1\} \cup \{\leq k \mid k \geq 2\}$. Then*

(1) $\mathcal{L}((k \cdot n)\text{-PDA}[-\lambda], g_1\text{-comp}) \supset \mathcal{L}(n\text{-PDA}[-\lambda], *\text{-comp})$ *and*
(2) $\mathcal{L}(n\text{-PDA}[-\lambda], g_2\text{-comp}) \supset \mathcal{L}(n\text{-PDA}[-\lambda], *\text{-comp})$.

Proof (1) Consider a set $\{\mathcal{A}_i \mid i \in [n]\}$ of PDAs and let $L_i = L(\mathcal{A}_i)$, $i \in [n]$, be the languages accepted by these PDAs. For each such language L_i, $i \in [n]$, we define the k different PDAs $\mathcal{A}_i^{(1)}, \mathcal{A}_i^{(2)}, \ldots, \mathcal{A}_i^{(k)}$ in such a way that for each move $(q, w, \alpha) \vdash_{\mathcal{A}_i^{(j)}} (s, x, \beta)$, we have $(q, w, \alpha) \vdash_{\mathcal{A}_i^{(t)}} (s, x, \beta)$ for all $t \in [k] \backslash \{j\}$ as well. This can be accomplished by augmenting their pushdown alphabets (or transition mappings) with some useless symbols (transitions).

Moreover, we require that any two PDAs $\mathcal{A}_i^{(j)}$ and $\mathcal{A}_s^{(t)}$, $i, s \in [n]$, $i \neq s$, $j, t \in [k]$, have disjoint pushdown alphabets. Consequently, we compose a team PDA $\mathcal{T}$ over the set of all these $k \cdot n$ PDAs. Clearly,

$$L_{=k}(\mathcal{T}) = L_{\geq k}(\mathcal{T}) = L_1 \| L_2 \| \cdots \| L_n.$$

From Example 2 we know that $L = \{a^n b^n c^n | n \geq 1\}$ is contained in both $\mathcal{L}(2\text{-PDA}[-\lambda], =2\text{-comp})$ and $\mathcal{L}(2\text{-PDA}[-\lambda], \geq 2\text{-comp})$. Furthermore, it is well known that $L \notin \mathcal{L}(\text{CF})$. It remains to prove that $L \notin m\text{-Shuf(CF)}$, for $m > 1$. Let $m > 1$ and assume the contrary. Then $L = L_1 \| L_2 \| \cdots \| L_m$, with $L_i \in \mathcal{L}(\text{CF})$ for all $i \in [m]$. Let $w_i \in L_i$, for all $i \in [m]$, be such that $w_1 w_2 \cdots w_m \in L$. Then we know that $w_1 = a w_1'$ and $w_m = w_m' c$, with $w_1', w_m' \in \{a, b, c\}^*$. However, then it must be the case that also $w_m w_1 w_2 \cdots w_{m-1} \in L$, which clearly does not hold. From this contradiction we conclude that $L \notin m\text{-Shuf(CF)}$ and thus we are done.

(2) The inclusion is straightforward. We just take n context-free languages, define n PDAs with pairwise disjoint pushdown alphabets accepting them, and compose a team PDA over these PDAs. From Example 2 and Proposition 1(1) we know that the non-context-free language

$$L' = \{a^m b^n c^m | 1 \leq m \leq n\} \cup \{a^k b^k c^m | 1 \leq k \leq m\} \in \mathcal{L}(2\text{-PDA}[-\lambda], g\text{-comp}),$$

for all $g \in \{\geq 1\} \cup \{\leq k | k \geq 2\}$. By a similar reasoning as above one can easily prove that L' cannot be expressed as the shuffle of m context-free languages, for any $m > 1$. It remains to provide a language that cannot be expressed as the shuffle of m context-free languages, for any $m > 1$, but which is accepted by a λ-free 2-team PDA in the $=1$-comp-mode. To this aim we define the two PDAs

$$\mathcal{A}_1 = (\{q_0, q_1\}, \{a, b, c\}, \{A, F\}, \delta_1, q_0, A) \text{ and } \mathcal{A}_2 = (\{s_0, s_1\}, \{b, c, d\}, \{A, F\}, \delta_2, s_0, A),$$

with transition mappings δ_1 and δ_2 defined by

$$\begin{aligned}
&\delta_1(q_0, a, A) = \{(q_0, AA), (q_1, A)\}, && \delta_2(s_0, b, A) = \{(s_0, AA), (s_1, A)\},\\
&\delta_1(q_0, b, A) = \{(q_1, F)\}, && \delta_2(s_0, c, A) = \{(s_1, F)\},\\
&\delta_1(q_1, c, A) = \{(q_1, \lambda)\} \text{ and} && \delta_2(s_1, d, A) = \{(s_1, \lambda)\}.
\end{aligned}$$

We claim that

$$L_{=1}(\mathcal{T}') = \bigcup_{m,n \geq 1} \{a^m\}\{b^n\}(\{c^m\} \| \{d^n\}),$$

where $\mathcal{T}'$ is the 2-team PDA composed over the above PDAs $\mathcal{A}_1$ and $\mathcal{A}_2$. First note that no word starting with any letter other than a can be accepted. Indeed, if it starts with a b, then the accepting process cannot start since both component PDAs are equally competent in the initial configuration. If it starts with a c or a d, then no move at all is possible. Therefore the input string must be of the form $a^m \alpha$, for some $m \geq 1$, where the first letter of α is not an a. An accepting process that might lead to a final configuration, *i.e.* one with the empty word vector as its stack, moves in the $=1$-comp-mode from the initial configuration $((q_0, s_0), a^m \alpha, (A, A))$ to the configuration $((q_1, s_0), \alpha, (A^m A))$.

By checking the transition mappings one can easily infer that the first letter of α must be a b, *i.e.* $\alpha = b^n \beta$, for some $n \geq 1$. Consequently, our accepting process has to move in the $=1$-comp-mode from the configuration $((q_1, s_0), b^n \beta, (A^m A))$ to the configuration $((q_1, s_1), \beta, (A^m, A^n))$. Since from now on the first and the second component PDA can read

only a c and a d, respectively, it follows that the accepting process developed so far can eventually reach a final configuration if and only if $\beta \in (\{c^m\} \| \{d^n\})$. Because it follows immediately that $L_{=1}(\mathcal{T}') \notin m\text{-Shuf(CF)}$, for any $m > 1$, we are done. ■

For two modes of competence we know that increasing the number of components in a team PDA does not lead to an infinite hierarchy. This is a direct result of the fact that a 3-team PDA in either the $=$2-comp-mode or the $\geq$2-comp-mode suffices to generate the class $\mathcal{L}$(RE) of recursively enumerable languages. To prove this claim we can make use of the fact that the class $\mathcal{L}$(PDA2) of languages generated by 2-stack PDAs equals $\mathcal{L}$(RE) because this implies that it suffices to prove that every 2-stack PDA can be simulated by a 3-team PDA in the $=$2-comp-mode or the $\geq$2-comp-mode.

LEMMA 1 $\mathcal{L}(\text{PDA2}) \subseteq \mathcal{L}(\text{3-PDA}, =\text{2-comp})$.

Proof Let $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q^0, Z_1^0, Z_2^0)$ be a 2-stack PDA. It should be clear that without loss of generality we can assume that from any of its initial configurations, only a unique letter $a \in \Sigma \cup \{\lambda\}$ can be read, or else $L(\mathcal{A}) = \emptyset$. Thus from such an initial configuration (q^0, aw, Z_1^0, Z_2^0), for some $w \in \Sigma^*$, only a unique move $(q^0, aw, Z_1^0, Z_2^0) \vdash (q', w, \alpha_1, \alpha_2)$, for some $q' \in Q$ and $\alpha_1, \alpha_2 \in \Gamma^*$, can be made. This means that $(q', \alpha_1, \alpha_2) \in \delta(q^0, a, Z_1^0, Z_2^0)$. For later use, we denote $[Z] = [q^0, a, Z_1^0, Z_2^0, q', \alpha_1, \alpha_2]$.

To simulate $\mathcal{A}$, we construct the 3-team PDA

$$\mathcal{T} = (Q_1 \times Q_2 \times Q_3, \Sigma, \Gamma \times \Gamma_2 \times \Gamma_3, (q_1^0, q_2^0, q_3^0), (Z_1^0, Z_1^0, [Z]))$$

over the component PDAs

$$\mathcal{A}_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_1^0, Z_1^0),$$

where $Q_1 = \{q_1 | q \in Q\}$ and δ_1 is defined by

$$\delta_1(p_1, a, Z_1) = \{(q_1, \alpha_1) | (q, \alpha_1, \alpha_2) \in \delta(p, a, Z_1, Z_2)\},$$

with $p_1 \in Q_1$, $a \in \Sigma \cup \{\lambda\}$, and $Z_1 \in \Gamma$,

$$\mathcal{A}_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2^0, Z_1^0),$$

where $Q_2 = \{q_2, \bar{q}_2 | q \in Q\}$, $\Gamma_2 = \Gamma \cup \{[p, a, Z_1, Z_2, q, \alpha_1, \alpha_2] | (q, \alpha_1, \alpha_2) \in \delta(p, a, Z_1, Z_2)$, $p, q \in Q, a \in \Sigma \cup \{\lambda\}, Z_1, Z_2 \in \Gamma, \alpha_1, \alpha_2 \in \Gamma^*\}$, and δ_2 is defined by

$$\delta_2(p_2, a, Z_1) = \{(\bar{q}_2, [p, a, Z_1, Z_2, q, \alpha_1, \alpha_2]) | (q, \alpha_1, \alpha_2) \in \delta(p, a, Z_1, Z_2)\},$$

with $p_2 \in Q_2$, $a \in \Sigma \cup \{\lambda\}$ and $Z_1 \in \Gamma$, and

$$\delta_2(\bar{q}_2, \lambda, [p, a, Z_1, Z_2, q, \alpha_1, \alpha_2]) = \{(q_2, \alpha_1) | (q, \alpha_1, \alpha_2) \in \delta(p, a, Z_1, Z_2)\},$$

with $\bar{q}_2 \in Q_2$ and $[p, a, Z_1, Z_2, q, \alpha_1, \alpha_2] \in \Gamma_2$, and

$$\mathcal{A}_3 = (Q_3, \emptyset, \Gamma_3, \delta_3, q_3^0, [Z]),$$

where $Q_3 = \{q_3 | q \in Q\}$, $\Gamma_3 = \Gamma_2 \backslash \Gamma$, and δ_3 is defined by

$$\delta_3(p_3, \lambda, [p, a, Z_1, Z_2, q, \alpha_1, \alpha_2]) = \{(q_3, \gamma) | (q, \alpha_1, \alpha_2) \in \delta(p, a, Z_1, Z_2), \gamma \in \Gamma_3^*, \text{stack}_2(\gamma) = \alpha_2\},$$

with $p_3 \in Q_3$, $[p, a, Z_1, Z_2, q, \alpha_1, \alpha_2] \in \Gamma_3$, and $\text{stack}_2: \Gamma_3^* \to \Gamma$ is a homomorphism defined by $\text{stack}_2([r, b, X, Y, s, \beta_1, \beta_2]) = Y$.

We now show how $\mathcal{A}$ can be simulated by $\mathcal{T}$ in the =2-comp-mode. Each move of $\mathcal{A}$ is simulated by two consecutive moves of $\mathcal{T}$, the first of which is brought about by its component PDAs $\mathcal{A}_1$ and $\mathcal{A}_2$ and the second by its component PDAs $\mathcal{A}_2$ and $\mathcal{A}_3$.

Let us assume that $aw \in L(\mathcal{A})$, for some $a \in \Sigma \cup \{\lambda\}$ and $w \in \Sigma^*$. Consequently, we recall that from the resulting initial configuration (q^0, aw, Z_1^0, Z_2^0) only a unique move $(q^0, aw, Z_1^0, Z_2^0) \vdash (q', w, \alpha_1, \alpha_2)$, for some $q' \in Q$ and $\alpha_1, \alpha_2, \in \Gamma^*$, could be made. Note that this means $(q', \alpha_1, \alpha_2) \in \delta(q^0, a, Z_1^0, Z_2^0)$. Recall furthermore that $[Z]$ denotes $[q^0, a, Z_1^0, Z_2^0, q', \alpha_1, \alpha_2]$.

This initial move is now simulated by $\mathcal{T}$ in the =2-comp-mode. In our initial configuration

$$((q_1^0, q_2^0, q_3^0), aw, (Z_1^0, Z_1^0, [Z]))$$

clearly only $\{\mathcal{A}_1, \mathcal{A}_2\}$ is =2-competent, due to the fact that both $\delta_1(q_1^0, a, Z_1^0)$ and $\delta_2(q_2^0, a, Z_1^0)$ are defined. Thus, since $(q_1', \alpha_1) \in \delta_1(q_1^0, a, Z_1^0)$ and $(\bar{q}_2', [Z]) \in \delta_2(q_2^0, a, Z_1^0)$, $\mathcal{T}$ can make the move

$$((q_1^0, q_2^0, q_3^0), aw, (Z_1^0, Z_1^0, [Z])) \vdash^{=2} ((q_1', \bar{q}_2', q_3^0), w, (\alpha_1, [Z], [Z])).$$

In the latter configuration only $\{\mathcal{A}_2, \mathcal{A}_3\}$ is =2-competent, due to the fact that both $\delta_2(\bar{q}_2', \lambda, [Z])$ and $\delta_3(q_3^0, \lambda, [Z])$ are defined. Thus, since $(q_2', \alpha_1) \in \delta_2(\bar{q}_2', \lambda, [Z])$ and $(q_3', \gamma) \in \delta_3(q_3^0, \lambda, [Z])$, where $\gamma \in \Gamma_3^*$ is such that $\text{stack}_2(\gamma) = \alpha_2$, it follows that $\mathcal{T}$ can make the move

$$((q_1', \bar{q}_2', q_3^0), w, (\alpha_1, [Z], [Z])) \vdash^{=2} ((q_1', q_2', q_3'), w, (\alpha_1, \alpha_1, \gamma)),$$

where we recall $\text{stack}_2: \Gamma_3^* \to \Gamma$ to be a homomorphism defined by $\text{stack}_2([r, b, X, Y, s, \beta_1, \beta_2]) = Y$.

At this point, it is important to explain the third element of the stack contents of $\mathcal{T}$, as this is the tricky part of the simulation. Even though the third element of the stack contents of $\mathcal{T}$ serves to simulate the stack of $\mathcal{A}$, we note that $\mathcal{T}$ after its first two moves has $\gamma \in \Gamma_3^*$ as the third element in its stack contents, while $\mathcal{A}$ after its first move has $\alpha_2 \in \Gamma^*$ in its stack. However, γ is chosen such that $\text{stack}_2(\gamma) = \alpha_2$. Hence, what actually happens is that $\mathcal{T}$ 'guesses', for each symbol of α_2, which move will, eventually, remove this symbol from the third element of its stack contents.

The remaining moves of $\mathcal{A}$ are likewise simulated by two moves of $\mathcal{T}$, alternatingly brought about by its component PDAs $\mathcal{A}_1$ and $\mathcal{A}_2$ and by its component PDAs $\mathcal{A}_2$ and $\mathcal{A}_3$. Since the input alphabet of $\mathcal{A}_3$ is empty, it is not difficult to see that no other moves than the ones sketched above can lead to the acceptance of aw by $\mathcal{T}$. Hence $L_{=2}(\mathcal{T}) = L(\mathcal{A})$. ■

Since in the simulation in this proof the input alphabet of $\mathcal{A}_3$ is empty, Proposition 1(4) implies

COROLLARY 1 $\mathcal{L}(\text{PDA2}) \subseteq \mathcal{L}(3\text{-PDA}, \geq 2\text{-comp})$.

Combining these two results with the fact that $L(\text{PDA2}) = L(\text{RE})$, we obtain

THEOREM 4 $\mathcal{L}(3\text{-PDA}, =2\text{-comp}) = \mathcal{L}(3\text{-PDA}, \geq 2\text{-comp}) = \mathcal{L}(\text{RE})$.

Note that in the proof of Lemma 1 the 3-team PDA simulating the 2-stack PDA makes explicit use of λ-moves. In fact, it remains an open problem to establish the accepting capacity of λ-free n-team PDAs, where $n \geq 3$. We conjecture that prohibiting λ-moves in general decreases the accepting capacity of n-team PDAs.

7 FUTURE WORK

Next to the open problems mentioned throughout the article, we would like to point out some more directions for future work. In this article, we have studied only team PDAs with non-deterministic component PDAs and with acceptance by empty stack. To begin with, it would be interesting to add final states to the component PDAs and to investigate the accepting capacity of team PDAs with acceptance by final states. This might lead to a different picture.

Furthermore, it would be interesting to restrict ourselves to team PDAs with deterministic component PDAs and to investigate their accepting capacity, both with acceptance by empty stack and with acceptance by final states. This might lead to a different picture, in particular due to the well-known fact that the family of languages accepted by deterministic PDAs is strictly included in $\mathcal{L}(\text{CF})$. As an initial result in this direction we note that – in analogy with Theorem 1 – the language accepted by an n-team PDA with deterministic component PDAs in the $*$-comp-mode and under acceptance by empty stack, equals the shuffle of the languages accepted by its n deterministic component PDAs.

In Section 4, consequently, we have shown that the language accepted by an n-team PDA in the $*$-comp-mode is characterized by the shuffle of the languages of its component PDAs (cf. Theorem 1). It would be interesting to find similar characterizations for n-team PDAs accepting languages in other modes of competence. To this aim a variant of the *synchronized shuffles* studied in Ref. [6] might be used. More precisely, we believe that a variant of the *fully synchronized shuffle* (fS-shuffle) is needed. We now define the fS-shuffle as studied in Ref. [6]. An fS-shuffle of two words is defined with respect to the alphabets of these two words. Like the ordinary shuffle, such an fS-shuffle still contains all symbols of both words but those symbols from the intersection of these two alphabets are now required to synchronize – and form a sort of 'backbone' of the fS-shuffle – whereas the remaining symbols are shuffled – while preserving the order within the two words. Formally, the fS-shuffles of two words $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$ with respect to Σ_1 and Σ_2, denoted by $u \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2} v$, is defined as $u \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2} v = \{w \in (\Sigma_1 \cup \Sigma_2)^* | \text{pres}_{\Sigma_1}(w) = u, \text{pres}_{\Sigma_2}(w) = v\}$, where $\text{pres}_\Gamma \colon \Sigma \to \Gamma^*$ is a homomorphism defined by $\text{pres}_\Gamma(a) = a$ if $a \in \Gamma$ and $\text{pres}_\Gamma(a) = \lambda$ otherwise. The fS-shuffle of two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, denoted by $L_1 \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2} L_2$, is defned as the union of all the fS-shuffles of a word from L_1 and a word from L_2. Hence $L_1 \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2} L_2 = \{w \in u \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2} v | u \in L_1, v \in L_2\} = \cup_{u \in L_1, v \in L_2} u \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2} v$.

As the set of symbols required to synchronize in an fS-shuffle is dynamically formed with respect to the alphabets of the words involved, it is nontrivial to prove [see, *e.g.*, Ref. 6] the fS-shuffle to be commutative and 'associative', *i.e.* $u \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2} v = v \,{}_{\Sigma_2}\underline{\|}_{\Sigma_1} u$ and $(u \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2} v)\Sigma_1 \cup \Sigma_2 \,{}\underline{\|}\, \Sigma_3 w = u \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2 \cup \Sigma_3} (v \,{}_{\Sigma_2}\underline{\|}_{\Sigma_3} w)$, for all words $u \in \Sigma_1^*$, $v \in \Sigma_2^*$, and $w \in \Sigma_3^*$, and likewise for languages.

There are examples of n-team PDAs whose languages accepted in the f-comp-mode equal the fS-shuffles of the languages accepted by their component PDAs. In Example 2, *e.g.*, $L_{=2}(\mathcal{T}_1) = L_{\geq 2}(\mathcal{T}_1) = L(\mathcal{A}_1) \,{}_{\{a,b,c\}}\underline{\|}_{\{a,b,c\}} L(\mathcal{A}_2)$. Unfortunately, it is just as easy to come up with counterexamples. It thus remains an open problem to find a suitable variant of the fS-shuffle which might characterize some families of languages defined by team PDAs.

Finally, in Ref. [31] team automata have been applied by giving a variety of known access control protocols a rigorous formal description in terms of synchronizations in team automata. The type of access control protocols that can be modelled by team automata, however, is limited due to the fact that team automata can only deal with pure communication: their constituting component automata can synchronize their actions, but they cannot exchange any information due to the absence of private memory. Since team PDAs augment team automata with a (distributed) pushdown memory – and thereby allow the flow of information among their constituting component automata – a larger variety of access control protocols can potentially be modelled. Team PDAs moreover seem to have the capacity to model numerous other intriguing groupware applications within CSCW.

Acknowledgements

The first author was supported by an ERCIM postdoctoral fellowship and the third author was supported by the Centre of Excellence in Information Technology, Computer Science and Control, ICA1-CT-2000-70025, HUN-TING project, WP5. The first author's research for this article was fully carried out during his stay at the Computer and Automation Research Institute of the Hungarian Academy of Sciences.

References

[1] E. Csuhaj-Varjú, V. Mitrana, and Gy. Vaszil, Distributed pushdown automata systems: computational power, In: Z. Ésik and Z. Fülöp (Eds.), *Proceedings of the DLT'03 Seventh International Conference on Developments in Language Theory, Szeged, Hungary*, Lecture Notes in Computer Science, Vol. 2710, Springer-Verlag, Berlin (2003) 218–229.

[2] J. Dassow and V. Mitrana, Stack cooperation in multistack pushdown automata, *Journal of Computer and System Sciences*, 58(3) (1999) 611–621.

[3] K. Krithivasan and M. Sakthi Balan, Distributed processing in deterministic PDA, In: R. Freund and A. Kelemenová (Eds.), *Proceedings of the International Workshop Grammar Systems 2000, Vienna, Austria*, Silesian University at Opava, Faculty of Philosophy and Science, Institute of Computer Science, Czech Republic (2000), pp. 127–145.

[4] K. Krithivasan, M. Sakthi Balan, and P. Harsha, Distributed processing in automata, *International Journal of Foundations of Computer Science*, 10(4) (1999), 443–463.

[5] M. Sakthi Balan, Distributed processing in automata, *Masters's thesis*, Department of Computer Science and Engineering, Indian Institute of Technology (2000).

[6] M. H. ter Beek, Team automata – A formal approach to the modeling of collaboration between system components, *Ph.D. thesis*, Leiden Institute of Advanced Computer Science, Universiteit Leiden (2003).

[7] M. H. ter Beek, C. A. Ellis, J. Kleijn, and G. Rozenberg, Synchronizations in team automata for groupware systems, *Computer Supported Cooperative Work – The Journal of Collaborative Computing*, 12(1) (2003), 21–69. (Also appeared as *Technical Report TR-99-12*, Leiden Institute of Advanced Computer Science, Universiteit Leiden (1999).)

[8] C. A. Ellis, Team automata for groupware systems, In: S. C. Hayne and W. Prinz (Eds.), *Proceedings of the GROUP'97 International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge*, ACM Press, New York (1997) pp. 415–424.

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

[10] J. Dassow, Gh. Păun, and G. Rozenberg, Grammar systems, Chapter 4 in Volume 2 of [31] (1997) 155–213.

[11] E. Csuhaj-Varjú and J. Dassow, On cooperating distributed grammar systems, *Journal of Information Processing and Cybernetics*, EIK(26) (1990) pp. 49–63.

[12] E. Csuhaj-Varjú and J. Kelemen, Cooperating grammar systems: a syntactical framework for the blackboard model of problem solving, In: I. Plander (Ed.), *Proceedings AI and Information-control Systems of Robots'89*, North-Holland Publishing Company, Amsterdam (1989) pp. 121–127.

[13] P. H. Nii, Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architectures. Part I, *The AI Magazine* 7(2) (1986) 38–53.

[14] P. H. Nii, Blackboard systems, In: A. Barr, P. R. Cohen and E. A. Feigenbaum (Eds.), *The Handbook of Artificial Intelligence Volume 4* Addison Wesley Publishers, Reading, Massachusetts (1989) pp. 1–82.

[15] M. H. ter Beek, Teams in grammar systems, *Master's thesis IR-96-32*, Department of Computer Science, Leiden University (1996).

[16] L. Kari, A. Mateescu, Gh. Păun, and A. Salomaa, Teams in cooperating grammar systems, *Journal of Experimental and Theoretical Articial Intelligence*, 7 (1995) 347–359.

[17] Gh. Păun and G. Rozenberg, Prescribed teams of grammars, *Acta Informatica* 31 (1994), 525–537.
[18] J. Smith, *Collective Intelligence in Computer Based Collaboration – A Volume in the Computers, Cognition, and Work Series*, Lawrence Erlbaum Associates, Mahwah, New Jersey (1994).
[19] H. Bordihn and E. Csuhaj-Varjú, On competence and completeness in CD grammar systems, *Acta Cybernetica*, 12(4) (1996), 347–361.
[20] M. H. ter Beek, E. Csuhaj-Varjú, M. Holzer, and Gy. Vaszil, On competence in cooperating distributed grammar systems, *Technical Report 2002/1*, Computer and Automation Research Institute of the Hungarian Academy of Sciences (2002).
[21] M. H. ter Beek, E. Csuhaj-Varjú, M. Holzer, and Gy. Vaszil, On competence in cooperating distributed grammar systems. Part II, *Technical Report 2002/2*, Computer and Automation Research Institute of the Hungarian Academy of Sciences (2002).
[22] M. H. ter Beek, E. Csuhaj-Varjú, M. Holzer, and Gy. Vaszil, On competence in cooperating distributed grammar systems. Part III, *Technical Report 2002/3*, Computer and Automation Research Institute of the Hungarian Academy of Sciences (2002).
[23] E. Csuhaj-Varjú, J. Dassow, and M. Holzer, On a competence-based cooperation strategy in CD grammar systems, *Submitted* (2002).
[24] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana, and Gy. Vaszil, Parallel communicating pushdown automata systems, *International Journal of Foundations of Computer Science*, 11(4) (2000), 633–650.
[25] T. Hirst and D. Harel, On the power of bounded concurrency II: pushdown automata, *Journal of the ACM*, 41(3) (1994), 540–554.
[26] R. M. Baecker (Ed.), *Readings in Groupware and Computer Supported Cooperation Work: Assisting Human–Human Collaboration*, Morgan Kaufmann Publishers, San Mateo, California (1992).
[27] J. Grudin, CSCW: history and focus, *IEEE Computer*, 27(5) (1994) 19–26.
[28] N. A. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, San Mateo, California (1996).
[29] N. A. Lynch and M. R. Tuttle, An introduction to input/output automata, *CWI Quarterly*, 2(3) (1989) 219–246. (Also appeared as *Technical Memo MIT/LCS/TM-373*, Massachusetts Institute of Technology, Cambridge, Massachusetts (1988).)
[30] M. H. ter Beek, C. A. Ellis, J. Kleijn, and G. Rozenberg, Team automata for CSCW, In: H. Weber, H. Ehrig, and W. Reisig, eds. *Proceedings of the 2nd International Colloquium on Petri Net Technologies for Modelling Communication Based Systems, Berlin, Germany*, Fraunhofer Institute for Software and Systems Engineering, Berlin (2001), pp. 1–20. (Also appeared as *Technical Report TR-01-07*, Leiden Institute of Advanced Computer Science, Universiteit Leiden (2001).)
[31] M. H. ter Beek, C. A. Ellis, J. Kleijn, and G. Rozenberg, Team automata for spatial access control, In: W. Prinz, M. Jarke, Y. Rogers, K. Schmidt, and V. Wulf (Eds.), *Proceedings of the ECSCW 2001 Seventh European Conference on Computer Supported Cooperative Work, Bonn, Germany*, Kluwer Academic Publishers, Dordrecht (2001), pp. 59–77. (Also appeared as *Technical Report TR-01-03*, Leiden Institute of Advanced Computer Science, Universiteit Leiden (2001).)
[32] M. H. ter Beek, E. Csuhaj-Varjú, and V. Mitrana, Teams of pushdown automata (extended abstract), In: A. Zamulin and M. Broy (Eds.), *Proceedings of the PSI'03 Fifth International Conference on Perspectives of System Informatics, Novosibirsk, Siberia, Russia*, Lecture Notes in Computer Science, Vol. 2890, Springer-Verlag, Berlin (2003) 329–337.
[33] G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Springer-Verlag, Berlin (1997).