Teams of Pushdown Automata*

Maurice H. ter Beek[1], Erzsébet Csuhaj-Varjú[2], and Victor Mitrana[3]

[1] Istituto di Scienza e Tecnologie dell'Informazione, CNR, Via G. Moruzzi 1, 56124 Pisa, Italy
mtbeek@iei.pi.cnr.it
[2] Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende utca 13–17, 1111 Budapest, Hungary
csuhaj@sztaki.hu
[3] University of Bucharest, Faculty of Mathematics, Str. Academiei 14, 70109 Bucharest, Romania,
mitrana@funinf.cs.unibuc.ro

Abstract. We introduce team pushdown automata as a theoretical framework capable of modelling various communication and cooperation strategies in complex, distributed systems. Team pushdown automata are obtained by augmenting distributed pushdown automata with the notion of team cooperation or — alternatively — by augmenting team automata with pushdown memory. Here we study their accepting capacity.

1 Introduction

During the last decade communication and cooperation strategies in complex, distributed systems have received considerable attention in many research areas. Examples include problem solving, multi-agent systems, groupware systems, and computer supported cooperative work (CSCW). Along this line of development, many attempts were made to provide theoretical models for such systems. These models often abstract from concrete data, configurations, and actions, but describe the system solely in terms of *(pushdown) automata*. The *team pushdown automata (team pda's)* we introduce here are in many ways a continuation of two such attempts, viz. the *distributed pushdown automaton* model [13,14,18] and the *team automaton* model [7,15]. However, team pda's also borrow ideas from other models, in particular from the theory of *grammar systems* [10,23].

Grammar systems consist of a set of grammars that, by interacting according to a protocol of communication and cooperation, generate one language. Best known are the sequential *cooperating distributed (CD) grammar systems* and the *parallel communicating (PC) grammar systems*. The grammars in CD grammar systems work together, in turn, by rewriting a common string [8]. When to

* ter Beek was supported by an ERCIM postdoctoral fellowship and his research was fully carried out during his stay at the Computer and Automation Research Institute of the Hungarian Academy of Sciences. Mitrana was supported by the Centre of Excellence in Information Technology, Computer Science and Control, ICA1-CT-2000-70025, HUN-TING project, WP5.

M. Broy and A.V. Zamulin (Eds.): PSI 2003, LNCS 2890, pp. 329–337, 2003.

transfer control from one grammar to another is determined by the CD grammar system's cooperation strategy. The classic cooperation strategies are $*$, t, $\leq k$, $=k$, and $\geq k$, for some $k \geq 1$, by 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, resp. We call these *CD strategies*. In [11] CD grammar systems were recognized as the formal language-theoretic counterpart of blackboard systems [21], multi-agent systems used in the blackboard model of problem solving [20]. The common string models the blackboard containing the current state of the problem solving, the grammars represent knowledge sources contributing to the problem solving by changing the contents of the blackboard based on 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 *teams* was introduced in CD grammar systems by grouping grammars that, in turn, rewrite a common string in parallel [17,22]. Teams are prescribed or formed automatically, and the CD strategies determine when to transfer control from one team to another. Since well-structured groups outperform individuals in a variety of tasks, the concept of teams has a clear practical motivation [24]. The concept of *competence* was introduced in CD grammar systems by defining the "cleverness" of a grammar w.r.t. a certain string as the number of nonterminals of this string it is able to rewrite [2,3,9]. This recognizes the practical reality of members of a group possessing different skills.

Replacing the grammars in CD grammar systems by (pushdown) automata leads to *distributed (pushdown) automata* [13,18]. Their pushdown memories (a.k.a. stacks) model the memory, or a notebook, of the knowledge sources of blackboard systems. A similar operation was carried out for PC grammar systems [12]. Independently, *multistack pushdown automata* with CD strategies controlling the use of the stacks were introduced [14]. These two models thus differ conceptually: in the latter case there is one pushdown automaton (pda) with n stacks, while in the former case the n stacks are distributed over n pda's. The distributed pda's of [18] moreover communicate by allowing transitions from states of one pda to states of another pda. We refer to both models as a *CD pda*.

CD pda's accept languages similar to the way ordinary pda's 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. A fixed head scans the tape from left to right, one symbol at a time, while a mobile head 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. This 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 it is part of the language accepted by a CD pda accepting by empty stack.

Team automata form a formal framework capable of modelling groupware systems [15], multi-user software systems used in CSCW [1,16]. Inspiration came from a call for models capturing concepts of group behaviour [24]. Technically, team automata are an extension of I/O automata [19]. Team automata consist of a set of component automata interacting in a coordinated way by synchronizations of common actions. Component automata differ from ordinary automata by their lack of final states and 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, while external actions are observable by other components. The latter are used for communication between components and comprise both input 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, thus showing their usefulness within CSCW [5,6,7]. Research on team automata focuses on their modelling capacity rather than on their accepting capacity.

Like CD pda's and team automata, team pda's are comprised of a specific type of automata — pda's in this case — that by means of a certain strategy accept one language. Abstracting from the alphabet partition, we thus augment team automata with stacks. Seen as memories, or as notebooks in which the components can make sketches, these stacks thus enhance the modelling capacity of team automata. A team pda accepts languages similar to the way CD pda's do. However, 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 pda's with team behaviour. The size of a team is based on the so-called *mode of competence* a team pda is working in. 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. By requiring the topmost symbol of each stack of a team to be equal, we model that the components forming a team are equally competent, i.e. have the same skills.

In this paper we focus on team pda's with non-deterministic pda's and acceptance by empty stack. We show that team pda's consisting of two pda's can generate non-context-free languages, while team pda's consisting of three pda's can accept all recursively enumerable languages. All proofs can be found in [4].

2 Preliminaries

We assume the reader to be familiar with the basic notions from formal language theory and automata theory, in particular concerning pushdown automata [23].

Set inclusion is denoted by $\subseteq$, while $\subset$ denotes strict inclusion. Set difference of sets V and W is denoted by $V \setminus W$. We denote the cardinality of a finite set V by $\#V$. The powerset of a set V, formed by finite parts of V only, is denoted by $\mathcal{P}_f(V)$. We denote the set $\{1, \ldots, n\}$ by $[n]$. Then $[0] = \varnothing$. The empty word

is denoted by λ and the empty word vector $(\lambda, \ldots, \lambda)$, its dimension being clear from the context, is denoted by Λ. We consider languages L_1 and L_2 to be equal, denoted by $L_1 = L_2$, iff $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$. The classes of context-free and recursively enumerable languages are denoted by $\mathcal{L}(\mathrm{CF})$ and $\mathcal{L}(\mathrm{RE})$, resp. The family of context-free languages that do not contain λ is denoted by $\mathcal{L}(\mathrm{CF}-\lambda)$.

A *pushdown automaton* (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 $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \mathcal{P}_f(Q \times \Gamma^*)$, initial state $q^0 \in Q$, and initial stack content $Z^0 \in \Gamma$. The family of languages recognized by [λ-free] pda's is denoted by $\mathcal{L}(\mathrm{PDA}[-\lambda])$ and it is known that $\mathcal{L}(\mathrm{PDA}[-\lambda]) = \mathcal{L}(\mathrm{CF}[-\lambda])$.

The *shuffle* of $u, v \in \Sigma^*$, denoted by $u \,||\, v$, is defined as $u \,||\, v = \{u_1 v_1 \cdots u_n v_n \mid u_i, v_i \in \Sigma^*,\ i \in [n],\ u_1 \cdots u_n = u,\ v_1 \cdots v_n = v\}$. The shuffle of $L_1, L_2 \subseteq \Sigma^*$, denoted by $L_1 \,||\, L_2$, is defined as $L_1 \,||\, L_2 = \{w \in u \,||\, v \mid u \in L_1,\ v \in L_2\} = \bigcup_{u \in L_1,\, v \in L_2} u \,||\, v$. The shuffle operation is commutative and associative. The family of shuffles of n languages from language family $\mathcal{L}(F)$, denoted by *n-Shuf*(F), is defined as *n-Shuf*$(F) = \{L_1 \,||\, \cdots \,||\, L_n \mid L_i \in \mathcal{L}(F),\ i \in [n]\}$. It is known that the shuffle of n context-free languages need not be context-free, for any $n \geq 2$.

3 Teams of Pushdown Automata

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

Given a set $\mathcal{S} = \{\, \mathcal{A}_i \mid i \in [n] \,\}$ of pda's $\mathcal{A}_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q_i^0, Z_i^0)$, the *team pushdown automaton* of degree n (n-team pda) composed over $\mathcal{S}$ is the quintuple

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

where $Q = Q_1 \times \cdots \times Q_n$ is its set of *states*, $\Sigma = \bigcup_{i \in [n]} \Sigma_i$ is its *input alphabet*, $\Gamma = \Gamma_1^* \times \cdots \times \Gamma_n^*$ is its *pushdown alphabet*, $q^0 = (q_1^0, \ldots, q_n^0)$ is its *initial state*, and $Z^0 = (Z_1^0, \ldots, Z_n^0)$ is *initial stack contents*. The pda's in $\mathcal{S}$ are called the *component pda's* of $\mathcal{T}$. We assumed $n \geq 2$ since a 1-team pda would be a pda.

Configuration of $\mathcal{T}$ contain its current — global — state defined by the — local — states its component pda's reside in, the part of the input word still to be read, and the current contents of its n stacks, i.e. they are triples

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

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

$\mathcal{T}$ can *move* from a configuration τ to a configuration τ', denoted by $\tau \vdash \tau'$, if one of its component pda's is competent in τ. Assuming that $\mathcal{A}_j$, 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 of the remaining part of the input word to be read, and replacing the topmost symbol of the j-th stack. Formally,

$$((p_1, \ldots, p_n), aw, (Z_1\beta_1, \ldots, Z_n\beta_n)) \vdash ((q_1, \ldots, q_n), w, (\alpha_1\beta_1, \ldots, \alpha_n\beta_n))$$

if there exists a $j \in [n]$ for which $(q_j, \alpha_j) \in \delta_j(p_j, a, Z_j)$ and for all $i \in [n] \setminus \{j\}$, $q_i = p_i$ 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*.

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, $\mathcal{T}$ *accepts* w if its stack content is the empty word vector when w has been completely read. All words that can be accepted in this way together form the language of $\mathcal{T}$. Consequently, the language accepted by $\mathcal{T}$, denoted by $L_*(\mathcal{T})$, is thus defined as

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

with $\vdash^*$ as reflexive transitive closure of $\vdash$. The family of languages recognized by [λ-free] n-team pda's is denoted by $\mathcal{L}(\text{n-PDA}[-\lambda], *\text{-comp})$.

Example 1. Consider the pda's $\mathcal{A}_1 = (Q_1, \{a,b,c\}, \{A\}, \delta_1, p^0, A)$ and $\mathcal{A}_2 = (Q_2, \{a,b,c\}, \{A\}, \delta_2, q^0, A)$, where $Q_1 = \{p^0\} \cup \{p_i \mid i \in [3]\}$ and $Q_2 = \{q^0\} \cup \{q_i \mid i \in [3]\}$, and whose transition mappings δ_1 and δ_2 are defined by

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

Clearly the languages accepted by these pda's are $L(\mathcal{A}_1) = \{\, a^m b^n c^m \mid m, n \geq 1 \,\}$ and $L(\mathcal{A}_2) = \{\, a^k b^{k+1} \mid k \geq 1 \,\} \cup \{\, a^k b^k c^m \mid k, m \geq 1 \,\}$. Next 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 $\{\mathcal{A}_1, \mathcal{A}_2\}$. One easily verifies that the language accepted by $\mathcal{T}_1$ is $L_*(\mathcal{T}_1) = L(\mathcal{A}_1) \mid\mid L(\mathcal{A}_2)$. □

No matter how many component pda's a team pda consists of, no team feature is actually used: each (global) move of a team pda is brought about by a (local) move of an arbitrary one of its component pda's. Any n-team pda thus recognizes the shuffle of n context-free languages. The converse is also true, viz.

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

Note that any λ-free n-team pda can be simulated by an n-team pda with λ-moves. Moreover, any n-team pda can be simulated by an $(n+1)$-team pda: if the n-team pda is composed over $\mathcal{S}$, then it suffices to compose the $(n+1)$-team pda over $\mathcal{S}$ augmented by a pda with an empty transition mapping. However,

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

4 Competence in Teams of Pushdown Automata

Until now an n-team pda can move if at least one component pda is competent. A global move is then brought about by the local move of an arbitrary competent pda, i.e. the component pda's of an n-team pda do not cooperate in any way.

In this section we define ways of cooperation between the pda's constituting an n-team pda $\mathcal{T}$ by requiring a precise number of them to be competent before $\mathcal{T}$ can move. These competent pda's are moreover required to be equally competent among each other, i.e. 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 $k \geq 1$, requiring at most k, exactly k, or at least k component pda's, resp., to be equally competent among each other and allowing no other component pda's to be equally competent with any of them. This resembles maximal competence defined in [2]. From now on we ignore the ≤ 1-mode of competence as it equals the $=1$-mode of competence.

Let us say that in the previous section n-team pda's accepted languages in the $*$-mode of competence. Note that this is not the same as the $=1$-mode of competence, since in the $*$-mode of competence an arbitrary one of the possibly many equally competent component pda's is chosen to bring about the global move of $\mathcal{T}$, thus completely disregarding whether or not any other component pda is equally competent with it. Below we will show this with an example.

For the sequel we let $\mathcal{T} = (Q, \Sigma, \Gamma, q^0, Z^0)$ be the n-team pda composed over the set $\mathcal{S} = \{ \mathcal{A}_i \mid i \in [n] \}$ of component pda's $\mathcal{A}_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q_i^0, Z_i^0)$, we let $f \in \{ \leq k \mid k \geq 2 \} \cup \{ =k, \geq k \mid k \geq 1 \}$ be the *modes of competence*, and we say that a natural number ℓ *satisfies* f iff ℓf (e.g. 4 satisfies ≤ 7 because $4 \leq 7$).

$\mathcal{T}$ can *move* from a configuration

$$\tau = ((p_1, \ldots, p_n), aw, (Z_1\beta_1, \ldots, Z_n\beta_n))$$

to a configuration

$$((q_1, \ldots, q_n), w, (\alpha_1\beta_1, \ldots, \alpha_n\beta_n))$$

in *f-comp-mode*, denoted by

$$((p_1, \ldots, p_n), aw, (Z_1\beta_1, \ldots, Z_n\beta_n)) \vdash^f ((q_1, \ldots, q_n), w, (\alpha_1\beta_1, \ldots, \alpha_n\beta_n))$$

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

$Z_j = X$, for some $X \in \bigcap_{t \in J} \Gamma_t$,
$(q_j, \alpha_j) \in \delta_j(p_j, a, X)$, i.e. all $\mathcal{A}_j$ are equally competent in τ,

and for all $i \in [n] \setminus J$, either

$Z_i \neq X$ or
$\delta_i(p_i, a, X)$ is undefined, i.e. $\mathcal{A}_i$ is not equally competent in τ with above $\mathcal{A}_j$'s.

If $a = \lambda$, then we have a λ-*move*, and an n-team pda without λ-moves is λ-*free*.

$\mathcal{T}$ can thus move from a configuration τ to a configuration τ' in the f-comp-mode if f component pda's are equally competent among each other in τ. A global move of $\mathcal{T}$ is now brought about by all these component pda's changing their local state, reading the first letter of the remaining part of the input word

to be read, and replacing the unique symbol on top of their stacks. Consequently, the *language* accepted by $\mathcal{T}$ in f-comp-mode, denoted by $L_f(\mathcal{T})$, is defined as

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

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

Example 2. (Ex. 1 cont.) As no initial configuration of $\mathcal{T}_1$ has only one competent component pda, the language accepted by $\mathcal{T}_1$ in the $=$1-comp-mode is $L_{=1}(\mathcal{T}_1) = \varnothing$, which does not equal $L_*(\mathcal{T}_1)$. Clearly the language accepted by $\mathcal{T}_1$ in the $=$2-comp-mode or the $\geq$2-comp-mode is the non-context-free language $L_{=2}(\mathcal{T}_1) = L_{\geq 2}(\mathcal{T}_1) = \{\, a^n b^n c^n \mid n \geq 1 \,\}$, while that accepted by $\mathcal{T}_1$ in the $\geq$1-comp-mode or the $\leq$2-comp-mode is the non-context-free language $L_{\geq 1}(\mathcal{T}_1) = L_{\leq 2}(\mathcal{T}_1) = \{\, a^m b^n c^m \mid 1 \leq m \leq n \,\} \cup \{\, a^k b^k c^m \mid 1 \leq k \leq m \,\}$. □

Note that any λ-free n-team pda in f-comp-mode can be simulated by an n-team pda with λ-moves in f-comp-mode. Moreover, any n-team pda in f-comp-mode can be simulated by an $(n+1)$-team pda in f-comp-mode. Finally,

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

We now study the accepting capacity of n-team pda's in the f-comp-mode.

Theorem 4. *Let* $g_1 \in \{=k, \geq k \mid k \geq 2\}$ *and* $g_2 \in \{=1, \geq 1\} \cup \{\leq k \mid k \geq 2\}$. *Then*
(1) $\mathcal{L}((\text{k}\cdot\text{n})\text{-PDA}[-\lambda], g_1\text{-comp}) \supset \mathcal{L}(\text{n-PDA}[-\lambda], *\text{-comp})$ *and*
(2) $\mathcal{L}(\text{n-PDA}[-\lambda], g_2\text{-comp}) \supset \mathcal{L}(\text{n-PDA}[-\lambda], *\text{-comp})$. □

By using the fact that the family of languages recognized by *two-stack pda's* equals $\mathcal{L}(\text{RE})$, we are able to prove that increasing the number of pda's in a team pda need not always lead to an infinite hierarchy.

Theorem 5. $\mathcal{L}(\text{3-PDA}, =2\text{-comp}) = \mathcal{L}(\text{3-PDA}, \geq 2\text{-comp}) = \mathcal{L}(\text{RE})$. □

As the proof of this theorem uses λ-moves, it remains an open problem to establish the accepting capacity of λ-free n-team pda's, with $n \geq 3$. We conjecture prohibiting λ-moves in general decreases the accepting capacity of n-team pda's.

5 Future Work

We could add final states to pda's and study the accepting capacity of team pda's with acceptance by final states. We could also study the accepting capacity of team pda's with deterministic component pda's. Since the family of languages accepted by deterministic pda's is strictly included in $\mathcal{L}(\text{CF})$, this could result in a different picture. As an initial result, we note that — in analogy with Theorem 1

— the language accepted by an n-team pda with deterministic component pda's in the $*$-comp-mode and under acceptance by empty stack, equals the shuffle of the languages accepted by its n deterministic component pda's.

In [6], a variety of access control protocols was formally modelled by team automata. The type of protocols modelled was limited, however, because team automata can only deal with pure communication: their constituting component automata can synchronize their actions, but the lack of a private memory blocks them from exchanging information. Since team pda's augment team automata with a (distributed) pushdown memory and thereby allow the flow of information among their consituting component automata, a larger variety of access control protocols can potentially be modelled.

Team pda's seem capable of modelling more groupware protocols in CSCW.

References

1. R.M. Baecker (ed.), *Readings in Groupware and CSCW*, Morgan Kaufmann, 1992.
2. H. Bordihn and E. Csuhaj-Varjú, On competence and completeness in CD grammar systems. *Acta Cybernetica* 12, 4 (1996), 347–361.
3. M.H. ter Beek, E. Csuhaj-Varjú, M. Holzer, and Gy. Vaszil, On Competence in Cooperating Distributed Grammar Systems, parts I-III. TR 2002/1–3, Computer and Automation Research Institute, Hungarian Academy of Sciences, 2002.
4. M.H. ter Beek, E. Csuhaj-Varjú, and V. Mitrana, Teams of Pushdown Automata. TR 2002/4, Computer and Automation Research Institute, Hungarian Academy of Sciences, 2002.
5. M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg, Team Automata for CSCW. In *Proc. 2nd Int. Coll. on Petri Net Technologies for Modelling Communication Based Systems*, Fraunhofer Institute for Software and Systems Engineering, 2001, 1–20.
6. M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg, Team Automata for Spatial Access Control. In *Proc. 7th Eur. Conf. on CSCW*, Kluwer Academic, 2001, 59–77.
7. M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg, Synchronizations in team automata for groupware systems. *CSCW* 12, 1 (2003), 21–69.
8. E. Csuhaj-Varjú and J. Dassow, On cooperating distributed grammar systems. *Journal of Information Processing and Cybernetics EIK* 26 (1990), 49–63.
9. E. Csuhaj-Varjú, J. Dassow, and M. Holzer, On a competence-based cooperation strategy in CD grammar systems. Submitted, 2002.
10. E. Csuhaj-Varjú, J. Dassow, J. Kelemen, and Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, 1994.
11. E. Csuhaj-Varjú and J. Kelemen, Cooperating Grammar Systems: a Syntactical Framework for the Blackboard Model of Problem Solving. In *Proc. AI and Information-control Systems of Robots*, North-Holland, 1989, 121–127.
12. E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana, and Gy. Vaszil, Parallel Communicating Pushdown Automata Systems. *IJFCS* 11, 4 (2000), 633–650.
13. E. Csuhaj-Varjú, V. Mitrana, and Gy. Vaszil, Distributed Pushdown Automata Systems: Computational Power. To appear in *Proc. 7th. Int. Conf. on Developments in Language Theory*, *LNCS*, Springer, 2003.
14. J. Dassow and V. Mitrana, Stack Cooperation in Multistack Pushdown Automata. *Journal of Computer and System Sciences* 58, 3 (1999), 611–621.

15. C.A. Ellis, Team Automata for Groupware Systems. In *Proc. Int. Conf. on Supporting Group Work: The Integration Challenge*, ACM Press, 1997, 415–424.
16. J. Grudin, CSCW: History and Focus. *IEEE Computer* 27, 5 (1994), 19–26.
17. L. Kari, A. Mateescu, Gh. Păun and A. Salomaa, Teams in cooperating grammar systems, *Journal of Experimental and Theoretical AI* 7 (1995), 347–359.
18. K. Krithivasan, M. Sakthi Balan, and P. Harsha, Distributed Processing in Automata. *IJFCS* 10, 4 (1999), 443–463.
19. N.A. Lynch, *Distributed Algorithms*, Morgan Kaufmann, 1996.
20. P.H. Nii, Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. Part I. *AI Magazine* 7, 2 (1986), 38–53.
21. P.H. Nii, Blackboard Systems. In *Handbook of AI* vol. 4, Addison-Wesley, 1989, 1–82.
22. Gh. Păun and G. Rozenberg, Prescribed teams of grammars. *Acta Informatica* 31 (1994), 525–537.
23. G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Springer, 1997.
24. J. Smith, *Collective Intelligence in Computer Based Collaboration — A Volume in the Computers, Cognition, and Work Series*, Lawrence Erlbaum, 1994.