Journal of Automata, Languages and Combinatorics **19** (2014) 1–4, 17–32
© Otto-von-Guericke-Universität Magdeburg

ON DISTRIBUTED COOPERATION AND SYNCHRONISED COLLABORATION

Maurice H. ter Beek
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"(ISTI), CNR
Area della Ricerca di Pisa, Via G. Moruzzi 1, 56124 Pisa
e-mail: maurice.terbeek@isti.cnr.it

and

Jetty Kleijn
Leiden Institute of Advanced Computer Science (LIACS), Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
e-mail: h.c.m.kleijn@liacs.leidenuniv.nl

ABSTRACT

In CD grammar systems, the rewriting process is distributed over component grammars that take turns in the derivation of new symbols. Team automata however collaborate by synchronising their actions. Here we investigate how to transfer this concept of synchronisation to grammars by defining grammar teams that agree on the generation of shared terminal symbols based on a novel notion of competence. We first illustrate this idea for the case of regular grammars and next propose an extension to the case of context-free grammars.

Keywords: CD grammar systems; Team automata; Synchronisation

1. Introduction

When a scientist encounters difficulties when trying to solve a complex problem, (s)he might try to approach the problem in cooperation or collaboration with other scientist(s). To this aim, modern means of communication like e-mail and skype are used but also blackboards. In Artificial Intelligence, the blackboard model of problem solving was defined as a method of cooperative problem solving. The problem is specified on a blackboard. A number of participants contribute, regulated by a cooperation strategy, to the solution of the problem by editing the blackboard. During this cooperative process, the participants communicate through the blackboard. If their cooperation is successful, the solution will appear on the blackboard.

Together with an expert in AI, Erzsébet Csuhaj-Varjú established a link between this blackboard model of problem solving and formal languages [13]. In [11], so-called Cooperating Distributed (CD) grammar systems have been proposed to formalise this link. The participants are modelled as grammars that edit the blackboard by rewriting a sentential form taking turns. Their cooperation strategy is regulated by so-called derivation modes and solutions are represented by terminal words. Contrary to the classical notion of a single grammar generating a language, the grammars forming a grammar system work together and generate a language by cooperating. This idea has given rise to a lively field of formal language theory known as grammar systems. In ter Beek's M.Sc. thesis, supervised by Csuhaj-Varjú, Kleijn, and Rozenberg, the idea of teams in grammar systems – originally introduced in [20] – was explored [1, 2]. Whereas in CD grammar systems, grammars rewrite sequentially taking turns, grammars forming a team within the system rewrite several occurrences of nonterminals in parallel. On the other hand, in team automata (see, e. g., ter Beek's Ph.D. thesis [3] and [6, 9, 10]), the basic concept is collaboration through synchronisation rather than distributed cooperation either by taking turns in rewriting or parallel rewriting. The automata forming a team automaton collaborate to recognise a language by synchronising on shared actions (i. e., they agree on the simultaneous execution of labelled transitions) according to a specific synchronisation strategy.

In this paper, we propose to transfer the concept of synchronisation to grammars. We define teams of grammars that synchronise on the generation of shared terminal symbols based on a novel notion of competence of the grammars forming the team. Thus grammars should now agree on which terminal to insert in the sentential form and so this concept adds to the blackboard model the idea that experts collaborate and agree on a common contribution to the solution.

After a preliminary section we formally introduce CD grammar systems and team automata, pointing out resemblances and differences between the distributed cooperation strategies of CD grammar systems and the synchronised collaboration strategies of team automata. Using the close relationship between finite automata and regular grammars, we first focus on the synchronisation of regular grammars. Next we demonstrate how this concept could be extended to the case of context-free grammars. In the concluding section we reflect on what we have achieved, mention related work, and indicate some interesting questions for future research. Although the investigations presented in this paper have no conclusive results yet, we would like to argue that they point towards a hitherto missing link between distributed cooperation and synchronised collaboration in the context of rewriting systems.

2. Preliminaries

The following notation is used: $\setminus$ for set difference; $\subseteq$ for set inclusion and $\subset$ for strict inclusion; and $[n]$ for the set $\{1, \ldots, n\}$. Let $V_1, \ldots, V_n$ be sets. Then $\prod_{i \in [n]} V_i$ is their cartesian product. If $v = (v_1, \ldots, v_n) \in \prod_{i \in [n]} V_i$ and $i \in [n]$, then the i-th entry of v is obtained by applying the projection function $\mathrm{proj}_i(v) = v_i$.

We assume some familiarity with formal language theory [24]. An alphabet is a finite nonempty set of symbols. A word over an alphabet Σ is a finite string of symbols from Σ. By $|w|$ we denote the length of a word w, i. e., the number of occurrences of symbols from Σ in w; and $\mathrm{alph}(w)$ is the set of symbols that occur at least once in w. We use λ to denote the empty word; $|\lambda| = 0$. The set of all nonempty words over Σ is denoted by Σ^+ and Σ^* consists of all words over Σ. Any subset of Σ^* is a language (over Σ).

A context-free grammar is a quadruple $G = (N, T, S, P)$, with disjoint finite sets N of nonterminal symbols and T of terminal symbols, axiom $S \in N$, and finite set P of productions of the form $A \to \alpha$, for $A \in N$ and $\alpha \in (N \cup T)^*$. The alphabet of a production $\pi : A \to \alpha$ is defined as $\mathrm{alph}(\pi) = \mathrm{alph}(A\alpha)$ and its terminal alphabet as $\mathrm{alph}(\pi) \cap T$. If all productions of G are of the form $A \to zB$ or $A \to z$, for $A, B \in N$ and $z \in T \cup \{\lambda\}$, then G is regular. G is in Greibach normal form (GNF) if each production in P is of the form $S \to \lambda$ or $A \to z\alpha$, for $A \in N$, $z \in T$, $\alpha \in (N \setminus \{S\})^*$, and $|\alpha| \leq 2$ [19]. In this paper, we often assume context-free grammars to be in GNF.

A word x directly derives a word y in G, $x \Rightarrow_G y$, iff $x = x_1Ax_2$, $y = x_1\alpha x_2$, and $A \to \alpha \in P$ for $x_1, x_2 \in (N \cup T)^*$. Such a one-step derivation is extended to a k-step derivation, $x \Rightarrow_G^k y$ for some $k \geq 0$, iff there are words $x_0, \ldots, x_k$ such that $x = x_0$, $y = x_k$ and $x_i \Rightarrow_G x_{i+1}$, for all $0 \leq i \leq k-1$. G may be omitted if it is clear from the context. The transitive (and reflexive) closure of $\Rightarrow$ is denoted by $\Rightarrow^+$ ($\Rightarrow^*$).

The language generated by G is defined as $L(G) = \{\, w \in T^* \mid S \Rightarrow_G^* w \,\}$. A word $w \in (N \cup T)^*$ is called a sentential form (word if $w \in T^*$) of G iff $S \Rightarrow_G^* w$. The family of context-free (regular) languages generated by context-free (regular) grammars is denoted by CF (REG, respectively). It is well known that $REG \subset CF$.

3. CD Grammar Systems and Team Automata

In this section we first recall CD grammar systems, in which grammars cooperate by taking turns in rewriting according to their own productions until they have to hand over the string as determined by a cooperation strategy (mode). Subsequently we recall team automata, which consist of a conglomerate of component automata forming a product automaton in which some combinations of components' transitions with a common label are synchronised.

3.1. CD Grammar Systems

The underlying philosophy is that grammars cooperate by individually rewriting a sentential form repeatedly until passing it to a colleague.

A *cooperating distributed grammar system* (CDGS) of degree $n \geq 1$ is a construct $G = (N, T, S, P_1, \ldots, P_n)$, with finite sets N of nonterminals and T of terminals, $N \cap T = \varnothing$, axiom $S \in N$, and finite sets $P_1, \ldots, P_n$ of context-free productions of the form $A \to \alpha$, for $A \in N$ and $\alpha \in (N \cup T)^*$. We refer to both the P_i and $G_i = (N, T, S, P_i)$ as *component grammars* or components for short.

Let $1 \leq i, k \leq n$ and $x, y \in (N \cup T)^*$. We define a (sequential) *rewriting step* as $x \Rightarrow_i y$ iff $x = x_1Ax_2$ and $y = x_1\alpha x_2$, for some $A \to \alpha \in P_i$ and we denote a k-step derivation by $\Rightarrow_i^k$. Subscript i refers to the component used. A CDGS G rewrites by means of a *cooperation strategy* $f \in \{\, {\leq}k, {=}k, {\geq}k \mid k \geq 1 \,\} \cup \{*, t\}$, defined as follows:

$x \Rightarrow_G^{\leq k} y$ iff there exists a P_i such that $x \Rightarrow_i^0 y$ or $x \Rightarrow_i^j y$ for some $j \leq k$,

$x \Rightarrow_G^{=k} y$ iff there exists a P_i such that $x \Rightarrow_i^k y$,

$x \Rightarrow_G^{\geq k} y$ iff there exists a P_i such that $x \Rightarrow_i^\ell y$ for some $\ell \geq k$,

$x \Rightarrow_G^* y$ iff there exists a P_i such that $x \Rightarrow_i^0 y$ or $x \Rightarrow_i^k y$ for some k,

$x \Rightarrow_G^t y$ iff there exists a P_i such that $x \Rightarrow_i^* y$ and no z exists for which $y \Rightarrow_i z$.

The *language* generated by a CDGS G depends on the cooperation strategy:

$$L^f(G) = \{ w \in T^* \mid S = w_1 \Rightarrow_G^f \cdots \Rightarrow_G^f w_n = w \text{ and } n \geq 1 \}.$$

The language family generated by CD grammar systems with context-free productions, at most n components, and cooperation strategy f, with

$$f \in \{ \leq k, =k, \geq k \mid k \geq 1 \} \cup \{*, t\},$$

is denoted by $CD_n(f)$. Denote $CD(f) = \bigcup_{n \geq 1} CD_n(f)$ and $CD_{REG}(f)$ when limited to regular productions.

Example 1 Consider the CDGS $G_1 = (\{S, A, B\}, \{a, b, c\}, S, P_1, P_2, P_3)$, with

$$P_1 = \{S \to aS, S \to aB\},\ P_2 = \{B \to bB, B \to bC\},\ P_3 = \{C \to cC, C \to c\}.$$

Then $L^t(G_1) = a^+b^+c^+$. Derivations are as follows:

$$\begin{aligned} S &\Rightarrow_1^t aS \Rightarrow_1^t \cdots \Rightarrow_1^t a^+B \Rightarrow_2^t a^+bB \Rightarrow_2^t \cdots \Rightarrow_2^t a^+b^+C \\ &\Rightarrow_3^t A^+b^+cC \Rightarrow_3^t \cdots \Rightarrow_3^t a^+b^+c^+. \end{aligned}$$

Example 2 Consider the CDGS $G_2 = (\{S, A, B, A', B'\}, \{a, b, c\}, S, P_1, P_2, P_3)$, with $P_1 = \{S \to \lambda, S \to aA'S', S' \to bB'S'', S'' \to cC', A \to aA', B \to bB', C \to cC'\}$, $P_2 = \{A' \to aA, B' \to bB, C' \to cC\}$, and $P_3 = \{A' \to a, B' \to b, C' \to c\}$. Then $L^t(G_2) = \{ a^n b^n c^n \mid n \text{ is even} \}$. Derivations are as follows:

$$S \Rightarrow_1^t aA'bB'cC' \Rightarrow_2^t a^2Ab^2Bc^2C \Rightarrow_1^t \cdots \Rightarrow_1^t a^{n-1}A'b^{n-1}B'c^{n-1}C' \Rightarrow_3^t a^nb^nc^n.$$

Hence CD grammar systems with context-free components can produce non-context-free languages. From [12] we know that for $g \in \{ =k, \geq k \mid k \geq 2 \}$, $k \geq 1$, and $k', k'' \geq 2$,

$$\begin{aligned} &CF = CD(=1) = CD(\geq 1) = CD(*) = CD(\leq k) \subset (CD(=k') \cap CD(\geq k'')), \\ &CF = CD_1(g) \subset CD_2(g) \subseteq CD_3(g) \subseteq \cdots \subseteq CD(g), \text{ and} \\ &CF = CD_1(t) = CD_2(t) \subset CD_3(t) = CD(t) = ET0L, \end{aligned}$$

the family of *ET0L* languages [24]; in case of regular components however

$$REG = CD_{REG}(f), \text{ for } f \in \{ \leq k, =k, \geq k \mid k \geq 1 \} \cup \{*, t\}.$$

3.2. Team Automata

The underlying philosophy is that (finite) automata collaborate by jointly executing (synchronising) transitions with the same label as agreed upon upfront.

A (component) *automaton* is a quintuple $\mathcal{A} = (Q, \Sigma, I, \delta)$, with finite sets Q of states, Σ of actions, $Q \cap \Sigma = \varnothing$, $I \subseteq Q$ of initial states, and $\delta \subseteq Q \times \Sigma \times Q$ of (Σ-labelled) transitions. In addition there is a partition of the actions into input, output, and internal actions (less relevant to this paper). Let $\delta_a = \{\, (q, q') \mid (q, a, q') \in \delta \,\}$ denote the set of all a-transitions, for $a \in \Sigma$. Note that no final states are specified: all states are viewed as accepting states and computations are seen as ongoing runs.

The set of *computations* of $\mathcal{A}$ is defined as

$$C(\mathcal{A}) = \{\, q_0 a_1 q_1 \cdots a_n q_n \mid (q_{i-1}, a_i, q_i) \in \delta \text{ for all } i \in [n] \text{ and } n \geq 0 \,\}.$$

Its *language* is defined as $L(\mathcal{A}) = \mathrm{pres}_\Sigma(C(\mathcal{A}))$ with the homomorphism $\mathrm{pres}_{\Gamma,\Sigma}$ that preserves the symbols from Σ and erases all other symbols in Γ, being defined by $\mathrm{pres}_{\Gamma,\Sigma}(a) = \lambda$ if $a \in \Gamma \setminus \Sigma$ and $\mathrm{pres}_{\Gamma,\Sigma}(a) = a$ if $a \in \Gamma \cap \Sigma$. Clearly, since each state is accepting, the languages of component automata are exactly the prefix-closed regular languages, with a language K being prefix closed if $\mathrm{pref}(K) \subseteq K$, where $\mathrm{pref}(K)$ denotes the set of prefixes of the words of K.

Team automata are composed of component automata and have as their actions the components' actions and as states the cartesian products of the components' states, while their transitions are synchronisations of components' transitions. Let $\mathcal{S} = \{\, \mathcal{A}_i = (Q_i, \Sigma_i, I_i, \delta_i) \mid i \in [n] \,\}$ be a set of component automata, $\Sigma = \bigcup_{i \in [n]} \Sigma_i$, and $a \in \Sigma$. The set of *synchronisations* on a is defined as

$$\begin{aligned}\Delta_a(\mathcal{S}) = \{\, (q, q') \in \prod_{i \in [n]} Q_i \times \prod_{i \in [n]} Q_i \mid \exists j \in [n] \colon (\mathrm{proj}_j(q), \mathrm{proj}_j(q)') \in \delta_{j,a} \\ \text{and } \forall i \in [n] \colon (\mathrm{proj}_i(q), \mathrm{proj}_i(q')) \in \delta_{i,a} \text{ or } \mathrm{proj}_i(q) = \mathrm{proj}_i(q') \,\}.\end{aligned}$$

Hence $\Delta_a(\mathcal{S})$ contains all possible combinations of a-transitions of the components in $\mathcal{S}$, with all non-participating components remaining in the same state. Note that in every synchronisation always *at least one* component takes part.

A specific team automaton is defined by *choosing*, for each action a, a subset of $\Delta_a(\mathcal{S})$, all possible synchronisations on a: A *team automaton* over $\mathcal{S}$ is a quadruple $\mathcal{T} = (Q, \Sigma, I, \delta)$, with $Q = \prod_{i \in [n]} Q_i$, $I = \prod_{i \in [n]} I_i$, and $\delta \subseteq Q \times \Sigma \times Q$ such that $\delta_a \subseteq \Delta_a(\mathcal{S})$ for all $a \in \Sigma$. Hence each choice of synchronisations defines a team automaton and every team automaton is again a component automaton.

Which synchronisations to select for a team automaton may be determined by an a priori defined *synchronisation strategy*. Such a strategy should imply the specification for each action a of a subset $\mathcal{R}_a(\mathcal{S})$ of all synchronisations on that action that should be included as a-transitions of the team. If we now let $\mathcal{R} = \{\, \mathcal{R}_a(\mathcal{S}) \mid a \in \Sigma \,\}$, then $\mathcal{T}$ is the $\mathcal{R}$-*team automaton* over $\mathcal{S}$ if $\delta_a = \mathcal{R}_a(\mathcal{S})$ for all $a \in \Sigma$.

We mention two obvious strategies. Action a is *action-indispensable* (ai) in $\mathcal{T}$ if all a-transitions of $\mathcal{T}$ are the result of a synchronisation of all components from $\mathcal{S}$ that have a as an action; and a is *state-indispensable* (si) in $\mathcal{T}$ if all its a-transitions are the result of a synchronisation of all components from $\mathcal{S}$ in which a is currently enabled;

a is enabled at state q in an automaton $\mathcal{A}$ if there exists a state q' such that (q, q') is an a-transition of $\mathcal{A}$. We now specify the corresponding synchronisation strategies:

$$\mathcal{R}^{ai}_a(\mathcal{S}) = \{(q, q') \in \Delta_a(\mathcal{S}) \mid \forall i \in [n] : \text{if } a \in \Sigma_i, \text{ then } (\mathrm{proj}_i(q), \mathrm{proj}_i(q')) \in \delta_{i,a}\},$$
$$\mathcal{R}^{si}_a(\mathcal{S})] = \{(q, q') \in \Delta_a(\mathcal{S}) \mid \forall i \in [n] : \text{if } a \in \Sigma_i, \text{ then } (\mathrm{proj}_i(q), \mathrm{proj}_i(q')) \in \delta_{i,a} \text{ whenever } a \text{ is enabled in } \mathcal{A}_i \text{ at state } q\}.$$

Note that the ai strategy defines the standard (synchronised) product of automata. In [6] also other synchronisation strategies have been defined based on the input-output role of actions to model more advanced forms of collaboration between components, such as peer-to-peer and master-slave relationships.

Example 3 Consider the two automata depicted in Fig. 1(a). We have $L(\mathcal{A}_1) = a^*b^*$ and $L(\mathcal{A}_2) = b^*c^*$. The $\mathcal{R}^{ai}$- and $\mathcal{R}^{si}$-team automata $\mathcal{T}^{ai}$ and $\mathcal{T}^{si}$ over $\{\mathcal{A}_1, \mathcal{A}_2\}$ are depicted in Figs. 1(b) and 1(c), respectively. We have $L(\mathcal{T}^{ai}) = a^* \cup a^*b^+c^*$, while $L(\mathcal{T}^{si}) = a^* \cup a^*b^+\{b, c\}^*$. Note that $bcb \in L(\mathcal{T}^{si})$ and hence the relation $\mathrm{pres}_{\{a,b,c\},\{b,c\}}(L(\mathcal{T}^{si})) \not\subseteq L(\mathcal{A}_2)$ even though the alphabet of $\mathcal{A}_2$ is $\{b, c\}$.

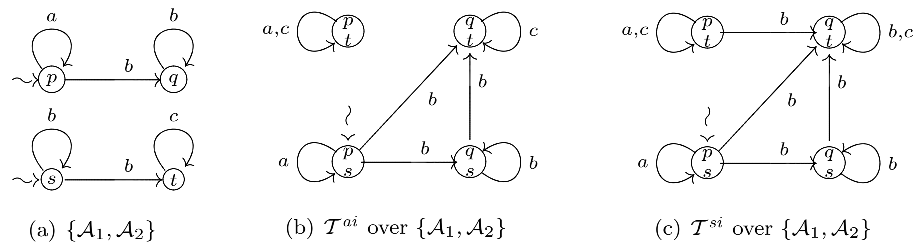
a
b
b
p
q
b
c
b
s
t
(a) $\{\mathcal{A}_1, \mathcal{A}_2\}$
a,c
c
b
b
a
b
b
(b) $\mathcal{T}^{ai}$ over $\{\mathcal{A}_1, \mathcal{A}_2\}$
a,c
b
b,c
b
b
a
b
b
(c) $\mathcal{T}^{si}$ over $\{\mathcal{A}_1, \mathcal{A}_2\}$

Figure 1: The $\mathcal{R}^{ai}$- and $\mathcal{R}^{si}$-team automata over $\{\mathcal{A}_1, \mathcal{A}_2\}$.

Every team automaton is a component automaton and thus its language is a prefix-closed regular language. Interesting features to investigate concern how, depending on the synchronisation strategy, the language of a team automaton can be defined in terms of the languages of its component automata [9, 7].

4. Teams of Regular Grammars

In this section we discuss how to transfer the idea of synchronisation in team automata to a team of collaborating regular grammars, the idea being that the grammars jointly generate a common terminal symbol. We consider the regular case first, as we can then rely on the close resemblance between automata and regular grammars. If we think of nonterminals as states and terminals as actions, then nonterminals are to be combined into cartesian products. It should be noted however that there is no obvious corresponding state once the rewriting in a grammar has terminated and it is no longer represented by a nonterminal. Suppose, e. g., that we want to introduce a terminal z when rewriting a sentential form $\binom{S}{S}$ by synchronising productions $S \to zB$

and $S \to z$. Then what should the resulting sentential form look like? An option would be to allow $\binom{S}{S} \Rightarrow z\binom{B}{\lambda}$ and use λ to indicate termination. We however prefer not to have 'empty' places in nonterminal vectors and therefore introduce a special symbol, #, to mark explicitly the termination of a component grammar. This symbol cannot be rewritten and corresponds to a final sink state in an automaton. For above example, this means we allow $\binom{S}{S} \Rightarrow z\binom{B}{\#}$, after which only the B can still be rewritten.

Formally, we consider #-extended regular grammars $G = (N, T \cup \{\#\}, S, P)$ in which all productions are of the form $A \to zB$ or $A \to z\#$, for $A, B \in N$, $z \in T \cup \{\lambda\}$, and $\# \notin N \cup T$. The underlying regular grammar $\underline{G}$ of G is obtained by removing # as a terminal symbol and replacing all occurrences of # by λ. Note that $L(G) = L(\underline{G})\#$.

We are now ready to lift the concept of synchronisation to regular grammars in a rather straightforward manner. For the remainder of this section, let $n \geq 1$, let $\mathcal{G} = \{\, G_i = (N_i, T_i, S_i, P_i) \mid G_i \text{ is a \#-extended regular grammar}, i \in [n] \,\}$, and $a \in T$. The set of *synchronisations* on a (within G) is defined as:

$$\begin{aligned}\Delta_a(\mathcal{G}) = \{\, (p,q) \in {\textstyle\prod_{i\in[n]}}(N_i \cup \{\#\}) \times {\textstyle\prod_{i\in[n]}}(N_i \cup \{\#\}) \mid \\ \exists j \in [n] : \mathrm{proj}_j(p) \to a\mathrm{proj}_j(q) \in P_j \text{ and} \\ \forall\, i \in [n] : \mathrm{proj}_i(p) \to a\mathrm{proj}_i(q) \in P_i \text{ or } \mathrm{proj}_i(p) = \mathrm{proj}_i(q) \,\}.\end{aligned}$$

Hence $\Delta_a(\mathcal{G})$ contains all possible combinations of a-productions (i. e., productions generating the terminal a) from the regular grammars constituting $\mathcal{G}$ such that in every synchronisation on a at least one grammar applies an a-production and all other grammars either do so as well or do not rewrite.

A (regular) grammar team is obtained by choosing, for each a, a subset of $\Delta_a(\mathcal{G})$.

Definition 1 (regular grammar team) *A* (regular) grammar team *over $\mathcal{G}$ is a grammar $\mathcal{T} = (Q, T, S, P)$, in which $Q = \prod_{i\in[n]}(N_i \cup \{\#\})$, $T = \bigcup_{i\in[n]} T_i \setminus \{\#\}$, $S = (S_1, S_2, \ldots, S_n)$, and $P \subseteq \bigcup_{a\in T} \Delta_a(\mathcal{S})$.*

Note that $\mathcal{T}$ is a regular grammar without terminating productions. We nevertheless refer to $\{\, w \in T^* \mid S \Rightarrow_{\mathcal{T}}^* w\begin{pmatrix}\#\\ \vdots\\ \#\end{pmatrix}\}$ as the language generated by $\mathcal{T}$ and denote it by $L(\mathcal{T})$.

Observe that $\prod_{i\in[n]} \#$ indicates termination of all grammars and if we would add a production $\begin{pmatrix}\#\\ \vdots\\ \#\end{pmatrix} \to \lambda$, thus defining the regular grammar $\mathcal{T}'$, then $L(\mathcal{T}) = L(\mathcal{T}')$. The family of languages generated by teams of regular grammars is denoted by TA_{REG}.

We now formalise two *synchronisation strategies* for grammar teams, which correspond to the ai and si synchronisation strategies from team automata. We use the term 'competence' in analogy to the concept of competence of component grammars in CD grammar systems. There however competence refers to the presence of the left-hand sides of productions in a sentential form, whereas here we are interested in the terminal symbols that can be introduced by a component grammar:

P_i is called *a-competent* iff $a \in \mathrm{alph}_T(\pi)$ for some $\pi \in P_i$, i. e., P_i is a-competent if at least one of its productions has a in its right-hand side.

P_i is called *a-competent on a nonterminal A* iff there exists a $\pi : A \to \alpha \in P_i$ such that $a \in \mathrm{alph}_T(\alpha)$, i. e., P_i is a-competent on A if at least one of its productions with A as its left-hand side has an occurrence of a in its right-hand side.

The ai and si synchronisation strategies are translated into the following synchronisation strategies for regular grammar teams:

$$\begin{aligned}\mathcal{R}_a^{ai}(\mathcal{G}) &= \{\, (p,q) \in \Delta_a(\mathcal{G}) \mid \forall i \in [n] : \text{if } P_i \text{ is } a\text{-competent,}\\ &\qquad\qquad \text{then } \mathrm{proj}_i(p) \to a\mathrm{proj}_i(q) \in P \,\},\\ \mathcal{R}_a^{si}(\mathcal{G}) &= \{\, (p,q) \in \Delta_a(\mathcal{G}) \mid \forall i \in [n] : \text{if } P_i \text{ is } a\text{-competent on } \mathrm{proj}_i(p),\\ &\qquad\qquad \text{then } \mathrm{proj}_i(p) \to a\mathrm{proj}_i(q) \in P \,\}.\end{aligned}$$

Let $\mathcal{R} = \{\, \mathcal{R}_a(\mathcal{G}) \mid a \in T \,\}$. $\mathcal{T}$ is the *$\mathcal{R}$-regular grammar team* over $\mathcal{G}$ if $P = \mathcal{R}$.

Example 4 Let $\mathcal{G} = \{G_1, G_2\}$ consist of the #-extended regular grammars

$$\begin{aligned}G_1 &= (\{A, A'\}, \{a,b\}, A, \{A \to aA,\ A \to bA',\ A' \to bA', A' \to b\#\}) \text{ and}\\ G_2 &= (\{B, B'\}, \{b,c\}, B, \{B \to bB, B \to bB', B' \to cB', B' \to c\#\}).\end{aligned}$$

We see that $L(G_1) = \{\, a^k b^\ell \# \mid k \geq 0,\ \ell \geq 2 \,\}$ and $L(G_2) = \{\, b^m c^n \# \mid m \geq 1,\ n \geq 1 \,\}$. The $\mathcal{R}^{ai}$- and $\mathcal{R}^{si}$-regular team grammars $\mathcal{T}_1$ and $\mathcal{T}_2$ over $\mathcal{G}$ both have terminals $\{a,b,c\}$, axiom $\tbinom{A}{B}$, and nonterminal $\{\tbinom{A}{B}, \tbinom{A}{B'}, \tbinom{A'}{B}, \tbinom{A'}{B'}, \tbinom{A}{\#}, \tbinom{A'}{\#}, \tbinom{\#}{B}, \tbinom{\#}{B'}, \tbinom{\#}{\#}\}$. The (ai) productions of $\mathcal{T}_1$ are

$$\begin{aligned}P_1 = {}&\{\tbinom{A}{B} \to a\tbinom{A}{B}, \tbinom{A}{B} \to b\tbinom{A'}{B}, \tbinom{A}{B} \to b\tbinom{A'}{B'}\} \cup \{\tbinom{A'}{B'} \to c\tbinom{A'}{B'}, \tbinom{A'}{B'} \to c\tbinom{A'}{\#}\}\\ &\cup \{\tbinom{A}{B'} \to a\tbinom{A}{B'}, \tbinom{A}{B'} \to c\tbinom{A}{B'}, \tbinom{A}{B'} \to c\tbinom{A}{\#}\} \cup \{\tbinom{\#}{B'} \to c\tbinom{\#}{B'}, \tbinom{\#}{B'} \to c\tbinom{\#}{\#}\}\\ &\cup \{\tbinom{A'}{B} \to b\tbinom{A'}{B}, \tbinom{A'}{B} \to b\tbinom{A'}{B'}, \tbinom{A'}{B} \to b\tbinom{\#}{B}, \tbinom{A'}{B} \to b\tbinom{\#}{B'}\} \cup \{\tbinom{A}{\#} \to a\tbinom{A}{\#}\}.\end{aligned}$$

The (si) productions of $\mathcal{T}_2$ are

$$\begin{aligned}P_2 = P_1 &\cup \{\tbinom{A}{B'} \to b\tbinom{A'}{B'}\} \cup \{\tbinom{A'}{B'} \to b\tbinom{\#}{B'}, \tbinom{A'}{B'} \to b\tbinom{A'}{B'}\} \cup \{\tbinom{A}{\#} \to b\tbinom{A'}{\#}\}\\ &\cup \{\tbinom{A'}{\#} \to b\tbinom{A'}{\#}, \tbinom{A'}{\#} \to b\tbinom{\#}{\#}\} \cup \{\tbinom{\#}{B} \to b\tbinom{\#}{B}, \tbinom{\#}{B} \to b\tbinom{\#}{B'}\}.\end{aligned}$$

Hence a possible derivation of $\mathcal{T}_1$ is

$$\tbinom{A}{B} \Rightarrow^* a^*\tbinom{A}{B} \Rightarrow a^*b\tbinom{A'}{B} \Rightarrow^* a^*bb^*\tbinom{A'}{B} \Rightarrow a^*bb^*b\tbinom{\#}{B'} \Rightarrow^* a^*bb^*bc^*\tbinom{\#}{B'} \Rightarrow a^*bb^*bc^*c\tbinom{\#}{\#}.$$

while $\mathcal{T}_2$ also allows derivations like

$$\tbinom{A}{B} \Rightarrow^* a^*\tbinom{A}{B} \Rightarrow a^*b\tbinom{A'}{B'} \Rightarrow^* a^*b\{b,c\}^*\tbinom{A'}{B'} \Rightarrow a^*b\{b,c\}^*b\tbinom{\#}{B'} \Rightarrow a^*b\{b,c\}^*bc\tbinom{\#}{\#}.$$

In fact, we have $L(\mathcal{T}_1) = \{\, a^k b^\ell c^n \tbinom{\#}{\#} \mid k \geq 0,\ \ell \geq 2,\ n \geq 1 \,\}$, whereas we have that $L(\mathcal{T}_2) = L(\mathcal{T}_1) \cup a^*b\{b,c\}^*bc\tbinom{\#}{\#} \cup a^*b\{b,c\}^*cb\tbinom{\#}{\#}$. Similar to Example 3, we note that $bcb \in L(\mathcal{T}_2)$ and thus $\mathrm{pres}_{\{a,b,c\},\{b,c\}}(L(\mathcal{T}_2)) \not\subseteq L(G_2)$ even though the alphabet of G_2 is $\{b,c\}$.

Regular Grammar Teams vs. Regular CD Grammar Systems

As already observed, the languages defined by regular grammar teams are exactly the regular languages, as is the case for regular CD grammar systems.

Theorem 1 *$TA_{REG} = REG = CD_{REG}(f)$, for $f \in \{\, \leq k, =k, \geq k \mid k \geq 1 \,\} \cup \{*, t\}$.*

Actually, it is more interesting to compare sequential distributed cooperation and synchronised collaboration in this context. We now show how these two mechanisms can simulate each other in the case of regular grammars.

From $\boldsymbol{TA_{REG}}$ to $\boldsymbol{CD_{REG}}$. First we demonstrate how the synchronisations in a grammar team can be simulated by a CDGS. Let $\mathcal{T} = (Q, T, S, P)$ be a regular grammar team over $\mathcal{G}$. Thus P has productions of the form $\left(\begin{smallmatrix}A_1\\ \vdots\\ A_n\end{smallmatrix}\right) \to z\left(\begin{smallmatrix}B_1\\ \vdots\\ B_n\end{smallmatrix}\right)$ for which $\exists\, i \in [n] : A_i \to zB_i \in P_i$ and $\forall j \in [n] : A_j \to zB_j \in P_j$ or $A_j = B_j$.

By interpreting the products $\left(\begin{smallmatrix}A_1\\ \vdots\\ A_n\end{smallmatrix}\right)$ as nonterminal symbols we immediately obtain a CDGS $G = (Q', T, [S_1 \cdots S_n], P')$ of degree 1 with

$$Q' = \{\, [A_1 \cdots A_n] \mid \left(\begin{smallmatrix}A_1\\ \vdots\\ A_n\end{smallmatrix}\right) \in Q \,\} \text{ and}$$

$$P' = \{\, [A_1 \cdots A_n] \to z[B_1 \cdots B_n] \mid \left(\begin{smallmatrix}A_1\\ \vdots\\ A_n\end{smallmatrix}\right) \to z\left(\begin{smallmatrix}B_1\\ \vdots\\ B_n\end{smallmatrix}\right) \in Q \,\} \cup \{[\# \cdots \#] \to \lambda\}.$$

Clearly, G simulates $\mathcal{T}$ in the sense that for every derivation of a sentential form w in $\mathcal{T}$ there exists a derivation in G that produces w. In fact, with G being of degree 1, any mode $f \in \{\, \leq k \mid k \geq 1 \,\} \cup \{=1, \geq 1, *, t\}$ will do and $L^f(G) = L(\mathcal{T})$. To simulate $\mathcal{T}$ in one of the modes $\{\, =k, \geq k \mid k \geq 2 \,\}$ dummy productions can be added to G.

From $\boldsymbol{CD_{REG}}$ to $\boldsymbol{TA_{REG}}$. Now we let $G = (N, T, S, P_1, \ldots, P_n)$ be a CDGS of degree n with regular components P_i, for $i \in [n]$. Hence each P_i has productions of the form $A \to zB$ and $A \to z$, with $A, B \in N$ and $z \in T \cup \{\lambda\}$.

For convenience, we assume that the component grammars make use of disjoint sets of nonterminals, i. e., $\{\, A \in N \mid A \to z \in P_i \,\} \cap \{\, A \in N \mid A \to z \in P_j \,\} = \varnothing$, for all $i \neq j \in [n]$. Moreover, in our construction we will make use of a #-extended version $P_i^{\#}$ for each component P_i.

The derivations in G can then be simulated by the regular grammar team $\mathcal{T} = (Q, T, \left(\begin{smallmatrix}S\\ \vdots\\ S\end{smallmatrix}\right), P)$ over the #-extended grammars $G_i = (N, T, S, P_i^{\#})$, for $i \in [n]$, where

$$Q = \{\left(\begin{smallmatrix}A\\ \vdots\\ A\end{smallmatrix}\right) \mid A \in N\} \cup \{\left(\begin{smallmatrix}\#\\ \vdots\\ \#\end{smallmatrix}\right)\} \quad \text{and} \quad P = \{\left(\begin{smallmatrix}A\\ \vdots\\ A\end{smallmatrix}\right) \to z\left(\begin{smallmatrix}B\\ \vdots\\ B\end{smallmatrix}\right) \mid A \to zB \in P_i^{\#},\ i \in [n]\}.$$

It is not difficult to see how $\mathcal{T}$ simulates derivations in G in mode $f \in \{\, \leq k \mid k \geq 1 \,\} \cup \{=1, \geq 1, *, t\}$. To simulate G in one of the remaining modes $\{\, =k, \geq k \mid k \geq 2 \,\}$ we add dummy productions for ‘counting’.

5. Teams of Context-Free Grammars

In this section we turn to the issue of synchronising context-free grammars. In this case, it is less straightforward to define the synchronised introduction of an (the same) occurrence of a nonterminal. This is due to the fact that a sentential form in general has several occurrences of nonterminals and moreover, in the right-hand side of productions several terminal symbols may occur in various places.

We therefore resort to context-free grammars in GNF and to leftmost derivations. The nonterminals in a context-free grammar in GNF, when rewritten, introduce a single terminal which moreover is the leftmost symbol of the right-hand side of the production used, thus making it the natural candidate to synchronise upon. This in combination with lefmost rewriting resembles the rewriting in regular grammars as well as the recognising process by an automaton (with a stack or pushdown for the remaining nonterminals). In case of GNF, the only terminal in a production is moreover followed by no more than two nonterminals, which significantly reduces the number of possible combinations of productions in synchronisations.

With leftmost derivations we rewrite sentential forms from left to right, symbol per symbol, as in the regular case. In addition we use λ-transpositions to 'push' nonterminals to the left. Suppose, e. g., that we want to introduce a terminal z in a sentential form $\binom{A}{D}$ by synchronising the productions $A \to zBC$ and $D \to zE$. Then what should the new sentential form look like? Intuitively, we would expect to derive $z\binom{BC}{E}$ with $\binom{BC}{E}$ partitioned as $\binom{B}{E}\binom{C}{\lambda}$, $\binom{B}{\lambda}\binom{C}{E}$, or even $\binom{B}{\lambda}\binom{C}{\lambda}\binom{\lambda}{E}$ in various orders. With leftmost rewriting, we favour the first option. Note, however, that the derivation could continue as $z\binom{B}{E}\binom{C}{\lambda} \Rightarrow zz\binom{F}{G}\binom{\lambda}{H}\binom{C}{\lambda} \Rightarrow zzz\binom{\lambda}{H}\binom{C}{\lambda}$ (assuming productions $B \to zF$, $E \to zGH$, $F \to z$, and $G \to z$). At this point, with C being the leftmost symbol for the first grammar, we would like to continue leftmost rewriting and synchronising with nonterminal vector $\binom{C}{H}$ rather than $\binom{\lambda}{H}$. Thus we need to move the C to the left, i. e., transpose the λ and C. This is implemented by auxiliary rewriting rules of the form $\binom{\lambda}{H}\binom{C}{\lambda} \to \binom{C}{H}$. We call such rules λ-transpositions. Thus, all but the leftmost occurrences of nonterminals in a sentential form act as a kind of stack, storing the nonterminals to be rewritten in the future.

In analogy with the regular case, we will use # as a special symbol to indicate that a grammar has finished rewriting. For context-free grammars this is slightly more involved though, since there is no unique event of a nonterminal being rewritten into a terminal string to mark the termination of the derivation. We therefore introduce a so-called '#-extended Greibach normal form', which uses # as a special (terminal) symbol to form the suffix of each derivation and uses a partition of the nonterminals in N and $N^{\#} = \{\, A^{\#} \mid A \in N \,\}$. The resulting (context-free) grammar thus generates the same language as before, but with # appended to every word.

Formally, we consider context-free grammars $(N \cup N^{\#}, T^{\#}, S^{\#}, P \cup P^{\#})$ in which all productions are of the form $A^{\#} \to zBC^{\#}$, $A \to zBC$, $A^{\#} \to zB^{\#}$, $A \to zB$, $A^{\#} \to z\#$, or $A \to z$, for $A, B, C \in N$, $A^{\#}, B^{\#}, C^{\#} \in N^{\#}$, and $z, \# \in T^{\#}$. We also allow $S^{\#} \to \#$ provided that $S^{\#}$ does not occur in the right-hand side of any production. Note that every context-free grammar G in GNF can be transformed into a $G^{\#}$ in #-extended GNF such that $L(G^{\#}) = L(G)\#$. Moreover, a production of the form $A^{\#} \to z\#$ is applied only once in each successful derivation and only to rewrite the rightmost symbol in the sentential form. Note that $G^{\#}$ would be in GNF if we were to consider $\#$ as a nonterminal symbol. This is exactly what we will do when formally defining context-free grammar teams (again similar to the regular case).

For the sequel, let $n \geq 1$, let $G_i = (N_i, T_i, S_i, P_i)$ be a context-free grammar in $\#$-extended GNF, for each $i \in [n]$, and let $\mathcal{G} = \{ G_i \mid i \in [n] \}$. Let $a \in T$, where $T = \bigcup_{i\in[n]} T_i \setminus \{\#\}$. The set of *synchronisations* on a (within G) is defined as:

$$\begin{aligned}\Delta_a(\mathcal{G}) = \{ (p, qr) \in \prod_{i\in[n]} N_i \cup \{\#\} \times \prod_{i\in[n]} N_i \cup \{\lambda, \#\} \prod_{i\in[n]} N_i \cup \{\lambda, \#\} \mid \\ \exists j \in [n] : (\mathrm{proj}_j(p) \to a\mathrm{proj}_j(q)\mathrm{proj}_j(r)) \in P_j \text{ and} \\ \forall\, i \in [n] : [(\mathrm{proj}_i(p) \to a\mathrm{proj}_i(q)\mathrm{proj}_i(r)) \in P_i \text{ and } \mathrm{proj}_i(q) \in \{\lambda, \#\} \\ \text{implies } \mathrm{proj}_i(r) = \lambda] \\ \text{or } [\mathrm{proj}_i(p) = \mathrm{proj}_i(q) \text{ and } \mathrm{proj}_i(r) = \lambda] \}.\end{aligned}$$

Hence $\Delta_a(\mathcal{G})$ contains all possible combinations of a-productions of the context-free grammars constituting $\mathcal{G}$ such that in every synchronisation on a at least one grammar applies an a-production and all other grammars either do so as well or do not rewrite.

Note how we deal with the introduction of λ's in nonterminal vectors. A production of the form $A \to zBC$ that synchronises with a production of the form $D \to zE$, with $A, B, C, D, E \in N$ and $z \in T$, will not lead to synchronised production $\binom{A}{D} \to z\binom{B}{\lambda}\binom{C}{E}$ but only to $\binom{A}{D} \to z\binom{B}{E}\binom{C}{\lambda}$ thanks to allowing $\mathrm{proj}_2(q) = \lambda$ only if $\mathrm{proj}_2(r) = \lambda$. To this we add the auxiliary rewriting rules, called λ-*transpositions*, now formally defined as:

$$\begin{aligned}\Lambda(\mathcal{G}) = \{ (pq, rs) \in \prod_{i\in[n]} N_i \cup \{\lambda, \#\} \prod_{i\in[n]} N_i \cup \{\lambda, \#\} \times \prod_{i\in[n]} N_i \cup \{\lambda, \#\} \prod_{i\in[n]} N_i \cup \{\lambda, \#\} \mid \\ \exists j \in [n] : \mathrm{proj}_j(p) = \mathrm{proj}_j(s) = \lambda, \mathrm{proj}_j(q) = \mathrm{proj}_j(r) \in N_j \cup \{\#\}, \text{and} \\ \exists i \neq j : \mathrm{proj}_i(q) \neq \lambda, \text{and} \\ \forall\, i \in [n] : \mathrm{proj}_i(p) = \mathrm{proj}_i(r) \text{ and } \mathrm{proj}_i(q) = \mathrm{proj}_i(s) \} \\ \cup \{ (pq, r) \in \prod_{i\in[n]} N_i \cup \{\lambda, \#\} \prod_{i\in[n]} N_i \cup \{\lambda, \#\} \times \prod_{i\in[n]} N_i \cup \{\lambda, \#\} \mid \\ \exists j \in [n] : \mathrm{proj}_j(p) = \mathrm{proj}_j(s) = \lambda, \mathrm{proj}_j(q) = \mathrm{proj}_j(r) \in N_j \cup \{\#\}, \text{and} \\ \forall i \neq j : \mathrm{proj}_i(q) = \lambda, \text{and} \\ \forall\, i \in [n] : \mathrm{proj}_i(p) = \mathrm{proj}_i(r) \}.\end{aligned}$$

A context-free grammar team is obtained by choosing, for each a, a subset of $\Delta_a(\mathcal{G})$.

Definition 2 (context-free grammar team) *A* context-free grammar team *over $\mathcal{G}$ is a grammar $\mathcal{T} = (Q, T, S, P \cup \Lambda(\mathcal{G}))$, in which $Q = (\prod_{i\in[n]}(N_i \cup \{\lambda, \#\})) \setminus \{\begin{pmatrix}\lambda \\ \vdots \\ \lambda\end{pmatrix}\}$, $T = \bigcup_{i\in[n]} T_i \setminus \{\#\}$, $S = (S_1, S_2, \ldots, S_n)$, and $P \subseteq \bigcup_{a\in\Sigma} \Delta_a(\mathcal{S})$.*

At each derivation step, the leftmost occurrence of a nonterminal vector is rewritten or a λ-transposition is applied. Formally, a word v directly derives a word w in $\mathcal{T}$,

$v \Rightarrow_{\mathcal{T}} w$, iff either $v = xpy$, $w = xzqry$ and $p \to zqr \in P$ for $x \in \Delta^*$ and $y \in (N \cup T)^*$, or $v = xpqy$ and either $w = xrsy$ and $pq \to rs \in \Lambda(\mathcal{G})$ or $w = xry$ and $pq \to r \in \Lambda(\mathcal{G})$ for $x, y \in (N \cup T)^*$. The language generated by $\mathcal{T}$ is $L(\mathcal{T}) = \{\, w \in T^* \mid S \Rightarrow_{\mathcal{T}}^* w\left(\begin{smallmatrix}\#\\ \vdots\\ \#\end{smallmatrix}\right) \,\}$.

Exactly as for the regular case, $\left(\begin{smallmatrix}\#\\ \vdots\\ \#\end{smallmatrix}\right)$ indicates termination of all grammars. The family of languages generated by teams of context-free grammars is denoted by TA_{CF}.

The fixed *synchronisation strategies* for teams of grammars defined for the regular case are extended in the obvious way for the case of context-free grammars, thus resulting in $\mathcal{R}^{ai}$- and $\mathcal{R}^{si}$-teams of context-free grammars:

$$\begin{aligned}
\mathcal{R}^{ai}_a(\mathcal{G}) = \{\, (p, qr) \in \Delta_a(\mathcal{G}) \mid \forall i \in [n] : {}& \text{if } P_i \text{ is } a\text{-competent}, \\
& \text{then } \mathrm{proj}_i(p) \to a\mathrm{proj}_i(q)\mathrm{proj}_i(r) \in P \,\}, \\
\mathcal{R}^{si}_a(\mathcal{G}) = \{\, (p, qr) \in \Delta_a(\mathcal{G}) \mid \forall i \in [n] : {}& \text{if } P_i \text{ is } a\text{-competent on } \mathrm{proj}_i(p), \\
& \text{then } \mathrm{proj}_i(p) \to a\mathrm{proj}_i(q)\mathrm{proj}_i(r) \in P \,\}.
\end{aligned}$$

Let $\mathcal{R} = \{\, \mathcal{R}_a(\mathcal{G}) \mid a \in \Sigma \,\}$. $\mathcal{T}$ is the $\mathcal{R}$-*context-free grammar team* over $\mathcal{G}$ if $P = \mathcal{R}$.

Example 5 Let $\mathcal{G} = \{G_1, G_2\}$, where $G_1 = (\{S, B\} \cup \{S^\#, B^\#\}, \{a, b, \#\}, S^\#, P_1)$ and $G_2 = (\{S, C\} \cup \{S^\#, C^\#\}, \{b, c, \#\}, S^\#, P_2)$ are context-free grammars in #-extended GNF, with

$$\begin{aligned}
P_1 &= \{S^\# \to aSB^\#, S \to aSB, S^\# \to aB^\#, S \to aB, B^\# \to b\#, B \to b\} \text{ and} \\
P_2 &= \{S^\# \to bSC^\#,\ S \to bSC,\ S^\# \to bC^\#, S \to bC,\ C^\# \to c\#,\ C \to c\}.
\end{aligned}$$

It is not difficult to see that $L(G_1) = \{\, a^n b^n \# \mid n \geq 1 \,\}$ and $L(G_2) = \{\, b^n c^n \# \mid n \geq 1 \,\}$. The $\mathcal{R}^{ai}$-context-free grammar team over $\mathcal{G}$ is

$$\mathcal{T}^{ai} = ((N_1^\# \times N_2^\#) \setminus \{\tbinom{\lambda}{\lambda}\}, \{a, b, c, \#\}, \tbinom{S^\#}{S^\#},\ P^{ai} \cup \Lambda(\mathcal{G})\,),$$

where

$$\begin{aligned}
N_1^\# &= \{S^\#, S, B^\#, B, \#, \lambda\}, \\
N_2^\# &= \{S^\#, S, C^\#, C, \#, \lambda\}, \\
P^{ai} &= \{\tbinom{S^\#}{S^\#} \to a\tbinom{S}{S^\#}\tbinom{B^\#}{\lambda}, \tbinom{S^\#}{S^\#} \to a\tbinom{B^\#}{S^\#}, \tbinom{S}{S^\#} \to a\tbinom{S}{S^\#}\tbinom{B}{\lambda}, \tbinom{S}{S^\#} \to a\tbinom{B}{S^\#}, \\
&\quad \ldots, \tbinom{B}{S^\#} \to b\tbinom{\lambda}{S}\tbinom{\lambda}{C^\#}, \tbinom{B}{S^\#} \to b\tbinom{\lambda}{C^\#}, \tbinom{B}{S} \to b\tbinom{\lambda}{S}\tbinom{\lambda}{C}, \tbinom{B}{S} \to b\tbinom{\lambda}{C}, \ldots, \\
&\quad \tbinom{B}{C^\#} \to c\tbinom{B}{\#}, \tbinom{B}{C} \to c\tbinom{B}{\lambda}, \ldots, \tbinom{\#}{C^\#} \to c\tbinom{\#}{\#}, \tbinom{\#}{C} \to c\tbinom{\#}{\lambda}\}, \text{ and} \\
\Lambda(\mathcal{G}) &= \{\tbinom{X}{\lambda}\tbinom{Y}{Z} \to \tbinom{X}{Z}\tbinom{Y}{\lambda},\ \tbinom{X}{\lambda}\tbinom{\lambda}{Z} \to \tbinom{X}{Z} \mid X \in N_1^\# \setminus \{\lambda\},\ Y, Z \in N_2^\# \setminus \{\lambda\}\} \\
&\quad \cup \{\tbinom{\lambda}{X}\tbinom{Y}{Z} \to \tbinom{Y}{X}\tbinom{\lambda}{Z},\ \tbinom{\lambda}{X}\tbinom{Y}{\lambda} \to \tbinom{Y}{X} \mid X \in N_1^\# \setminus \{\lambda\},\ Y, Z \in N_2^\# \setminus \{\lambda\}\}.
\end{aligned}$$

Hence a possible (leftmost) derivation of $\mathcal{T}^{ai}$ is:

$$\begin{aligned}
\binom{S^\#}{S^\#} &\Rightarrow^{n-1} a^{n-1}\binom{S}{S^\#}\binom{B}{\lambda}^{n-2}\binom{B^\#}{\lambda} \Rightarrow a^n\binom{B}{S^\#}\binom{B}{\lambda}^{n-2}\binom{B^\#}{\lambda} \\
&\Rightarrow a^n b\binom{\lambda}{S}\binom{\lambda}{C^\#}\binom{B}{\lambda}^{n-2}\binom{B^\#}{\lambda} \Rightarrow a^n b\binom{\lambda}{S}\binom{B}{C^\#}\binom{B}{\lambda}^{n-1}\binom{B^\#}{\lambda} \\
&\Rightarrow a^n b\binom{B}{S}\binom{\lambda}{C^\#}\binom{B}{\lambda}^{n-1}\binom{B^\#}{\lambda} \Rightarrow a^n b^2\binom{\lambda}{S}\binom{\lambda}{C}\binom{\lambda}{C^\#}\binom{B}{\lambda}^{n-1}\binom{B^\#}{\lambda} \\
&\Rightarrow^3 a^n b^2\binom{B}{S}\binom{\lambda}{C}\binom{\lambda}{C^\#}\binom{B}{\lambda}^{n-2}\binom{B^\#}{\lambda} \Rightarrow^+ a^n b^{n-2}\binom{\lambda}{S}\binom{\lambda}{C}^{n-3}\binom{\lambda}{C^\#}\binom{B}{\lambda}\binom{B^\#}{\lambda} \\
&\Rightarrow^{n-1} a^n b^{n-2}\binom{B}{S}\binom{\lambda}{C}^{n-3}\binom{\lambda}{C^\#}\binom{B^\#}{\lambda} \Rightarrow a^n b^{n-1}\binom{\lambda}{S}\binom{\lambda}{C}^{n-2}\binom{\lambda}{C^\#}\binom{B^\#}{\lambda} \\
&\Rightarrow^n a^n b^{n-1}\binom{B^\#}{S}\binom{\lambda}{C}^{n-2}\binom{\lambda}{C^\#} \Rightarrow a^n b^n\binom{\#}{C}\binom{\lambda}{C}^{n-2}\binom{\lambda}{C^\#} \\
&\Rightarrow a^n b^n c\binom{\#}{\lambda}\binom{\lambda}{C}^{n-2}\binom{\lambda}{C^\#} \Rightarrow a^n b^n c\binom{\#}{\lambda}\binom{\lambda}{C}^{n-2}\binom{\lambda}{C^\#} \\
&\Rightarrow a^n b^n c\binom{\#}{C}\binom{\lambda}{C}^{n-3}\binom{\lambda}{C^\#} \Rightarrow^{n-3} a^n b^n c^{n-2}\binom{\#}{\lambda}\binom{\lambda}{C}\binom{\lambda}{C^\#} \\
&\Rightarrow \; a^n b^n c^{n-2}\binom{\#}{C}\binom{\lambda}{C^\#} \Rightarrow a^n b^n c^{n-1}\binom{\#}{\lambda}\binom{\lambda}{C^\#} \Rightarrow a^n b^n c^{n-1}\binom{\#}{C^\#} \\
&\Rightarrow a^n b^n c^n\binom{\#}{\#}.
\end{aligned}$$

No other type of derivation can end successfully and we have

$$L(\mathcal{T}^{ai}) = \{\, a^n b^n c^n \mid n \geq 1 \,\}.$$

Finally, it is left to the reader to verify that the $\mathcal{R}^{si}$-team over $\mathcal{G} = \{G_1, G_2\}$ does not generate $L(\mathcal{T}^{ai})$.

6. Discussion and Future Work

For over two decades now, CD grammar systems have been recognised as an interesting and challenging model for the description of coordinated rewriting processes. Investigations in this area focus to a large extent on aspects of team work in concurrent systems that can be caught in the framework of grammar systems [14, 16, 18, 22].

There are two main classes of grammar systems, viz. CD grammar systems and parallel communicating (PC) grammar systems (originally introduced in [23], see also [12, 24]). Contrary to the sequential rewriting of CD grammar systems, the grammars in a PC grammar system work simultaneously, each on its own sentential form. The operations are synchronized by means of a global clock and in each time unit the grammars either all rewrite their current sentential form or they communicate via queries, sending sentential forms from one component to another. Many PC grammar system variants exist, mainly varying the protocol for transferring sentential forms.

CD grammar systems are intrinsically sequential, but concurrent (team) work can be achieved by grouping components (called teams) to rewrite different nonterminals in the sentential form in parallel [20, 1, 2, 15]. We already proposed a framework in which grammars collaborate by synchronising productions [8]: In Petri net controlled grammar systems they do so subject to the control exercised by a Petri net describing a concurrent rewriting strategy for the team of participating grammars. The grammars however all had their own sentential form to work on.

In this paper, we have proposed a novel concept for team collaboration between grammars following the approach of the team automata model. This has led to regular or context-free grammar teams in which the grammars forming a system collaborate by synchronising the introduction of terminals in a common sentential form rather than synchronising nonterminals, rewriting in parallel, or individual sentential forms. Different synchronisation strategies can be defined, depending on the competence of the participating grammars, i. e., based on the current nonterminals that can be rewritten and on the terminal they should introduce at that point in the derivation.

Rewriting according to the competence of the components of a CD grammar system is a concept on which ter Beek and Csuhaj-Varjú have closely collaborated, starting with [4]. They have also adapted this notion to automata when introducing teams of pushdown automata [5]. The latter differ from the grammar teams introduced in this paper, if not only because they consider stacks (i. e., pushdown memory), but also because they do not synchronise but instead rewrite in parallel, similar to the way this is done in teams of CD grammar systems. Also the sequential distributed (pushdown) automata models studied by Csuhaj-Varjú et alii in [17, 21] are different. These are obtained by replacing the grammars in CD grammar systems by (pushdown) automata, after which the latter take turns rewriting the sentential form in a sequential manner according to the known modes. As a side effect of the context-free case, we also seem to be close to a concept of synchronising pushdown automata.

The study presented in this paper is no more than a first step. Many issues concerning the relation between distributed cooperation and synchronised collaboration in the context of rewriting systems deserve further investigation. For one, we only studied simulation for the case of regular grammars. In case of context-free grammars, we know that $CD_{CF}(t) = ET0L$ and we have seen in Example 5 that also context-free grammar teams can generate non-context-free languages, but the exact relation between TA_{CF} and $CD_{CF}(f)$, for any f, is an open problem. It would also be worth to study the conditions under which a hierarchy of grammar teams can be defined.

Finally, inspired by team automata, it would be interesting to distinguish input, output, and internal actions in sycnhronisations and to study whether grammar teams are compositional: Does there exists a set-theoretic (shuffle) operation that when applied to the languages of the component grammars, yields the language of the grammar team (constructed according to a specific synchronisation strategy)?

Acknowledgements

We dedicate this paper to Erzsébet Csuhaj-Varjú. We are indebted to her for introducing us many years ago to the area of CD grammar systems. The first author is grateful for her encouragement in the initial phase of his academic career. Finally, we thank the anonymous reviewers for their useful suggestions.

References

[1] M. H. ter Beek, Teams in Grammar Systems: Hybridity and Weak Rewriting. *Acta Cybernetica* **12** (1996) 4, 425–444.

[2] M. H. ter Beek, Teams in Grammar Systems: Sub-Context-Free Cases. In: *New Trends in Formal Languages*. LNCS 1218, Springer-Verlag, 1997, 197–216.

[3] M. H. ter Beek, *Team Automata: A Formal Approach to the Modeling of Collaboration Between System Components*. Ph.D. thesis, Leiden University, 2003.

[4] M. H. ter Beek, E. Csuhaj-Varjú, M. Holzer, G. Vaszil, On Competence in CD Grammar Systems. In: *Proceedings 8th International Conference on Developments in Language Theory*. LNCS 3340, Springer-Verlag, 2004, 76–88.

[5] M. H. ter Beek, E. Csuhaj-Varjú, V. Mitrana, Teams of Pushdown Automata. *International Journal of Computer Mathematics* **81** (2004) 2, 141–156.

[6] M. H. ter Beek, C. A. Ellis, J. Kleijn, G. Rozenberg, Synchronizations in Team Automata for Groupware Systems. *Computer Supported Cooperative Work* **12** (2003) 1, 21–69.

[7] M. H. ter Beek, F. Gadducci, D. Janssens, A Calculus for Team Automata. *Electronic Notes in Theoretical Computer Science* **195** (2008), 41–55.

[8] M. H. ter Beek, J. Kleijn, Petri Net Control for Grammar Systems. In: *Formal and Natural Computing: Essays Dedicated to Grzegorz Rozenberg*. LNCS 2300, Springer-Verlag, 2002, 220–243.

[9] M. H. ter Beek, J. Kleijn, Team Automata Satisfying Compositionality. In: *Proceedings 12th International Symposium of Formal Methods*. LNCS 2805, Springer-Verlag, 2003, 381–400.

[10] M. H. ter Beek, J. Kleijn, Vector Team Automata. *Theoretical Computer Science* **429** (2012), 21–29.

[11] E. Csuhaj-Varjú, J. Dassow, On Cooperating/Distributed Grammar Systems. *Journal of Information Processing and Cybernetics / EIK* **26** (1990) 1-2, 49–63.

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

[13] E. Csuhaj-Varjú, J. Kelemen, Cooperating Grammar Systems: a Syntactical Framework for the Blackboard Model of Problem Solving. In: *Proceedings 5th International Conference on Artificial Intelligence and Information-Control Systems of Robots*. Elsevier, 1989, 121–127.

[14] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, G. Păun, Eco-Grammar Systems: A Grammatical Framework for Studying Life-Like Interaction. *Artificial Life* **3** (1997) 1, 1–28.

[15] E. Csuhaj-Varjú, A. Kelemenová, Team Behaviour in Eco-Grammar Systems. *Theoretical Computer Science* **209** (1998) 1–2, 213–224.

[16] E. Csuhaj-Varjú, A. Mateescu, On a Connection Between Cooperating Distributed Grammar Systems and Basic Process Algebra. *Fundamenta Informaticae* **73** (2006) 1–2, 37–50.

[17] E. Csuhaj-Varjú, V. Mitrana, G. Vaszil, Distributed Pushdown Automata Systems: Computational Power. In: *Proceedings 7th International Conference on Developments in Language Theory*. LNCS 2710, Springer-Verlag, 2003, 218–229.

[18] E. Csuhaj-Varjú, G. Păun, G. Vaszil, Grammar Systems versus Membrane Computing: The Case of CD Grammar Systems. *Fundamenta Informaticae* **76** (2007) 3, 271–292.

[19] S. A. Greibach, A New Normal-Form Theorem for Context-Free Phrase Structure Grammars. *Journal of the ACM* **12** (1965) 1, 42–52.

[20] L. Kari, A. Mateescu, G. Păun, A. Salomaa, Teams in Cooperating Grammar Systems. *Journal of Experimental & Theoretical Artificial Intelligence* **7** (1995) 4, 347–359.

[21] K. Krithivasan, M. S. Balan, P. Harsha, Distributed Processing in Automata. *International Journal of Foundations of Computer Science* **10** (1999) 4, 443–464.

[22] K. A. Lázár, E. Csuhaj-Varjú, A. Lörincz, Peer-to-Peer Networks: A Language Theoretic Approach. *Computing and Informatics* **27** (2008) 3, 403–422.

[23] G. Păun, L. Sântean, Parallel Communicating Grammar Systems: The Regular Case. *Analele Universitatii din Bucureşti, Seria Matematică–Informatică* **2** (1989), 55–63.

[24] G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*. Springer-Verlag, 1997.

(Received: February 4, 2014; revised: April 23, 2014)