V. Geffert, G. Pighizzini (eds.): Proceedings of the 9th workshop
Descriptional Complexity of Formal Systems. High Tatras, Slovakia, July 20 – 22, 2007.
(Pages 141 – 152)

Remarks on Context-free Parallel Communicating Grammar Systems Generating Crossed Agreements

Bianca Truthe

Facultat de Lletres – GRLMC – Universitat Rovira i Virgili
Plaça Imperial Tàrraco 1 – E-43005 Tarragona – Spain
bianca@truthe.de

Abstract. When modeling natural languages, three phenomena should be considered: the so-called multiple agreements, crossed agreements and replication. These aspects are represented by the three languages $K_1 = \{ a^n b^n c^n \mid n \geq 1 \}$, $K_2 = \{ a^n b^m c^n d^m \mid m \geq 1,\ n \geq 1 \}$, and $K_3 = \{ ww \mid w \in \{ a, b \}^+ \}$, respectively.

In the present paper, we give parallel communicating grammar systems (PC grammar systems) that are context-free and that generate the non-context-free language K_2 but have less components than those systems published so far. In two cases, the results are optimal. This paper follows [5], where linear and right-linear systems have been considered for generating the language K_2, besides systems for the languages K_1 and K_3.

Keywords: parallel communicating grammar systems, number of components, crossed agreements

1 Introduction and Definitions

A parallel communicating grammar system (PC grammar system) consists of several grammars. For solving a task, the components (grammars) work simultaneously and are allowed to communicate. According to the monograph [2], a communication is done by request: a component can request the whole word generated by another component. A minimal synchronization is assumed: In each time unit every component carries out a rewriting step or the system performs a communication.

If no component wants to communicate, then each grammar derives its current sentential form by a rewriting step according to its production rules (if a grammar has reached a terminal word, it keeps it; if the sentential form of a grammar contains only nonterminals for which the grammar has no rules, then the component blocks the whole system). If a component wants to get the sentential form of another component, then it introduces a query symbol that indicates to which component the query is sent. When the query is satisfied, the query symbol is rewritten by the sentential form obtained. In one rewriting step, a grammar may

The research was supported by the Alexander von Humboldt Foundation of the Federal Republic of Germany.

introduce several query symbols. All queries of a component are satisfied simultaneously and only, if all the requested sentential forms do not contain query symbols, otherwise the sentential form is not changed in this step. If a component which has been asked has sent a query to another component, then it first waits for an answer and, after receiving, it sends its own answer.

The grammars do not perform further rewriting steps before all sentential forms are query free; especially those components that are not involved in a communication do nothing in this moment. It can happen that two components issue a query to each other at the same time. Then none of the queries is satisfied and the components wait for each other – this causes a deadlock of the system.

In the returning mode, each component restarts with its start symbol after satisfying a query; in the non-returning mode the components continue with their current words.

We give the formal definitions now. For more details, we refer to [2]. By $\#(A)$ and $|w|_A$ we quote the cardinality of the set A and the number of occurrences of a letter of the alphabet A in the word w, respectively.

Let $n \geq 1$ be a natural number. A PC grammar system of degree n is an $(n+3)$-tuple

$$\Gamma^{(n)} = (N, T, K, (P_1, S_1), (P_2, S_2), \ldots, (P_n, S_n))$$

where

- N is a finite set (the set of nonterminals),
- T is a finite set (the set of terminals),
- $K = \{ Q_1, \ldots, Q_n \}$ is a set of n query symbols (with the sets N, T, K being pairwise disjoint) and
- (P_i, S_i) are the components of the system, where $S_i \in N$ and

  $$G_i = (N \cup K, T, P_i, S_i)$$

  is a Chomsky grammar $(i = 1, \ldots, n)$.

A PC grammar system is called centralized, if only one component (the so-called master) issues query symbols, and non-centralized otherwise. A PC grammar system is called regular, right-linear, linear, context-free etc., if the rules in the components have the respective property. In [6], it was shown that centralized PC grammar systems with right-linear components are more powerful than centralized systems with regular components. In a regular rule, exactly one terminal appears on the right hand side, whereas in right-linear rules, arbitrary many terminals are allowed. A rule $A \to B$ with $A, B \in N$ (without terminals) is called a chain rule.

Let $\Gamma^{(n)} = (N, T, K, (P_1, S_1), \ldots, (P_n, S_n))$ be a PC grammar system with n components. A configuration of the PC grammar system $\Gamma^{(n)}$ is an n-tuple

$$(w_1, \ldots, w_n) \text{ with } w_i \in (N \cup T \cup K)^*.$$

The PC grammar system $\Gamma^{(n)}$ derives a configuration $(x_1, \ldots, x_n)$ to a configuration $(y_1, \ldots, y_n)$ in the returning or non-returning mode (written formally as

$$(x_1, \ldots, x_n) \Longrightarrow (y_1, \ldots, y_n)$$

maybe with the index R for 'returning' or N for 'non-returning' appended to the symbol $\Longrightarrow$), if one of the following cases holds:

(I) No word x_i, $i = 1, \ldots, n$, contains a query symbol.
For $i = 1, \ldots, n$, either we have $x_i \Longrightarrow_{G_i} y_i$ or $x_i \in T^*$ and $y_i = x_i$.

(II) A word x_i, $i = 1, \ldots, n$, contains a query symbol.
For each index $i = 1, \ldots, n$ we have: If the word x_i contains a query symbol, then there are a number $m_i \geq 1$ (the number of query symbols), words $z_{i,1}, \ldots, z_{i,m_i+1} \in (N \cup T)^*$ and natural numbers $t_{i,1}, \ldots, t_{i,m_i} \in \{\, 1, \ldots, n \,\}$ such that the word x_i is composed as

$$x_i = z_{i,1} Q_{t_{i,1}} \cdots z_{i,m_i} Q_{t_{i,m_i}} z_{i,m_i+1}.$$

If a word $x_{t_{i,j}}$ $(j \in \{\, 1, \ldots, m_i \,\})$ contains a query symbol, then $y_i = x_i$ otherwise

$$y_i = z_{i,1} x_{t_{i,1}} z_{i,2} x_{t_{i,2}} \cdots z_{i,t_i} x_{t_{i,m_i}} z_{i,m_i+1}.$$

If the word x_i does not contain a query symbol, we have $y_i = x_i$ in the non-returning mode and there are two cases in the returning mode:

- There is a word x_k with the query symbol Q_i and such that $|x_{t_{k,j}}|_K = 0$ for all $j = 1, \ldots, m_k$. Then $y_i = S_i$ (if the i-th component is asked but does not ask itself, it returns to the start symbol after answering).
- There is no such word x_k. Then we have $y_i = x_i$ (if the i-th component is not involved in communication, the sentential form does not change).

By $\Longrightarrow^*$ we denote the reflexive and transitive closure of the relation $\Longrightarrow$.

The configuration $(S_1, \ldots, S_n)$ is called the start configuration; each configuration $(w_1, \ldots, w_n)$ with $w_1 \in T^*$ is called an end configuration of the system Γ, if it can be reached from the start configuration, i. e. if

$$(S_1, \ldots, S_n) \Longrightarrow^* (w_1, \ldots, w_n).$$

The languages $L_{\mathsf{R}}(\Gamma^{(n)})$ and $L_{\mathsf{N}}(\Gamma^{(n)})$ generated by a PC grammar system $\Gamma^{(n)}$ in the returning and non-returning mode, respectively, are the sets containing all words $w \in T^*$, for which there exist words $w_2, \ldots, w_n$ such that $(w, w_2, \ldots, w_n)$ is an end configuration of $\Gamma^{(n)}$:

$$L_x(\Gamma^{(n)}) = \{\, w \in T^* \mid (S_1, \ldots, S_n) \Longrightarrow_x^* (w, w_2, \ldots, w_n) \,\} \text{ for } x \in \{\, \mathsf{R}, \mathsf{N} \,\}.$$

The sets of regular, right-linear, linear and context-free grammars, we denote by *REG*, *RL*, *LIN*, and *CF*, respectively.

For $Y \in \{\, PC, CPC \,\}$ and $X \in \{\, REG, RL, LIN, CF \,\}$, we denote by

- PC_nX the set of all PC grammar systems (centralized or not) with at most n components of type X,

- CPC_nX the set of all centralized PC grammar systems with at most n components of type X,
- $\mathcal{L}_{\mathsf{R}}(Y_nX)$ the set of all languages that are generated by a Y_nX-system in the returning mode,
- $\mathcal{L}_{\mathsf{N}}(Y_nX)$ the set of all languages that are generated by a Y_nX-system in the non-returning mode.

By cooperation, even regular grammars are able to generate non-context-free languages (examples can be found in [2] and [4]). This is important since context-free languages are not sufficient for modeling natural languages and also artificial languages (a more detailed discussion can be read in [3]).

Three non-context-free phenomena occurring with natural languages are the so-called multiple agreements, crossed agreements and replication. They are represented by the languages

$$\begin{aligned} K_1 &= \{\, a^n b^n c^n \mid n \geq 1 \,\}, \\ K_2 &= \{\, a^n b^m c^n d^m \mid m \geq 1,\ n \geq 1 \,\} \text{ and} \\ K_3 &= \left\{\, ww \;\middle|\; w \in \{\, a, b \,\}^+ \,\right\}. \end{aligned}$$

In [5], we gave some PC grammar systems for generating these languages and discussed the tightness of the number of their components with respect to their types.

In the present paper, we give context-free PC grammar systems that generate the non-context-free language K_2 but have less components than those systems published so far. In two cases, the results are optimal. This paper follows [5], where only linear and right-linear systems have been considered for generating the language K_2 (besides further results for generating the languages K_1 and K_3). When allowing context-free rules, the number of necessary components can be further reduced.

2 PC grammar systems for the language K_2

In [5], right-linear and linear systems for generating the language K_2 are given. The present paper follows these investigations by studying context-free but non-linear PC grammar systems.

In [4], a context-free PC grammar system with three components and a context-free PC grammar system with ten components were given which generate the language K_2 the in the returning and non-returning mode, respectively. In [1], a context-free, centralized PC grammar system with four components was given that generates this language in the returning mode and another one with five components for the non-returning mode.

According to the number of components, the best PC grammar systems for generating the language K_2 with non-linear components are so far

- a CPC_5CF-system in the non-returning mode ([1]),
- a CPC_4CF-system in the returning mode ([1]) and

- a PC_3CF-system in the returning mode ([4]).

In this section, we give

- a PC_2CF-system for the returning mode,
- a CPC_3CF-system for the returning mode and
- a CPC_2CF-system for the non-returning mode

and we hereby improve the known results with respect to the number of necessary components.

We start with a PC_2CF-system working in the returning mode.

Theorem 1. *The PC_2CF-system*

$$\Gamma_1 = (\{ S_1, S_2, S_1', T, B, D, B', D' \}, \{ a, b, c, d \}, \{ Q_1, Q_2 \}, (P_1, S_1), (P_2, S_2))$$

with

$$\begin{aligned} P_1 &= \{ S_1 \to aTcD,\, T \to aTc,\, T \to B,\, B \to bB',\, D \to dD' \} \\ &\quad \cup \{ B \to b,\, D \to d,\, S_1 \to S_1',\, S_1' \to Q_2 \}, \\ P_2 &= \{ S_2 \to S_2,\, S_2 \to Q_1,\, B' \to B,\, D' \to D \}. \end{aligned}$$

generates the language K_2 in the returning mode.

Proof. The Figure 1 illustrates the system.

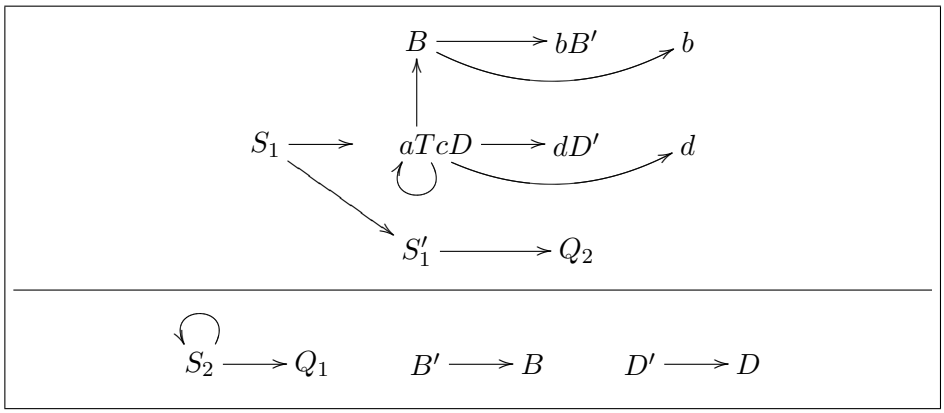

Figure 1. Illustration of the PC_2CF-system Γ_1 in the returning mode

The PC grammar system Γ_1 works in the returning mode as follows. The first component generates the as and cs and exactly a b and a d and possibly a nonterminal for more bs and ds. The second component first waits and then issues a query to the first component. If this happens too early, it receives an 'unknown' nonterminal, and, hence, can rewrite one nonterminal at most. Then the systems blocks, because the second components is not ready to issue a query. If the second component does not issue a query in the appropriate time, the first one blocks. The system also blocks if the first component contains only one nonterminal (B' or D') when being asked. This can be derived in the second component, but it blocks

after one step because there is no more rule applicable and the first component is not ready for a query. Hence, when the second component issues a query, the first component must contain exactly one B' and D'. These two nonterminals are rewritten in the second component. After this, the first component requests this word, derives both nonterminals (generates a terminal both times) and has to be asked by the second component again in time (after two steps). By this method, the bs and ds are generated in the same number of occurrences. This procedure repeats until the first component generates terminals.

We remark a specialty of this system here. All terminals are generated only in the first component. The rules of the second component all are chain rules. They serve to 'steer' the first component. By this manner, the first component is forced to produce in an ordered way. All 'wrong' derivations are omitted.

If the first component applies the rule $S_1 \to S_1'$ first, then the system will eventually block because S_1' will arrive in the second component or S_2 reaches the first component but both nonterminals are not derivable in the respective other component, or the components issue a query to each other at the same time. Hence, we first apply $S_1 \to aTcD$ to S_1. If the second component issues a query, while the sentential form of the first component contains a nonterminal T, B or D, then the second component can perform at most one rewriting step and blocks because the first component is not able to query after one step.

After applying the rule $T \to aTc$ repeatedly and rewriting T, B and D, the first component finally obtains a word of the form $a^n\beta c^n\delta$ with $n \geq 1$, $\beta \in \{ b, bB' \}$ and $\delta \in \{ d, dD' \}$. Then, we have three cases ($n \geq 1$ and $w_2 \in \{ S_2, Q_1 \}$):

(I) $(S_1, S_2) \Longrightarrow^{n+3} (a^n bc^n d, w_2)$,

(II) $(S_1, S_2) \Longrightarrow^{n+3} (a^n\beta c^n\delta, w_2)$ with $\beta = b$ and $\delta = dD'$ or $\beta = bB'$ and $\delta = d$ or

(III) $(S_1, S_2) \Longrightarrow^{n+3} (a^n bB'c^n bD', w_2)$

In the first case, the first component has derived a terminal word. If $w_2 = S_2$ in the second or third case, then the systems blocks because the first component can apply no rule any more. In the second case with $w_2 = Q_1$, the second component can perform only one rewriting step after its query has been satiesfied. Since the first component needs two steps before it can issue a query, the second component blocks the system. Hence, only the third case can occur with $w_2 = Q_1$: $(a^n bB'c^n dD', Q_1) \Longrightarrow (S_1, a^n bB'c^n dD')$. If now S_1 is not derived to S_1', the system blocks after two steps. Thus

$$(S_1, a^n bB'c^n dD') \Longrightarrow 2(Q_2, a^n bBc^n dD) \Longrightarrow (a^n bBc^n dD, S_2).$$

By the same argumentation as above, we have two possibilities for a non-blocking continuation:

(I) $(a^n bBc^n dD, S_2) \Longrightarrow 2(a^n b^2c^n d2, *)$ or

(II) $(a^n bBc^n dD, S_2) \Longrightarrow 2(a^n b^2B'c^n d^2D', Q_1)$.

The star marks that the corresponding component does not influence the system any more and hence its contents is not important. Summarizing, only the following derivations lead to terminal words:

$$(S_1, S_2) \Longrightarrow^{n+3} (a^n b c^n d, *)$$

or

$$\begin{aligned}(S_1, S_2) &\Longrightarrow^{n+3} (a^n b B' c^n d D', Q_1) && n \geq 1\\ &\Longrightarrow^{6k} (a^n b^{k+1} B' c^n d^{k+1} D', Q_1) && k \geq 0\\ &\Longrightarrow 6 \quad (a^n b^{k+2} c^n d^{k+2}, *),\end{aligned}$$

which yields together

$$(S_1, S_2) \Longrightarrow^{n+3+6k} (a^n b^{k+1} c^n d^{k+1}, *) \quad n \geq 1,\ k \geq 0.$$

Since no other terminal words are generated, the PC_2CF-system Γ_1 generates the language K_2 in the returning mode. □

One context-free component is not sufficient to generate the language K_2; hence, this result is optimal. Now we give a CPC_3CF-system for the returning mode.

Theorem 2. *The CPC_3CF-system*

$$\Gamma_2 = (N, \{ a, b, c, d \}, \{ Q_1, Q_2, Q_3 \}, (P_1, S_1), (P_2, S_2), (P_3, S_3))$$

with

$$\begin{aligned}N &= \{ S_1, S_2, S_3, S_2', S_3', Z_1, Z_2, Z_3, X, X', Y, Y', T, T', U, U', B, D, \bar{B}, \bar{D} \},\\ P_1 &= \{ S_1 \to Z_1,\ Z_1 \to Z_2,\ Z_2 \to Z_3,\ Z_3 \to Z_2,\ Z_3 \to Q_2,\ B \to Q_2,\ D \to Q_3 \}\\ &\quad \cup \{ \bar{B} \to b,\ \bar{D} \to d \},\\ P_2 &= \{ S_2 \to X,\ X \to X',\ X' \to aUc\bar{D},\ U \to \bar{B},\ U \to U',\ U' \to aUc \}\\ &\quad \cup \{ S_2 \to Y,\ Y \to Y',\ Y' \to aTcD,\ T \to B,\ T \to T',\ T' \to aTc \}\\ &\quad \cup \{ S_2 \to S_2',\ S_2' \to bB,\ S_2 \to \bar{B} \},\\ P_3 &= \{ S_3 \to X,\ S_3 \to S_3',\ X \to S_3',\ S_3' \to X,\ S_3' \to dD \}\\ &\quad \cup \{ S_3 \to Y,\ Y \to Y',\ Y' \to \bar{D} \}\end{aligned}$$

generates the language K_2 in the returning mode.

Proof. The Figure 2 illustrates the system.

We now give the formal details of the system. The first component issues a query to the second component after $4 + 2k$ steps ($k \geq 0$) for the first time. The word in the second component is then $a^{k+1}\bar{\beta}c^{k+1}\bar{D}$ with $\bar{\beta} \in \{ U', \bar{B} \}$ or $a^{k+1}\beta c^{k+1}D$ with $\beta \in \{ T', B \}$. If $\bar{\beta} = U'$ or $\beta = T'$, the nonterminal U' or T', respectively, enters the first component but cannot be derived there. Since the system is centralized, these nonterminals cannot disappear. In this situation, we do not obtain a terminal word. After $4 + 2k$ steps, the third component contains X, S_3' or dD. Since D cannot be derived, we only need to consider the other two situations.

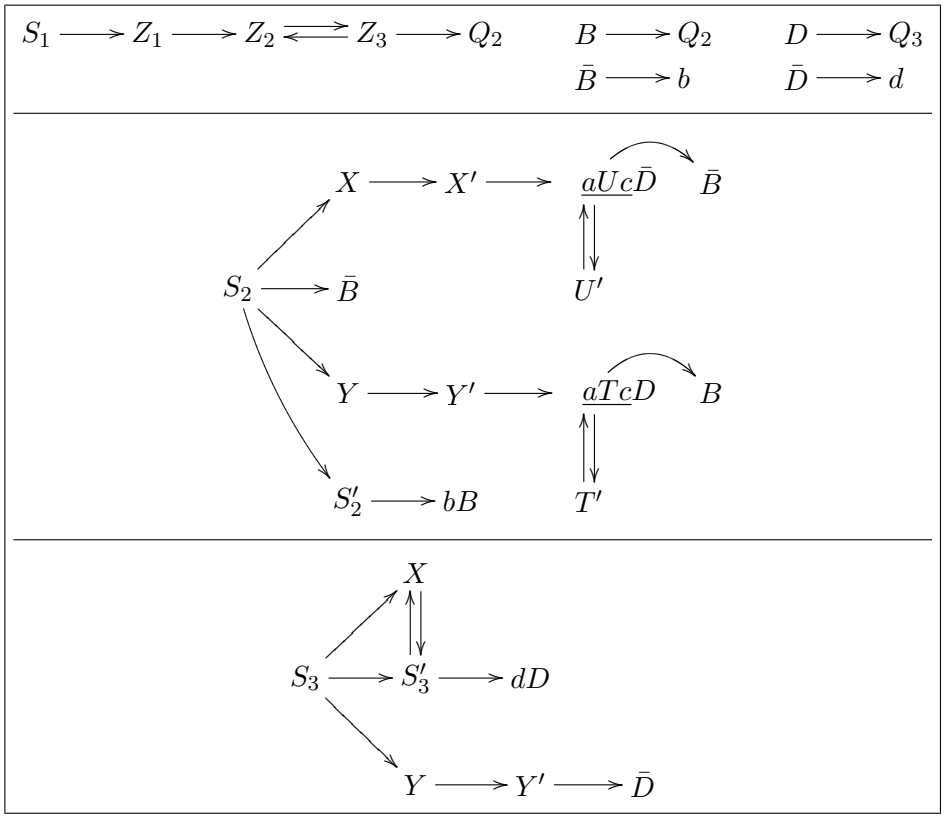

Figure 2. Illustration of the $CPC_3\,CF$-system Γ_2 in the returning mode

There are the following cases:

1. $(S_1, S_2, S_3) \Longrightarrow^{4+2k} (Q_2, a^{k+1}\bar{B}c^{k+1}\bar{D}, w_3)$ with $w_3 \in \{ X, S_3' \}$,
2. $(S_1, S_2, S_3) \Longrightarrow^{4+2k} (Q_2, a^{k+1}Bc^{k+1}D, X)$ or
3. $(S_1, S_2, S_3) \Longrightarrow^{4+2k} (Q_2, a^{k+1}Bc^{k+1}D, S_3')$.

Case 1. This case leads to $(a^{k+1}\bar{B}c^{k+1}\bar{D}, S_2, w_3) \Longrightarrow 2(a^{k+1}bc^{k+1}d, *, *)$.
Case 2. We show that the configuration $(a^{k+1}Bc^{k+1}D, S_2, X)$ does not lead to a terminal word. There are two possibilities of the derivation:

(I) $B \to Q_2$: Then $(a^{k+1}Bc^{k+1}D, S_2, X) \Longrightarrow (a^{k+1}Q_2c^{k+1}D, \bar{B}, S_3')$, because another derivation of S_2 would yield a nonterminal which is not derivable in the first component. The configuration changes to $(a^{k+1}\bar{B}c^{k+1}D, S_2, S_3')$. Only two of the possible successors do not lead immediatelly to a blocking of the system:

(A) $\Longrightarrow (a^{k+1}bc^{k+1}D, *, X) \Longrightarrow (a^{k+1}bc^{k+1}Q_3, *, S_3')$
$\Longrightarrow (a^{k+1}bc^{k+1}S_3', *, *)$.
The first component cannot derive the nonterminal S_3' and blocks.

(B) $\Longrightarrow (a^{k+1}\bar{B}c^{k+1}Q_3, *, dD) \Longrightarrow (a^{k+1}\bar{B}c^{k+1}dD, *, S_3)$.
If now D is derived to Q_3, the system blocks after the communication.

Hence, the derivation continues to $(a^{k+1}bc^{k+1}dD, *, S_3')$ and further to

$$\begin{aligned}(a^{k+1}bc^{k+1}dQ_3, *, dD) &\Longrightarrow (a^{k+1}bc^{k+1}d^2D, *, S_3)\\ &\Longrightarrow (a^{k+1}bc^{k+1}d^2Q_3, *, S_3').\end{aligned}$$

After the communication the first component blocks.

(II) $D \to Q_3$: Then $(a^{k+1}Bc^{k+1}D, S_2, X) \Longrightarrow (a^{k+1}Bc^{k+1}Q_3, w_2, S_3')$. Since the first component has no rule for S_3' it blocks after the communication, regardless the value w_2 of the second component.

The configuration $(a^{k+1}Bc^{k+1}D, S_2, X)$ does not lead to a terminal word. Hence, in this case, no terminal word is obtained.

Case 3. In this case, we obtain the configuration $(a^{k+1}Bc^{k+1}D, S_2, S_3')$. Again, there are two possibilities of the derivation:

(I) $B \to Q_2$:
Then $(a^{k+1}Bc^{k+1}D, S_2, S_3') \Longrightarrow (a^{k+1}Q_2c^{k+1}D, \bar{B}, X)$, because another derivation of S_2 yields a nonterminal which is not rewritable in the first component and another derivation of S_3' causes the blocking. This configuration leads to $(a^{k+1}\bar{B}c^{k+1}D, S_2, X)$. The only continuations that do not lead to an immediate blocking are:

(A) $\Longrightarrow (a^{k+1}bc^{k+1}D, *, S_3') \Longrightarrow (a^{k+1}bc^{k+1}Q_3, *, dD)$
$\Longrightarrow (a^{k+1}bc^{k+1}dD, *, S_3)$.
The next communication step moves a nonterminal into the first component which is not derivable.

(B) $\Longrightarrow (a^{k+1}\bar{B}c^{k+1}Q_3, S_2, S_3')$.
Also this configuration does not lead to a terminal word.

(II) $D \to Q_3$:
The application of this rule leads to $(a^{k+1}Bc^{k+1}Q_3, w_2, dD)$. In the third component, S_3' must have been rewritten by dD because the first component cannot derive the nonterminal X. This leads to the configuration $(a^{k+1}Bc^{k+1}dD, w_2, S_3)$. Now D must not be derived because otherwise the arriving nonterminal from the third component cannot be rewritten. Hence, we apply the rule $B \to Q_2$. In order to have success, $w_2 = S_2'$ must hold. Then, the configuration $(a^{k+1}Bc^{k+1}dD, S_2', S_3)$ is changed to $(a^{k+1}Q_2c^{k+1}dD, bB, w_3)$ with $w_3 \in \{ X, Y, S_3' \}$.
The configuration $(a^{k+1}bBc^{k+1}dD, S_2, X)$ does not lead to a terminal word according to Case 2.
The configuration $(a^{k+1}bBc^{k+1}dD, S_2, Y)$ leads to $(a^{k+1}bQ_2c^{k+1}dD, \bar{B}, Y')$ because deriving D or deriving S_2 by another rule cause the blocking of the system. From this configuration, we obtain $(a^{k+1}b\bar{B}c^{k+1}dD, S_2, Y')$. Since Y' will be rewritten by $\bar{D}$ in the next step and the third component blocks thereafter, we have to apply the rule $D \to Q_3$. This leads to $(a^{k+1}b\bar{B}c^{k+1}dQ_3, *, \bar{D})$ and in the next step to $(a^{k+1}b\bar{B}c^{k+1}d\bar{D}, *, S_3)$. In two steps, we obtain the word $a^{k+1}b^2c^{k+1}d2$ in the first component. Hence, $(a^{k+1}Bc^{k+1}D, S_2, S_3')$ can yield the word $a^{k+1}b^2c^{k+1}d2$.

In the case of $w_3 = S_3'$, the configuration $(a^{k+1}bBc^{k+1}dD, S_2, S_3')$ was derived from $(a^{k+1}Bc^{k+1}D, S_2, S_3')$. Thus, this yields every configuration $(a^{k+1}b^mBc^{k+1}d^mD, S_2, S_3')$ with $m \geq 1$. According to the argumentation above, such a configuration yields finally the word $a^{k+1}b^{m+2}c^{k+1}d^{m+2}$.

The configuration $(a^{k+1}Bc^{k+1}D, S_2, S_3')$ leads by alternating applications of the rules $D \to Q_3$ and $B \to Q_2$ and suitable derivations of the second and third component to the words $a^nb^mc^nd^m$ with $n \geq 1$ and $m \geq 2$. Other words are not produced.

Together with Case 1, the language generated in the returning mode by the CPC_3CF-system Γ_2 is $L_{\mathsf{R}}(\Gamma_2) = K_2$, which proves the claim. □

The 'trick' with the PC grammar system Γ_2 is that one query has success only in even numbered rewriting steps and the other query has success only in odd numbered rewriting steps. By this means, the alternating production of bs and ds is enforced.

Now we give a context-free, centralized PC grammar system with two components that generates in the non-returning mode the language K_2.

Theorem 3. *The CPC_2CF-system*

$$\Gamma_3 = (N, \{ a, b, c, d \}, \{ Q_1, Q_2 \}, (P_1, S_1), (P_2, S_2))$$

with

$$\begin{aligned}
N &= \{ S_1, S_2, S_1', S_2', B, B', D, D', \bar{T}, \bar{B}, E \}, \\
P_1 &= \{ S_1 \to aQ_2cd,\ S_1 \to aQ_2cD,\ S_1 \to S_1',\ S_1' \to S_1,\ D \to dD' \} \\
&\quad \cup \{ D' \to D,\ D' \to d,\ \bar{T} \to Q_2,\ \bar{T} \to aQ_2c,\ \bar{B} \to b \}, \\
P_2 &= \{ S_2 \to \bar{T},\ S_2 \to S_2',\ S_2' \to aS_2c,\ \bar{T} \to \bar{B},\ \bar{B} \to E,\ \bar{T} \to bB,\ B \to B' \} \\
&\quad \cup \{ B' \to bB,\ B' \to \bar{B} \}
\end{aligned}$$

generates the language K_2 in the non-returning mode.

Proof. The Figure 3 illustrates the system.

The first component issues a query to the second component in an odd numbered step. The only nonterminals that can occur in the second component and that are derivable by the first component are $\bar{T}$ and $\bar{B}$. Of these two, only $\bar{T}$ occurs in an odd numbered step. In order to suceed with the derivation, the first component has to catch the $\bar{T}$ when it occurs (since it appears only once). Consequently, we only need to consider derivations that start with the transition

$$(S_1, S_2) \Longrightarrow^{2k+1} (aQ_2c\delta, a^k\bar{T}c^k), \quad \text{for } k \geq 0,\ \delta \in \{ d, D \}.$$

In the case of $\delta = d$, we obtain

$$\begin{aligned}
(aQ_2cd, a^k\bar{T}c^k) &\Longrightarrow (a^{k+1}\bar{T}c^{k+1}d, a^k\bar{T}c^k) \\
&\Longrightarrow (a^{k+1+i}Q_2c^{k+1+i}d, a^k\bar{B}c^k) \quad 0 \leq i \leq 1 \\
&\Longrightarrow (a^{2k+1+i}\bar{B}c^{2k+1+i}d, a^k\bar{B}c^k) \\
&\Longrightarrow (a^{2k+1+i}bc^{2k+1+i}d, a^kEc^k).
\end{aligned}$$

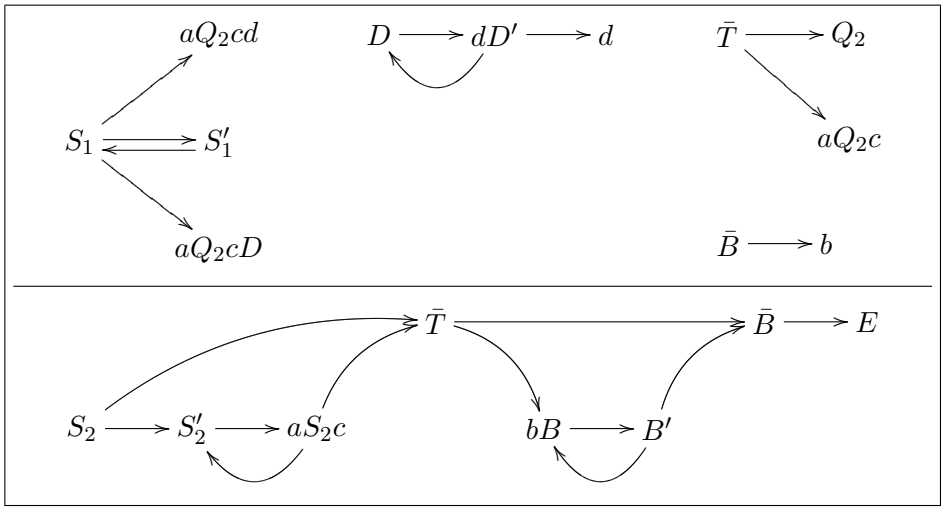

Figure 3. Illustration of the CPC_2CF-system Γ_3 in the non-returning mode

In the second component, $\bar{T}$ has to be rewritten by $\bar{B}$ because otherwise the first component would receive an underivable nonterminal. In the case of $\delta = d$, we get all words a^nbc^nd with $n \geq 1$.

Let now $\delta = D$. When the second component receives a query for the second time, it must contain $\bar{B}$ because the other possible nonterminals are not derivable by the first component. After this communication, the second component can perform only one rewriting step any more. Hence, in the first component, the just arrived $\bar{B}$ must be the only nonterminal left (all other nonterminals have to be eliminated before). Other derivations do not have to be considered. The only successful derivation is

$$\begin{aligned}
(aQ_2cD, a^k\bar{T}c^k) &\Longrightarrow (a^{k+1}\bar{T}c^{k+1}D, a^k\bar{T}c^k) \\
&\Longrightarrow (a^{k+1}\bar{T}c^{k+1}dD', a^kbBc^k) \\
&\Longrightarrow^{2l} (a^{k+1}\bar{T}c^{k+1}d^{l+1}D', a^kb^{l+1}Bc^k) && l \geq 0 \\
&\Longrightarrow (a^{k+1}\bar{T}c^{k+1}d^{l+2}, a^kb^{l+1}B'c^k) \\
&\Longrightarrow (a^{k+1+i}Q_2c^{k+1+i}d^{l+2}, a^kb^{l+1}\bar{B}c^k) && 0 \leq i \leq 1 \\
&\Longrightarrow (a^{2k+1+i}b^{l+1}\bar{B}c^{2k+1+i}d^{l+2}, a^kb^{l+1}\bar{B}c^k) \\
&\Longrightarrow (a^{2k+1+i}b^{l+2}c^{2k+1+i}d^{l+2}, a^kb^{l+1}Ec^k).
\end{aligned}$$

This derivation yields all words $a^nb^mc^nd^m$ with $n \geq 1$ and $m \geq 2$. Together with the case that $\delta = d$, we also obtain the words for $m = 1$. These are the only words that are produced. The CPC_2CF-system Γ_3 generates in the non-returning mode the language K_2. □

Also the PC grammar system Γ_3 uses the parity of the rewriting steps and controls the communication.

The Theorems 1, 2 and 3 yield the following statement.

Corollary 4. *We have* $K_2 \in \mathcal{L}_{\mathsf{R}}(PC_2CF) \cap \mathcal{L}_{\mathsf{R}}(CPC_3CF) \cap \mathcal{L}_{\mathsf{N}}(CPC_2CF)$.

Since $CPC_nCF \subseteq PC_nCF$ for any natural number n, also $K_2 \in \mathcal{L}_{\mathsf{N}}(PC_2CF)$ follows. It is not known, whether there is a CPC_2CF system which generates the language K_2 in the returning mode. However, as shown in [4], a special case of crossed agreements that is represented by the language $\{a^nb^ma^nb^m \mid m \geq 1, n \geq 1\}$ can be generated by an CPC_2CF system in the returning mode.

3 Summary

The present paper gives three context-free parallel communicating grammar systems that generate the non-context-free language $K_2 = \{\, a^nb^mc^nd^m \mid m, n \geq 1 \,\}$ but need less components than systems published so far. For the non-returning mode, the number of components was reduced from five to two; for the returning mode, the number of components was reduced from three to two. In these two cases, the result is tight. For the returning mode, the number of components in centralized PC grammar systems was reduced from four to three.

Acknowledgements

The author thanks Jürgen Dassow and György Vaszil for the inspiration and helpful discussions.

References

[1] A. Chiţu, PC Grammar Systems Versus Some Non-Context-Free Constructions from Natural and Artificial Languages. In: G. Paun, A. Salomaa (eds.), *New Trends in Formal Languages*. Lecture Notes in Computer Science 1218, Springer, 1997, 278–287.

[2] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, G. Păun, *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Topics in Computer Science 5, Gordon and Breach Science Publishers, 1994.

[3] J. Dassow, G. Păun, *Regulated Rewriting in Formal Language Theory*. EATCS 18, Springer, 1989.

[4] J. Dassow, G. Păun, G. Rozenberg, Grammar systems. In: G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*. Springer, 1997, 155–213.

[5] J. Dassow, B. Truthe, On the degree complexity of special non-context-free languages with respect to PC grammar systems. In: H. Leung, G. Pighizzini (eds.), *8th International Workshop on Descriptional Complexity of Formal Systems, DCFS 2006, Las Cruces, NM, USA, June 21–23, 2006, Proceedings*. Computer Science Technical Report NMSU-CS-2006-001, New Mexico State University Las Cruces, 2006, 241–249.

[6] S. Dumitrescu, G. Păun, On the Power of Parallel Communicating Grammar Systems with Right-Linear Components. *RAIRO Informatique Théorique et Applications/Theoretical Informatics and Applications* **31** (1997) 4, 331–354.