ELSEVIER
Theoretical Computer Science 255 (2001) 511–538
Theoretical Computer Science
www.elsevier.com/locate/tcs

On context-free parallel communicating grammar systems: synchronization, communication, and normal forms [☆]

Erzsébet Csuhaj-Varjú [*], György Vaszil

Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende utca 13-17, 1111 Budapest, Hungary

Received April 1998; revised May 1999
Communicated by A. Salomaa

Abstract

In this paper we study the generative power of context-free returning parallel communicating grammar systems using different synchronization mechanisms and communication protocols. We demonstrate the equivalence of several types of these systems and present normal form theorems showing that all languages generated by context-free returning parallel communicating grammar systems can also be generated by such systems having only rules of the form $X \rightarrow \alpha$, where α consists of at most two symbols and if $X \rightarrow \alpha$ is a query rule, then α is a single query symbol. © 2001 Elsevier Science B.V. All rights reserved.

1. Introduction

Parallel communicating grammar systems (PC grammar systems in short) have been introduced in [7] for modelling parallel and distributed computation in terms of formal grammars and languages. In these systems several grammars derive their own sentential forms in parallel and their work is organized in a communicating system to generate a single language. The parallel communicating frame has the following basic properties: the work of the components is synchronized by a universal clock, each component executes one rewriting step in each time unit, and communication is done by request through special non-terminals called query symbols, one different symbol referring to each component of the system. When a component introduces a query symbol in its sentential form, the rewriting process stops and one or more communication steps are performed by replacing all occurrences of the query symbols with the current sentential

[☆] Research supported by Hungarian Scientific Research Fund "OTKA" Grant no. T 017105 and INCO-COPERNICUS ALTEC-KIT.
[*] Corresponding author.
E-mail addresses: csuhaj@sztaki.hu (E. Csuhaj-Varjú), vaszil@sztaki.hu (Gy. Vaszil).

forms of the queried component grammars. When no more query symbol is present in any of the sentential forms, the rewriting process starts again. In the so-called returning systems after communicating its current sentential form the component returns to its start symbol and begins to generate a new string. In non-returning systems the components continue the rewriting of their current sentential forms. Rewriting steps and communication steps determine a computation. The language defined by the system is the set of terminal words which appear as sentential forms of a dedicated component grammar, the master, at some step of a computation starting from the initial configuration of the system. The PC grammar system is in initial configuration if the current sentential form of each component grammar is the start symbol of this grammar.

Parallel communicating grammar systems have been the subject of detailed study over the last years: see [1, 3, 6] for a summary of results and open problems. The investigations mainly concentrated on the generative power of PC grammar systems, and on studying how this power is influenced by changes in the basic characteristics of these systems. Recently, it has been shown in [2, 4] that contex-free PC grammar systems form a computationally complete class of generative devices, they determine the class of recursively enumerable languages.

Three of the most important features of PC grammar sytems are synchronization, communication, and the form of the components. Basic synchronization of the components is done by the universal clock, but the study of additional synchronizational mechanisms is also of interest. Additional synchronization can be realized through prescribing actions to be performed simultaneously by the component grammars, see [5] for several variants. One of the most natural variants is rule-synchronization, where the set of executable transitions (rewriting rules applied by the component grammars in the same rewriting step) is restricted.

Another determinant feature of parallel communicating grammar systems is the communication protocol. According to the original definition only complete information is sent, strings containing queries cannot be communicated, and all information requested by a component must be communicated to this component in one communication step.

Another reasonable communication protocol, called immediate communication, was proposed and examined in [12]. Here all the requested sentential forms which do not contain query symbols are communicated as soon as possible, thus, at each communication step all the available complete information is transmitted.

While in both of the above cases only complete information is sent (sentential forms not containing any query symbol), in distributed systems incomplete information can also be communicated. This is why we propose and consider a third communication protocol, where sentential forms both with and without query symbols (except strings containing a self-query) can be communicated, the only restriction is that each grammar is allowed to send its sentential form only once during the sequence of subsequent communication steps.

The way of synchronization and the way of communication are expected to have impact on the power of PC grammar systems. It is also an important question whether or not the form of the components of the PC grammar system influences the generative power, that is, whether or not PC grammar systems have equivalent systems with components in a fixed simple form, in normal form.

In this paper we examine parallel communicating grammar systems from these aspects, and study the relation of the above-mentioned determinant features. Moreover, we develop techniques which prove to be useful in constructing equivalents for PC grammar systems.

We first show how to construct for any rule-synchronized context-free PC grammar system an equivalent generic context-free parallel communicating grammar system (without rule-synchronization). The idea of the construction is to simulate each transition of the rule-synchronized PC grammar system with some dedicated components which reproduce the effect of the transition and only that. The equal generative power of the rule-synchronized and the generic context-free returning parallel communicating grammar systems is a direct consequence of this result. We note that the statement concerning the equal power of the two variants follows from [5, 2, 4], but here we effectively construct the simulating system.

By applying this decomposition technique, for each context-free returning PC grammar system we can construct an equivalent one where the effect of the complicated communication sequences of the original grammar system can be simulated by sequences of very simple communication steps in which each grammar has at most one occurrence of query symbols in its sentential form. By using these constructions, we can prove the existence of a Chomsky-like normal form for context-free returning PC grammar systems. Namely, we can show that for each PC grammar system of this type we can construct an equivalent one where each component has only rules with right-hand sides either being a query symbol or a string which consists of at most two non-query (nonterminal and/or terminal) symbols. The statement is valid for all the three communication protocols mentioned above. As a consequence of the results, we obtain that context-free returning PC grammar systems using any of these communication protocols determine the same class of languages.

2. Preliminaries

The reader is assumed to be familiar with the basics of formal language theory; further details can be found in [8].

The set of all words over an alphabet V and the empty word are denoted by V^* and ε. The family of regular, linear, and context-free grammars and their language classes are denoted by REG, LIN, CF, $\mathscr{L}(REG)$, $\mathscr{L}(LIN)$, and $\mathscr{L}(CF)$, respectively. $|X|$ denotes the number of elements of a finite set X, $|w|$ and $|w|_X$, $w \in V^*$, $X \subseteq V$, denote the length of a word w and the number of occurrences of symbols from set X in w, respectively.

We recall the notion of a parallel communicating grammar system from [7], for more information see [1, 3].

Definition 2.1. A *parallel communicating grammar system* with n components (a PC grammar system in short) is an $(n+3)$-tuple $\Gamma=(N,K,T,G_1,\dots,G_n)$, where N is a *nonterminal alphabet*, T is a *terminal alphabet*, and $K=\{Q_1,Q_2,\dots,Q_n\}$ is an alphabet of *query symbols*. N,T, and K are pairwise disjoint sets. $G_i=(N\cup K,T,P_i,S_i)$, $1\leqslant i\leqslant n$, called a *component* of Γ, is a usual Chomsky grammar with nonterminal alphabet $N\cup K$, terminal alphabet T, set of rewriting rules P_i, and *axiom* (or start symbol) S_i. G_1 is said to be the *master grammar* (or master) of Γ.

Definition 2.2. Let $\Gamma=(N,K,T,G_1,\dots,G_n)$ with $G_i=(N\cup K,T,P_i,S_i)$, $1\leqslant i\leqslant n$, be a parallel communicating grammar system. An n-tuple $(x_1,\dots,x_n)$, where $x_i\in(N\cup T\cup K)^*$, $1\leqslant i\leqslant n$, is called a *configuration* of Γ. $(S_1,\dots,S_n)$ is said to be the *initial configuration.*

PC grammar systems change their configurations by performing direct derivation steps.

Definition 2.3. Let $\Gamma=(N,K,T,G_1,\dots,G_n), n\geqslant 1$, be a parallel communicating grammar system and let $(x_1,\dots,x_n)$ and $(y_1,\dots,y_n)$ be two configurations of Γ.

We say that $(x_1,\dots,x_n)$ *directly derives* $(y_1,\dots,y_n)$, denoted by $(x_1,\dots,x_n)\Rightarrow(y_1,\dots,y_n)$, if one of the following two cases holds:

1. There is no x_i which contains any query symbol, that is, $x_i\in(N\cup T)^*$ for $1\leqslant i\leqslant n$. Then for each i, $1\leqslant i\leqslant n$, $x_i\Rightarrow_{G_i} y_i$ (y_i is obtained from x_i by a direct derivation step in G_i) for $x_i\notin T^*$ and $x_i=y_i$ for $x_i\in T^*$.

2. There is some $x_i, 1\leqslant i\leqslant n$, which contains at least one occurrence of a query symbol.

We distinguish three cases:

(a) In systems with communication protocol (a) configuration $(y_1,\dots,y_n)$ is obtained from configuration $(x_1,\dots,x_n)$ as follows:

For each $x_i, 1\leqslant i\leqslant n$, with $|x_i|_K\neq 0$ we write $x_i=z_1Q_{i_1}z_2Q_{i_2}\dots z_tQ_{i_t}z_{t+1}$, where $z_j\in(N\cup T)^*, 1\leqslant j\leqslant t+1$, and $Q_{i_l}\in K, 1\leqslant l\leqslant t$. If $|x_{i_l}|_K=0$ for each $l, 1\leqslant l\leqslant t$, then $y_i=z_1x_{i_1}z_2x_{i_2}\dots z_tx_{i_t}z_{t+1}$ and in *returning* systems $y_{i_l}=S_{i_l}$, in *non-returning* systems $y_{i_l}=x_{i_l}$, $1\leqslant l\leqslant t$. If $|x_{i_l}|_K\neq 0$ for some $l, 1\leqslant l\leqslant t$, then $y_i=x_i$. For all $j, 1\leqslant j\leqslant n$, for which y_j is not specified above, $y_j=x_j$.

(b) In systems with communication protocol (b) we obtain $(y_1,\dots,y_n)$ from $(x_1,\dots,x_n)$ as follows:

For each $x_i, 1\leqslant i\leqslant n$, with $|x_i|_K\neq 0$ we write $x_i=z_1Q_{i_1}z_2Q_{i_2}\dots z_tQ_{i_t}z_{t+1}$, where $z_j\in(N\cup T)^*, 1\leqslant j\leqslant t+1$, and $Q_{i_l}\in K, 1\leqslant l\leqslant t$. Then $y_i=z_1u_{i_1}z_2u_{i_2}\dots z_tu_{i_t}z_{t+1}$, where $u_{i_l}=x_{i_l}$ if $|x_{i_l}|_K=0$ and $u_{i_l}=Q_{i_l}$ if $|x_{i_l}|_K\neq 0, 1\leqslant l\leqslant t$. If $u_{i_l}=x_{i_l}$, then in *returning* systems $y_{i_l}=S_{i_l}$, in *non-returning* systems $y_{i_l}=x_{i_l}, 1\leqslant l\leqslant t$. For all $j, 1\leqslant j\leqslant n$, for which y_j is not specified above, $y_j=x_j$.

(c) In systems with communication protocol (c) configuration $(y_1,\dots,y_n)$ is obtained from configuration $(x_1,\dots,x_n)$ as follows:

For a fixed Q_j, $1 \leqslant j \leqslant n$, for which $|x_k|_{\{Q_j\}} \neq 0$ for some k, $1 \leqslant k \leqslant n$, and $|x_j|_{\{Q_j\}} = 0$ we write $x_i = z_1 Q_j z_2 Q_j \ldots z_t Q_j z_{t+1}$, $z_l \in (N \cup T \cup (K \setminus \{Q_j\}))^*, 1 \leqslant l \leqslant t+1$, for each x_i, where $|x_i|_{\{Q_j\}} \neq 0, 1 \leqslant i \leqslant n$. Then $y_i = z_1 x_j z_2 x_j \ldots z_t x_j z_{t+1}$ and in *returning* systems $y_j = S_j$, in *non-returning* systems $y_j = x_j$. For all $i, 1 \leqslant i \leqslant n$, for which y_i is not specified above, $y_i = x_i$.

Let $\Rightarrow^*$ denote the reflexive and transitive closure of $\Rightarrow$.

The first case is the description of a rewriting step. If no query symbol is present in any of the sentential forms, then each component grammar uses one of its rewriting rules except those which have already produced a terminal string. The derivation is blocked if a sentential form is not a terminal string but no rule can be applied to it.

The second case describes communication: if a query symbol Q_j appears in a sentential form x_i, $1 \leqslant i, j \leqslant n$, then the rewriting stops and communication must be performed.

In systems with communication protocol (a) (the communication protocol for PC grammar systems introduced in [7]), all the query symbols $Q_{i_j}, 1 \leqslant j \leqslant t$ which appear in a sentential form x_i must be replaced by the current sentential form x_{i_j} of component G_{i_j} in the same communication step provided that no x_{i_j} has any occurrence of a query symbol. If one of these sentential forms, x_{i_j}, contains a query symbol, then x_{i_j} must be made free from the queries before changing anything in x_i. The derivation gets blocked if in the obtained configuration none of the sentential forms with an occurrence of a query symbol can be made free from the queries in the above manner. This is the case when the sentential forms define only circular queries. (We consider the self-query to be a circular query.) Note that a rewriting step can result in a blocking configuration, but if a rewriting step introduces a circular query it might be several communication steps performed till the blocking configuration is reached.

In systems with communication protocol (b) (systems with immediate communications introduced in [12]), the occurrences of a query symbol Q_{i_j} in a sentential form x_i are replaced with the requested query symbol free string, x_{i_j}, independently from that whether the other query symbols in x_i refer to a query symbol free sentential form or not. If a requested string, x_{i_j}, does not contain any query symbol, then every occurrence of Q_{i_j} in each sentential form is replaced by x_{i_j} in the same communication step. Observe that different query symbols occurring in the same sentential form can be satisfied in different communication steps. The derivation, as above, gets blocked if the condition of the replacement of the sentential forms cannot be satisfied.

In systems with communication protocol (c), a query symbol Q_{i_j} in a sentential form x_i can be replaced with the requested string x_{i_j} independently from the other query symbols in x_i, even if the requested string, x_{i_j}, contains further queries. The only restrictions are that no replacement of a self-query is allowed and each occurrence of a certain query symbol Q_j in all sentential forms must be replaced by the corresponding sentential form in the same communication step. Again, the derivation gets blocked if no more replacement can be performed, but there is a sentential form with an occurrence of a query symbol.

After communicating its sentential form to another component, the grammar can continue its own work in two ways: In *returning* systems the component must return to its axiom and begin to generate a new string. In *non-returning* systems the components do not return to their axioms, but continue the generation of their current strings. This holds for all the three protocols: (a), (b), and (c).

In the following we denote by $\Rightarrow_{\text{rew}}$ and $\Rightarrow_{\text{com}}$ a rewriting and a communication step, respectively.

Definition 2.4. The *language* generated by a parallel communicating grammar system $\Gamma = (N, K, T, G_1, \ldots, G_n)$ with $G_i = (N \cup K, T, P_i, S_i)$, $1 \leqslant i \leqslant n$, is

$$L(\Gamma) = \{\alpha_1 \in T^* \mid (S_1, \ldots, S_n) \Rightarrow^* (\alpha_1, \ldots, \alpha_n)\}.$$

Thus, the generated language consists of the terminal strings appearing as sentential forms of the master grammar, G_1.

We denote the classes of returning and non-returning PC grammar systems with at most n components of type Y, where $Y \in \{REG, LIN, CF\}, n \geqslant 1$, by PC_nY and NPC_nY. The corresponding language classes generated by these systems using communication protocol X, where $X \in \{(a), (b), (c)\}$, are denoted by $\mathscr{L}(PC_nY, X)$ and $\mathscr{L}(NPC_nY, X)$, respectively. When an arbitrary number of components is considered, we use $*$ in the subscript instead of n.

Throughout the paper we call a PC grammar system *context-free* (regular, linear, etc.) if its *components* are *context-free* (regular, linear, etc.) *grammars*.

Example 1. Let $\Gamma = (N, K, T, G_1, G_2, G_3)$ be a returning PC grammar system and let $a \in T$. Let us suppose that Γ is in the following configuration:

$$(Q_2Q_3, Q_3, a).$$

With communication protocol (a) we have the following two communication steps:

$$(Q_2Q_3, Q_3, a) \Rightarrow_{\text{com}} (Q_2Q_3, a, S_3) \Rightarrow_{\text{com}} (aS_3, S_2, S_3).$$

With communication protocol (b) we obtain

$$(Q_2Q_3, Q_3, a) \Rightarrow_{\text{com}} (Q_2a, a, S_3) \Rightarrow_{\text{com}} (aa, S_2, S_3).$$

With communication protocol (c) we have two possibilities, we can get

$$(Q_2Q_3, Q_3, a) \Rightarrow_{\text{com}} (Q_2a, a, S_3) \Rightarrow_{\text{com}} (aa, S_2, S_3),$$

or we can obtain

$$(Q_2Q_3, Q_3, a) \Rightarrow_{\text{com}} (Q_3Q_3, S_2, a) \Rightarrow_{\text{com}} (aa, S_2, S_3).$$

In the following, we define some auxiliary notions that we shall use later in the proofs. The first one is the *communication sequence*.

Let $\Gamma = (N, K, T, G_1, \dots, G_n), n \geqslant 1$, be a parallel communicating grammar system and let $(x_1^{(0)}, \dots, x_n^{(0)}) \Rightarrow_{\text{rew}} (x_1^{(1)}, \dots, x_n^{(1)}) \Rightarrow_{\text{com}} (x_1^{(2)}, \dots, x_n^{(2)}) \Rightarrow_{\text{com}} \cdots \Rightarrow_{\text{com}} (x_1^{(t)}, \dots, x_n^{(t)})$, $t \geqslant 1$, be a derivation in Γ, where either $|x_i^{(t)}|_K = 0$, $1 \leqslant i \leqslant n$, or $(x_1^{(t)}, \dots, x_n^{(t)})$ is a blocking configuration where the derivation is blocked by a circular query. Then the sequence of communication steps $(x_1^{(1)}, \dots, x_n^{(1)}) \Rightarrow_{\text{com}} (x_1^{(2)}, \dots, x_n^{(2)}) \Rightarrow_{\text{com}} \cdots \Rightarrow_{\text{com}} (x_1^{(t)}, \dots, x_n^{(t)})$ is said to be a *communication sequence.*

The next notions concerns circular queries. When a circular query is introduced by a rewriting step, it might still be possible to satisfy some of the queries. For example, from configuration $(Q_4, Q_3Q_5, Q_2, \alpha_4, \alpha_5)$, where α_4 and α_5 do not contain further queries, we get $(\alpha_4, Q_3Q_5, Q_2, S_4, \alpha_5)$ with returning communication according to protocol (a) or $(\alpha_4, Q_3\alpha_5, Q_2, S_4, S_5)$ according to protocol (b). With protocol (c) we can reach one of the two following blocking configurations: $(\alpha_4, Q_2\alpha_5, S_3, S_4, S_5)$, or $(\alpha_4, S_2, Q_3\alpha_5, S_4, S_5)$. The query of the first component can be satisfied in all three cases.

We say that a sentential form is *part of a circle* in a configuration obtained by a rewriting step of a PC grammar system Γ using communication protocol X, $X \in \{(a), (b), (c)\}$, if it contains a query symbol which cannot be replaced by any other sentential form using communication mode (a).

These are, for example, Q_3Q_5 and Q_2 in the circular query above, while the others, Q_4, α_4, and α_5, are *not part of a circle.*

Note that we have formulated the condition of being part of a circle through communication protocol (a) for all the three communication modes. If a configuration obtained by a rewriting step contains a circular query, then it will enter into a blocking configuration using any of the communication protocols (a), (b), and (c). But, for all the three communication modes only those sentential forms of the configuration can possibly turn to be terminal words at the end of the communication sequence which are not involved in a circular query according to protocol (a), that is, which satisfy the condition being not part of a circle. Thus, if a word of the language generated by a PC grammar system is obtained with a communication step of a communication sequence leading to a blocking configuration using any of protocols $(a), (b), (c)$, then the sentential form of the master component must not be part of a circle in the configuration obtained by the rewriting step preceding the communication sequence. We shall use this property later.

3. Synchronization

In this section we focus our attention on rule-synchronized context-free PC grammar systems. Rule-synchronization is an additional control mechanism, introduced in [5]. A rule-synchronized PC grammar system is given with a set of so-called transitions (n-tuples of rules, where n is the number of component grammars) which prescribe those combinations of rules that can be used simultaneously during derivations to rewrite the sentential forms of the individual components. In this section we show that rule-synchronization does not change the generative power of context-free returning parallel communicating grammar systems using any of the communication protocols (a), (b), (c). Moreover, we effectively construct for each rule-synchronized context-free returning parallel communicating grammar system an equivalent one (generating the same language using the same communication protocol) without rule-synchronization.

We note that in the case of communication protocol (a) the equal generative power of the two classes of parallel communicating grammar systems follows by [5, 2, 4], however these results imply only the existence of an equivalent parallel communicating grammar system (without rule-synchronization) for each rule-synchronized context-free PC grammar system.

First, we recall the notion of a transition in a PC grammar system from [9].

Definition 3.1. Let $\Gamma = (N, K, T, G_1, \ldots, G_n)$ be a PC grammar system with components $G_i = (N \cup K, T, P_i, S_i)$, $1 \leqslant i \leqslant n$. A *transition* of Γ is an n-tuple $\bar{r} = (r_1, \ldots, r_n)$, where $r_i \in (P_i \cup \{\sharp\})$, $1 \leqslant i \leqslant n$, and $\sharp$ is an additional symbol, $\sharp \notin (N \cup K \cup T)$.

A transition $\bar{r} = (r_1, \ldots, r_n)$ is applied in the rewriting step $(\alpha_1, \ldots, \alpha_n) \Rightarrow_{\text{rew}} (\beta_1, \ldots, \beta_n)$ of Γ, if $\alpha_i \Rightarrow_{G_i} \beta_i$ by applying r_i for $r_i \in P_i$ and $\beta_i = \alpha_i$, $\alpha_i \in T^*$ for $r_i = \sharp$, $1 \leqslant i \leqslant n$.

Definition 3.2. A *PC grammar system with rule-synchronization* is an $(n+4)$-tuple $\Gamma = (N, K, T, G_1, \ldots, G_n, R)$, $n \geqslant 1$, where $(N, K, T, G_1, \ldots, G_n)$ is a PC grammar system and R is a set of its transitions.

In each rewriting step $(x_1, \ldots, x_n) \Rightarrow_{\text{rew}} (y_1, \ldots, y_n)$ of a rule-synchronized PC grammar system Γ one of the transitions $\bar{r} \in R$ must be applied. Communication is defined in the same way as in the generic case (Definition 2.3).

We denote by RZ_nY the class of rule-synchronized PC grammar systems of type Z with n components of type Y, where $Z \in \{PC, NPC\}$, $Y \in \{REG, LIN, CF\}$, $n \geqslant 1$. $\mathscr{L}(RZ_nY, X)$ denotes the corresponding language class generated by these systems using communication protocol X, where $X \in \{(a), (b), (c)\}$. If an arbitrary number of components is considered, we put $*$ in the subscript instead of n.

In rule-synchronized PC grammar systems the set R consists of all the transitions which can ever be applied to the sentential forms in course of a derivation. We can associate a set with the same property also to PC grammar systems without rule-synchronization, it is the set of all possible transitions that can be constructed from the rules and $\sharp$.

Definition 3.3. Let $\Gamma = (N, K, T, G_1, \ldots, G_n)$, $n \geqslant 1$, be a PC grammar system. The set $M = ((P_1 \cup \{\sharp\}) \times \cdots \times (P_n \cup \{\sharp\}))$, where P_i is the production set of G_i, $1 \leqslant i \leqslant n$, and $\sharp$ is the symbol defined in Definition 3.1 is called the *transition set* of Γ.

If we take a PC grammar system $\Gamma = (N, K, T, G_1, \ldots, G_n)$ and its transition set M, then the rule-synchronized system $\Gamma' = (N, K, T, G_1, \ldots, G_n, M)$ with the same communication protocol obviously generates the same language as Γ, so we have the following statement by the definitions (for protocol (a) the statement can be found in [5]):

Theorem 3.1.

$$\mathscr{L}(Z_nY,X) \subseteq \mathscr{L}(RZ_nY,X),$$

where $n \geqslant 1$, $Z \in \{PC, NPC\}$, $Y \in \{REG, LIN, CF\}$, $X \in \{(a),(b),(c)\}$.

In the following, we are going to study the converse inclusion. We show that for any rule-synchronized context-free returning PC grammar system we can construct a context-free returning PC grammar system without rule-synchronization such that using the same communication protocol the two systems generate the same language. The statement is valid for all the three communication protocols, (a), (b), and (c). Before turning to the construction, we present a statement comparing the three communication protocols in a particular case where all sentential forms contain at most one query symbol.

Lemma 3.2. *Let Γ be a context-free returning PC grammar system where each production contains at most one query symbol on its right-hand side.*

Then Γ generates the same language using any of the three communication protocols, (a), (b), *or* (c).

Proof. First, let us compare communication protocols (b) and (c). If a sentential form containing a query symbol is not part of a circle in a configuration obtained by a rewriting step, then at the end of the possible following communication sequence the query symbol is replaced by the same string in both cases, since all components communicate their strings at most once during the communication sequence. If a sentential form is part of a circle, then the query symbol it contains is either unchanged (using protocol (b)), or it is replaced only with strings containing further queries (using protocol (c)). At the end of the corresponding communication sequence the same query symbols are replaced by query symbol free strings both in the case of protocol (b) and protocol (c), and also the replacing strings coincide.

This also holds for communication protocol (a). If the sentential forms contain at most one query symbol, then every queried component communicates its sentential form in at most one of the communication steps during any communication sequence. This means that the communication sequences not only produce the same result, but the sequences themselves are identical with the communication sequences using protocol (b). □

We note that for communication protocol (a) and (b) a stronger result can be found in [12] which states the following: if all rules of a PC grammar system Γ with at least one occurrence of a query symbol on their right-hand sides are homogeneous, that is, they contain any number of occurrences but only of one query symbol, then Γ generates the same language with both communication protocols, (a) or (b).

Now, we are going to show how to construct an equivalent context-free returning PC grammar system (without rule-synchronization) for a rule-synchronized one. The construction is based on creating an n-tuple of grammars for each transition which is allowed to be applied in the rule-synchronized system. Rule-synchronization then can be simulated with a "server" n-tuple which sends the actual sentential forms to one of those n-tuples that can simulate the application of a certain transition. After the application of the transition the sentential forms are sent back to the server, which communicates them to another n-tuple to simulate another transition in the next step.

Theorem 3.3. *For every rule-synchronized context-free returning PC grammar system Γ we can construct a context-free returning PC grammar system Γ_R such that Γ_R and Γ generate the same language using the same communication protocol X, $X \in \{(a),(b),(c)\}$.*

Proof. Let $\Gamma = (N,K,T,G_1,\dots,G_n,R)$, $n \geqslant 1$, be a rule-synchronized context-free returning PC grammar system. We are going to construct a PC grammar system Γ_R without rule-synchronization which generates the same language when uses the same communication protocol as Γ. The construction is independent from the chosen protocol, Γ_R is constructed in the same way for all three cases.

We first introduce a notation. Let us denote the transitions in R by $\bar{r}_k = (r_{k,1},\dots,r_{k,n})$, where $r_{k,i} \in (P_i \cup \{\sharp\})$, $1 \leqslant k \leqslant |R|$, $1 \leqslant i \leqslant n$. Then let $R' = \{\bar{r}'_k = (r'_{k,1},\dots,r'_{k,n}) \mid 1 \leqslant k \leqslant |R|\}$, where $\bar{r}'_k$ is defined as follows: if $r_{k,i} = X \to \alpha$, $1 \leqslant k \leqslant |R|$, $1 \leqslant i \leqslant n$, then $r'_{k,i} = X \to \alpha'$, where $\alpha' = \alpha$ for $|\alpha|_K = 0$ and $\alpha' = \alpha_1 Q'_{i_1} \alpha_2 \dots \alpha_t Q'_{i_t} \alpha_{t+1}$ for $\alpha = \alpha_1 Q_{i_1} \alpha_2 \dots \alpha_t Q_{i_t} \alpha_{t+1}$ with $|\alpha_j|_K = 0$, $1 \leqslant j \leqslant t+1$, $t \geqslant 1$.

Now we construct Γ_R. Let

$$\begin{aligned}\Gamma_R = (N',K',T,\ & G_M, G_1^{serv},..,G_n^{serv}, G_1^{a_1},..,G_n^{a_1},\\ & G_1^{(1)},..,G_n^{(1)},.....,G_1^{(|R|)},..,G_n^{(|R|)}, G_1^{(1)'},..,G_n^{(1)'},.....,G_1^{(|R|)'},..,G_n^{(|R|)'},\\ & G_1^{a_2},..,G_n^{a_2}, G_1^{a_3},..,G_n^{a_3}, G^{sel}, G'_1,..,G'_n, G_1^{a_4},..,G_n^{a_4}, G^{a_5},\\ & G_1^{a_6},..,G_n^{a_6}, G_1^{a_7},..,G_n^{a_7}),\end{aligned}$$

where

$G_M = (N' \cup K', T, P_M, S_M)$ is the master grammar,

$G_i^{serv} = (N' \cup K', T, P_i^{serv}, S_i)$,

$G_i^{\alpha} = (N' \cup K', T, P_i^{\alpha}, S_i^{\alpha})$,

$1 \leqslant i \leqslant n$, $\alpha \in \{a_1,a_2,a_3,a_4,a_6,a_7\} \cup \{'\} \cup \{(k),(k)' \mid 1 \leqslant k \leqslant |R|\}$,

and

$G^{\beta} = (N' \cup K', T, P^{\beta}, S^{\beta})$, $\beta \in \{sel, a_5\}$.

The sets of non-terminals and query symbols are the following:

$$N' = \{S_M, S_i^{serv^{(1)}}, S_i^{serv^{(2)}}, S_i^{a_1}, S_i^{a_2}, S_i^{a_3}, S_i^{a_3(1)}, S_i^{a_3(2)}, S_i^{a_3(3)},$$

$$
\begin{aligned}
&S^{sel}, S_i^{a_4}, S^{a_5}, S_i^{a_6}, S_i^{a_6(1)}, S_i^{a_6(2)}, S_i^{a_7}, S_i^{(k)}, S_i^{(k)'}, (k),\\
&(k)^{(1)}, (k)^{(2)}, (k)^{(3)}, S_i', S_i'^{(1)} \mid 1 \leqslant i \leqslant n, 1 \leqslant k \leqslant |R|\}\\
&\cup \{X, [X], [X]^{(1)} \mid X \in N\}\\
&\cup \{A, B\},
\end{aligned}
$$

and

$$
\begin{aligned}
K' = \{&Q_M, Q_i^{serv}, Q_i^{a_1}, Q_i^{a_2}, Q_i^{a_3}, Q^{sel}, Q_i^{a_4}, Q^{a_5}, Q_i^{a_6}, Q_i^{a_7}, Q_i^{(k)}, Q_i^{(k)'}, Q_i' \mid\\
&1 \leqslant i \leqslant n, 1 \leqslant k \leqslant |R|\}.
\end{aligned}
$$

The rule sets of the component grammars are as follows:

$$P_M = \{S_M \to S_M, S_M \to Q_1^{(k)} \mid 1 \leqslant k \leqslant |R|\}.$$

This is the master component, the terminal strings generated by the system are selected by this component.

$$
\begin{aligned}
P_i^{serv} = &\{X \to [X] \mid X \in N\} \cup \{S_i'^{(1)} \to [S_i]\}\\
&\cup \{S_i \to S_i^{serv^{(1)}}, S_i^{serv^{(1)}} \to S_i^{serv^{(2)}}, S_i^{serv^{(2)}} \to Q_i'\}
\end{aligned}
$$

for $1 \leqslant i \leqslant n$. These are the so-called server components. They choose the nonterminals to be rewritten according to a certain transition chosen by G^{sel}, then they communicate the sentential forms to one of the transition simulating n-tuple of components and wait for the result.

$$P_i^{a_1} = \{S_i^{a_1} \to Q_i^{serv}, S_i^{serv^{(1)}} \to S_i^{serv^{(1)}}\}$$

for $1 \leqslant i \leqslant n$. These assistant components are used to synchronize the work of the system. They query the components G_i^{serv}, $1 \leqslant i \leqslant n$, after the initial rewriting step, forcing them to restart the derivation from their axiom.

$$
\begin{aligned}
P_i^{(k)} = &\{S_i^{(k)} \to Q^{sel}, (k) \to Q_i^{serv}\}\\
&\cup \{[X] \to [X]^{(1)}, [X]^{(1)} \to \alpha \mid X \to \alpha = r'_{k,i}\}\\
&\cup \{(j) \to (j)^{(1)}, (j)^{(1)} \to (j)^{(2)}, (j)^{(2)} \to (j)^{(3)},\\
&\quad (j)^{(3)} \to Q^{sel} \mid 1 \leqslant j \leqslant |R|,\ j \neq k\}
\end{aligned}
$$

for $1 \leqslant i \leqslant n$, $1 \leqslant k \leqslant |R|$. These are the transition simulating n-tuples, they simulate the application of the kth transition of the rule-synchronized system.

$$
\begin{aligned}
P_i^{(k)'} = &\{(k) \to AB, A \to Q_i^{a_6}, [X] \to [X] \mid X \to \alpha = r'_{k,i}\}\\
&\cup \{(k) \to A, A \to Q_i^{a_6} \mid r'_{k,i} = \sharp\}\\
&\cup \{(j) \to (j)^{(1)}, (j)^{(1)} \to (j)^{(2)}, (j)^{(2)} \to (j)^{(3)},\\
&\quad (j)^{(3)} \to Q^{sel} \mid 1 \leqslant j \leqslant |R|,\ j \neq k\},\\
&\cup \{S_i^{(k)'} \to Q^{sel}\}
\end{aligned}
$$

for $1 \leqslant i \leqslant n$, $1 \leqslant k \leqslant |R|$. These n-tuples work together with the transition simulating components, they check whether the kth transition can be applied to the sentential forms. If this transition is selected, but it is not applicable, they block the work of the system.

$$P_i^{a_2} = \{(k) \to (k)^{(1)}, (k)^{(1)} \to (k)^{(2)} S_i^{a_3}, (k)^{(2)} \to Q_i^{(k)'} \mid 1 \leqslant k \leqslant |R|\}$$
$$\cup \{S_i^{a_2} \to Q^{sel}\},$$
$$P_i^{a_3} = \{S_i^{a_3} \to S_i^{a_3(1)}, S_i^{a_3(1)} \to S_i^{a_3(2)}, S_i^{a_3(2)} \to S_i^{a_3(3)}, S_i^{a_3(3)} \to Q_i^{a_2}\}$$

for $1 \leqslant i \leqslant n$. These components query components $G_i^{(k)'}$, $1 \leqslant i \leqslant n$, to remove their sentential forms when they are not needed any more.

$$P^{sel} = \{S^{sel} \to (k), (k) \to (k) \mid 1 \leqslant k \leqslant |R|\}.$$

This component selects the index of the transition of the rule-synchronized system that is going to be simulated.

$$P_i' = \{S_i' \to S_i'^{(1)}, S_i'^{(1)} \to Q^{a_5}, (k) \to Q_i^{(k)} \mid 1 \leqslant k \leqslant |R|\} \cup \{X \to X \mid X \in N \cup \{B\}\}$$

for $1 \leqslant i \leqslant n$. After the simulation of a transition, the resulting sentential forms are transferred to these components, and then sent back to G_i^{serv}, $1 \leqslant i \leqslant n$.

$$P_i^{a_4} = \{S_i^{a_4} \to Q_i', S_i'^{(1)} \to S_i'^{(1)}\}$$

for $1 \leqslant i \leqslant n$. These components are used to synchronize the work of the system. They query components G_i', $1 \leqslant i \leqslant n$, after the initial rewriting step, forcing them to restart their work.

$$P^{a_5} = \{S^{a_5} \to (k), (k) \to (k) \mid 1 \leqslant k \leqslant |R|\}.$$

This component assists the work of G_i', $1 \leqslant i \leqslant n$. It selects the index of a transition which refers to the n-tuple of grammars that will be queried by components G_i' in order to send their sentential forms back to G_i^{serv}.

$$P_i^{a_6} = \{S_i^{a_6} \to S_i^{a_6(1)}, S_i^{a_6} \to Q_i^{serv}, S_i^{a_6(1)} \to S_i^{a_6(2)}, S_i^{a_6(2)} \to Q_i^{serv},$$
$$[X] \to [X] \mid X \in N\},$$
$$P_i^{a_7} = \{S_i^{a_7} \to Q_i^{a_6}, S_i^{a_6(1)} \to S_i^{a_6(1)}\}$$

for $1 \leqslant i \leqslant n$. These components assist the work of $G_i^{(k)'}$, $1 \leqslant i \leqslant n$, $1 \leqslant k \leqslant |R|$. They query the server components G_i^{serv}, receive their sentential forms and then send them to $G_i^{(k)'}$.

Now we explain how Γ_R simulates the application of a transition of Γ. First Γ_R selects k, the index of the transition to be simulated by introducing the non-terminal (k) at component G^{sel}. The application of this transition is simulated by components $G_1^{(k)}, \ldots, G_n^{(k)}$, so the sentential forms of $G_1^{serv}, \ldots, G_n^{serv}$ are sent to these components and also to components $G_1^{a_6}, \ldots, G_n^{a_6}$ which communicate them to components

$G_1^{(k)'}, \ldots, G_n^{(k)'}$. These primed components check whether or not the selected transition $\bar{r}'_k$ can be applied to the current sentential forms sent by the server components. If the answer to this test is positive, the sentential forms are rewritten according to $\bar{r}'_k$ and then they are returned to $G_1^{serv}, \ldots, G_n^{serv}$ through the components $G'_1, \ldots, G'_n$. If a terminal string appears in $G_1^{(k)}$, it can be transferred to G_M, the master component, producing the result of the derivation. The components $G_1^{a_1}, \ldots, G_n^{a_1}$, $G_1^{a_2}, \ldots, G_n^{a_2}$, $G_1^{a_3}, \ldots, G_n^{a_3}$, $G_1^{a_4}, \ldots, G_n^{a_4}$, $G_1^{a_6}, \ldots, G_n^{a_6}$, and $G_1^{a_7}, \ldots, G_n^{a_7}$ are used to synchronize the system by querying certain components, forcing them this way to restart their work by returning to their axioms if it is necessary.

Now we describe the simulation in detail. First, we show how Γ_R simulates the initial rewriting step of Γ.

Γ_R starts with the following configuration:

$$\begin{aligned}
&(S_M, S_1, .., S_n, S_1^{a_1}, .., S_n^{a_1},\\
&S_1^{(1)}, .., S_n^{(1)}, ..., S_1^{(k)}, .., S_n^{(k)}, ..., S_1^{(|R|)}, .., S_n^{(|R|)},\\
&S_1^{(1)'}, .., S_n^{(1)'}, ..., S_1^{(k)'}, .., S_n^{(k)'}, ..., S_1^{(|R|)'}, .., S_n^{(|R|)'},\\
&S_1^{a_2}, .., S_n^{a_2}, S_1^{a_3}, .., S_n^{a_3}, S^{sel}, S'_1, .., S'_n, S_1^{a_4}, .., S_n^{a_4},\\
&S^{a_5}, S_1^{a_6}, .., S_n^{a_6}, S_1^{a_7}, .., S_n^{a_7}).
\end{aligned}$$

The master component G_M can either leave its start symbol S_M unchanged, or it can introduce a query symbol $Q_1^{(k)}$, $1 \leqslant k \leqslant |R|$. In the latter case the derivation is blocked unless $G_1^{(k)}$ contains a terminal string, so we assume that S_M remains unchanged.

Now, component G^{sel} selects a transition to be simulated by introducing the nonterminal (k), $1 \leqslant k \leqslant |R|$, corresponding to transition $\bar{r}'_k$. Then (k) is communicated to the other components. After a rewriting step and the following communication sequence we obtain configuration:

$$\begin{aligned}
&(S_M, S_1, .., S_n, S_1^{serv^{(1)}}, .., S_n^{serv^{(1)}},\\
&(k), .., (k), \ldots, (k), .., (k), \ldots, (k), .., (k),\\
&(k), .., (k), \ldots, (k), .., (k), \ldots, (k), .., (k),\\
&(k), .., (k), S_1^{a_3(1)}, .., S_n^{a_3(1)}, S^{sel}, S'_1, .., S'_n, S_1'^{(1)}, .., S_n'^{(1)},\\
&(l), S_1^{a_6}, .., S_n^{a_6}, S_1^{a_6(1)}, .., S_n^{a_6(1)}).
\end{aligned}$$

Now the components $G_i^{(k)}$ are going to query G_i^{serv}, $1 \leqslant i \leqslant n$. Then $G_i^{(k)'}$ will check whether $\bar{r}'_k$, the selected transition, can be applied to the sentential forms received from G_i^{serv} and $G_i^{(k)}$ will apply $\bar{r}'_k$.

After a rewriting step, a communication sequence, and again a rewriting step, we obtain

$$\begin{aligned}&(S_M, S_1^{serv^{(1)}}, .., S_n^{serv^{(1)}}, S_1^{serv^{(1)}}, .., S_n^{serv^{(1)}},\\ &(k)^{(2)}, .., (k)^{(2)}, \ldots, [S_1]^{(1)}, .., [S_n]^{(1)}, \ldots, (k)^{(2)}, .., (k)^{(2)},\\ &(k)^{(2)}, .., (k)^{(2)}, \ldots, u_1^{(k)'}, .., u_n^{(k)'}, \ldots, (k)^{(2)}, .., (k)^{(2)},\\ &(k)^{(2)} S_1^{a_3}, .., (k)^{(2)} S_n^{a_3}, S_1^{a_3(3)}, .., S_n^{a_3(3)}, (m), Q^{a_5}, .., Q^{a_5}, S_1'^{(1)}, .., S_n'^{(1)},\\ &(l), [S_1], \ldots, [S_n], S_1^{a_6(1)}, .., S_n^{a_6(1)}).\end{aligned}$$

Now the components $G_1^{(k)'}, \ldots, G_n^{(k)'}$ are going to check whether $\bar{r}'_k = (r'_{k,1}, \ldots, r'_{k,n})$, the transition selected for simulation, can be applied to the current n-tuple of sentential forms, $[S_1], \ldots, [S_n]$. This is done in the following way: if $r'_{k,j} = \sharp$, then $u_j^{(k)'} = Q_j^{a_6}$, and if $r'_{k,j} = X \to \alpha,\ X \in N,\ \alpha \in (N \cup T \cup K')^*$, then $u_j^{(k)'} = Q_j^{a_6} B$. In the first case, when $r'_{k,j} = \sharp$, the sentential form of G_j^{serv} which is now present at $G_j^{a_6}$, should be a terminal word. If it is not, then the system is going to get blocked under rewriting after the communication, since $G_j^{(k)'}$ cannot rewrite any symbol from $N \cup \{[X] \mid X \in N\}$. In the second case, when $r'_{k,j} = X \to \alpha$, the sentential form of $G_j^{a_6}$ should contain non-terminal $[X]$. If it does not contain $[X]$, but contains another non-terminal $[Y]$, $[Y] \neq [X]$ instead, the system is going to be blocked under rewriting after the communication, since $G_j^{(k)'}$ cannot rewrite $[Y]$. In this second case, if the sentential form of G_j^{serv} is a terminal word, the system also blocks, since $G_j^{(k)'}$ cannot rewrite symbol B.

Let us suppose that $\bar{r}'_k$ is applicable to the current sentential forms and let $r'_{k,i} = S_i \to \alpha_i,\ \alpha_i \in (N \cup T \cup K')^*,\ 1 \leqslant i \leqslant n$. In this case after communication and one more rewriting step we have

$$\begin{aligned}&(u_M, S_1^{serv^{(2)}}, .., S_n^{serv^{(2)}}, S_1^{serv^{(1)}}, .., S_n^{serv^{(1)}},\\ &(k)^{(3)}, .., (k)^{(3)}, \ldots, \alpha_1, .., \alpha_n, \ldots, (k)^{(3)}, .., (k)^{(3)},\\ &(k)^{(3)}, .., (k)^{(3)}, \ldots, [S_1]B, .., [S_n]B, \ldots, (k)^{(3)}, .., (k)^{(3)},\\ &Q_1^{(k)'} S_1^{a_3}, .., Q_n^{(k)'} S_n^{a_3}, Q_1^{a_2}, .., Q_n^{a_2}, (m), Q_1^{(l)}, .., Q_n^{(l)}, S_1'^{(1)}, .., S_n'^{(1)},\\ &(s), S_1^{a_6(1)}, .., S_n^{a_6(1)}, S_1^{a_6(1)}, .., S_n^{a_6(1)}),\end{aligned}$$

where $u_M = S_M$ or $u_M = Q_1^{(t)}$ for some $t,\ 1 \leqslant t \leqslant |R|$, and α_i is the right-hand side of $r'_{k,i}$.

The sentential forms of the components $G'_1, \ldots, G'_n$ are $Q_1^{(l)}, \ldots, Q_n^{(l)}$, for some $l,\ 1 \leqslant l \leqslant |R|$. If $l = k$, the sentential forms are $Q_1^{(k)}, \ldots, Q_n^{(k)}$, then after the next communication, the result of the communication sequence introduced by the sentential forms $\alpha_1, \ldots, \alpha_n$ appears at the components $G'_1, \ldots, G'_n$. If Γ_R uses the same communication protocol as Γ, then these sentential forms are the same that we would obtain by communication in Γ starting from configuration $(\alpha_1, \ldots, \alpha_n)$ supposing that we replace the possible occurrences of query symbols and axioms with their primed versions. If $l \neq k$, then the sentential forms of $G'_1, \ldots, G'_n$ are $Q_1^{(l)}, \ldots, Q_n^{(l)}$, and then the system blocks after communication. To see this, notice that the sentential forms $\alpha_1, \ldots, \alpha_n$ cannot contain Q'_i for all i, $1 \leqslant i \leqslant n$ (in this case they would introduce a circular query), so at least one of the sentential forms of $G'_1, \ldots, G'_n$ should be $(k)^{(3)}$ after communication, which symbol cannot be rewritten in G'_i, $1 \leqslant i \leqslant n$.

The sentential form that is sent to G'_1 may also appear in G_M if $u_M = Q_1^{(k)}$. In this case either a terminal string is derived, or if the string is not a terminal one, the system is blocked. If $u_M = S_M$, the derivation can continue. After the communication sequence we have

$$
\begin{aligned}
&(u'_M, S_1^{serv^{(2)}}, .., S_n^{serv^{(2)}}, S_1^{serv^{(1)}}, .., S_n^{serv^{(1)}},\\
&(k)^{(3)}, .., (k)^{(3)}, \ldots, S_1^{(k)}, .., S_n^{(k)}, \ldots, (k)^{(3)}, .., (k)^{(3)},\\
&(k)^{(3)}, .., (k)^{(3)}, \ldots, S_1^{(k)'}, .., S_n^{(k)'}, \ldots, (k)^{(3)}, .., (k)^{(3)},\\
&S_1^{a_2}, .., S_n^{a_2}, [S_1]BS_1^{a_3}, .., [S_n]BS_n^{a_3}, (m), \beta'_1, .., \beta'_n, S_1'^{(1)}, .., S_n'^{(1)},\\
&(s), S_1^{a_6(1)}, .., S_n^{a_6(1)}, S_1^{a_6(1)}, .., S_n^{a_6(1)})
\end{aligned}
$$

with $\beta'_i = \beta_i$ if $\beta_i \neq S_i$, and $\beta'_i = S'_i$ if $\beta_i = S_i$, $1 \leqslant i \leqslant n$, where $\beta_1, .., \beta_n$ is the result of the communication sequence possibly following the application of transition $\bar{r}'_k$. u'_M is either S_M, or a terminal string, or the derivation is blocked. If the derivation can continue, then after the next rewriting step and a communication sequence we obtain the configuration

$$
\begin{aligned}
&(S_M, \beta''_1, .., \beta''_n, S_1^{serv^{(1)}}, .., S_n^{serv^{(1)}},\\
&(m), .., (m), \ldots, (m), .., (m), \ldots, (m), .., (m),\\
&(m), .., (m), \ldots, (m), .., (m), \ldots, (m), .., (m),\\
&(m), .., (m), [S_1]BS_1^{a_3(1)}, .., [S_n]BS_n^{a_3(1)}, S^{sel}, S'_1, .., S'_n, S_1'^{(1)}, .., S_n'^{(1)},\\
&(s), S_1^{a_6(2)}, .., S_n^{a_6(2)}, S_1^{a_6(1)}, .., S_n^{a_6(1)}),
\end{aligned}
$$

where $\beta''_i = S_i'^{(1)}$ for $\beta'_i = S'_i$ or $\beta''_i = \beta'_i$ otherwise, $1 \leqslant i \leqslant n$. Now the system is ready to simulate the application of a transition $\bar{r}'_m$ in the same manner as above. Note that the sentential form of the component G_i^{serv} is the same string that we would obtain after the application of the simulated transition and the possible following communication sequence at the component $G_i, 1 \leqslant i \leqslant n$, supposing that the possible occurrences of query symbols and axioms are replaced with their primed versions.

If we start from configuration

$$
\begin{aligned}
&(S_M, \alpha_1, .., \alpha_n, S_1^{serv^{(1)}}, .., S_n^{serv^{(1)}},\\
&(m), .., (m), \ldots, (m), .., (m), \ldots, (m), .., (m),\\
&(m), .., (m), \ldots, (m), .., (m), \ldots, (m), .., (m),\\
&(m), .., (m), \gamma_1 S_1^{a_3(1)}, .., \gamma_n S_n^{a_3(1)}, S^{sel}, S'_1, .., S'_n, S_1'^{(1)}, .., S_n'^{(1)},\\
&(s), S_1^{a_6(2)}, .., S_n^{a_6(2)}, S_1^{a_6(1)}, .., S_n^{a_6(1)}),
\end{aligned}
$$

where $\alpha_i, 1 \leqslant i \leqslant n$, is an arbitrary sentential form and $\gamma_i \in (N' \cup T)^*$ we obtain

$$\begin{array}{l}(S_M, S_1, .., S_n, S_1^{serv^{(1)}}, .., S_n^{serv^{(1)}},\\ (m)^{(1)}, .., (m)^{(1)}, \ldots, [\alpha_1], .., [\alpha_n], \ldots, (m)^{(1)}, .., (m)^{(1)},\\ (m)^{(1)}, .., (m)^{(1)}, \ldots, u_1^{(m)'}, .., u_n^{(m)'}, \ldots, (m)^{(1)}, .., (m)^{(1)},\\ (m)^{(1)}, .., (m)^{(1)}, \gamma_1 S_1^{a_3(2)}, .., \gamma_n S_n^{a_3(2)}, (r), S_1'^{(1)}, .., S_n'^{(1)}, S_1'^{(1)}, .., S_n'^{(1)},\\ (s), [\alpha_1], .., [\alpha_n], S_1^{a_6(1)}, .., S_n^{a_6(1)}),\end{array}$$

where $[\alpha_i]$ denotes the string α_i with one of its non-terminals in brackets, that is, $[\alpha_i] = \alpha_{i_1}[X]\alpha_{i_2}$, $\alpha_i = \alpha_{i_1} X \alpha_{i_2}$, $X \in N$ or if $\alpha_i \in T^*$, then $[\alpha_i] = \alpha_i$. The string $u_i^{(m)'}$ is AB if $r'_{m,} \neq \sharp$ or $u_i^{(m)'} = A$ if $r'_{m,} = \sharp$, $1 \leqslant i \leqslant n$. Now the system checks the applicability of transition $\bar{r}'_m$ and applies $\bar{r}'_m$ in the way we have previously described.

Repeating these steps, we can see that each derivation of Γ can be simulated by a derivation of Γ_R. Moreover, by the construction of Γ_R, each derivation of Γ_R corresponds to a derivation in Γ, the only possible way for Γ_R to work is to simulate the application of the transitions of Γ in the above described manner. Any other way of functioning leads to a blocking configuration without generating a terminal word. Note that the construction of Γ_R is independent of the communication protocol used by Γ. If it is the same as in Γ, then the two PC grammar systems, Γ and Γ_R, generate the same language. To see this, consider Lemma 3.2 and the fact that apart from the transition simulating ones, all components of Γ_R introduce at most one query symbol in their sentential forms. This means, that only the work of the transition simulating components is dependent on the communication protocol used, the others work in the same way in all of the three cases. By these considerations we can see that Γ_R generates the same language using the same communication protocol as Γ. □

As a direct consequence of Theorems 3.1 and 3.3 we obtain

Corollary 3.4.

$$\mathscr{L}(RPC_*CF, X) = \mathscr{L}(PC_*CF, X), \quad \textit{where } X \in \{(a), (b), (c)\}.$$

4. Communication and normal forms

In this section we examine the relationships among the language classes generated by PC grammar systems using different communication protocols. First we study the case of non-returning systems and the case of returning systems with regular or linear components. A PC grammar system of one of these types generates the same language with any of the three communication protocols.

Next, we look at the remaining case, the case of context-free returning PC grammar systems. We show that every context-free returning PC grammar system can be transformed to a context-free returning system which uses rules of a very simple form while generating the same language. The equality of language classes generated by context-free returning systems using different protocols will follow from this normal form result.

First we look at non-returning systems. In this case the use of different communication protocols has no effect on the generated language.

Lemma 4.1.

$$\mathscr{L}(X_nY,(a)) = \mathscr{L}(X_nY,(b)) = \mathscr{L}(X_nY,(c)),$$

where $X \in \{NPC, RNPC\}$, $Y \in \{REG, LIN, CF\}$.

Proof. In the case of non-returning systems the components do not return to their axioms during the communication sequence, so if we start from a configuration without a circular query, then each occurrence of the same query symbol in all sentential forms will be replaced with the same string for any of the three communication protocols. For protocols (a) and (b) the statement obviously holds and for protocol (c) it follows from the property that during a communication sequence under protocol (c) each component communicates its sentential form in exactly one of the communication steps of the communication sequence. Similarly, if the starting configuration contains a circular query, then after finishing the communication sequence the sentential forms which are not part of the circle will determine the same subconfiguration for any of the three communication protocols. This implies that non-returning PC grammar systems generate the same language in any of the three communication modes. □

By Lemma 3.2 the same statement can be presented for returning systems with regular or linear components.

Lemma 4.2.

$$\mathscr{L}(X_nY,(a)) = \mathscr{L}(X_nY,(b)) = \mathscr{L}(X_nY,(c)),$$

where $X \in \{PC, RPC\}$, $Y \in \{REG, LIN\}$.

Now, we turn to context-free returning PC grammar systems. We show that each system of this type can be transformed into a system that uses rules of a simple form while generates the same language. As a consequence of this result and Lemma 3.2 the equal power of classes of context-free returning PC grammar systems using any of the three communication protocols follows.

First, we present a theorem showing that all languages generated by context-free returning PC grammar systems can also be generated by systems of the same type with rules having at most two symbols on their right-hand side.

Definition 4.1. Let $\Gamma = (N, K, T, G_1, \dots, G_n)$, $n \geqslant 1$, be a context-free PC grammar system. A rule $X \to \alpha$ of Γ is in *strong binary form* if one of the following cases holds: $\alpha = Q_jQ_l$, $\alpha = AQ_j$, $\alpha = Q_j$, $\alpha = A$, $\alpha = \varepsilon$, where $Q_j, Q_l \in K$, $j \neq l$, $1 \leqslant j, l \leqslant n$ and $A \in (N \cup T)$. A *PC grammar system is in strong binary form*, if all of its rules are in strong binary form.

Theorem 4.3. *For each returning context-free PC grammar system Γ with communication protocol X, where $X \in \{(a),(b),(c)\}$, we can construct a context-free returning PC grammar system Γ' in strong binary form that generates the same language and uses the same communication protocol as Γ.*

Proof. Let $\Gamma = (N, K, T, G_1, \dots, G_n)$ be a context-free returning PC grammar system with component grammars $G_i = (N \cup K, T, P_i, S_i)$, $1 \leqslant i \leqslant n$, and let R be the set of all its transitions. To prove the statement, we first construct a set of transitions R' with rules in strong binary form and a rule-synchronized context-free returning PC grammar system Δ with transition set R' such that Δ uses the same communication protocol and generates the same language as Γ. Then, by Theorem 3.3 we can construct a context-free returning PC grammar system Γ'_R without rule-synchronization which generates the same language and uses the same communication protocol as Γ. Moreover, since the rules in R' are in strong binary form, Γ'_R is also in strong binary form.

We first prove the statement for PC grammar systems with communication protocol (a). Let $t = |\alpha|$, where $X \to \alpha$ is the rule with the longest right-hand side in Γ. Let us define

$$\begin{aligned}\Delta = (&N', K', T,\\ &G_{1,1}, .., G_{1,t}, \dots, G_{n,1}, .., G_{n,t},\\ &G^a_{1,1}, .., G^a_{1,t}, \dots, G^a_{n,1}, .., G^a_{n,t}, R')\end{aligned}$$

with

$$N' = N \cup \{S^a_{i,j} \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant t\},$$

$$K' = \{Q_{i,j}, Q^a_{i,j} \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant t\},$$

where $G_{i,j} = (N' \cup K', T, P_{i,j}, S_i)$ and $G^a_{i,j} = (N' \cup K', T, P^a_{i,j}, S^a_{i,j})$, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant t$.

Now we construct R'. First, we modify the transitions in R in the following manner: Those transitions which have no rule with an occurrence of a query symbol remain unchanged. Now, suppose that some production of a transition $\bar{r}_k$, $1 \leqslant k \leqslant |R|$, has at least one query symbol at the right-hand side. Then we can replace $\bar{r}_k$ by another transition $\bar{r}''_k$ such that at the end of the communication sequence following the application of $\bar{r}''_k$ the obtained configuration will be the same as the configuration we would get at the end of the communication sequence following the application of $\bar{r}_k$ and during the communication sequence following the application of $\bar{r}''_k$ every queried component communicates its string in exactly one of the communication steps. To see this, let us consider a returning communication sequence where there is a component, G_j, $1 \leqslant j \leqslant n$, which communicates its sentential form in more than one communication steps during the communication sequence. Then G_j sends S_j, its axiom to all querying components which did not receive its sentential form for the first time, because it had already returned to the axiom. Since we can establish which occurrences of Q_j will be replaced by S_j in the sentential forms, we can modify the rules $X \to \alpha Q_j \beta$ in $\bar{r}_k$ by changing these occurrences of Q_j for S_j. If we make these modifications for each query symbol Q_j with the above property, we obtain a new transition which has the same effect as the original one and which has the required property, namely, that each communicating component is active in exactly one of the communication steps of the communication sequence. (For illustration see Example 2.)

By modifying the transitions in R in the above manner, we can immediately see that we obtain a new set of transitions $\bar{R}$ such that the rule-synchronized context-free returning PC grammar system $\bar{\Gamma} = (N, K, T, G_1, \ldots, G_n, \bar{R})$ generates the same language as Γ under communication protocol (a).

Starting from $\bar{R}$, we are going to construct R', the transition set of Δ. The transitions in R' will use $2nt$ components to simulate the transitions in $\bar{R}$, where t is the length of the longest right-hand side of the rules in $\bar{R}$. The effect of each rule of $\bar{R}$ is reproduced in t components by introducing in each component one symbol of the right-hand side of the rule together with a query symbol which is used to collect the rest of the symbols in a "communication chain".

For each transition $\bar{r}_k = (r_{k,1}, \ldots, r_{k,n})$ of $\bar{R}$ we construct a transition $\bar{r}'_k = (r'_{k,1}, \ldots, r'_{k,2nt})$. First, we construct a t-tuple of rules for all rules $r_{k,i}$, $1 \leqslant k \leqslant |R|$, $1 \leqslant i \leqslant n$.

If

$$r_{k,i} = Y \to A_1 A_2 \ldots A_s,$$

where $Y \in N$, $A_j \in (N \cup T \cup K)$, $1 \leqslant k \leqslant |R|$, $1 \leqslant i \leqslant n$, $2 \leqslant j \leqslant s$, $s \leqslant t$, then the t-tuple of rules are

$$\begin{aligned}
r'_{k,(i-1)t+1} &= Y \to A'_1 Q_{i,2},\\
r'_{k,(i-1)t+2} &= S_i \to A'_2 Q_{i,3},\\
&\vdots\\
r'_{k,(i-1)t+s} &= S_i \to A'_s,\\
r'_{k,(i-1)t+s+1} &= S_i \to S_i,\\
&\vdots\\
r'_{k,it} &= S_i \to S_i,
\end{aligned}$$

where $A'_j = A_j$ if $A_j \in (N \cup T)$ and $A'_j = Q^a_{i,j}$ if $A_j \in K$.

If

$$r_{k,i} = Y \to A,$$

then $r'_{k,(i-1)t+1} = Y \to A'$, where $A' = A$ if $A \in (N \cup T)$ and $A' = Q^a_{i,1}$ if $A \in K$ and $r'_{k,(i-1)t+j} = S_i \to S_i$, for all $2 \leqslant j \leqslant t$.

If

$$r_{k,i} = Y \to \varepsilon \quad \text{or} \quad r_{k,i} = \sharp,$$

then $r'_{k,(i-1)t+1} = Y \to \varepsilon$ or $r'_{k,(i-1)t+1} = \sharp$, respectively, and $r'_{k,(i-1)t+j} = S_i \to S_i$, for all $2 \leqslant j \leqslant t$.

The next t-tuple of rules is needed to assist the collection of the symbols. In communication mode (a) all query symbols occurring in the same sentential form must be replaced in the same communication step, so the system needs to store sentential forms until they can be sent to their original destination. The rules used by these assistant components are the following:

$$\begin{aligned}
r'_{k,nt+1} &= S^a_{1,1} \to X_{1,1},\\
r'_{k,nt+2} &= S^a_{1,2} \to X_{1,2},\\
&\vdots\\
r'_{k,nt+t} &= S^a_{1,t} \to X_{1,t},\\
&\vdots\\
r'_{k,(2n-1)t+1} &= S^a_{n,1} \to X_{n,1},\\
&\vdots\\
r'_{k,2nt} &= S^a_{n,t} \to X_{n,t},
\end{aligned}$$

where $X_{i,j} = Q_{l,1}$ if $r_{k,i} = Y \to A_1A_2\ldots A_s$ and $A_j = Q_l$, otherwise $X_{i,j} = S^a_{i,j}$, $1 \leqslant j \leqslant t$, $1 \leqslant i \leqslant n$, $1 \leqslant k \leqslant |R|$.

Now if we consider the rule-synchronized PC grammar system Δ with

$$P_{i,j} = \bigcup_{k=1}^{|R|} r'_{k,(i-1)t+j}, \quad 1 \leqslant i \leqslant n, \ 1 \leqslant j \leqslant t,$$

$$P^a_{i,j} = \bigcup_{k=1}^{|R|} r'_{k,(i+n-1)t+j}, \quad 1 \leqslant i \leqslant n, \ 1 \leqslant j \leqslant t,$$

then we shall have $L(\Delta) = L(\Gamma)$. This can be seen by the following considerations: when transition $\bar{r}'_k$ is applied, the effect of applying rules $r_{k,i}$, $1 \leqslant i \leqslant n$, is reproduced at the components, each rule in $\bar{r}'_k$ introducing a symbol of the right-hand side of $r_{k,i}$ and a query to collect the rest of the right-hand side in a "communication chain". If the right-hand side of $r_{k,i}$ has a query symbol Q_l as the jth symbol, then instead of Q_l, $Q^a_{i,j}$ is introduced, querying component $G^a_{i,j}$, where $Q_{l,1}$ is present. This query symbol $Q_{l,1}$ will be replaced by the already collected sentential form which can be passed on to replace $Q^a_{i,j}$. (See Example 2.) Thus, after the communication sequence following the application of $\bar{r}''_k$, the components $G_{j,1}$ of Δ, $1 \leqslant j \leqslant n$, will have the same sentential forms as components G_j of Γ have after applying transition $\bar{r}_k$ and completing the possible following communication sequence. The other components of Δ will have their axioms as the current sentential forms.

By the construction of R' and Δ we can easily verify that there is a one-to-one correspondence between the transitions of Γ and Δ, each transition of Δ simulates a transition of Γ and reversely.

Now starting from Δ, in the same way as described in the proof of Theorem 3.3, we can construct a context-free returning PC grammar system Γ'_R in strong binary form (without rule-synchronization) which generates the same language and uses the same communication protocol as Γ.

We can use similar considerations for communication protocols (b) and (c). Since in these cases the possible occurrences of the different query symbols in the same string can be replaced in different communication steps, independently from each other, the construction of R' and Δ can be simplified. Let

$$\Delta = (N', K', T, G_{1,1}, \ldots, G_{1,t}, \ldots, G_{n,1}, \ldots, G_{n,t}, R'),$$

where

$$R' = \{\bar{r}'_k = (r'_{k,1}, \ldots, r'_{k,nt}) \mid 1 \leqslant k \leqslant |R|\}$$

and let us define $r'_{k,j}$, $1 \leqslant j \leqslant nt$, as follows: if

$$r_{k,i} = Y \to A_1 A_2 \ldots A_s,$$

where $Y \in N$, $A_j \in (N \cup T \cup K)$, $1 \leqslant k \leqslant |R|$, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant s$, $s \leqslant t$, then

$$\begin{aligned}
r'_{k,(i-1)t+1} &= Y \to A'_1 Q_{i,2},\\
r'_{k,(i-1)t+2} &= S_i \to A'_2 Q_{i,3},\\
&\vdots\\
r'_{k,(i-1)t+s} &= S_i \to A'_s,\\
r'_{k,(i-1)t+s+1} &= S_i \to S_i,\\
&\vdots\\
r'_{k,it} &= S_i \to S_i,
\end{aligned}$$

where $A'_j = A_j$ if $A_j \in (N \cup T)$ and $A'_j = Q_{l,1}$ if $A_j = Q_l \in K$.

If

$$r_{k,i} = Y \to A,$$

then $r'_{k,(i-1)t+1} = Y \to A'$, where $A' = A$ if $A \in (N \cup T)$ and $A' = Q_{l,1}$ if $A = Q_l \in K$ and $r'_{k,(i-1)t+j} = S_i \to S_i$, for all $2 \leqslant j \leqslant t$.

If

$$r_{k,i} = Y \to \varepsilon, \quad \text{or} \quad r_{k,i} = \sharp,$$

then $r'_{k,(i-1)t+1} = Y \to \varepsilon$ or $r'_{k,(i-1)t+1} = \sharp$, respectively, and $r'_{k,(i-1)t+j} = S_i \to S_i$, for all $2 \leqslant j \leqslant t$.

With similar arguments to those that we used in the case of protocol (a) we can show that these transitions have the same effect as the transitions in R, that is, the transitions of Δ simulate the transitions of Γ in a one-to-one manner. (We leave the technical details for the reader). As a consequence, we obtain that $L(\Gamma)=L(\Delta)$.

Now, starting from Δ, in the same way as described in the proof of Theorem 3.3, we can construct a returning PC grammar system Γ_R' without rule-synchronization which generates the same language as Δ. This way we obtain a PC grammar system in strong binary form generating the same language and using the same communication protocol as Γ. □

Example 2. Let

$$\bar{r}=(X_1\to Q_3Q_2,\ X_2\to a,\ X_3\to Q_2Q_2)$$

be a transition of a context-free returning PC grammar system Γ.

If we apply $\bar{r}$ to

$$(X_1,X_2,X_3)$$

and we use communication protocol (a), we obtain

$$(Q_3Q_2,a,Q_2Q_2)\Rightarrow_{\text{com}}(Q_3Q_2,S_2,aa)\Rightarrow_{\text{com}}(aaS_2,S_2,S_3),$$

so we may consider the transition

$$(X_1\to Q_3S_2,\ X_2\to a,\ X_3\to Q_2Q_2)$$

instead of $\bar{r}$.

Now if we assume that the longest rule of R has two symbols on its right-hand side, then the construction described in the previous theorem produces the following result:

$$\begin{aligned}\bar{r}' = (&X_1\to Q^a_{1,1}Q_{1,2},S_1\to S_2,\\ &X_2\to a,S_2\to S_2,\\ &X_3\to Q^a_{3,1}Q_{3,2},S_3\to Q^a_{3,2},\\ &S^a_{1,1}\to Q_{3,1},S^a_{1,2}\to S^a_{1,2},\\ &S^a_{2,1}\to S^a_{2,1},S^a_{2,2}\to S^a_{2,2},\\ &S^a_{3,1}\to Q_{2,1},S^a_{3,2}\to Q_{2,1}).\end{aligned}$$

If we apply $\bar{r}'$ to

$$(X_1,S_1,X_2,S_2,X_3,S_3,S^a_{1,1},S^a_{1,2},S^a_{2,1},S^a_{2,2},S^a_{3,1},S^a_{3,2}),$$

we get

$$(Q^a_{1,1}Q_{1,2},S_2,a,S_2,Q^a_{3,1}Q_{3,2},Q^a_{3,2},Q_{3,1},S^a_{1,2},S^a_{2,1},S^a_{2,2},Q_{2,1},Q_{2,1})\Rightarrow_{\text{com}}$$

$$(Q^a_{1,1}Q_{1,2},S_2,S_2,S_2,Q^a_{3,1}Q_{3,2},Q^a_{3,2},Q_{3,1},S^a_{1,2},S^a_{2,1},S^a_{2,2},a,a)\Rightarrow_{\text{com}}$$

$$(Q^a_{1,1}Q_{1,2},S_2,S_2,S_2,Q^a_{3,1}Q_{3,2},a,Q_{3,1},S^a_{1,2},S^a_{2,1},S^a_{2,2},a,S^a_{3,2})\Rightarrow_{\text{com}}$$

$$(Q^a_{1,1}Q_{1,2}, S_2, S_2, S_2, aa, S_3, Q_{3,1}, S^a_{1,2}, S^a_{2,1}, S^a_{2,2}, S^a_{3,1}, S^a_{3,2}) \Rightarrow_{\mathrm{com}}$$

$$(Q^a_{1,1}Q_{1,2}, S_2, S_2, S_2, S_3, S_3, aa, S^a_{1,2}, S^a_{2,1}, S^a_{2,2}, S^a_{3,1}, S^a_{3,2}) \Rightarrow_{\mathrm{com}}$$

$$(aaS_2, S_1, S_2, S_2, S_3, S_3, S^a_{1,1}, S^a_{1,2}, S^a_{2,1}, S^a_{2,2}, S^a_{3,1}, S^a_{3,2}),$$

which produces the same result in $G_{1,1}, G_{2,1}, G_{3,1}$ as $\bar{r}$ in G_1, G_2, G_3.

By the next theorem we show that all languages generated by context-free returning PC grammar systems can be generated by systems with rules not only in strong binary form, but also having at most one query symbol in their right-hand sides. From this result and Lemma 3.2 the equality of language classes generated by context-free returning PC grammar systems using different communication protocols follows. We note that the equality can be directly obtained by using arguments similar to the proof of Theorem 4.3.

Definition 4.2. Let $\Gamma = (N, K, T, G_1, \ldots, G_n)$, $n \geqslant 1$, be a context-free PC grammar system. A rule $X \to \alpha$ of Γ is in *normal form* if one of the following cases holds: $\alpha = AB$, $\alpha = A$, $\alpha = a$, $\alpha = \varepsilon$, or $\alpha = Q_j$, where $A, B \in N, a \in T$, and $Q_j \in K$, $1 \leqslant j \leqslant n$. *A PC grammar system is in normal form*, if all of its rules are in normal form.

Theorem 4.4. *For each context-free returning PC grammar system Γ using communication protocol X, where $X \in \{(a), (b), (c)\}$, we can construct a context-free returning PC grammar system Γ' in normal form that generates the same language and uses the same communication protocol as Γ.*

Proof. Let $\Gamma = (N, K, T, G_1, \ldots, G_n)$ be context-free returning PC grammar system with components $G_i = (N \cup K, T, P_i, S_i)$, $1 \leqslant i \leqslant n$. We show that $L(\Gamma)$ can be generated by a PC grammar system Γ', where Γ' is in normal form and uses the same communication protocol as Γ.

By Theorem 4.3, without the loss of generality, we can assume that Γ is in strong binary form. Let R be the set of all transitions of Γ. To prove the statement, starting from R, we first construct a new set of transitions R' containing only rules in normal form and a rule-synchronized PC grammar system Δ with R' which generates the same language and uses the same communication protocol as Γ. Then, by Theorem 3.3 we construct from Δ a context-free returning PC grammar system Γ'_R without rule-synchronization such that $L(\Gamma'_R) = L(\Gamma)$ holds and the two PC grammar systems use the same communication protocol. Since the rules in R' are in normal form, Γ'_R is also in normal form, so we have a PC grammar system in normal form generating the same language as Γ.

We construct R' by replacing each transition $\bar{r}_k \in R$, $1 \leqslant k \leqslant |R|$ with a set of new transitions that can only be applied in a certain order and if applied in this order they simulate $\bar{r}_k$. The correct order of application will be ensured by a new, $(n+1)$th component.

Let t_k denote the number of different query symbols appearing on the right-hand side of the rules of transition $\bar{r}_k$, $1 \leqslant k \leqslant |R|$, and let

$$\Delta = (N', K, T, G'_1, \ldots, G'_n, G'_{n+1}, R')$$

with components $G'_i = (N' \cup K, T, P'_i, S_i)$, $1 \leqslant i \leqslant n$, and $G'_{n+1} = (N' \cup K, T, P'_{n+1}, \$)$, where

$$N' = N \cup \{[Q_i] \mid 1 \leqslant i \leqslant n\} \cup \{[k, j] \mid 1 \leqslant k \leqslant |R|, 1 \leqslant j \leqslant t_k\} \cup \{\$\}.$$

Now, we show how to replace each transition $\bar{r}_k$ in R with a set $\mathscr{R}'_k$ of one or more new transitions containing rules only in normal form.

1. If $\bar{r}_k = (r_1, \ldots, r_n)$, $r_i \in (P_i \cup \{\sharp\})$, $1 \leqslant i \leqslant n$, and the rules are in normal form, then we replace $\bar{r}_k$ with $\mathscr{R}'_k = \{(r_1, \ldots, r_n, \$ \rightarrow \$)\}$.

2. If $\bar{r}_k = (r_1, \ldots, r_n)$, $r_i \in (P_i \cup \{\sharp\})$, $1 \leqslant i \leqslant n$, and the rules are not in normal form then we do the following. (We note that in this case there is at least one rule in $\bar{r}_k$ which introduces a query symbol.) Without the loss of generality, we can assume that during the communication sequence following $\bar{r}_k$ every queried component communicates its string only in exactly one of the communication steps. (This assumption holds by definition for PC grammar systems using communication protocols (b) or (c). For systems with protocol (a) see the proof of Theorem 4.3.)

Now, let us define an order $Q_{s_1} <_k Q_{s_2} <_k \cdots <_k Q_{s_{t_k}}$, $1 \leqslant s_j \leqslant n$, $1 \leqslant j \leqslant t_k$, $1 \leqslant t_k \leqslant n$, of the different query symbols introduced by the rules of $\bar{r}_k$. (t_k is the number of different query symbols introduced by $\bar{r}_k$.) If transition $\bar{r}_k$ does not introduce a circular query, this order can be arbitrary, otherwise we assign this order in such a way that the query symbols appearing in the sentential forms that are part of a circle in the configuration obtained by the application of $\bar{r}_k$ form a suffix of the ordering sequence. If we replace the query symbols in the sentential forms that are not part of a circle with the strings they refer to according to this order, then we obtain the same string at each of these components as we would get at the end of the communication sequence using the corresponding communication protocol. This holds because each component communicates only in one of the communication steps, therefore all ocurrences of any Q_{s_j}, $1 \leqslant s_j \leqslant n, 1 \leqslant j \leqslant t_k$, will be replaced with the same string, namely the sentential form of G_{s_j}, and thus the different orders lead to the same string at each of those components which are not part of a circle.

Once we determined the order $Q_{s_1} <_k Q_{s_2} <_k \cdots <_k Q_{s_{t_k}}, 1 \leqslant s_j \leqslant n, 1 \leqslant j \leqslant t_k, 1 \leqslant t_k \leqslant n$, we construct sets of transitions $\mathscr{R}_k^{(0)}, \mathscr{R}_k^{(1)}, \ldots, \mathscr{R}_k^{(t_k)}$ to replace $\bar{r}_k$. The first one is

$$\mathscr{R}_k^{(0)} = \{(r_1^{(0)}, \ldots, r_n^{(0)}, \$ \rightarrow [k, 1])\}$$

with

$$r_i^{(0)} = \begin{cases} \sharp & \text{if } r_i = \sharp, \\ X \rightarrow \alpha & \text{if } r_i = X \rightarrow \alpha,\ |\alpha|_K = 0, \\ X \rightarrow [\alpha] & \text{if } r_i = X \rightarrow \alpha,\ |\alpha|_K > 0, \end{cases}$$

$1 \leqslant i \leqslant n$, where $[\alpha]$ denotes the string obtained from α by replacing each occurrence of the query symbol Q_j in α with the new non-terminal $[Q_j]$, $1 \leqslant j \leqslant n$.

This transition rewrites the sentential forms in the same way as $\bar{r}_k$ but introduces new nonterminals instead of the occurrences of query symbols. The rule $\$ \rightarrow [k,1]$ introduces $[k,1]$, a non-terminal indicating that a transition from the set $\mathscr{R}_k^{(1)}$, the second set of the replacing transitions will have to be used next.

The transitions contained by the replacing transition sets will change the new non-terminals with brackets for query symbols in the previously defined order. The transitions in $\mathscr{R}_k^{(j)}$, $1 \leqslant j \leqslant t_k$, rewrite each occurrence of $[Q_{s_j}]$ to Q_{s_j}, and leave the rest of the sentential forms unchanged. The correct order of applying transitions from these sets is ensured by the $(n+1)$th component.

Now we show how to construct the set of transitions $\mathscr{R}_k^{(j)}$, $1 \leqslant j \leqslant t_k$. First, we define rule sets $P_i^{(j)}$, $1 \leqslant i \leqslant (n+1)$, $1 \leqslant j \leqslant t_k$ in the following way: Let

$$P_i^{(j)} = \begin{cases} \{[Q_{s_j}] \rightarrow Q_{s_j}\} & \text{if } r_i = X \rightarrow \alpha,\ \alpha = \alpha_1 Q_{s_j} \text{ or } \alpha = Q_{s_j}\alpha_2, \\ & \text{where } Q_{s_j} \text{ is the } j\text{th} \\ & \text{symbol in the order} \\ & Q_{s_1} <_k \cdots <_k Q_{s_{t_k}} \\ \{\sharp\} \cup \{Y \rightarrow Y \mid Y \in N'\} & \text{if } r_i = X \rightarrow \alpha, \text{ where} \\ & |\alpha|_{\{Q_{s_j}\}} = 0, \text{ or } r_i = \sharp \end{cases}$$

for $1 \leqslant i \leqslant n$ and

$$P_{n+1}^{(j)} = \begin{cases} \{[k,j] \rightarrow [k,j+1]\} & \text{if } 1 \leqslant j \leqslant t_k - 1 \\ \{[k,t_k] \rightarrow \$\} \end{cases}$$

$1 \leqslant j \leqslant t_k$.

The sets of replacing transitions are

$$\mathscr{R}_k^{(j)} = \{(r_1^{(j)}, \ldots, r_{n+1}^{(j)}) \mid r_i^{(j)} \in P_i^{(j)},\ 1 \leqslant i \leqslant n+1\}$$

for $1 \leqslant j \leqslant t_k$, and we have

$$\mathscr{R}'_k = \bigcup_{j=0}^{t_k} \mathscr{R}_k^{(j)}.$$

These transitions reproduce the effect of the application of $\bar{r}_k$ in the following way: the transition in $\mathscr{R}_k^{(0)}$ rewrites the sentential forms and introduces the bracketed query symbols. Now a series of transitions follow, one from each set $\mathscr{R}_k^{(j)}$, $1 \leqslant j \leqslant t_k$, where t_k is the number of different query symbols introduced by $\bar{r}_k$. These sets contain transitions that replace each non-terminal $[Q_{s_j}]$ marking the jth element of the sequence $Q_{s_1} <_k Q_{s_2} <_k \cdots <_k Q_{s_{t_k}}$, with the corresponding query symbol. Since our aim is to leave the rest of the sentential forms unchanged, we consider all possible combinations of the rules $X \rightarrow X$, $X \in N'$ and $\sharp$, obtaining the whole set of transitions $\mathscr{R}_k^{(j)}$. One or more of the elements surely has the desired effect, while those which do not have this effect cannot be used.

The $(n+1)$th component makes sure that the transitions are applied in a correct order, that is, we use $\mathscr{R}_k^{(j)}$ after $\mathscr{R}_k^{(j-1)}$ and before $\mathscr{R}_k^{(j+1)}$. This is done with the aid of the non-terminals $[k,j]$, $1 \leqslant j \leqslant t_k$. To be able to apply a transition from $\mathscr{R}_k^{(j)}$, we have to have $[k,j]$ as sentential form of the $(n+1)$th component, and all transitions in $\mathscr{R}_k^{(j)}$ change it to $[k,j+1]$ enabling a transition from $\mathscr{R}_k^{(j+1)}$ to be applied next. Then using $\mathscr{R}_k^{(t_k)}$, this $(n+1)$th sentential form is changed back to \$.

From the construction of the set $\mathscr{R}'_k$ we can see that its transitions either reproduce the effect of $\bar{r}_k$ or they cannot be used on the sentential forms. If $\bar{r}_k$ is successfully simulated, then the first n sentential forms in the configuration obtained after the application of the elements of $\mathscr{R}'_k$ in the correct order and completing the possible following communication sequence will coincide with the sentential forms obtained after the application of $\bar{r}_k$ and the possible following communication sequence, respectively.

Thus, if we consider Δ and

$$R' = \bigcup_{k=1}^{|R|} \mathscr{R}'_k,$$

we obtain a rule-synchronized PC grammar system with set of transitions R' such that $L(\Delta) = L(\Gamma)$. Moreover, the two PC grammar systems use the same communication protocol. Now, as described in the proof of Theorem 3.3, starting from Δ we can construct a context-free returning PC grammar system Γ'_R in normal form generating the same language and using the same protocol as Γ. □

Example 3. We illustrate the construction of the proof of the above theorem by an example.

Let

$$\bar{r}_k = (X_1 \to AB,\ X_2 \to Q_1Q_3,\ X_3 \to Q_1A)$$

be a transition of a context-free returning PC grammar system Γ. Let us consider configuration

$$(X_1, X_2, X_3)$$

and apply the above transition using communication protocol (a). Then we obtain

$$(AB, Q_1Q_3, Q_1A) \Rightarrow_{\text{com}}$$

$$(S_1, Q_1Q_3, ABA) \Rightarrow_{\text{com}}$$

$$(S_1, S_1ABA, S_3),$$

so we may consider transition $(X_1 \to AB,\ X_2 \to S_1Q_3,\ X_3 \to Q_1A)$ instead of $\bar{r}_k$. Next, we determine the order in which the query symbols will be replaced. We have $t_k = 2$ and the order will be $Q_1 <_k Q_3$. Now we construct the sets of transitions replacing $\bar{r}$. These are:

$$\mathscr{R}^{(0)} = \{(X_1 \to AB,\ X_2 \to S_1[Q_3],\ X_3 \to [Q_1]A, \$ \to [k,1])\},$$

$\mathscr{R}^{(1)}$ has an element

$$(A \to A,\ [Q_3] \to [Q_3],\ [Q_1] \to Q_1, [k,1] \to [k,2]),$$

and $\mathscr{R}^{(2)}$ has an element

$$(S_1 \to S_1,\ [Q_3] \to Q_3,\ A \to A, [k,2] \to \$).$$

If we apply these transitions to $(X_1, X_2, X_3, \$)$, we obtain the following:

$$(AB, S_1[Q_3], [Q_1]A, [k,1]) \Rightarrow$$

$$(AB, S_1[Q_3], Q_1A, [k,2]) \Rightarrow_{\text{com}}$$

$$(S_1, S_1[Q_3], ABA, [k,2]) \Rightarrow$$

$$(S_1, S_1Q_3, ABA, \$) \Rightarrow_{\text{com}}$$

$$(S_1, S_1ABA, S_3, \$).$$

Thus, the first three sentential forms are the same that can be found in the configuration we obtain at the end of the communication sequence following the application of $\bar{r}_k$.

In the previous theorem we proved that each context-free returning PC grammar system using any of the three communication protocols can be transformed to a system in normal form, that is, having rules with at most two non-terminals or one query symbol on their right-hand side. If we combine this result with Lemma 3.2, we have the following corollary.

Corollary 4.5.

$$\mathscr{L}(PC_*CF,(a)) = \mathscr{L}(PC_*CF,(b)) = \mathscr{L}(PC_*CF,(c)).$$

Finally, we add that motivated by these results similar investigations were conducted concerning parallel communicating Lindenmayer systems. [11] contains normal forms for extended PC Lindenmayer systems and [10] shows that the class of languages generated using any of the three communication protocols are also the same in the case of extended PC Lindenmayer systems.

References

[1] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun, Grammar Systems. A Grammatical Approach to Distribution and Cooperation, Gordon and Breach, Yverdon, 1994.

[2] E. Csuhaj-Varjú, Gy. Vaszil, On the computational completeness of contex-free parallel communicating grammar systems, Theoret. Comput. Sci. 215 (1–2) (1999) 349–358.

[3] J. Dassow, Gh. Păun, G. Rozenberg, Grammar Systems, in: G. Rozenberg and A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 2, Springer, Berlin, 1997, pp. 155–213 (Chapter 4).

[4] N. Mandache, On the computational power of context-free PC grammer systems, Theoret. Comput. Sci. 237 (2000) 135–148.

[5] Gh. Păun, On the synchronization in parallel communicating grammar systems, Acta Inform. 30 (1993) 351–367.
[6] Gh. Păun, Parallel communicating grammar systems: recent results, open problems, Acta Cybernet. 12 (1996) 381–395.
[7] Gh. Păun, L. Sântean, Parallel communicating grammar systems: the regular case, Ann. Univ. Bucharest Ser. Matem.-Inform. 38 (2) (1989) 55–63.
[8] A. Salomaa, Formal Languages, Academic Press, New York, 1973.
[9] F.L. Ţiplea, C. Ene, C.M. Ionescu, O. Procopiuc, Some decision problems for parallel communicating grammar systems, Theoret. Comput. Sci. 134 (1994) 365–385.
[10] Gy. Vaszil, Communication in parallel communicating Lindenmayer systems, in: A. Kelemenová (Ed.), Proc. MFCS'98 Satellite Workshop on Grammar Systems, Silesian University at Opava, Opava, 1998, pp. 139–147.
[11] Gy. Vaszil, On parallel communicating Lindenmayer systems, in: Gh. Păun, A. Salomaa (Eds.), Grammatical Models of Multi-Agent Systems, Gordon and Breach Science Publishers, Amsterdam, 1999, pp. 99–112.
[12] Gy. Vaszil, Various communications in parallel communicating grammar systems, Acta Cybernet. 13 (1997) 173–196.