CHAPTER 2

PARALLEL COMMUNICATING GRAMMAR SYSTEMS

Bringing PC Grammar Systems Closer to Hoare's CSP's[1]

Lila KARI, Hanan LUTFIYYA

Department of Computer Science, University of Western Ontario
N6A 5B7 London, Ontario, Canada
lkari/hanan@csd.uwo.ca

Carlos MARTÍN-VIDE

Research Group on Mathematical Linguistics and Language Engineering
(GRLMC)
Rovira i Virgili University
Pl. Imperial Tárraco 1, 43005 Tarragona, Spain
cmv@astor.urv.es

Gheorghe PĂUN

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
gpaun@imar.ro

Abstract. We consider here Parallel Communicating (PC) grammar systems with features inspired from Hoare's model of Communicating Sequential Processes (CSP). Specifically, we consider (1) *nondeterministic queries* (in two variants: using sets of query symbols, such that one symbol from each set has to be answered, and querying by nonterminals, such that the queried grammars are those having as axioms the specified symbol), (2) *flags* indicating the fact that a component of the system is ready to communicate, and (3) *patterns* of the communicated strings. The generative power of the obtained PC grammar systems is investigated, in comparison with the power of usual classes of PC grammar systems and with classes of grammars in the Chomsky hierarchy.

[1]Research supported by Grants OGP0007877 and S365A2 of the Natural Sciences and Engineering Research Council of Canada, and University of Western Ontario start-up grant

1 Introduction

The PC grammar systems were introduced in [13] as a grammatical model of parallel computing. In short, such a system consists of a set of usual (context-free) Chomsky grammars, which start from separate axioms, work synchronously (each component grammar uses one rewriting rule in each time unit, rewriting its own sentential form), and communicate by request. That is, special *query symbols* are provided; when they appear in a sentential form of a grammar, a *communication* step is done: the sentential form of the component identified by a query symbol replaces the occurrence(s) of that query symbol in the sentential form of the component which has introduced it. The communication has priority over rewriting.

There are several features of the communication in PC grammar systems which are significantly different from the way the communication takes place in CSP's of [6]. Three of them will be considered here:

First, the communication in a PC grammar system is *deterministic*, in the following sense: a query symbol is of the form Q_j, it precisely identifies the component from which a message is required.

Second, the communication has priority over rewriting, when a query symbol is introduced, it has to be immediately satisfied, irrespective of the "state" of the queried component.

Third, any sentential form may be communicated, there is no filtering condition in the receiving component.

Ready-to-communicate and filtering features are necessary when modelling distributed applications like natural language syntax, see [7].

All these three "drawbacks" (from the point of view of CSP's) can be easily "corrected". We shall do it in this paper, after briefly introducing the PC grammar systems and the CSP's frameworks. This again shows the great versatility of the grammar system paradigm, in its various instances (see also [1]).

For instance, instead of introducing only one query symbol Q_i, we can introduce a set $\{Q_{i_1}, \ldots, Q_{i_k}\}$, with the meaning: "(any) one of the components $i_1, \ldots, i_k$ has to communicate". A natural variant is not to use query symbols, but to use nonterminals for starting a communication step: if a nonterminal A cannot be rewritten by a component i, but A is the start symbol (axiom) of other components $i_1, \ldots, i_k$, then any one of these latter components has to send its sentential form to component i, replacing A (this is similar to replacing A with a set $\{Q_{i_1}, \ldots, Q_{i_k}\}$.

Then, we can also introduce "ready-to-communicate" symbols, as in CSP's, pointing to components of the system which can be targets of communication steps. A communication from a component j to a component i is performed only if i introduces the query symbol Q_j and j introduces the "ready symbol" R_i.

Finally, a filtering of messages can be implemented by considering regular sets associated to query symbols: a sentential form is communicated (and accepted by the receiver) only if it is an element of the regular language associated to the currently used query symbol. (A regular language is a simple enough filter, for instance, because the membership with respect to such a language can be decided in real time.)

Although these modifications of PC grammar systems are very similar to the corresponding features of CSP's, we still remain here in the framework of formal language theory: we are mainly interested in the generative power of the obtained systems (the language of the system is the language of a designated component, the "master") and not, say, in the sequence of communication steps and its properties (deadlocks, circularities, waiting intervals, etc). We hope to return to such topics in a forthcoming paper.

2 PC Grammar Systems

We refer to [15] for elements of formal language theory we use, and we specify only some general notations and notions.

For an alphabet V we denote by V^* the free monoid generated by V under the operation of concatenation; the empty word is denoted by λ and $V^* - \{\lambda\}$ (the set of all non-empty words) is denoted by V^+. For $x \in V^*$, $|x|$ is the length of x, $|x|_a$ is the number of occurrences in x of the symbol $a \in V$, and $alph(x)$ is the set of symbols appearing in x. The families of regular, context-free, context-sensitive, and recursively enumerable languages are denoted by *REG, CF, CS, RE*, respectively.

A *PC grammar system* of degree $n, n \geq 1$ ([13], [1]), is a construct

$$\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n)),$$

where N, K, T are pairwise disjoint alphabets, with $K = \{Q_1, \ldots, Q_n\}$, $S_i \in N$, and P_i are finite sets of rewriting rules over $N \cup T \cup K, 1 \leq i \leq n$; the elements of N are *nonterminal* symbols, those of T are *terminals*; the elements of K are called *query symbols*; the pairs (S_i, P_i) are the *components* of the system. Note that, by their indices, the query symbols are associated with the components. When discussing the type of the components in Chomsky hierarchy, the query symbols are interpreted as nonterminals.

For $(x_1, \ldots, x_n), (y_1, \ldots, y_n)$, with $x_i, y_i \in (N \cup T \cup K)^*, 1 \leq i \leq n$ (we call such an n-tuple a *configuration*), and $x_1 \notin T^*$, we write $(x_1, \ldots, x_n) \Longrightarrow_r (y_1, \ldots, y_n)$ if one of the following two cases holds:

(i) $|x_i|_K = 0$ for all $1 \leq i \leq n$; then $x_i \Longrightarrow_{P_i} y_i$ or $x_i = y_i \in T^*, 1 \leq i \leq n$;

(ii) there is $i, 1 \leq i \leq n$, such that $|x_i|_K > 0$; we write such a string x_i as

$$x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1},$$

for $t \geq 1, z_i \in (N \cup T)^*, 1 \leq i \leq t+1$; if $|x_{i_j}|_K = 0$ for all $1 \leq j \leq t$, then

$$y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1},$$

[and $y_{i_j} = S_{i_j}, 1 \leq j \leq t$]; otherwise $y_i = x_i$. For all unspecified i we have $y_i = x_i$.

Point (i) defines a *rewriting* step (componentwise, synchronously, using one rule in all components whose current strings are not terminal), (ii) defines a *communication* step: the query symbols Q_{i_j} introduced in some x_i are replaced by the

associated strings x_{i_j}, providing that these strings do not contain further query symbols. The communication has priority over rewriting (a rewriting step is allowed only when no query symbol appears in the current configuration). The work of the system is blocked when circular queries appear, as well as when no query symbol is present but point (i) is not realized because a component cannot rewrite its sentential form, although it is a nonterminal string. Note that the derivation stops when the first component produces a terminal string.

The above considered relation $\Longrightarrow_r$ is said to be performed in the *returning* mode: after communicating, a component resumes working from its axiom. If the brackets, [and $y_{i_j} = S_{i_j}, 1 \leq i \leq t$], are removed, then we obtain the *non-returning* mode of derivation: after communicating, a component continues the processing of the current string. We denote by $\Longrightarrow_{nr}$ the obtained relation.

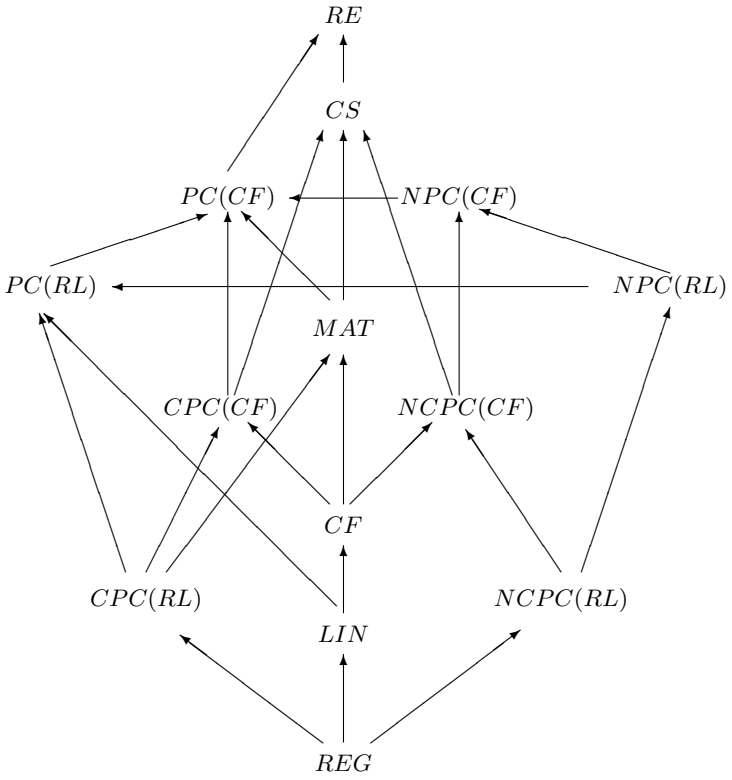

Figure 1: Hierarchies of PC families

The language generated by Γ is the language generated by its first component (G_1 above), when starting from $(S_1, \ldots, S_n)$, that is

$$L_f(\Gamma) = \{w \in T^* \mid (S_1, \ldots, S_n) \Longrightarrow_f^* (w, \alpha_2, \ldots, \alpha_n), \text{ for } \alpha_i \in (N \cup T \cup K)^*, 2 \leq i \leq n\}, \ f \in \{r, nr\}.$$

(No attention is paid to strings in the components $2, \ldots, n$ in the last configuration

of a derivation; moreover, it is supposed that the work of Γ stops when a terminal string is obtained by the first component.)

Two basic classes of PC grammar systems can be distinguished: *centralized* (only G_1, the *master* of the system, is allowed to introduce query symbols), and *non-centralized* (no restriction is imposed on the introduction of query symbols). Therefore, we get four basic families of languages: we denote by $PC(X)$ the family of languages generated in the returning mode by non-centralized PC grammar systems with rules of type X (and of arbitrary degree); when centralized systems are used, we add the symbol C, when the non-returning mode of derivation is used, we add the symbol N, thus obtaining the families $CPC(X), NPC(X), NCPC(X)$. In what concerns X, we can consider λ-free right-linear (RL) or context-free (CF) rules, or arbitrary right-linear (RL^λ) or context-free (CF^λ) rules. If the language we consider contains the empty string, then a rule $S \to \lambda$ is allowed in the master grammar. (Note that, because the derivation stops in that moment, λ cannot be communicated to another component.)

The diagram in Figure 1 indicates the relations between the eight basic families of languages defined above, as well as their relationships with families in the Chomsky hierarchy (MAT denotes the family of languages generated by matrix grammars with λ-free context-free rules and without appearance checking and LIN is the family of linear languages). The arrows indicate inclusions, not necessarily proper; the families not connected by a path are not necessarily incomparable. The families $CPC(RL), NCPC(RL)$ are incomparable. Proofs of these relations can be found in [1], [4], [5], [8], [9].

Let us consider two **examples**. For the system

$$\begin{aligned}\Gamma_1 \;=\; & (\{S_1,S_2,S_3\},K,\{a,b,c\},(S_1,P_1),(S_2,P_2),(S_3,P_3)),\\ & P_1=\{S_1\to abc, S_1\to a^2b^2c^2, S_1\to aS_1,\\ & \qquad S_1\to a^3Q_2, S_2\to b^2Q_3, S_3\to c\},\\ & P_2=\{S_2\to bS_2\},\\ & P_3=\{S_3\to cS_3\},\end{aligned}$$

we obtain

$$L_r(\Gamma)=L_{nr}(\Gamma)=\{a^nb^nc^n \mid n\geq 1\},$$

hence this language belongs to both $CPC(RL)$ and $NCPC(RL)$.

Here is a derivation in Γ_1:

$$\begin{aligned}(S_1,S_2,S_3) &\Longrightarrow_f (aS_1,bS_2,cS_3)\Longrightarrow_f \ldots\Longrightarrow_f (a^nS_1,b^nS_2,c^nS_3),\\ &\Longrightarrow_f (a^{n+3}Q_2,b^{n+1}S_2,c^{n+1}S_3)\Longrightarrow_f (a^{n+3}b^{n+1}S_2,y_2,c^{n+1}S_3)\\ &\Longrightarrow_f (a^{n+3}b^{n+3}Q_3,y_2',c^{n+2}S_3)\Longrightarrow_f (a^{n+3}b^{n+3}c^{n+2}S_3,y_2',y_3)\\ &\Longrightarrow_f (a^{n+3}b^{n+3}c^{n+3},y_2'',y_3'),\ n\geq 0,\end{aligned}$$

for $f\in\{r,nr\}$; in the returning case we have $y_2=S_2, y_2'=bS_2, y_2''=b^2S_2$, $y_3=S_3, y_3'=cS_3$, in the non-returning case $y_2=b^{n+1}S_2, y_2'=b^{n+2}S_2, y_2''=b^{n+3}S_2$, $y_3=c^{n+2}S_3, y_3'=c^{n+3}S_3$. Because the second and the third components communicate only once to the first component, there is no difference between

the language generated in the returning mode and the language generated in the non-returning mode. This is not the case for the following system.

$$\begin{aligned}\Gamma_2 \;=\; & (\{S_1, S_2\}, \{a\}, K, (S_1, P_1), (S_2, P_2)),\\ & P_1 = \{S_1 \to aQ_2, S_2 \to aQ_2, S_2 \to a\},\\ & P_2 = \{S_2 \to aS_2\}.\end{aligned}$$

The reader might check that we obtain

$$\begin{aligned}L_r(\Gamma_2) &= \{a^{2n+1} \mid n \geq 1\},\\ L_{nr}(\Gamma_2) &= \{a^{\frac{(m+1)(m+2)}{2}} \mid m \geq 1\}.\end{aligned}$$

If the synchronization feature is removed (this amounts to suppose that in each component of a PC grammar system there is a rule of the form $A \to A$ for each nonterminal A), then we speak about *unsynchronized* PC grammar systems. The families of languages obtained in this way are denoted by adding the letter U in front of the notations above; thus, we obtain $UPC(CF), UNCPC(CF)$, etc.

In all sections below we consider only PC grammar systems with context-free rules (λ-free or arbitrary), but many of the results hold true also for right-linear rules; we leave to the reader the task of particularizing the proofs for the right-linear case, when this is possible.

3 Communicating Sequential Processes

This section provides a brief description of the syntax and meaning of CSP commands. Full details of CSP are contained in [6].

Communicating Sequential Processes (CSP) was proposed as a preliminary solution to the problem of defining a synchronous message-based language.

The basic idea of CSP is that multiple concurrent (parallel) processes can synchronize with each other most easily by synchronizing their I/O. A CSP program consists of a static collection of processes. The basic command of CSP is $[\rho_1 || \ldots || \rho_n]$ expressing concurrent execution of sequential processes $\rho_1, \ldots, \rho_n$. Each individual process ρ_i has a distinct address space and consists of statements S_i. We can also express parallelism between program statements as well as between processes.

Coordination between processes is implemented by message exchange between pairs of processes. It involves the *synchronized execution* of *send* (output) and *receive* (input) operations by both processes. The proposed way to do communication is to allow communication to occur only when: process ρ_i states that it is ready to output (send) to process ρ_j specifically, and process ρ_j is ready to input (receive) from process ρ_i specifically. The send and receive operations in processes ρ_j and ρ_i take the following forms: $\rho_i!y$ and $\rho_j?x$, respectively.

Input command $\rho_j?x$ expresses a request to ρ_j to assign a value to the (local) variable x of ρ_i. Output command $\rho_i!y$ expresses a request to ρ_i to receive a value from ρ_j. Execution of $\rho_j?x$ and $\rho_i!y$ is synchronized and results in assigning the value of y to x. $\rho_j?x$ and $\rho_i!y$ are said to be a *matching pair* of communication

statements. If one of these happens in a process without the other, the process is blocked until the other process is ready.

The *alteration command* allows for a path to be non-deterministically chosen from a set of paths. The *repetition rule* allows for repeated non-deterministic choosing of a path from a set of paths. The alteration and repetition commands are as follows:

$$[b_1; c_1 \rightarrow S_1$$
$$\ldots\ldots$$
$$b_n; c_n \rightarrow S_n]$$

$$*[b_1; c_1 \rightarrow S_1$$
$$\ldots\ldots$$
$$b_n; c_n \rightarrow S_n]$$

Alteration and repetition are formed from sets of guarded commands. A guarded command $b; c \rightarrow S$ consists of a guard $b; c$ and a command S. In the guard, b is a boolean expression and c is either skip or one of the communication primitives. The symbol ";" is used as a delimiter for separating different program statements. If b is false, the guard is failed. If b is true and $c = skip$, the guard is ready. If b is true and c is one of the communication primitives, then the guard is prepared to communicate with the process named in the communication primitive. It is ready when the other process is prepared to communicate and blocked at other times.

Execution of an alteration command selects a guarded command with a ready guard and executes the sequence $c; S$. If c is skip, execution is independent of other processes. If c is a communication command, then a matching communication command must be executed simultaneously. When some guards are blocked and none are ready, the process is blocked and must wait. If all guards are failed, the process aborts.

Execution of the repetitive command is the same except that, whereas execution of alternation selects one guarded command and is completed, for repetition the selection is repeated until all guards are failed, at which time execution of the repetition is completed.

In the case that more than one guard evaluates to true, then one is arbitrarily chosen to execute while the others are ignored.

Let us now examine a simple banking application that creates, deletes, and performs rudimentary transactions on bank accounts. This is provided by a *server* process that we call ρ_1. ρ_1 allows other processes (called clients) to contact it and request create, delete and other simple transaction operations on bank accounts. Let us assume that the clients are called $\rho_2, \rho_3, \rho_4, \rho_5$.

The CSP representation of ρ_1 follows this pattern:

$$*[true; \rho_2?r \rightarrow \ ...process\ request...; \rho_2!a$$
$$true; \rho_3?r \rightarrow \ ...process\ request...; \rho_3!a$$
$$true; \rho_4?r \rightarrow \ ...process\ request...; \rho_4!a$$
$$true; \rho_5?r \rightarrow \ ...process\ request...; \rho_5!a]$$

This means that ρ_1 repetively receives requests from ρ_2, ρ_3, ρ_4 and ρ_5 for either creating, deleting or performing rudimentary bank account transactions. A guard is true if a request has actually been received. If several guards are true then ρ_1 arbitrarily chooses which one to execute. After it has chosen, it processes the request and the sends a response back.

In ρ_2, ρ_3, ρ_4, ρ_5, the CSP representation follows this pattern;

$$\ldots\ \rho_1!r; \rho_1?a$$

When process ρ_i $(2 \leq i \leq 5)$ sends a request to ρ_1 then it blocks itself until ρ_1 responds.

It is the non-determinism and matching pair of communication observed in the above presented definitions and example that motivates us to extend the PCGS model.

4 PC Grammar Systems with Set Queries

One possibility of introducing non-determinism in the querying process specific to PC grammar systems is the following one.

A PC grammar system *with set queries* is a construct

$$\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n)),$$

where all components are as in a usual PC grammar system, except that the sets P_i contain either context-free rules over $N \cup T$ or rules of the form

$$A \to z_1 K_1 z_2 \ldots z_r K_r z_{r+1},$$

where $z_i \in (N \cup T)^*, 1 \leq i \leq s+1$, and $K_i \subseteq K, 1 \leq i \leq r$, for $r \geq 1$.

The rewriting steps are defined as usual in a PC grammar system (componentwise, synchronized), but a communication step is defined as follows: for two configurations $(x_1, \ldots, x_n), (y_1, \ldots, y_n)$, with $x_i, y_i \in (N \cup T \cup 2^K)^*, 1 \leq i \leq n$, $x_1 \notin T^*$ (the subsets of K are considered symbols), we write $(x_1, \ldots, x_n) \Longrightarrow_r (y_1, \ldots, y_n)$ if and only if:

1. there is $i, 1 \leq i \leq n$, with x_i containing an occurrence of some $K' \subseteq K$;
2. for x_i as above we have

   $$x_i = z_1 K_1 z_2 \ldots z_s K_s z_{s+1},$$

   with $z_j \in (N \cup T)^*, 1 \leq j \leq s+1$, and $K_j \subseteq K, 1 \leq j \leq s, s \geq 1$; then

   $$y_i = z_1 x_{j_1} z_2 \ldots z_s x_{j_s} z_{s+1},$$

   provided that $Q_{j_t} \in K_t$ and $x_{j_t} \in (N \cup T)^*, 1 \leq t \leq s$;
   moreover, $y_{j_t} = S_{j_t}, 1 \leq t \leq s$;
3. for all i for which y_i is not defined above we have $y_i = x_i$.

Thus, a query symbol from each set K_j has to be choosen and satisfied; the communicated strings should not contain further query symbols (sets of query symbols), otherwise the communication is not accepted. If at least one set K_t cannot be satisfied in these conditions, then the communication does not take place, the string x_i is not modified.

This is a communication step done in the *returning* mode. If, after communicating, a component continues to process the current string (hence in condition 2 above we have $y_{j_t} = x_{j_t}$ instead of $y_{j_t} = S_{j_t}$), then we speak about a communication done in the *non-returning* mode, denoted by $\Longrightarrow_{nr}$.

The language generated by a PC grammar system with query sets Γ in a mode $\alpha \in \{r, nr\}$, is denoted by $L_\alpha(\Gamma)$ and is defined in the usual way, as the language of the master component, the first one in Γ.

We denote by $SPC(CF)$ the family of languages $L_r(\Gamma)$, generated in the returning mode by PC grammar systems with set queries, with arbitrarily many λ-free context-free components. As in the case of usual systems, we add the letter C when only centralized systems are used, and the letter N when we work in the non-returning mode. In this way, we obtain the families $CSPC(CF), NSPC(CF), NCSPC(CF)$.

The usual PC grammar systems are a particular case of PC grammar systems with set queries: the sets of query symbols appearing in rewriting rules are singletons. Therefore, we obtain:

Lemma 1. $XPC(CF) \subseteq XSPC(CF),\ X \in \{-, C, N, NC\}$.

The converse inclusions also holds. The non-determinism of a set of query symbols can be simulated by the intrinsic non-determinism of the set of rewriting rules able to rewrite a given nonterminal symbol. Specifically, each rule of the form

$$A \to z_1 K_1 z_2 \dots z_s K_s z_{s+1}, \qquad (1)$$

with $z_i \in (N \cup T)^*, 1 \leq i \leq s+1$ and $K_i \subseteq K, 1 \leq i \leq s$, can be replaced by the set of rules

$$A \to z_1 Q_{j_1} z_2 \dots z_s Q_{j_s} z_{s+1}, \qquad (2)$$

for all possible $q_{j_i} \in K_i, 1 \leq i \leq s$. Non-deterministically satisfying one query symbol from each set K_i in a rule of type (1) means exactly the using of the corresponding rule of type (2). The type of the system (centralized or not) and the type of the derivation (returning or non-returning) are not changed, that is we have:

Lemma 2. $XSPC(CF) \subseteq XPC(CF),\ X \in \{-, C, N, NC\}$.

Combining these two lemmas, we can write

Theorem 1. $XSPC(CF) = XPC(CF),\ X \in \{-, C, N, NC\}$.

The generative power of the new systems is equal to the power of the corresponding classes of "old" systems. Identical results are obtained when λ-rules are allowed.

5 PC Grammar Systems with Queries by Nonterminals

Another natural way of specifying the queries in a non-deterministic manner is to use the nonterminal symbols for starting communication steps.

A PC grammar system *with queries by nonterminals* is a construct

$$\Gamma = (N, T, (N_1, A_1, P_1), \ldots, (N_n, A_n, P_n)),$$

where N is a nonterminal alphabet, T is a terminal alphabet, $N_i \subseteq N$, $A_i \in N_i$, and P_i are finite sets of context-free rules over $N \cup T$ of the form $A \to w$, $A \in N_i$, $w \in (N \cup T)^*$, $1 \leq i \leq n$.

(The components of the system have different nonterminal alphabets, subsets of the nonterminal alphabet of the system.)

The derivation starts from the configuration $(A_1, \ldots, A_n)$ and proceeds by componentwise derivation steps defined as in a usual PC grammar system and communication steps, defined as follows: for two configurations $(x_1, \ldots, x_n), (y_1, \ldots, y_n)$, $x_i, y_i \in (N \cup T)^*, 1 \leq i \leq n$, $x_1 \notin T^*$, we write $(x_1, \ldots, x_n) \Longrightarrow_r (y_1, \ldots, y_n)$ if and only if:

1. there is $i, 1 \leq i \leq n$, such that x_i contains symbols in $N - N_i$;
2. for x_i of the form above we have

   $$x_i = z_1 B_1 z_2 \ldots z_s B_s z_{s+1},$$

   for $z_j \in (N_j \cup T)^*, 1 \leq j \leq s+1$, $B_j \in N - N_j, 1 \leq j \leq s, s \geq 1$; then

   $$y_i = z_1 x_{j_1} z_2 \ldots z_s x_{j_s} z_{s+1},$$

   for $B_t = A_{j_t}$ and $x_{j_t} \in (N_i \cup T)^*, 1 \leq t \leq s$;
   moreover, $y_{j_t} = A_{j_t}, 1 \leq t \leq s$;
3. for all i for which y_i is not defined above we have $y_i = x_i$.

In plain words, when a component i of the system introduces a nonterminal A which cannot be rewritten by its rules, then a communication step should be done; one component of the system which has A as its start symbol has to communicate its sentential form to component i, providing that this sentential form does not contain nonterminal symbols which cannot be rewritten by component i.

The generated language is defined in the usual way (for both returning and non-returning communications). The obtained families are denoted by $XMPC(CF), X \in \{-, C, N, NC\}$ (thus, $XMPC(CF)$ corresponds to $XSPC(CF)$ in the previous section, S in front of PC is replaced by M).

Also in this case the generative power of PC grammar systems of any type is not modified.

Lemma 3. $XPC(CF) \subseteq XMPC(CF),\ X \in \{-, C, N, NC\}$.

Proof. Consider a PC grammar system $\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n))$ and construct the PC grammar system with queries by nonterminals

$$\Gamma' = (N \cup K, T, (N \cup \{Q_1\}, Q_1, P_1'), \ldots, (N \cup \{Q_n\}, Q_n, P_n')),$$

with

$$P_i' = P_i \cup \{Q_i \to x \mid S_i \to x \in P_i\},\ 1 \leq i \leq n.$$

The query symbols of Γ are usual nonterminals for Γ', they precisely identify the components of Γ', and play the same role in Γ as in Γ': when a symbol Q_j is introduced by a component i (of Γ or of Γ'), then a communication from component j to component i must be performed – providing that the sentential form of component j contains no further query symbols. (If Q_i appears in this sentential form, then a circular query is met and the system is blocked both in the case of Γ and of Γ'.)

Consequently, $L_\alpha(\Gamma) = L_\alpha(\Gamma')$, $\alpha \in \{r, nr\}$. Because Γ' is of the same type as Γ, the proof is complete. □

Lemma 4. $XMPC(CF) \subseteq XSPC(CF),\ X \in \{-, C, N, NC\}$.

Proof. Consider a PC grammar system with queries by nonterminals $\Gamma = (N, T, (N_1, A_1, P_1), \ldots, (N_n, A_n, P_n))$ and construct the PC grammar system with query sets

$$\Gamma' = (N, K, T, (A_1, P_1'), \ldots, (A_n, P_n')),$$

where

$$\begin{aligned}
K &= \{Q_1, \ldots, Q_n\},\\
P_i' &= \{A \to x \mid A \to x \in P_i, x \in (N_i \cup T)^*\}\\
&\cup \{A \to z_1 K_1 z_2 \ldots z_s K_s z_{s+1} \mid A \to z_1 A_{j_1} z_2 \ldots z_s A_{j_s} z_{s+1} \in P_i,\\
&\quad z_r \in (N_i \cup T)^*, 1 \leq r \leq s+1, A_{j_r} \in N - N_i, 1 \leq r \leq s, s \geq 1,\\
&\quad \text{and } K_r = \{Q_t \mid A_t = A_{j_r}\}, 1 \leq r \leq s\},\ 1 \leq i \leq n.
\end{aligned}$$

The sets K_r identify the components of Γ' (hence those of Γ, too) which have as a starting symbol the nonterminal A_{j_r}. Satisfying a query symbol in K_r is the same as replacing A_{j_r} by the sentential form of some component of Γ having A_{j_r} as a starting symbol. Consequently, $L_\alpha(\Gamma) = L_\alpha(\Gamma'), \alpha \in \{r, nr\}$. □

Combining these two latter lemmas with the equalities in Theorem 1, we obtain the following equalities:

Theorem 2. $XPC(CF) = XSPC(CF) = XMPC(CF),\ X \in \{-, C, N, NC\}$.

The two modifications of the way of starting a communication in a PC grammar system considered above are equal in power; moreover, the modified systems have the same power as the usual PC grammar systems. This is true also in the case of using λ-rules.

6 PC Grammar Systems with Flags

We now consider also a way of indicating that a component of a PC grammar system is ready to communicate to another component. To this aim we shall consider *flags* associated to the symbols of a PC grammar system, namely sequences of the form $i?, i!$ similar to those in CPS's: the symbol $i?$ will be equivalent to a query symbol Q_i and $i!$ will indicate that the component which introduces $i!$ is ready to communicate to component i.

A PC grammar system *with flags* is a construct

$$\Gamma = (N, T, (S_1, P_1), \ldots, (S_n, P_n)),$$

where N is a nonterminal alphabet, T is a terminal alphabet, $S_i \in N$, and P_i are finite sets of context-free rules of the form $A \to x$, with $A \in N, x \in (N \cup T)^* \cup (N \cup T \cup N_?)^* \cup (N \cup T \cup N_!)^*$, $1 \leq i \leq n$, where

$$N_? = \{[i?B] \mid B \in N\},$$
$$N_! = \{[i!\alpha] \mid 1 \leq i \leq n, \alpha \in N \cup T\}.$$

The elements of $N_?, N_!$ are called flagged symbols, $i?, i!$ are called flags. Note that only the nonterminal symbols can be flagged with $i?$, but both the nonterminal and the terminal symbols can be flagged with $i!$. (In fact, the nonterminal symbol in $[i?B]$ is useless, it is written mainly for uniformity with the notation used for CPS's: the whole $[i?B]$ is replaced by a communicated string, irrespective which is the symbol B.)

The work of such a system is defined according to the following rules (we prefer here a non-formal formulation):

- we start from the configuration $(S_1, \ldots, S_n)$;
- when no flag is present in a configuration, then each component has to use a rule for rewriting its sentential form, unless this sentential form is a string in T^*;
- if in some components of a configuration we have flagged symbols, but there are no matching pairs of the form $[i?B], [j!\alpha]$, with $[i?B]$ appearing in the jth component and $[j!\alpha]$ appearing in the ith component, then all components of the system use a rule for rewriting their sentential forms, excepting those components whose sentential forms either are terminal or contain flagged symbols;
- when at least a pair of matching flags appear (that is a component j has introduced $[i?B]$ and the component i has introduced $[j!\alpha]$), then a communication step is performed: the sentential form of component i is transmitted to component j, where it replaces the flagged occurrence of B; the flags $i?$ from component j are removed and similarly with the flags $j!$ appearing in component i; all other flags, not involved in this communication, remain unchanged.

Therefore, a communication is done only when two matching flags appear, linking the two communicating components. A component introducing a flag does not rewrite its sentential form, but waits for the partner components to introduce matching flags.

We have formulated no condition on the communicated string. Two variants can be considered: any string can be communicated or only strings which do not contain flags different from that in the matching pair can be communicated. In the first case we have to specify how the additional flags are processed. Again two possibilities exist: either they are removed after communication, or they remain and have to be satisfied by subsequent communications.) We adopt here the most complex variant: strings containing flags can be communicated, and the flags are removed only by communications which satisfy them.

After communicating, a component can return to its axiom or it can continue to rewrite the current sentential form (providing that it has no remaining flag). The centralized systems are defined with respect to flags of the form $i?$: only the master component can introduce such flags. Of course, the master cannot handle flags $i!$, hence such flags are neither introduced by the master, nor communicated to the master.

We denote by $XFPC(CF)$ the families of languages generated in this way: as above, $X \in \{-, C, N, NC\}$.

Here is an *example*, clarifying the previous definition:

$$\begin{aligned}
\Gamma &= (N, T, (S_1, P_1), (S_2, P_2), (S_3, P_3)),\\
N &= \{S_1, S_2, S_3, A\},\\
T &= \{a, b, c\},\\
P_1 &= \{S_1 \to aS_1, S_1 \to [2?A][2?A][3?A]\},\\
P_2 &= \{S_2 \to bS_2b, S_2 \to [1!a]\},\\
P_3 &= \{S_3 \to cS_3, S_3 \to [1!c]\}.
\end{aligned}$$

A derivation in Γ starts with a phase of the form

$$(S_1, S_2, S_3) \Longrightarrow^* (a^n S_1, b^n S_2 b^n, c^n S_3), n \geq 0.$$

In any moment, any component can introduce a flagged symbol. If only P_1 or only P_2 or P_3 introduces such symbols, then the rewriting continues in the components without flags until a matching pair of flags is produced. For instance, we can have

$$(a^n S_1, b^n S_2 b^n, c^n S_3) \Longrightarrow (a^n[2?A][2?A][3?A], b^{n+1} S_2 b^{n+1}, c^{n+1} S_3).$$

No communication is possible, the first component has to wait until 1! is introduced in one of the other components. We can continue with

$$\begin{aligned}
&(a^n[2?A][2?A][3?A], b^{n+1} S_2 b^{n+1}, c^{n+1} S_3)\\
&\Longrightarrow^* (a^n[2?A][2?A][3?A], b^{n+m} S_2 b^{n+m}, c^{n+m} S_3),\ m \geq 0\\
&\Longrightarrow (a^n[2?A][2?A][3?A], b^{n+m}[1!a] b^{n+m}, c^{n+m+1} S_3).
\end{aligned}$$

A communication must now be performed, leading to

$$(a^n b^{n+m} a b^{n+m} b^{n+m} a b^{n+m}[3?A], S_2, c^{n+m+1} S_3).$$

We have considered here the returning case; in the non-returning mode, S_2 above is replaced with $b^{n+m}ab^{n+m}$. This string is never rewritten, hence never communicated (no flag can be introduced). After a number of steps, say $p \geq 0$, also the component P_3 will introduce a flag and then again a communication takes place.

If the second component or the third one introduces a flag before having a flag in the first component, then these components have to wait. Therefore there is no relation between the number of occurrences of the symbol a introduced by P_1, the number of occurrences of the symbol b introduced by P_2, and the number of occurrences of the symbol c introduced by P_3. The generated language is

$$L_\alpha(\Gamma) = \{a^n b^m a b^m b^m a b^m c^p \mid n \geq 0, m \geq 0, p \geq 1\}.$$

This is a non-context-free language; the non-context-freeness appears due to the reduplication of the string produced by the second component.

The system Γ is centralized. One can easily see that the same language as above can be generated by a centralized PC grammar system with flags having only two components: we produce both the prefix a^n and the suffix c^p in the first component, before introducing flagged symbols.

Somewhat expected, at least in the returning centralized case the PC grammar systems with flags are equivalent with the unsynchronized PC grammar systems.

Lemma 5. $CFPC(CF) \subseteq UCPC(CF)$.

Proof. For a PC grammar system with flags $\Gamma = (N, T, (S_1, P_1), \ldots, (S_n, P_n))$ we construct the unsynchronized PC grammar system

$$\Gamma' = (N', K, T, (S_1, P_1'), (S_2^{(2)}, P_2'), \ldots, (S_n^{(n)}, P_n')),$$

where

$$\begin{aligned}
N' &= N \cup \{A^{(i)} \mid A \in N, 1 \leq i \leq n\} \cup \{\overline{A} \mid A \in N\},\\
K &= \{Q_1, \ldots, Q_n\},\\
P_1' &= \{A \to x \in P_1 \mid A \in N, x \in (N \cup T)^*\}\\
&\cup \{A \to x' \mid A \to x \in P_1, A \in N, x \in (N \cup T \cup N_!)^*,\\
&\quad \text{and } x' \text{ is obtained by replacing each symbol } [j?B] \text{ in } x\\
&\quad \text{by } Q_j, 2 \leq j \leq n\}\\
&\cup \{\overline{A} \to A \mid A \in N\},\\
P_i' &= \{A \to x \in P_i \mid A \in N, x \in (N \cup T)^*\}\\
&\cup \{A^{(i)} \to x_1 B^{(i)} x_2 \mid A \to x_1 B x_2 \in P_i, A, B \in N, x_1, x_2 \in (N \cup T)^*\}\\
&\cup \{A^{(i)} \to \overline{x} \mid A \to x \in P_2, A \in N, x \in (N \cup T \cup N_!)^*\\
&\quad \text{and } \overline{x} \text{ is obtained by replacing each symbol } [1!\alpha] \text{ in } x\\
&\quad \text{by } \alpha \text{ if } \alpha \in T \text{ and by } \overline{\alpha} \text{ if } \alpha \in N\},\ 2 \leq i \leq n.
\end{aligned}$$

We have the equality $L_r(\Gamma) = L_r(\Gamma')$: The components $2-n$ of Γ' start from $S_2^{(2)}, \ldots, S_n^{(n)}$, and each sentential form of them contains a symbol $A^{(2)}, \ldots, A^{(n)}$, respectively. Such symbols cannot be rewritten in the component P_1, hence should not be communicated. Their superscripts $(2), \ldots, (n)$ are eliminated only by rules

in $P'_2, \ldots, P'_n$ corresponding to rules in $P_2, \ldots, P_n$ which introduce flagged symbols $[1!\alpha]$. This means that the sentential forms are ready to be communicated to the master. The nonterminals introduced at the same moment with a flagged symbol $[1!\alpha]$ are not rewritten in components $2-n$. That is why they are introduced in a barred form. The bars can be removed in P'_1 by rules $\overline{A} \to A, A \in N$. In turn, the master component uses query symbols Q_j instead of flagged symbols $[j?A]$. Because of the waiting possibilities in Γ and of the unsynchronization of Γ', no restriction appears on the length of the derivations in the components of the two systems.

Consequently, we have $L_r(\Gamma) = L_r(\Gamma')$, that is $CFPC(CF) \subseteq UCPC(CF)$.□

Lemma 6. $UCPC(CF) \subseteq CFPC(CF)$.

Proof. Consider an unsynchronized PC grammar system $\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n))$ and construct a PC grammar system with flags

$$\Gamma' = (N, T, (S_1, P'_1), \ldots, (S_n, P'_n)),$$

with

$$\begin{aligned}
P'_1 &= \{A \to x \in P_1 \mid A \in N, x \in (N \cup T)^*\} \\
&\cup \{A \to x' \mid A \to x \in P_1, A \in N, x \in (N \cup T \cup K)^*, \text{ where} \\
&\quad x' \text{ is obtained by replacing each } Q_i \text{ in } x \text{ by } [i?A], 2 \le i \le n\}, \\
P'_i &= P_i \cup \{A \to x' \mid A \to x \in P_i, A \in N, x \in (N \cup T)^*, |x|_N \ge 1, \\
&\quad \text{and } x' \text{ is obtained by replacing a symbol } \alpha \text{ of } x \\
&\quad \text{by } [1!\alpha], \alpha \in N \cup T\}, \ 2 \le i \le n.
\end{aligned}$$

The two systems Γ, Γ' are equivalent when working in the returning mode: a communication can be performed in Γ' only after having a matching pair of flags, but a flag 1! can be introduced at any time, hence no restriction appear in Γ' when simulating derivations in Γ. □

Theorem 3. $UCPC(CF) = CFPC(CF)$.

As a consequence of this equality, we can formulate for languages generated by PC grammar systems with flags the pumping lemma given in [10]:

Lemma 7. *For each infinite language $L \in CFPC(CF), L \subseteq V^*$, there are a string $z \in L$, a constant p, and two strings $z', z'' \in V^*, z'z'' \neq \lambda$, such that*

$$z = \alpha_1\beta_1\alpha_2\beta_2 \ldots \alpha_k\beta_k\alpha_{k+1},$$

for $1 \le k \le p, \beta_i \in \{z', z''\}, 1 \le i \le k, \alpha_i \in V^*, 1 \le i \le k+1$, and

$$\alpha_1\beta_1^j\alpha_2\beta_2^j \ldots \alpha_k\beta_k^j\alpha_{k+1}$$

is in L for all $j \ge 1$.

Note the essential fact here that one pumps occurrences of the same two substrings z', z'' of the string z. This ensures the fact that, for instance, the language $\{a^n b^n c^n \mid n \ge 1\}$ cannot be generated by a PC grammar system with flags.

Lemma 8. *The family $NCFPC(CF)$ contains one-letter non-regular languages.*

Proof. Consider the system

$$\Gamma = (\{A\}, \{a\}, (A, P_1), (A, P_2)),$$
$$P_1 = \{A \to [2?A], A \to a\},$$
$$P_2 = \{A \to a[1!A]\}.$$

A derivation in Γ proceeds as follows:

$$(A, A) \Longrightarrow^* ([2?A], a[1!A]) \Longrightarrow_{nr} (aA, aA) \Longrightarrow^* (a[2?A], aa[1!A])$$
$$\Longrightarrow_{nr} (aa^2A, a^2A) \Longrightarrow^* (aa^2[2?A], aaa[1!A]) \Longrightarrow_{nr} (aa^2a^3A, a^3A)$$
$$\Longrightarrow^* (aa^2a^3 \dots a^nA, a^nA) \Longrightarrow^* (aa^2a^3 \dots a^na, x).$$

Consequently,

$$L_{nr}(\Gamma) = \{a^m \mid m = \frac{n(n+1)}{2} + 1, n \geq 0\}.$$

This is a one-letter non-regular language. □

Clearly, the language above does not have the pumping property in Lemma 7. Therefore, we obtain the following result (for PC grammar systems of the usual form such a result is not yet known):

Theorem 4. $NCFPC(CF) - CFPC(CF) \neq \emptyset$.

The above results hold true also when λ-rules are allowed.

The power of non-centralized PC grammar systems with flags remains to be investigated.

7 PC Grammar Systems with Filters

A loose selection of the messages exists already in the previous variants of PC grammar systems, as well as in the basic classes of PC grammar systems: a string containing query symbols (or nonterminals which cannot be rewritten in the receiving component in the case of queries by nonterminals) cannot be communicated. Communicating only terminal strings has been also considered in [1].

A variant as in [6] is introduced here: each query symbol is paired with a target language, which is a regular one, and a sentential form can satisfy that query symbol only if it is an element of the associated language.

Formally, a PC grammar system *with local targets* is a construct

$$\Gamma = (N, K, T, (S_1, P_1), \dots, (S_n, P_n)),$$

where each component is exactly as in a usual PC grammar system, except that in rules of $P_1, \dots, P_n$ the query symbols appear in pairs of the form $[Q_i, R]$, where R is a regular language over $N \cup T$ associated with that occurrence of Q_i. (Different occurrences of Q_i can be paired with different regular languages.)

When a communication is intended to satisfy an occurrence of a query symbol Q_i paired with a language R, then only strings in R can be used.

If all regular languages used in such pairs are equal, then the system is written in the form

$$\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n), R),$$

where $(N, K, T, (S_1, P_1), \ldots, (S_n, P_n))$ is a usual PC grammar system and $R \subseteq (N \cup T)^*$ is the common target language. (The query symbols appear in rules of $P_1, \ldots, P_n$ as in a PC grammar system, but they are satisfied only by strings belonging to R.) Such a system is said to be *with a global target*.

The language generated by a PC grammar system with local or global targets is defined in the natural way. We leave the formal details to the reader. The families of such languages are denoted by $XT_lPC(CF), XT_gPC(CF), X \in \{-, C, N, NC\}$, where T_l indicates *local targets* and T_g indicates *global targets*.

By the definitions, we have

Lemma 9. $XPC(Y) \subseteq XT_gPC(Y) \subseteq XT_lPC(Y), X \in \{-, C, N, NC\}, Y \in \{CF, CF^\lambda\}$.

Using targets, even in the global version, seems to significantly increase the power of PC grammar systems at least in the non-centralized returning case (the relation between families $XPC(CF)$ and CS, even when using only λ-free rules in PC grammar systems, is not settled yet). The explanation lies in the fact that by means of the filtering restriction we introduce a powerful context-sensing ability in the system.

Lemma 10. $RE \subseteq T_gPC(CF)$.

Proof. Let $G = (N, T, S, P)$ be a type-0 grammar in Kuroda normal form, that is with P containing rules of the following forms:

1. $A \to x$, with $A \in N, x \in (N \cup T)^*, |x| \leq 2$,
2. $AB \to CD$, with $A, B, C, D \in N$.

We consider the rules of type 2 labelled in a one-to-one manner, $r : AB \to CD$.

We construct the PC grammar system with a global target

$$\Gamma = (N', K, T, (S_1, P_1), (S_2, P_2)),$$

where

$$\begin{aligned}
N' &= N \cup \{C_r, D_r \mid r : AB \to CD \in P\} \cup \{S_1, S_2\},\\
K &= \{Q_1, Q_2\},\\
P_1 &= \{S_1 \to S_2, S_1 \to S, S_1 \to Q_2\}\\
&\quad \cup \{C_r \to C, D_r \to D \mid r : AB \to CD \in P\}\\
&\quad \cup \{A \to x \mid A \to x \in P\},\\
P_2 &= \{S_2 \to S_2, S_2 \to Q_1\}\\
&\quad \cup \{A \to C_r, B \to D_r \mid r : AB \to CD \in P\},\\
R &= (N \cup T)^* \cup (N \cup T)^*\{C_rD_r \mid r : AB \to CD \in P\}(N \cup T)^*.
\end{aligned}$$

We obtain $L(G) = L_r(\Gamma)$.

Let us examine the possible derivations in Γ.

If P_1 introduces the symbol Q_2 while P_2 is still processing S_2, then the system is blocked: $S_2 \notin R$, hence this symbol cannot be communicated. If both components introduce query symbols at the same time, then the system is blocked by circularity of queries.

Thus, P_1 has to eventually use the rule $S_1 \to S$, which will entails a context-free derivation simulating a context-free derivation in G. If a terminal string is obtained, then the derivation stops by producing a string in $L(G)$.

Assume that at some moment we obtain a configuration of the form (w, Q_2). If $w \in (N \cup T)^*$, then w can be communicated (this is the case at the first step when Q_2 is introduced). We obtain (S_1, w). The only way of continuing in P_2 is to use rules of the form $A \to C_r, B \to D_r$, maybe associated to different rules in P. If no query symbol is introduced in P_1, then the derivation will either finish with a terminal string in P_1 or it will be blocked when P_2 will have no symbol to rewrite.

Assume that P_1 introduces Q_2. The string z of P_2 should be in R, otherwise the communication is not permitted. At least a rule $A \to C_r, B \to D_r$ has been used, hence z should be of the form $z = z_1 C_r D_r z_2$, for some $z_1, z_2 \in (N \cup T)^*$, $r : AB \to CD \in P$. This means that the second component has received from the first one, two steps ago, a string $z' = z_1 AB z_2$. By communication, we get the configuration $(z_1 C_r D_r z_2, S_2)$. The first component can use context-free rules of P and can replace C_r, D_r by C, D, respectively. By replacing C_r, D_r by C, D, we get the string $z'' = z_1 CD z_2$. Clearly, $z' \Longrightarrow z''$ by the rule $r : AB \to CD$ of P. In order to communicate the string of the first component to the second one, either no subscripted nonterminal should exist, or both C_r, D_r should be still present. In the second case the string is lost: P_2 has to use one of its rules of the form $X \to X_s$, for s being a non-context-free rule in P, and no further communication can use this string. Therefore, P_1 should replace C_r, D_r by C, D if a further communication will be done.

Consequently, each derivation in the grammar G can be simulated in the system Γ and, conversely, each returning derivation in Γ corresponds to a derivation in G. That is, $L(G) = L_r(\Gamma)$. □

Theorem 5. $T_g PC(CF) = T_l PC(CF) = RE$.

Proof. The inclusion $T_g PC(CF) \subseteq T_l PC(CF)$ is pointed out in Lemma 9, the inclusion $T_l PC(CF) \subseteq RE$ follows from the Turing-Church thesis (or can be directly proved in a straightforward way), whereas the inclusion $RE \subseteq T_g PC(CF)$ is proved in Lemma 10. □

When considering PC grammar systems with targets, non-centralized, working in the returning mode and using only λ-free rules, we obtain a characterization of context-sensitive languages:

Corollary 1. $T_g PC(CF) = T_l PC(CF) = CS$.

It is worth emphasizing in these results the equivalence of local targets with global targets, which can be interpreted in terms of decentralized multi-agent systems.

In the centralized systems we do not have a result as above:

Lemma 11. $CT_gPC(CF) \subseteq CPC(CF)$.

Proof. Let $\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n), R)$ be a PC grammar system with a global target, $R \subseteq (N \cup T)^*, R \in REG$. Consider a deterministic finite automaton $M = (H, N \cup T, s_0, F, \delta)$ recognizing R (H is the set of states, s_0 is the initial state, F is the set of final states, and $\delta : H \times (N \cup T) \longrightarrow H$ is the next state mapping).

For each $i = 2, 3, \ldots, n$ let m_i be the length of the longest derivation which can be performed using the rules in P_i, starting from S_i. If a terminal derivation is possible in (S_i, P_i), then we put $m_i = \infty$. (Because P_i contains only context-free rules, the value of m_i can be computed algorithmically.) Denote

$$m = \max\{m_i \mid 2 \leq i \leq n, m_i < \infty\}.$$

We construct the PC grammar system

$$\Gamma' = (N', K, T, (S_1, P_1), (S'_2, P'_2), \ldots, (S'_n, P'_n)),$$

where

$$\begin{aligned} N' \;=\; & N \cup \{S'_2, \ldots, S'_n\} \times \{X_j \mid 1 \leq j \leq m\} \\ & \cup\; \{X\} \cup \{(s, A, s') \mid s, s' \in H, A \in N\}, \end{aligned}$$

and for $i = 2, 3, \ldots, n$ we put

$$\begin{aligned} P'_i \;=\; & \{S'_i \to \lambda \mid \text{if } S_i \to \lambda \in P_i\} \\ & \cup\; \{S'_i \to \alpha'_1\alpha'_2 \ldots \alpha'_{t+1} \mid t \geq 0, S_i \to \alpha_1\alpha_2 \ldots \alpha_{t+1} \in P_i, \\ & \quad \alpha_j \in N \cup T, 1 \leq j \leq t+1, \text{ there are } s_1, \ldots, s_{t+1} \in H, \\ & \quad s_{t+1} \in F \text{ such that } \delta(s_j, \alpha_{j+1}) = s_{j+1}, 0 \leq j \leq t, \\ & \quad \text{and } \alpha'_j \in \{\alpha_j, (s_{j-1}, \alpha_j, s_j)\}, 1 \leq j \leq t+1\} \\ & \cup\; \{(s, A, s') \to \alpha'_1\alpha'_2 \ldots \alpha'_{t+1} \mid t \geq 0, S_i \to \alpha_1\alpha_2 \ldots \alpha_{t+1} \in P_i, \\ & \quad \alpha_j \in N \cup T, 1 \leq j \leq t+1, \text{ there are } s_1, \ldots, s_{t+1} \in H, \\ & \quad \text{such that } \delta(s_j, \alpha_{j+1}) = s_{j+1}, 0 \leq j \leq t, \\ & \quad \text{and } \alpha'_j \in \{\alpha_j, (s_{j-1}, \alpha_j, s_j)\}, 1 \leq j \leq t+1\} \\ & \cup\; \{S'_i \to X, X \to X \mid \text{if } m_i = \infty\} \\ & \cup\; \{S'_i \to X_1, X_1 \to X_2, \ldots, X_{m_i-1} \to X_{m_i} \mid \text{if } m_i < \infty\}. \end{aligned}$$

The idea behind this construction is the following.

The master component of Γ' is the same as that of $\Gamma, (S_1, P_1)$, hence it can rewrite only symbols in N. The other components of Γ', that is (S'_2, P'_2), $\ldots$, (S'_n, P'_n), can choose two ways of working: producing a string in R or introducing a symbol X or X_1. In the latter case, X will be rewritten for ever by $X \to X$, whereas X_1 will impose a derivation of length at most m_i. This path of deriving is important when no communication will be performed from the corresponding component, but there is a bound on the number of steps a derivation in Γ can

continue after the last communication from that component. By means of X and $X_1, \ldots, X_{m_i}$, this bound is "imported" in Γ, too. When choosing to produce a string in R, then both the terminals and the nonterminals could be bracketed with states in H. The triples of the form (s, A, s') with $A \in N$ can be rewritten in P_i', but triples $(s, a, s'), a \in T$, not. Therefore, no terminal symbols should appear in such a triple and communicated to the first component. Moreover, when communicating to the first component, the string cannot contain triples (s, A, s') either. If the string still contain nonterminal symbols, then the last triple of the form (s, A, s') has to be rewritten exactly in the step when the master introduces the query symbols: the nonterminals in N cannot be rewritten in $P_i', 2 \leq i \leq n$, hence the derivation is blocked. This means that the derivation in P_i' has to correspond to the derivation in P_i which uses the same rules but without bracketing the nonterminals with states in H.

Consequently, $L_r(\Gamma') = L_r(\Gamma)$, completing the proof. □

Theorem 6. $CT_gPC(CF) = CPC(CF)$.

Proof. The inclusion $CPC(CF) \subseteq CT_gPC(CF)$ is pointed out in Lemma 8, the converse inclusion is proved in Lemma 11. □

A result similar to Lemma 11 is probably true also for local targets, but the proof seems not to be similarly simple. One difficulty is the following: if we have two rules in the master, one introducing a query (Q_i, R_1) and the other one introducing (Q_i, R_2), with the same i, but different languages R_1, R_2, then, on the one hand, we need two "twin" components P_i', P_i'', one producing strings in R_1 and one in R_2, on the other hand, we cannot handle two different components associated to P_i, because after a communication the component has to return to the axiom; if P_i' communicates first and then we have to communicate from P_i'', then the string of the latter component is incorrectly derived, during a derivation which has started before the last communication from P_i.

Also the case of non-returning systems remains *open*.

8 Final Remarks

We have introduced in the architecture of PC grammar systems features suggested by three basic characteristics of CPS's: nondeterminism of queries (in two variants), ready-to-communicate symbols (with waiting possibilities until producing matching queries and ready symbols), and filtering procedures for the communicated strings (in a local or in a global way). The effect of these modifications of the basic definition of PC grammar systems on the generative power is different from a case to the other one: the nondeterminism of queries does not modify the generative power, ready-to-communicate symbols associated with waiting possibilities decreases the power (because of the lost of the synchronization), both global and local targets increase the power of non-centralized systems (because of the additional context-sensitivity brought by the filters). The possibility (and the effect) of introducing other CPS's features in PC grammar systems remains to be investigated.

We close this discussion by mentioning that two of the previous variants of PC grammar systems have some resemblance with earlier variants present in the

literature: a non-deterministic way of querying is also considered in [14] (any component can answer a query, there is only one query symbol in the system), filtering features appear also in [12] (each component has an associated regular language and the sentential form obtained by the receiver after a communication step – hence not the communicated string – should be an element of this language) and in [2], [3] (the communication is started by the sending component, like in the WAVE paradigm of communicating in distributed computing, [16], while the receiver selects the messages according to associated regular languages, as above).

References

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

[2] E. Csuhaj-Varju, J. Kelemen, Gh. Păun, Grammar systems with WAVE-like communication, *Computers and AI*, 15, 5 (1996), 419 – 436.

[3] E. Csuhaj-Varju, A. Salomaa, Networks of parallel language processors, in *New Trends in Formal Languages. Control, Cooperation, Combinatorics* (Gh. Păun, A. Salomaa, eds.), *Lecture Notes in Computer Science* 1218, Springer-Verlag, 1997, 299 – 318.

[4] S. Dumitrescu, Non-returning PC grammar systems can be simulated by returning systems, *Theoretical Computer Sci.*, 161 (1996), 463 – 474.

[5] S. Dumitrescu, Gh. Păun, On the generative power of PC grammar systems with right-linear rules, *RAIRO*, 1997.

[6] C. A. R. Hoare, Communicating sequential processes, in *Communications of the ACM*, 21 (1978), 666-677.

[7] C. Martin-Vide, Natural language understanding: a new challenge for grammar systems, *Acta Cybernetica*, 12, 4 (1996), 461 – 472.

[8] V. Mihalache, Matrix grammars versus parallel communicating grammar systems, in *Mathematical Aspects of Natural and Formal Languages* ed. Gh. Păun (World Sci. Publ., Singapore, 1994) pp. 293 – 318.

[9] V. Mihalache, On the generative capacity of parallel communicating grammar systems with regular components, *Computers and AI*, **15** (1996) 155 – 172.

[10] Gh. Păun, On the synchronization in parallel communicating grammar systems, *Acta Informatica*, 30 (1993), 351 – 367.

[11] Gh. Păun, Parallel communicating grammar systems: Recent results, open problems, *Acta Cybernetica*, 12, 4 (1996), 381 – 395.

[12] Gh. Păun, L. Polkowski, A. Skowron, Parallel communicating grammar systems with negociation, *Fundamenta Informaticae*, 28 (1996), 315 – 330.

[13] Gh. Păun, L. Sântean, Parallel communicating grammar systems: the regular case, *Ann. Univ. Buc., Matem.-Inform. Series*, 38, 2 (1989), 55 – 63.

[14] D. Popescu, Parallel communicating grammar systems with comunication by signals, in *New Trends in Formal Languages. Control, Cooperation, and Combinatorics* (Gh. Păun, A. Salomaa, eds.), *Lecture Notes in Computer Science* 1218, Springer-Verlag, 1997, 267 – 277.

[15] G. Rozenberg, A. Salomaa, Eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, Heidelberg, 1997.

[16] P. S. Sapaty, The WAVE paradigm, *Internal Report* 17/92, Dept. of Informatics, Univ. of Karlsruhe, 1992.