Acta Cybernetica 12 (1996) 381-395.

Parallel Communicating Grammar Systems: Recent Results, Open Problems*

Gheorghe PĂUN†

Abstract

First, we recall several recent results concerning the generative power of parallel communicating (PC) grammar systems, including characterizations of recursively enumerable (RE) languages starting from PC grammar systems and their languages. Then, we prove that the simple matrix languages can be generated by PC grammar systems and finally we introduce a new class of PC grammar systems: when a component has to communicate, it may transmit any non-empty prefix of its current sentential form. Each RE language is the morphic image of the intersection with a regular language of a language generated by such a system. A series of open problems are pointed out in this context.

1 Introduction

This paper deals with only one class of grammar systems, the *parallel communicating* (PC) grammar systems, introduced in [24]. We do not discuss here cooperating distributed (CD) grammar systems, introduced in [4]. Of course, also in the case of PC grammar systems we do not cover all the recent results; for instance, we are not concerned here at all with a series of variants introduced in the last time.

Informally speaking, a PC grammar system consists of several usual grammars, each of them having its own sentential form. In each time unit (a common clock divides the time in units, in a uniform way for all components) each component uses a rule, rewriting the associated sentential form. Special (query) symbols are provided, pointing to components of the system. When a component i introduces the query symbol Q_j, then the current sentential form of the component j will be sent to the component i, replacing the occurrence(s) of Q_j. One component is distinguished as the *master*, and the language generated by it, alone or involving communications, is the language generated by the system. Several variants can be

*Research supported by the Academy of Finland, Project 11281, and by Hungarian Scientific Research Fund OTKA T 017105

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

considered, depending on the shape of the communication graph, on the action a component has to perform after communicating, and so on.

The work of PC grammar systems is quite intricate, systems with a small number of components can generate one-letter non-regular languages, [5], characterizations of recursively enumerable languages are obtained by (non-centralized) systems with context-sensitive components, [12], [25], each matrix language (generated without appearance checking) can be generated by a PC grammar system, too, [17], etc. Moreover, many basic questions proved to be very resistent and (with the exception of some particular cases) are still open. For instance, does the number of components induce an infinite hierarchy of families of languages generated by PC grammar systems with context-free components ? Which is the relation between families of languages generated by non-centralized PC grammar systems with context-free (arbitrary or λ-free) rules and the family of context-sensitive languages ? Both grammatical techniques and complexity techniques were used, but without settling this latter question.

Recently, several results were obtained which shed more light on the power of PC grammar systems. We recall some of them in the next section. Without solving the above mentioned questions, they provide a new indication about the difficulty of these questions: characterizations of recursively enumerable (RE) languages were obtained by adding to PC grammar systems certain features usual in language theory (for instance, lefmost derivation). We shall recall some results of this type in Section 3 below.

These results are not the first of this type. For instance, characterizations of *RE* appear also in [19], using query words instead of query symbols, and in [6] and [14], using a variant of PC grammar systems where the communication is done by command, not on request (the component which sends the string to another component starts the communication and the communicated string is accepted only if it passes a given filter associated with the receiving component).

Because PC grammar systems with leftmost derivation characterize *RE*, they trivially generate each simple matrix language; this has been proved in [17] without noticing the equality with *RE*. However, the leftmost restriction is not necessary in order to cover the power of simple matrix languages; we prove this in Section 4.

Then, we introduce a new class of PC grammar systems, where prefixes of the current sentential forms may be communicated. Such systems are both very natural from the point of view of the returning-non-returning feature (when the whole string is communicated, then the component resumes working from its axiom; if a part of the sentential form remains, then one continue from it) and because a nice characterization of RE languages is again obtained: as the morphic image of the intersection of a regular language with a language generated by a system as above. (This is similar to the well-known Chomsky-Schützenberger characterization of context-free languages.) The proof makes use of a powerful result in formal language theory: a characterization of recursively enumerable languages starting from a rather restricted class of languages, the so-called *twin-shuffle* languages, and the operations of intersection with regular languages and erasing morphisms. This result appears in [11]; a proof can be also found in [28]. A twin-shuffle language over a given alphabet V is the set of all strings obtained by arbitrarily shuffling each string over V with a "twin" of the string, obtained by marking each symbol with a bar. Modulo an intersection with a regular language, such a language can be generated in a relatively easy way by a PC grammar system with (λ-free) context-free rules allowed to communicate prefixes.

Several open problems are formulated, both for usual PC grammar systems and for the new variant of PC grammar systems.

2 Parallel communicating grammar systems

As usual, for an alphabet V we denote by V^* the free monoid generated by V under the operation of concatenation; the empty string is denoted by λ and $V^* - \{\lambda\}$ is denoted by V^+. For $x \in V^*, U \subseteq V$, $|x|$ is the length of x and $|x|_U$ is the number of occurrences in x of symbols in U. A Chomsky grammar is denoted by $G = (N, T, S, P)$, where N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom and P is the set of rewriting rules. The language generated by G is denoted by $L(G)$ and *REG, LIN, CF, CS, RE* are the families of regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively. We also denote by MAT, MAT^λ the families of languages generated by matrix grammars (without appearance checking) with λ-free context-free rules, and with arbitrary context-free rules, respectively. Two languages L_1, L_2 are considered equal if they differ only in the empty string, that is if $L_1 - \{\lambda\} = L_2 - \{\lambda\}$.

For basic elements of formal language theory we refer to [7], [26], [27].

A *parallel communicating* (PC, for short) *grammar system* of degree $n, n \geq 1$ ([24], [5]), is a construct

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

where N, T, K are pairwise disjoint alphabets, with $K = \{Q_1, \dots, Q_n\}$, $S_i \in N$, 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 (P_i, S_i) are the *components* of the system (often, the sets P_i are called components). Note that the query symbols are associated in a one-to-one manner with the components. When discussing the type of the components in Chomsky hierarchy, the query symbols are interpreted as nonterminals. In general, the axiom of component i is denoted by S_i and its associated query symbol by Q_i; when this is the case, we do not explicitly specify these elements; if this is not the case, then the axioms and the query symbols are explicitly defined for each component of a PC grammar system.

For $(x_1, \dots, x_n), (y_1, \dots, y_n)$, 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 $(x_1, \dots, x_n) \Longrightarrow_r (y_1, \dots, y_n)$ 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

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

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

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

[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 fulfilled because a component cannot rewrite its sentential form, although it is a nonterminal 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.

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

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

(No attention is paid to strings in the components $2, \dots, n$ 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.)

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

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

we obtain

$$L_r(\Gamma) = L_{nr}(\Gamma) = \{a^n b^n c^n \mid n \geq 1\}.$$

Here is a derivation in Γ_1:

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

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.

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

The reader might check that we obtain

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

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: denote by $PC_n(X), n \geq 1$, the family of languages generated in the returning mode by non-centralized PC grammar systems with at most n components and with rules of type X; 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 $CPC_n(X), NPC_n(X), NCPC_n(X)$. When no restriction on the number of components is imposed, then we remove the subscript n, obtaining $PC(X), CPC(X), NPC(X), NCPC(X)$. In what concerns the type X of rules, they can be λ-free right-linear (denoted by RL), λ-free context-free (CF), arbitrary right-linear (denoted by RL^λ), arbitrary context-free (CF^λ), and so on. Note that because we consider as equal the languages differing at most by λ, we need no λ-rule for introducing the empty string in our languages.

The diagram in Figure 1 indicates the relations between the eight basic families of languages defined above, for the λ-free case, as well as their relationships with families in the Chomsky hierarchy. The arrows indicate inclusions, not necessarily proper; the families not connected by a path are not necessarily incomparable.

Among the newest relations contained in this diagram, we mention:

1. $NPC(RL) \subseteq PC(RL)$ and $NPC(CF) \subseteq PC(CF)$. (The first result of this type has been given in [18], $NCPC(CF) \subseteq PC(CF)$, hence starting from centralized systems, then a proof for the inclusion $NPC(LIN) \subseteq PC(LIN)$ has been done in [29]; the question was settled in [9].)

2. $MAT \subseteq PC(CF)$ ([17]).

3. $CPC(RL) \subset MAT$ ([20]).

4. $LIN \subset PC(RL)$ ([10]).

5. The families $CPC(RL), NCPC(RL)$ are incomparable and also incomparable with LIN ([5] and [10]).

From the last item above we get the strictness of the inclusions of families $CPC(RL), NCPC(RL)$ in the families above them in this diagram. Not contained in the diagram is the inclusion $PC(RL) \subseteq CS$ proved in [3] (where, in fact, the stronger result is proved that $PC(LIN) \subseteq CS$; the inclusion $PC(RL) \subseteq CS$ is already proved in [2]).

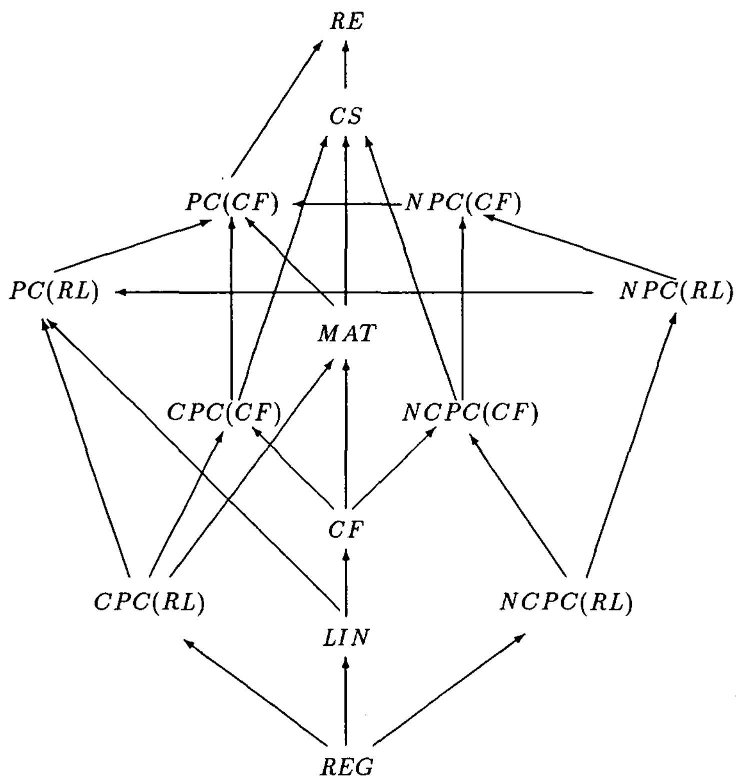

Figure 1

Several problems concerning the generative power of PC grammar systems are still open. We list here some of them.

1. Which of the hierarchies $Y_n(X), n \geq 1$, $Y \in \{PC, CPC, NPC, NCPC\}, X \in \{RL, CF\}$, are infinite ? The answer is known only for $CPC_n(RL)$ and $NCPC_n(RL)$, which, as expected, are infinite hierarchies; see [15].

2. Which of the inclusions not mentioned above as being proper are proper ?

3. Which is the relation between families $CPC(CF)$ and $NCPC(CF)$?

4. Which of the inclusions $Y(X) \subseteq Y(X^\lambda)$, for all possible X, Y, are proper ?
5. Which is the relation between $PC(CF), NPC(CF)$ and CS ? The same when λ-rules are allowed. Several authors have announced proofs of the inclusion $PC(CF) \subseteq CS$, but none of them is confirmed yet.
6. Which are the relations between LIN and $NPC(RL)$? The same for the families MAT and each of $PC(RL), NPC(RL), NPC(CF)$.

3 Characterizing RE

First, we recall a result in [23], concerning PC grammar systems with leftmost derivation. It is known from regulated rewriting area, [7], that such a restriction increases the power of grammars with controlled derivation. This is the case also for PC grammar systems. Moreover, the rather surprising result is obtained that RE can be characterized by such systems with λ-free rules. (The explanation lies in the fact that we can use the components of the system other than the master as working space where no erasing is necessary, because we ignore the contents of these components at the end of a derivation.)

We say that a context-free rule $A \rightarrow v$ is applied in the leftmost mode to a string x, and we denote by $x \Longrightarrow_l y$ the derivation, if $x = x_1Ax_2, y = x_1vx_2$ and $|x_1|_{dom(P_i)} = 0$, where $dom(P_i) = \{B \in N \mid B \rightarrow z \in P_i\}$. We denote by $L_{g,l}(\Gamma), g \in \{r, nr\}$, the language generated by a PC grammar system Γ in the mode g when using leftmost derivations. By $PC_l(X)$ we denote the family of languages $L_{r,l}(\Gamma)$, for Γ a PC grammar system of type X; in the non-returning case we write $NPC_l(X)$.

The inclusions $PC_l(CF) \subseteq PC_l(CF^\lambda), NPC_l(CF) \subseteq NPC_l(CF^\lambda)$ are obvious. We do not know how large the families $NPC_l(CF), NPC_l(CF^\lambda)$ are, but, surprisingly, we have

Theorem 1. $PC_l(CF) = PC_l(CF^\lambda) = RE$.

The idea of the proof is the following.

Take a language $L \subseteq T^*, L \in RE$. It is known (see [27]) that there are two new symbols c_1, c_2 and a language $L' \in CS$ such that $L' \subseteq Lc_1c_2^*$ and for each $w \in L$ there is $i \geq 0$ such that $wc_1c_2^i \in L'$.

Take a (λ-free) grammar $G = (N_0, T \cup \{c_1, c_2\}, S_0, P_0)$ for the language L', in Kuroda normal form, with the non-context-free rules labelled in a one-to-one manner, $r : AB \rightarrow CD$. Assume that for all $A, B \in N_0$ there also is a rule $AB \rightarrow AB$ in P_0.

One constructs the PC grammar system Γ working as follows.

Certain components of it generate strings of the form $w'c_1'c_2'^{\,i}E$, for $wc_1c_2^i \in L'$ (w' is obtained from w by priming its symbols). Then, other components take the string $w'c_1'c_2'^{\,i}E$ generated by the previous group and adjoin to it a string $y''Z$, where $y \in T^+$ and y'' contains double primes. At the same time, one of the components (specifically, P_4 in the construction) produces a terminal string equal to y. The string $w'c_1'c_2'^i Ey''Z$ is took by another group of components which check whether or not $w = y$. When this is true, the master component can ask for the string of P_4. In this way, P_1 receives a terminal string equal to w, hence a string in L.

In the characterization above, the use of context-free rules is esential. Because LIN is incomparable with $CPC(RL^\lambda)$ and $NCPC(RL^\lambda)$ and it is conjectured that the same result holds true for $NPC(RL^\lambda)$, the known characterizations of RE languages starting from linear languages, [1], [16], cannot be directly extended to these classes of PC grammar systems. Still, such results are true for the family $NPC(RL^\lambda)$ at least. Moreover, the proof shows a very close similarity of linear languages and copy languages. Note that every linear language L can be written in the form $L = \{h(x\ mi(\bar{x})) \mid x \in L_0\}$, for a regular language L_0 and a morphism h. Removing the mirror image, we get the copy languages, which characterize RE in the same way as linear languages.

For a language L, denote $copy(L) = \{x\bar{x} \mid x \in L\}$. Proofs of the following lemmas can be found in [23].

Lemma 1. *For each language $L \in RE$ there are two regular languages L_1, L_2 and three morphisms h_1, h_2, h_3 such that $L = h_3(h_1(copy(L_1)) \cap h_2(copy(L_2)))$.*

Lemma 2. *For each language $L \in RE$ there are two regular languages L_1, L_2 and two morphisms h_1, h_2 such that $L = h_1(copy(L_1))\backslash h_2(copy(L_2))$.*

($\backslash$ denotes the left quotient: $L\backslash L' = \{x \mid zx \in L', z \in L\}$.)

Lemma 3. *For each regular language L and morphism h we have $h(copy(L)) \in NPC(RL^\lambda)$.*

Synthesizing Lemmas 1, 2, 3 above, we get

Theorem 2. *For each language $L \in RE$ we can find $L_1, L_2, L_3, L_4 \in NPC(RL^\lambda)$ and a morphism h such that $L = h(L_1 \cap L_2) = L_3\backslash L_4$.*

In the proofs of Theorems 1, 2 above no bound on the number of components of PC grammar systems characterizing the family RE is imposed. This is not the case in [25] and [12], where two context-sensitive components in the non-returning case and three in the returning case are enough (and necessary) for characterizing RE using PC grammar systems. It is an *open problem* whether or not a bounded number of components is enough also in the above theorems. It is also open the case of non-returning PC grammar systems with context-free rules and leftmost derivation; we *conjecture* that such systems cannot characterize RE.

4 Simple matrix grammars versus PC grammar systems

In [17] it is proved that PC grammar systems with leftmost derivation can generate each simple matrix language of [13]. The previous Theorem 1 trivially implies this result. Still, one can prove that the simple matrix languages can be generated by

PC grammar systems with arbitrary context-free components in the usual mode of derivation.

A *simple matrix grammar* (of degree $n, n \geq 1$) is an $(n+3)$-tuple $G = (N_1, \dots, N_n, T, S, M)$, where

1. $N_1, \dots, N_n, T$ are disjoint alphabets ($N_i, 1 \leq i \leq n$, are nonterminal alphabets and T is the terminal one); we denote $N = \bigcup_{i=1}^{n} N_i$;
2. $S \notin N \cup T$ (the axiom);
3. M is a finite set of matrix rules of the forms:
   a) $(S \rightarrow x),\ x \in T^*$;
   b) $(S \rightarrow A_1 A_2 \dots A_n),\ A_i \in N_i, 1 \leq i \leq n,$;
   c) $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n),\ A_i \in N_i, x_i \in (N_i \cup T)^*,\ 1 \leq i \leq n,$
   and $|x_i|_{N_i} = |x_j|_{N_j}$ for all $1 \leq i, j \leq n$.

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if one of the following two cases holds:

(i) $w = S$ and $(S \rightarrow z) \in M$;

(ii) $w = u_1 A_1 v_1 u_2 A_2 v_2 \dots u_n A_n v_n,\ z = u_1 x_1 v_1 u_2 x_2 v_2 \dots u_n x_n v_n,$
where $u_i \in T^*, v_i \in (N_i \cup T)^*, 1 \leq i \leq n$, and $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n) \in M$.

Therefore, the derivation is done in the leftmost manner on each of the n substrings in $(N_i \cup T)^*$ of the derived string. Then,

$$L(G) = \{x \in T^* \mid S \Longrightarrow^* x\}.$$

We denote by SM the family of languages generated by simple matrix grammars (of arbitrary degree) with λ-free context-free rules; when λ-rules are allowed, we write SM^λ for the corresponding family.

The following results are known (see proofs and references in [7]):

1. $CF \subset SM \subset SM^\lambda \subset CS$;
2. Each language in SM^λ is semilinear.

We shall essentially use below the following characterization of languages in the family SM^λ.

Let V be an alphabet and n be a natural number. Denote

$$[V, n] = \{(a, i) \mid a \in V, 1 \leq i \leq n\},$$

and define the mapping $\tau_n : [V, n]^* \longrightarrow (V^*)^n$ by

1. $\tau_n(\lambda) = (\lambda, \dots, \lambda)$,
2. $\tau_n((a, i)x) = (x_1, \dots, x_{i-1}, a x_i, x_{i+1}, \dots, x_n)$,
   for $a \in V, 1 \leq i \leq n, x \in [V, n]^*, \tau_n(x) = (x_1, \dots, x_n)$.

Consider also the mapping $f : (V^*)^n \longrightarrow V^*$ defined by

$$f(x_1, x_2, \ldots, x_n) = x_1 x_2 \ldots x_n.$$

Extend these mappings in the natural way to languages.

From Lemma 1.5.2 in [7] we get

Lemma 4. *A language $L \subseteq T^*$ is in the family SM^λ if and only if there is an integer $n \geq 1$ and a language $L' \in CF$, $L \subseteq [T,n]^*$, such that $L = f(\tau_n(L'))$.*

Using this characterization, we can obtain the following result.

Theorem 3. $SM^\lambda \subset PC(CF^\lambda)$.

Proof. Because $PC(CF^\lambda)$ contains non-semilinear languages (see [5]), it is enough to prove the inclusion.

Consider a simple matrix language $L \subseteq T^*$. If L is finite, then trivially $L \in PC(CF^\lambda)$. Assume that L is infinite. According to Lemma 4, consider $L' \in CF, L' \subseteq [T,n]^*$, such that $L = f(\tau_n(L'))$. Let $G = (N_0, [T,n], S_0, P_0)$ be a context-free grammar for the language L'. We construct the PC grammar system

$$\Gamma = (N, T, K, (S_1, P_1), (S_2, P_2), (S_3, P_3), (S_4, P_4), (S_{4+1}, P_{4+1}), \ldots, (S_{4+n}, P_{4+n})),$$

with

$$\begin{aligned}
N &= \{S_i, S_i' \mid 1 \leq i \leq 4+n\} \cup \{(a,i) \mid a \in T, 1 \leq i \leq n\} \cup N_0 \cup \{Z\},\\
P_1 &= \{S_1 \rightarrow S_1, S_1 \rightarrow Q_5 Q_6 \ldots Q_{4+n}\},\\
P_2 &= \{S_2 \rightarrow S_2, S_2 \rightarrow Q_3, S_3' \rightarrow S_3'\},\\
P_3 &= \{S_3 \rightarrow Z, S_3 \rightarrow S_3', S_3' \rightarrow S_3'\},\\
P_4 &= \{S_4 \rightarrow S_0\} \cup P_0,\\
P_{4+i} &= \{S_{4+i} \rightarrow S_{4+i}', S_{4+i}' \rightarrow S_{4+i}', S_{4+i}' \rightarrow Q_3, Z \rightarrow Q_4\}\\
&\cup \{(a,j) \rightarrow \lambda \mid a \in T, 1 \leq j \leq n, j \neq i\} \cup \{(a,i) \rightarrow a \mid a \in T\},\\
&\quad \text{for } i = 1, 2, \ldots, n.
\end{aligned}$$

The idea behind this construction is the following. The component P_4 generates a string in the language L' (over the alphabet $[T,n]$). When the work of P_4 is finished, all the components $P_{4+i}, i = 1, 2, \ldots, n$, ask for the produced string. The synchronization of these queries (and the fact that each component P_{4+i} can introduce only once the query symbol Q_3) is ensured by the "trigger technique" made possible by the synchronization feature of PC grammar systems and accomplished here by the components P_2, P_3 (see details below). Each component P_{4+i} erases from the received string all symbols (a,j) with $j \neq i$, and replaces (a,i) by $a, a \in T$. In this way, together with P_1, they simulate at the same time the action of τ_n and of f: when communicated to the master, which introduces the string $Q_5 Q_6 \ldots Q_{4+n}$, the strings of $P_5, \ldots, P_{4+n}$ must contain only terminals and they are now arranged in the order imposed by τ_n and f.

Here are some details of the work of Γ.

If P_2 starts by introducing the symbol Q_3, then it will receive either the symbol Z and the derivation is blocked, or the symbol S_3' and no terminal string will be obtained, because $P_{4+i}, 1 \leq i \leq n$, cannot ask for Z at the first step. Thus, we have to start with $S_2 \rightarrow S_2$ in the second component and $S_3 \rightarrow S_3'$ in the third one (if we introduce Z in the third component, then the derivation is blocked, Z cannot be rewritten here or communicated). This means that P_3 will work an arbitrarily large number of steps just using $S_3' \rightarrow S_3'$. It can return to S_3 only when P_2 introduces Q_3. After receiving S_3', the component P_2 will continue for ever with the rule $S_3' \rightarrow S_3'$. Therefore, at the next step P_3 has to use $S_3 \rightarrow Z$, otherwise Z will be never introduced. If not all components $P_{4+i}, 1 \leq i \leq n$, introduce Q_3 at the same time, they must introduce it at the next step, otherwise they cannot receive the symbol Z. But, after receiving Z, any component P_{4+i} has to use $Z \rightarrow Q_4$. At the same step, P_3 will either introduce S_3' and no terminal string will be obtained (S_3' is communicated to components P_{4+i} which have not introduced Q_3 before), or P_3 will introduce Z. After satisfying the query symbols, P_3 returns to its axiom, and P_4 does the same; the components which have received the symbol Z will introduce Q_4 and they will receive S_0 from the fourth component. The derivation is blocked.

The only case when the derivation will continue leading to a terminal string is that when all components $P_{4+i}, 1 \leq i \leq n$, ask for the string of P_4 at the same time.

At any moment, the component P_1 can ask for the strings of $P_{4+i}, 1 \leq i \leq n$. If it receives strings containing symbols in N_0 or in $[T, n]$, then the derivation is blocked. Thus, the only terminal strings produced by Γ are those in $f(\tau_n(L(G_0)))$, which completes the proof. □

5 Prefix communication in PC grammar systems

Let us consider a slight modification in the definition of a communication step in a PC grammar system: when a component i introduces the query symbol Q_j, then component j communicates to component i a non-empty prefix of its current sentential form. If the whole string is communicated, then component j resumes working from its axiom; if a non-empty string remains in component j, then component j continues processing this string. We denote by $L_p(\Gamma)$ the language generated by a system Γ in this way. We denote by $PPC_n(X)$ the family of languages generated by prefix communicating PC grammar systems with at most $n, n \geq 1$, components of type X; when n is not specified, we remove it. When centralized systems are used, then we add the letter C, as usual.

One can consider several variants: to communicate only a terminal prefix, or, deterministically, the maximal terminal prefix, or to allow also the communication of the empty word. Their study, as well as the systematic study of the non-restricted class considered above, is left to the reader. Here we give only one result, again a characterization of RE languages.

Let x, y be strings over some alphabet V. Their *shuffle* is the set

$$\begin{aligned} x \sqcup\!\sqcup\, y = \{ & x_1 y_1 x_2 y_2 \dots x_n y_n \mid x = x_1 x_2 \dots x_n, y = y_1 y_2 \dots y_n, \\ & x_i, y_i \in V^*, 1 \leq i \leq n, n \geq 1\}. \end{aligned}$$

Consider an alphabet V, take a new symbol $\bar{a}$ for each $a \in V$, denote $\overline{V} = \{\bar{a} \mid a \in V\}$, and define the coding $h : V^* \longrightarrow \overline{V}^*$ by $h(a) = \bar{a}, a \in V$. The string $h(x)$ is also denoted by $\bar{x}$.

The *twin-shuffle* language over V, denoted $twin(V)$, is defined by

$$twin(V) = \bigcup_{x \in V^*} (x \sqcup\!\sqcup\, \bar{x}).$$

In [11] (see also [28], Theorem 6.10) one proves the following characterization of recursively enumerable languages:

Lemma 5. *For every recursively enumerable language L there is a twin-shuffle language $twin(V)$, a regular language R and a weak coding h such that $L = h(twin(V) \cap R)$.*

Based on this result, we can obtain

Theorem 4. *For every recursively enumerable language L there is a PC grammar system Γ, a regular language R, and a weak coding h such that*

$$L = h(L_p(\Gamma) \cap R).$$

Proof. For a language $L \in RE$, consider the morphism h and the regular language R as in the previous lemma. Construct the PC grammar system

$$\Gamma = (N, V \cup \overline{V} \cup \{c, \bar{c}\}, K, (P_1, S_1), (P_2, S_2), (P_3, S_3), (P_4, S_4)),$$

with

$$\begin{aligned} N &= \{S_1, S_2, S_3, S_4, X\} \cup \{X_a \mid a \in V\}, \\ P_1 &= \{S_1 \to S_1,\ S_1 \to Q_2 S_1,\ S_1 \to Q_3 S_1,\ S_1 \to Q_2 Q_3,\ S_1 \to Q_3 Q_2\}, \\ P_2 &= \{S_2 \to Q_4,\ X \to c\} \cup \{X_a \to a S_2 \mid a \in V\}, \\ P_3 &= \{S_3 \to Q_4,\ X \to \bar{c}\} \cup \{X_a \to \bar{a} S_3 \mid a \in V\}, \\ P_4 &= \{S_4 \to X_a,\ X_a \to X_a \mid a \in V\} \cup \{S_4 \to X\}. \end{aligned}$$

No communication from the first component to another component is ever performed. Component P_4 introduces symbols X_a for $a \in V$, at each step components P_2, P_3 ask for these symbols, hence component P_4 has to send it to P_2, P_3 and resume working from its axiom. Components P_2, P_3 produce in this way strings $x, \bar{x}$, for the same $x \in V^*$. When P_4 introduces the symbol X, then it becomes c in P_2 and $\bar{c}$ in P_3. Asking for prefixes of the strings produced by P_2 and P_3, in all possible orders, component P_1 builds a shuffle of the two strings, x and $\bar{x}$. Therefore, $twin(V) \subseteq L_p(\Gamma)$.

The opposite inclusion is not true, because of the possibility of sending any prefix to P_1 (not necessarily covering the whole strings of P_2, P_3). However, $L_p(\Gamma) \cap ((V \cup \overline{V})^*\{c\bar{c}\}) = twin(V)\{c\bar{c}\}$: we have to communicate to P_1 a string of the form xc from P_2 and a string $\bar{y}\bar{c}$ from P_3 (and nothing else); as we have seen above, we must have $x = y$.

Consequently, $L = h(L_p(\Gamma) \cap R')$, where

$$R' = R \cap ((V \cup \overline{V})^*\{c\bar{c}\}).$$

This completes the proof. □

Corollary 1. *For each family FL of language such that $FL \subset RE$ and FL is closed under intersection with regular languages and arbitrary morphisms we have $PPC_4(CF) - FL \neq \emptyset$.*

Proof. In view of Lemma 5 and the properties of family FL, the inclusion $PPC_4(CF) \subseteq FL$ would imply $RE \subseteq FL$, a contradiction. □

Important families having the properties of FL above are MAT^λ and $ET0L$ (the family of languages generated by tabled extended L systems without interaction, known to be a full AFL strictly included in CS, [26]). Therefore, $PPC_n(CF)$, contains languages outside these families for all $n \geq 4$. On the other hand, we believe that MAT and $ET0L$ contain languages which are not in $PPC(CF^\lambda)$. If confirmed, this conjecture will imply the incomparability of $PPC(CF), PPC(CF^\lambda)$ with these families, as well as the fact that $PPC(CF^\lambda)$ is not closed under intersection with regular languages (it is obviously closed under arbitrary morphisms).

References

[1] B. Baker, R. Book, Reversal-bounded multipushdown machines, *J. Computer System Sci.*, 8 (1974), 315 - 332.

[2] L. Cai, The computational complexity of PCGS with right-linear components, *Proc. Conf. DLT*, Magdeburg, 1995, World Sci. Publ., Singapore, 1996, 209 - 219.

[3] L. Cai, The computational complexity of linear PCGS, *Computers and AI*, 15, 2-3 (1996), 199 - 210.

[4] E. Csuhaj-Varjú, J. Dassow, On cooperating distributed grammar systems, *J. Inf. Process. Cybern., EIK*, 26, 1-2 (1990), 49 - 63.

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

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

[7] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, Heidelberg, 1989.

[8] J. Dassow, Gh. Păun, G. Rozenberg, Grammar systems, in *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, Heidelberg, 1996.

[9] S. Dumitrescu, Non-returning PC grammar systems can be simulated by returning systems, *Theor. Computer Sci.*, 165 (1996), 463-474.

[10] S. Dumitrescu, Gh. Păun, On the power of PC grammar systems with right-linear rules, submitted, 1996.

[11] J. Engelfriet, G. Rozenberg, Fixed point languages and representations of recursively enumerable languages, *Journal of the ACM*, 27, 3 (1980), 499 - 518.

[12] R. Freund, Gh. Păun, C. M. Procopiuc, O. Procopiuc, Parallel communicating grammar systems with context-sensitive components, in *Artificial Life. Grammatical Models* (Gh. Păun, ed.), Black Sea Univ. Press, Bucharest, 1995, 166 - 174.

[13] O. Ibarra, Simple matrix languages, *Inform. Control*, 17 (1970), 359 - 394.

[14] L. Ilie, A. Salomaa, 2-testability and relabeling produce everything, submitted, 1995.

[15] J. Kari, L. Sântean, The impact of the number of cooperating grammars on the generative power, *Theor. Computer Sci.*, 98 (1992), 249 - 263.

[16] M. Latteux, B. Leguy, B. Ratoandromanana, The family of one-counter languages is closed under quotient, *Acta Informatica*, 22 (1985), 579 - 588.

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

[18] V. Mihalache, On parallel communicating grammar systems with context-free rules, in vol. *Mathematical Linguistics and Related Topics* (Gh. Păun, ed.), Ed. Academiei, Bucureşti, 1995, 147 - 160.

[19] V. Mihalache, Parallel communicating grammar systems with query words, *Ann. Univ. Buc., Matem.-Inform. Series*, 45, 1 (1996), 81 - 93.

[20] V. Mihalache, On the generative capacity of PCGS with regular components, *Computers and AI*, 15, 2-3 (1996), 27 - 36.

[21] Gh. Păun, Grammar systems: a grammatical approach to distribution and cooperation, ICALP '95, *LNCS* 944 (Z. Fülöp, F. Gécseg, eds.), Springer-Verlag, 1995, 429 - 443.

[22] Gh. Păun, Parallel communicating grammar systems. A survey, *Proc. XI Congress on Natural and Formal Languages*, Tortosa, 1995 (C. Martin-Vide, ed.), 257 - 283.

[23] Gh. Păun, Characterizations of recursively enumerable languages by means of grammar systems, submitted, 1996.

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

[25] O. Procopiuc, C. M. Ionescu, F. L. Ţiplea, Parallel communicating grammar systems: the context-sensitive case, *Intern. J. Computer Math.*, 49 (1993), 145 - 156.

[26] G. Rozenberg, A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.

[27] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

[28] A. Salomaa, *Jewels of Formal Language Theory*, Computer Science Press, Rockville, Maryland, 1981.

[29] Gy. Vaszil, Linear non-returning PCGS can be simulated by returning PCGS, manuscript, 1996.