Parallel Communicating Grammar Systems and Analysis by Reduction by Restarting Automata

Dana Pardubská[***1], and Martin Plátek[2]

[1] Dept. of Computer Science, Comenius University, Bratislava
pardubska@dcs.fmph.uniba.sk
[2] Dept. of Computer Science, Charles University, Prague
Martin.Platek@mff.cuni.cz

Abstract. This paper studies the relation between Parallel Communicating Grammar Systems (PCGS) and Freely Rewriting Restarting Automata (FRR). It is shown that analysis by reduction for a PCGS with m components, and with communication complexity bounded by constant k can be implemented by a strongly linearized deterministic FRR-automaton with t rewrites per cycle, where $t \leq m(m-1)^k$. We show that this implementation is almost optimal. As consequences we obtain a pumping lemma for the class of languages generated by PCGS with communication complexity bounded by a constant, and the fact that this class of languages is semi-linear.

1 Introduction

This paper deals with the comparison of Freely Rewriting Restarting Automata (FRR, [6]), and Parallel Communicating Grammar Systems (PCGS, [10,2]). Namely, the so-called linearized FRR-automaton and returning PCGS with regular components are used for this purpose. The motivation for our study is the usefulness of both models in computational linguistic.

Freely rewriting restarting automata create a suitable tool for modelling the so-called *analysis by reduction*, namely its topological properties connecting the (lexical) valences and word-order in sentences with a simple segmentation structure. Analysis by reduction in general facilitates the development and testing of categories for syntactic and semantic disambiguation of sentences of natural languages with a simple segmentation structure. The Functional Generative Description for the Czech language developed in Prague (see, e.g., [3]) is based on this method.

FRR-automata work on so-called *characteristic languages*, that is, on languages with auxiliary symbols (categories) included in addition to the input

* Partially supported by the Slovak Grant Agency for Science (VEGA-1/3106/06) under contract "Theory of Models, Complexity and Algorithms".
** Partially supported by the Grant Agency of the Czech Republic under Grant-No. 405/08/0681 and by the program Information Society under project 1ET100300517.

symbols. The *proper language* is obtained from a characteristic language by removing all auxiliary symbols from its sentences. By requiring that the automata considered are *linearized* we restrict the number of auxiliary symbols allowed on the tape by a function linear in the number of terminals on the tape. We mainly focus on the deterministic restarting automata in order to ensure the *correctness preserving property* for the analysis, i.e., after any restart within an accepting computation the content of the tape is a word from the characteristic language, after any restart within a non-accepting computation the content of the tape is a word from the complement of the characteristic language, and if there is an non-accepting computation for a word w then the word w is from the complement of the characteristic language. In fact, we consider *strongly linearized* restarting automata. This additional restriction requires that all rewrite operations must be deletions.

Parallel Communicating Grammar Systems handle the creation of copies of generated strings and their regular mappings in a natural way. This ability has a strong similarity to the generation of coordinations in the Czech language (and some other natural languages). However, the synonymy of coordinations has not been appropriately modelled yet.

In this paper we study the relation between PCGS's and FRR's. It is shown that analysis by reduction for a returning PCGS with m regular components, and with communication complexity bounded by constant k can be implemented by a strongly linearized deterministic FRR-automaton with t rewrites per cycle, where $t \leq m(m-1)^k$.

We show that this implementation is almost optimal. As a consequence of the proof we obtain a pumping lemma for the class of languages generated by PCGS with communication complexity bounded by a constant, and the fact that this class of languages is semi-linear. Let us recall that the class of unconstrained PCGS is not semi-linear.

The presented results show the potential ability of restricted PCGS's to model the valency and the (free) word-order of the simply segmented sentences of natural languages (i.e. sentences without coordinations, and segmented in a rather simple way).

The paper is organized as follows. In Section 2 we give the basic definitions and results concerning Freely Rewriting Automata. Parallel Communicating Grammar Systems are introduced in Section 3. We give the definitions, summarize known relevant facts and motivate the technical notions utilized in the proof of the main result of the paper there. Results are presented in Section 4 and finally, some closing remarks are given in Section 5.

2 Restarting Automata

A *freely rewriting restarting automaton*, abbreviated as FRR-automaton, is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$. It consists of a finite-state control, a flexible tape, and a read/write window of a fixed size $k \geq 1$. Here Q denotes a finite set of (internal) states that contains the initial state q_0, Σ is

a finite input alphabet, and Γ is a finite tape alphabet that contains Σ. The elements of $\Gamma \setminus \Sigma$ are called *auxiliary symbols*. The additional symbols $¢, \$ \notin \Gamma$ are used as markers for the left and right end of the workspace, respectively. They cannot be removed from the tape. The behavior of M is described by a transition function δ that associates transition steps to certain pairs of the form (q, u) consisting of a state q and a possible content u of the read/write window. There are four types of transition steps: *move-right steps*, *rewrite steps*, *restart steps*, and *accept steps*. A *move-right step* simply shifts the read/write window one position to the right and changes the internal state. A *rewrite step* causes M to replace a non-empty prefix u of the content of the read/write window by a shorter word v, thereby shortening the length of the tape, and to change the state. Further, the read/write window is placed immediately to the right of the string v. A *restart step* causes M to place its read/write window over the left end of the tape, so that the first symbol it sees is the left sentinel ¢, and to reenter the initial state q_0. Finally, an *accept step* simply causes M to halt and accept.

A *configuration* of M is described by a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \varepsilon$ (the empty word) and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0 ¢ w \$$, where $w \in \Gamma^*$.

Any computation of M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration. The window is shifted along the tape by move-right and rewrite operations until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, the computation necessarily finishes in a halting configuration – such a phase is called a *tail*. It is required that in each cycle M performs at least one rewrite step. As each rewrite step shortens the tape, we see that each cycle reduces the length of the tape. We use the notation $u \vdash_M^c v$ to denote a cycle of M that begins with the restarting configuration $q_0 ¢ u \$$ and ends with the restarting configuration $q_0 ¢ v \$$; the relation $\vdash_M^{c^*}$ is the reflexive and transitive closure of $\vdash_M^c$.

A word $w \in \Gamma^*$ is *accepted* by M, if there is a computation which starts from the restarting configuration $q_0 ¢ w \$$, and ends with an application of an accept step. By $L_C(M)$ we denote the language consisting of all words accepted by M. It is the *characteristic language* of M.

By Pr^Σ we denote the projection from Γ^* onto Σ^*, that is, Pr^Σ is the morphism defined by $a \mapsto a$ $(a \in \Sigma)$ and $A \mapsto \varepsilon$ $(A \in \Gamma \setminus \Sigma)$. If $v := \mathsf{Pr}^\Sigma(w)$, then v is the *Σ-projection* of w, and w is an *expanded version* of v. For a language $L \subseteq \Gamma^*$, $\mathsf{Pr}^\Sigma(L) := \{ \mathsf{Pr}^\Sigma(w) \mid w \in L \}$. Further, for $K \subseteq \Gamma$, $|x|_K$ denotes the number of occurrences of symbols from K in x.

In recent papers (see, e.g., [5]) restarting automata were mainly used as acceptors. The (*input*) *language* accepted by a restarting automaton M is the set $L(M) := L_C(M) \cap \Sigma^*$. Here, motivated by linguistic considerations to model the analysis by reduction with parallel processing, we are rather interested in the so-called *proper language of* M, which is the set of words $L_P(M) := \mathsf{Pr}^\Sigma(L_C(M))$.

Hence, a word $v \in \Sigma^*$ belongs to $L_{\mathrm{P}}(M)$ if and only if there exists an expanded version u of v such that $u \in L_{\mathrm{C}}(M)$.

For each type X of restarting automata, we use $\mathcal{L}_{\mathrm{C}}(\mathsf{X})$ and $\mathcal{L}_{\mathrm{P}}(\mathsf{X})$ to denote the class of all characteristic languages and the class of all proper languages of automata of this type.

The following basic properties of FRR-automata are often used in proofs.

(Correctness Preserving Property.) Each deterministic FRR-automaton M is *correctness preserving*, i.e., if $u \in L_{\mathrm{C}}(M)$ and $u \vdash_M^{c^*} v$, then $v \in L_{\mathrm{C}}(M)$, too.

(Cycle Pumping Lemma.) For any FRR-automaton M, there exists a constant p such that the following property holds. Assume that $uxvyz \vdash_M^c ux'vy'z$ is a cycle of M, where $u = u_1u_2 \cdots u_n$ for some non-empty words $u_1, \ldots, u_n$ and an integer $n > p$. Then there exist $r, s \in \mathbb{N}_+$, $1 \leq r < s \leq n$, such that $u_1 \cdots u_{r-1}(u_r \cdots u_{s-1})^i u_s \cdots u_n xvyz \vdash_M^c u_1 \cdots u_{r-1}(u_r \cdots u_{s-1})^i u_s \cdots u_n x'vy'z$ holds for all $i \geq 0$, that is, $u_r \cdots u_{s-1}$ is a "pumping factor" in the above cycle. Similarly, such a pumping factor can be found in any factorization of length greater than p of v or z as well as in any factorization of length greater than p of a word accepted in a tail computation.

We focus our attention on FRR-automata, for which the use of auxiliary symbols is less restricted than in [6]—we allow the number of auxiliary symbols to be linear in the length of the characteristic word.

Definition 1. *Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an* FRR*-automaton.*

(a) *M is called* linearized *if there exists a constant $j \in \mathbb{N}_+$ such that $|w|_{\Gamma-\Sigma} \leq j \cdot |w|_\Sigma + j$ for each $w \in L_C(M)$.*
(b) *M is called* strongly linearized *if it is linearized, and if each of its rewrite operations just deletes some symbols.*

Since linearized FRR-automata operate in linear space, we have the following:

Corollary 1. *If M is a linearized* FRR*-automaton, then the proper language $L_{\mathrm{P}}(M)$ is context-sensitive.*

Due to our linguistic motivations we are mainly interested in the strongly linearized FRR-automata and their proper languages in what follows. We denote by SLnRR the class of strongly linearized deterministic FRR-automata, and, for $t \in \mathbb{N}_+$, we use the prefix t- to denote the types of FRR-automata that execute at most t rewrite steps in any cycle.

3 Parallel Communicating Grammar Systems

To be consistent with our previous works concerning the returning parallel communicating grammar system with regular components we follow the definitions and abbreviation PCGS from the early papers [10,2,7]. The same system is defined in a slightly different way and denoted PCREG in [1]; the differences are only in formalism and denotations.

PCGS *of degree* m, $m \geq 1$, is an $(m+1)$-tuple $\Pi = (G_1, \ldots, G_m, K)$, where, for all $i \in \{1, \ldots, m\}$, $G_i = (N_i, T, S_i, P_i)$ are regular grammars, called *component grammars*, satisfying $N_i \cap T = \emptyset$ and $K \subseteq \{Q_1, \ldots, Q_m\} \bigcap \bigcup_{i=1}^{m} N_i$ is a set of special symbols, called *communication symbols*.
A *configuration* is an m-tuple $C = (x_1, \ldots, x_m)$, $x_i = \alpha_i A_i$, $\alpha_i \in T^*, A_i \in (N_i \cup \varepsilon)$; we call x_i the *i-th component of the configuration* (resp. component). The *nonterminal cut* of configuration C is the $m-$tuple $N(C) = (A_1, A_2, \ldots, A_m)$. If $N(C)$ contains at least one communication symbol, it is denoted $NC(C)$ and called an *NC-cut*.

We say a *configuration* $X = (x_1, \ldots, x_m)$ *directly derives a configuration* $Y = (y_1, \ldots, y_m)$ and write $X \Rightarrow Y$, if Y is derived from X by one *generative* or *communication* step (see below). Informally, in a communication step any occurrence of a communication symbol Q_i in X is substituted by the i-th component of X (assuming that this component does not contain any communication symbol).

Let $X = (x_1, \ldots, x_m)$, $x_i = \alpha_i A_i$, $\alpha_i \in T^*, A_i \in (N_i \cup \varepsilon)$:

1. (generative step) if $A_i \notin K$ for all i, $1 \leq i \leq m$, then
   $x_i \overset{G_i}{\Rightarrow} y_i$ for $x_i \in T^* N_i$,
   $y_i = x_i$ for $x_i \in T^+$;
2. (communication step) if $A_i \in K$ for some i, $1 \leq i \leq m$, then for each k such that $x_k = z_k Q_{j_k}$, $z_k \in T^*, Q_{j_k} \in K$, the following holds:
   (a) If $A_{j_k} \notin K$, then $y_k = z_k x_{j_k}$ and $y_{j_k} = S_{j_k}$.
   (b) If $A_{j_k} \in K$, then $y_k = x_k$.
   For all remaining indices t, for which x_t does not contain a communication symbol and Q_t does not occur in any of x_i's, we put $y_t = x_t$.

Now, we describe the derivations in a PCGS. A *derivation* of a PCGS Π is a sequence of configurations $D = C_1, C_2, \ldots, C_t$, where $C_i \Rightarrow C_{i+1}$ in Π. If the first component of C_t is a terminal word w, then we usually write $D(w)$ instead of D. Analogously, we denote by $W(D)$ the terminal word generated within the derivation D. Every derivation can be viewed as a sequence of *generative steps* and *communication steps*.

If no communication symbol appears in any of the components, then we perform a *generative step* consisting of rewriting steps synchronously performed in each of the component grammars $G_i, 1 \leq i \leq m$. If any of the components is a terminal string, it is left unchanged. If any of the components contains a nonterminal that cannot be rewritten, the derivation is blocked. If the first component is a terminal word y, then y is the word generated by Π in this derivation.

If a communication symbol is present in any of the components, then a *communication step* is performed. It consists of replacing those communication symbols with the phrases they refer to for which the phrases do not contain communication symbols. Such an individual replacement is called a *communication*. Obviously, in one communication step at most $m-1$ communications can be performed. If some communication symbol was not replaced in this communication

step, it may be replaced in one of the next communication steps. Communication steps are performed until no more communication symbols are present or the derivation is blocked because no communication symbol can be replaced in the last communication step. This maximal sub-sequence of communication steps form a *communication section*.

Generative section is a non-empty sequence of generative steps between two consecutive communication steps (resp. communication sequences) in $D(w)$, resp. before the first and/or after the last communication step in $D(w)$.

Thus, communication steps divide the derivation into generative and communication sections.

The *(terminal) language* $L(\Pi)$ generated by a PCGS Π is the set of terminal words that appears in a component G_1(this component is also called a *master* of the system):

$$L(\Pi) = \{\, \alpha \in T^* \mid (S_1, \ldots, S_m) \Rightarrow^+ (\alpha, \beta_2, \ldots, \beta_m) \,\}.$$

To illustrate the derivation and to motivate/demonstrate the bellow defined notions let us give the illustrative example.

Example 1. Consider a PCGS $\Pi_{ab}(2) = (G_1, G_2, G_3, \{Q_2, Q_3\})$ generating [8] the language

$$L_{ab}(2) = \{a^{i_1} b^{i_1} a^{i_2} b^{i_1+i_2} \mid i_j \in \mathbb{N}\}.$$

Bellow, the grammars are implicitly given by their productions; a, b are terminals, all other symbols are non-terminals.

P_1 : $\{S_1 \to aS_1 | aQ_2,\ Z_2 \to aZ_2 | aQ_3,\ Z_3 \to b\}$
P_2 : $\{S_2 \to bZ_2,\quad Z_2 \to bZ_2\}$
P_3 : $\{S_3 \to Z_3,\quad Z_3 \to bZ_3\}$

Let us consider the following derivation $D_1(w)$; we write the configurations into columns and separate them by sign | when the particular step is generative, resp. by sign || to point out the communication step.

$$\underbrace{\left|\begin{matrix} S_1 \\ S_2 \\ S_3 \end{matrix}\right| \left|\begin{matrix} aS_1 \\ bZ_2 \\ Z_3 \end{matrix}\right| \left|\begin{matrix} a^2S_1 \\ b^2Z_2 \\ bZ_3 \end{matrix}\right| \left|\begin{matrix} a^3Q_2 \\ b^3Z_2 \\ b^2Z_3 \end{matrix}\right|}_{1st\ generative\ section} \left\| \begin{matrix} a^3b^3Z_2 \\ S_2 \\ b^2Z_3 \end{matrix} \right| \begin{matrix} a^3b^3aZ_2 \\ bZ_2 \\ b^2bZ_3 \end{matrix} \left| \begin{matrix} a^3b^3a^2Q_3 \\ b^2Z_2 \\ b^2b^2Z_3 \end{matrix} \right\| \begin{matrix} a^3b^3a^2b^2b^2Z_3 \\ b^2Z_2 \\ S_3 \end{matrix} \left| \begin{matrix} a^3b^3a^2b^2b^2b \\ b^2bZ_2 \\ Z_3 \end{matrix} \right|$$

The derivation has three generative sections; the content of the first grammar at the end of the first generative section is a^3Q_2, thus the terminal string a^3 has been generated in grammar G_1 within this section. ◇

Let $D = D(w) = C_0, C_1, \ldots, C_t$ be the derivation of w by Π; $D(w)$, Π, and w are fixed in what follows. Several notions can be associated with the derivation $D(w)$ which help to analyze the derivation of Π and to unambiguously describe w. We start with those describing the structure of w first.

$g(i,j)$, resp. $g(i,j,D(w))$ denotes the terminal part generated by G_i within the j-th generative section of $D(w)$, we call it (i,j)-(generative) factor (by $D(w)$)
$n(i,j)$, resp. $n(i,j,D(w))$ denotes the number of occurrences of $g(i,j)$ in w. Note that $n(i,j) > 1$ is possible, when there are at least two different component grammars requiring the content of the same component grammar *at the same time* (see Example 2).

Communication structure $CS(D(w))$ captures the connection between the terminal word w and its particular derivation $D(w)$)—it determines, how w is composed from individual $g(i,j)$'s:

$CS(D(w)) = (i_1,j_1),(i_2,j_2),\ldots,(i_r,j_r)$, where $w = g(i_1,j_1)g(i_2,j_2)\ldots g(i_r,j_r)$.

The *set* of the tuples of indices of $CS(D(w))$ is denoted $I(D(w))$. Realize that a communication has been involved if $(i,j) \in I(D(w))$ for $i \neq 1$.

Let us analyze the derivation $D_1(w)$ from Example 1:

$$\underbrace{\begin{array}{lll} g(1,1) = a^3 & g(1,2) = a^2 & g(1,3) = b \\ g(2,1) = b^3 & g(2,2) = b^2 & g(2,3) = b \\ g(3,1) = b^2 & g(3,2) = b^2 & g(3,3) = \epsilon \end{array}}$$

$$w = a^3b^3a^2b^5 = g(1,1)g(2,1)g(1,2)g(3,1)g(3,2)g(1,3)$$

We can see that the communication structure $CS(D_1(w))$, $I(D_1(w))$ and particular $n(i,j)$'s have the following form:

$$\begin{array}{l} CS(D_1(w)) = (1,1)(2,1)(1,2)(3,1)(3,2)(1,3) \\ I(D_1(w)) = \{(1,1),(2,1),(1,2),(3,1),(3,2),(1,3)\} \\ n(1,1) = n(1,2) = n(2,1) = n(3,1) = n(3,2) = 1 \\ n(1,3) = n(2,2) = n(2,3) = n(3,3) = 0 \end{array}$$

The last couple of notions is mostly connected with the derivations themselves.

The *trace* of a (sub)derivation D is the sequence $T(D)$ of nonterminal cuts of individual configurations of D; $T(D)$=$N(C_0), N(C_1), \ldots, N(C_t)$. Note that (in general) the trace does not unambiguously identify the derivation.

The *communication sequence*, resp. *NC-sequence* is defined analogously; $NCS(D)$ is the sequence of all NC-cuts in the (sub)derivation D. Let us recall that any NC-cut contains at least one communication symbol. Realize also that the communication sequence $NCS(D(w))$ unambiguously defines the communication structure of w. Moreover, the set of words with the same communication sequence/structure might, in general, be infinite.

A *cycle in the derivation* is such a smallest (continuous) sub-derivation $\mathcal{C}$ od D, $\mathcal{C} = C_1,\ldots,C_j$, in which the corresponding first and last nonterminal cuts are the same; $N(C_1) = N(C_j)$.

If *none* of nonterminal cuts involved in $\mathcal{C}$ contains a communication symbol then, obviously, whole cycle is contained in one generative section; we speak about a *generative cycle* in this case. If the only nonterminal cuts of $\mathcal{C}$ containing a communication symbols are the first and the last one ($N(C_1) = N(C_j)$), then the cycle is called *communication cycle*.

Note that, if there is a cycle in the derivation $D(w)$ then manifold repetition[3] of the cycle is possible and the resulting derivation is again a derivation of some terminal word. We call a derivation $D(w)$ *reduced* if every repetition of its cycle leads to a *longer* terminal word ω; $|w| < |\omega|$. Obviously, to every derivation $D(w)$ there is an equivalent reduced derivation $D'(w)$ of the same word. In what follows, we consider only the derivations that are reduced.

Fact 1 *Repetition/deletion of a generative cycle does not change the communication sequence and communication structure of the derivation.*

Again, we illustrate the defined notions on the derivation D_1 from Example 1.

$$T(D_1) = \begin{pmatrix} S_2 \\ S_2 \\ S_3 \end{pmatrix} \underbrace{\begin{pmatrix} S_1 \\ Z_2 \\ Z_3 \end{pmatrix} \begin{pmatrix} S_1 \\ Z_2 \\ Z_3 \end{pmatrix}}_{\text{generative cycle}} \begin{pmatrix} Q_2 \\ Z_2 \\ Z_3 \end{pmatrix} \begin{pmatrix} Z_2 \\ S_2 \\ Z_3 \end{pmatrix} \begin{pmatrix} Q_3 \\ Z_2 \\ Z_3 \end{pmatrix} \begin{pmatrix} Z_3 \\ Z_2 \\ S_3 \end{pmatrix} \begin{pmatrix} \varepsilon \\ Z_2 \\ Z_3 \end{pmatrix}$$

$$NCS(D_1) = \begin{pmatrix} Q_2 \\ Z_2 \\ Z_3 \end{pmatrix} \begin{pmatrix} Q_3 \\ Z_2 \\ Z_3 \end{pmatrix}$$

Delete the first generative cycle from the derivation D_1. As a result, we have the derivation D_2:

$$\left|\begin{matrix} S_1 \\ S_2 \\ S_3 \end{matrix}\right| \left|\begin{matrix} aS_1 \\ bZ_2 \\ Z_3 \end{matrix}\right| \left|\begin{matrix} a^2Q_2 \\ b^2Z_2 \\ bZ_3 \end{matrix}\right| \left\|\begin{matrix} a^2b^2Z_2 \\ S_2 \\ bZ_3 \end{matrix}\right\| \left|\begin{matrix} a^2b^2aZ_2 \\ bZ_2 \\ bbZ_3 \end{matrix}\right| \left|\begin{matrix} a^2b^2a^2Q_3 \\ b^2Z_2 \\ bb^2Z_3 \end{matrix}\right| \left\|\begin{matrix} a^2b^2a^2bb^2Z_3 \\ b^2Z_2 \\ S_3 \end{matrix}\right\| \left|\begin{matrix} \mathbf{a^2b^2a^2bb^2b} \\ b^2bZ_2 \\ Z_3 \end{matrix}\right|$$

Deletion of the generative cycle has caused the change of the terminal strings generated within the first generative section and consequently, the terminal word ω generated within this changed derivation D_2.

$$g(1,1,D_2) = a^2, g(2,1,D_2) = b^2, g(3,1,D_2) = b \text{ and } CS(D_1) = CS(D_2)$$
$$\Downarrow$$
$$\omega = g(1,1,D_2)g(2,1,D_2)g(1,2,D_2)g(3,1,D_2)g(3,2,D_2)g(1,3,D_2) = \mathbf{a^2b^2a^2bb^2b}$$

Finally, let us define the notion of *communication complexity*. This paper utilizes the notion of communication complexity from [2,7]. Informally, communication complexity of a derivation D (denoted $com(D)$) is defined as the number of communications performed within the derivation D. E.g., the communication complexity of the derivation D_1 from Example 1 is equal 2.

Then, communication complexity of the language and associated complexity class are also defined in the usual way (always considering the corresponding maximum):

[3] Deletion of a cycle is also possible.

Definition 2. *Let $\Pi = (G_1, \ldots, G_m, K)$ be a* PCGS *and $D(w) = C_0, C_1, \ldots, C_t$, where $C_i = (C_{i,1}, C_{i,2}, \ldots, C_{i,m})$, be the derivation of w by Π. Denote by I, $I = \{t_1, t_2, \ldots, t_k \mid t_1 < t_2 < \ldots < t_k\}$, the set of all communication steps of $D(w)$ for which the $(t_i - 1)-st$ step is rewriting. Then $com(D(w)) = \sum_{i=1}^{k} |C_{t_i}|_K$.*

Definition 3. *Let $f : \mathbb{N} \to \mathbb{N}^+$ be a function, Π be a* PCGS*, and $w \in L(\Pi)$. We say that Π* is of communication complexity *f if, for any derivation $D(w)$ by Π, $com(D(w)) \leq f(|w|)$. We say that a* language *L* has communication complexity *f, if there exists a* PCGS *Π of communication complexity f such $L = L(\Pi)$. The* class of languages *with communication complexity f is denoted* COM(f(n)); *for $f(n) = k$ the corresponding class is denoted* COM(k).

Realize that if $L \in COM(k)$, then there is a PCGS Π generating L such that none of its derivations contains a communication cycle. We say that Π is a PCGS without a communication cycle or equivalently that Π is a PCGS with constant communication complexity. The problem of deciding whether a given PCGS is of constant communication complexity or not is decidable.

Here, we are mainly interested in the class COM(k). Relevant observations characterizing the derivations of PCGS with constant communication complexity (see [7] for more information) are summarized in the following facts:

Fact 2 *Let Π be a* PCGS *without a communication cycle. Then there are constants $d(\Pi), \ell(\Pi), s(\Pi)$ such that*

1. *the number $n(i,j)$ of occurrences of individual $g(i,j)'s$ in* reduced *derivation is bounded by $d(\Pi); n(i,j) \leq d(\Pi)$;*
2. *the length of the communication structure for every* reduced *derivation is bounded by $\ell(\Pi)$;*
3. *the cardinality of the set of the possible communication structures corresponding to a reduced derivation by Π is bounded by $s(\Pi)$.*

Fact 3 *Let Π be a* PCGS *without a communication cycle, $D(w)$ a* reduced *derivation of a terminal word w. Then there is a constant $e(\Pi)$ such that, if more than $e(\Pi)$ generative steps of j-th generative section were performed than at least one $g(i,j,D(w))$ has been changed.*

Corollary 2. *Let Π be a* PCGS *without a communication cycle. Then the set of all generative-cycle free derivations by Π is finite.*

4 Results

We start the section showing that a language generated by a PCGS Π with constant communication complexity can be analyzed (by reduction) by a t-SLnRR-automaton M.

The high-level idea is to merge the terminal word w with the information describing its *reduced* derivation $D(w)$ in a way allowing simultaneously the

"simulation/reduction" of the derivation $D(w)$ and the correctness checking. Analysis by reduction is based on the deletion of the parts of a (characteristic) word which correspond to parts generated within one generative cycle. To be able to identify necessary sub-words corresponding to one generative cycle we insert the delimiters $[b,i,j],[e,i,j]$ to denote the beginning and end of individual $g(i,j)$'s into a characteristic word. To make the computation deterministic, we prefix the characteristic word with communication structure. Such a merged (characteristic) word is then called Π-description of w.

Now, let us describe the construction of Π-description of w in more details. Assume the derivation rules in individual component grammars are ordered. Let $(\alpha_1 A_1,\ldots,\alpha_m A_m)$ be the configuration at the beginning of the j-th generative section,

$$\begin{pmatrix}\alpha_1 A_1\\ \alpha_2 A_2\\ \cdots\\ \alpha_m A_m\end{pmatrix} \overset{(i_{1,1},\ldots,i_{1,m})}{\rightarrow} \begin{pmatrix}\alpha_1\alpha_{1,1}A_{1,1}\\ \alpha_2\alpha_{1,2}A_{1,2}\\ \cdots\\ \alpha_m\alpha_{1,m}A_{1,m}\end{pmatrix} \overset{(i_{2,1}\ldots,i_{2,m})}{\rightarrow} \cdots$$
$$\overset{(i_{s,1}\ldots,i_{s,m})}{\rightarrow} \begin{pmatrix}\alpha_1\alpha_{1,1}\alpha_{2,1}\ldots\alpha_{s,1}A_{s,1}\\ \alpha_2\alpha_{1,2}\alpha_{2,2}\ldots\alpha_{s,2}A_{s,2}\\ \cdots\\ \alpha_m\alpha_{1,m}\alpha_{2,m}\ldots\alpha_{s,m}A_{s,m}\end{pmatrix}$$

be the sub-derivation corresponding to this generative section; $\overset{(i_1,\ldots,i_m)}{\rightarrow}$ means, that j-th component grammar has applied its i_j-th derivation rule. Merging the description of this sub-derivation into $g(i,j)$ we obtain *extended version* $ex\text{-}g(i,j)$ of $g(i,j)$:

$$ex\text{-}g(i,j) = [b,i,j]\begin{pmatrix}i_{1,1}\\ i_{1,2}\\ \cdots\\ i_{1,m}\end{pmatrix}\alpha_{1,i}\begin{pmatrix}i_{2,1}\\ i_{2,2}\\ \cdots\\ i_{2,m}\end{pmatrix}\alpha_{2,i}\cdots\begin{pmatrix}i_{s,1}\\ i_{s,2}\\ \cdots\\ i_{s,m}\end{pmatrix}\alpha_{s,i}[e,i,j]$$

Note that the terminal strings $\alpha_1,\ldots,\alpha_m$ from the configuration at the beginning of the j-th generative section have *not* been generated within the j-th generative section and thus are *not* involved in $ex\text{-}g(i,j)$. Realize that all nonterminals $A_1,\ldots,A_m,\ldots,A_{1,1},\ldots,A_{s,m}$ and thus also all nonterminal/communication cuts and cycles are implicitly given in $ex\text{-}g(i,j)$.

We use $ex\text{-}g(i,j)$ to merge the (topological) information about derivation $D(w)$ into w. Obviously, we can speak about *traces* and *factor cycles* in the extended factor $ex\text{-}g(i,j)$ similarly as we speak about traces and generative cycles in derivations.

Replace any occurrence of $g(i,j)$ in w by $ex\text{-}g(i,j)$; the result is denoted $ex\text{-}w$. Then, concatenating NC-sequence of $D(w)$ and $ex\text{-}w$ we obtain *Π-description of w*:

$$\Pi d(D(w)) = \#NCS(D(w))ex\text{-}w\#$$

Remember that the Π-description of w is closely related to some (fixed) derivation $D(w)$ of w. In general, there might be several Π-descriptions of the same

word w. Choosing one Π-description of w, we have fixed one of its derivations, namely $D(w)$.

Let the derivation rules in the component grammars in Example 1 be ordered from left to right. Then, for the derivation $D_1(w)$ we have

$$ex\text{-}g(1,1) = [b,1,1] \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} a \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} a \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} a[e,1,1]$$

From the above discussion the following facts should be obvious:

Fact 4 *Let Π be a* PCGS *with constant communication complexity, $D(w)$ be a reduced derivation and $\Pi d(D(w))$ be the Π-description of w. Then*

(a) the terminal word w is obtained from $\Pi d(D(w))$ by deleting all symbols which are not the terminal symbols of Π;
(b) the length of $\Pi d(D(w))$ is bounded from above by $c_\Pi \cdot |w| + c_\Pi$, where c_Π is a constant depending on Π only;
(c) the sets of possible NC-sequences and communication structures of Π are finite. Thus, a finite automaton is sufficient to check if a given string x is a correct $ex\text{-}g(i,j)$, $NCS(D(w))$, resp. $CS(D(w))$.

We are ready for the main result of this paper. Based on the analysis by reduction the construction of a k-SLnRR-automaton M accepting the characteristic language $L_C(M)=\{\Pi d(D(w)) \mid w \in L(\Pi)\}$ is outlined in the proof of the next theorem.

Theorem 1. *Let $k \in \mathbb{N}$ be a constant, Π be a* PCGS *of degree m with communication complexity bounded by k. Then there is a* t-SLnRR*-automaton M such that $L(\Pi) = L_P(M)$, and $t \leq m(m-1)^k$.*

Proof. We outline the construction of a k-SLnRR-automaton M accepting the characteristic language $L_C(M) = \{\Pi d(D(w)) \mid w \in L(\Pi)\}$. Let ω be an input word to M. M starts to "simulate", resp. reduce the derivation $D(w)$ supposing $\omega = \Pi d(D(w))$. The main idea is to reduce ω in a cycle of M to $\omega_1 \in L_C(M)$ accordingly to a reduction (deletion) of a chosen occurrence of a generative cycle of $D(w)$. To keep the automaton deterministic, the generative cycle taken for reduction has to be deterministically identified. Therefore, we take the generative cycle implicitly described in the leftmost $ex\text{-}g(i,j)$ in the actual ω. Let us denote the newly reduced derivation $D(w_1)$ (using the corresponding shortened terminal word w_1 for that).

Simultaneously, we need to ensure that an $\omega \notin L_C(M)$ should either be directly rejected or reduced to some $\omega_1 \notin L_C(M)$. The set of "short" derivations (resp.Π-descriptions) is finite (Corollary 2), thus checkable by finite automaton. The long word $\omega \notin L_C(M)$ will either in successive steps be reduced to such a short ω' that is not a correct Π-description from $L_C(M)$, or some

inconsistency will be found within the reduction. In this way we ensure by the finite control of M the correctness preserving property of M.

Let $short(\Pi)$ be the finite set of all generative-cycle-free Π-descriptions, $\ell = \max_{\omega \in short(\Pi)} |\omega|$. If $|\omega| \leq \ell$ then M decides in an accepting/rejecting tail. Otherwise, M is trying to perform a cycle.

A cycle by M. At the beginning of its cycle M assumes the content ω of the tape to be $\omega = NCS(D(w))ex\text{-}w$. In order to check the consistence of ω, M stores $NCS(D(w))$ and $SC(D(w))$ in its finite control; remember that $SC(D(w))$ is unambiguously given by $NCS(D(w))$. From now on, M can be viewed as a pair of two simultaneously working automata. One, that is responsible for *reduction* and the second, that is responsible for *verification* of consistency. For simplicity, we will describe those "two" automata separately.

Let us start with the description of the **reduction**: M deterministically finds the leftmost factor cycle in $ex\text{-}w$. Let this (occurrence of the) generative cycle be found between the delimiters $[b,i,j]$ and $[e,i,j]$ thus indicating that we are going to reduce the first generative cycle of the j-th generative section; denote this occurrence of the cycle $c(i,j)$.

More precisely, the prefix of the content of the M's window is of the form

$$[b,i,j]\begin{pmatrix} i_{1,1} \\ i_{1,2} \\ \cdots \\ i_{1,m} \end{pmatrix} \alpha_{1,i} \cdots \alpha_{r-1,i} \underbrace{\begin{pmatrix} i_{r,1} \\ i_{r,2} \\ \cdots \\ i_{r,m} \end{pmatrix} \alpha_{r,i} \cdots \begin{pmatrix} i_{s,1} \\ i_{s,2} \\ \cdots \\ i_{s,m} \end{pmatrix} \alpha_{s,i}}_{c(i,j)} \begin{pmatrix} i_{s+1,1} \\ i_{s+1,2} \\ \cdots \\ i_{s+1,m} \end{pmatrix} \cdots$$

unambiguously identifying the first $s+1$ steps of the j-th generative section (including those of the generative cycle) as

$$pref(j) = \begin{pmatrix} i_{1,1} \\ i_{1,2} \\ \cdots \\ i_{1,m} \end{pmatrix} \cdots \begin{pmatrix} i_{r,1} \\ i_{r,2} \\ \cdots \\ i_{r,m} \end{pmatrix} \cdots \begin{pmatrix} i_{s,1} \\ i_{s,2} \\ \cdots \\ i_{s,m} \end{pmatrix} \begin{pmatrix} i_{s+1,1} \\ i_{s+1,2} \\ \cdots \\ i_{s+1,m} \end{pmatrix}$$

Now, if $pref(j)$ is a possible sub-derivation in Π and the terminal string $\alpha_{1,i} \ldots \alpha_{s,i}$ is consistent with this sub-derivation then M:

- remembers $[j, pref(j)]$ in its state
- deletes $c(i,j)$ from the tape; realize that the left-side nonterminals in production $i_{r,p}$ and $i_{r+1,p}$, $1 \leq p \leq m$, are the same
- remembers identification of the particular $g(i,j)$ in which the successful reduction has just been performed[4] and
- moves its window

From now on M continues its left-right move and reduces relevant parts of the remembered generative cycle from all occurrences of $ex\text{-}g(i,j)$, $1 \leq i \leq m$.

[4] For example, that particular $g(i,j)$ might be deleted from the remembered $NCS(D(w))$

For that, M repeats right-move steps until (1): $[b, i', j]$, $1 \leq i' \leq m$, becomes the leftmost symbol in its window or (2): # becomes the rightmost symbol in its window.

(1) M tries to apply the deletion:

`if`

$$[b, i', j] \begin{pmatrix} i_{1,1} \\ i_{1,2} \\ \cdots \\ i_{1,m} \end{pmatrix} \alpha_{1,i'} \cdots \alpha_{r-1,i'} \underbrace{\begin{pmatrix} i_{r,1} \\ i_{r,2} \\ \cdots \\ i_{r,m} \end{pmatrix} \alpha_{r,i'} \cdots \begin{pmatrix} i_{s,1} \\ i_{s,2} \\ \cdots \\ i_{s,m} \end{pmatrix} \alpha_{s,i'}}_{c(i',j)} \begin{pmatrix} i_{s+1,1} \\ i_{s+1,2} \\ \cdots \\ i_{s+1,m} \end{pmatrix} \cdots$$

is the prefix of M's window [5], `and`

$\alpha_{1,i'} \ldots \alpha_{s,i'}$ is consistent with this sub-derivation,

`then` M deletes $c(i', j)$ from the tape, moves its window and continues reduction procedure in this cycle

`else` the computation is blocked

(2) if all occurrences of the first generative cycle in j-th generative section has successfully been reduced and verification automaton has not found any inconsistences then M restarts; otherwise it `rejects`

To guarantee the correctness preserving property the **verification** automaton checks the consistency of explicitly and implicitly stored information. More precisely, it verifies that

- the content of the tape is a correct Π-description
- each $ex\text{-}g(i, j)$, $1 \leq i \leq m, 1 \leq j \leq k+1$, describes the possible sub-derivation in Π
- the first and last nonterminal cuts implicitly given in $ex\text{-}g(i, j)$ are consistent with stored $NCS(D(w))$

According to the Fact 4(c) such an finite automaton exists.

Rejecting tail computation is identified when either the verification automaton finds the inconsistency within its left-right move or when the content of the tape is a short word that does not belong to $short(\Pi)$.

Accepting tail computation. The only situation in which the accepting tail computation occurs is when a short word ω becomes the content of the tape and $\omega \in short(\Pi)$.

From the above discussion and the construction of M is not hard to see that M is a deterministic t-SLnRR-automaton that accepts the above described language $L_C(M)$. Moreover, we can obtain $L_P(M)$ from $L_C(M)$ by a projection of $L_C(M)$ into the terminal symbols of Π. That means $L_P(M) = L(\Pi)$.

[5] note that we require the described sub-derivation be the same as $pref(j)$, which is stored in the state

Now, let us bound the parameter t. Obviously, the number of rewrites/deletes within one cycle of M is bounded from above by $\max_{1\leq j\leq k+1}\sum_{i=1}^{m} n(i,j)$. To bound $n(i,j)$ realize that the number of occurrences of some $g(i,j)$ in configuration of Π can be increased only by manifold communication: Consider a configuration C

$$(\alpha_1 A_1,\ \cdots,\ \alpha_a A_a,\ \cdots,\ \alpha_m A_m)$$

such that $n'(i,j)$ be the number of occurrences of $g(i,j)$'s in G_a and there are m' component grammars $G_{i_1},\ldots,G_{i_{m'}}$ such that $A_{i_b} = Q_a, 1 \leq b \leq m'$. Then, there are $(m-2)\times n'(i,j)$ new occurrences of $g(i,j)$ in the configuration after the realized communications. Since at the end of j-th generative section there is only one occurrence of $g(i,j)$ in the single component grammar G_i and since the communication complexity of Π is bounded by k, $n(i,j) \leq (m-1)^k$ obviously holds for every i,j. Thus

$$t \leq m \times (m-1)^k$$

completes the proof.

□

The following Corollary 3 easily follows from the proof of the Theorem 1. Then, the Corollary 4 is a consequence of the Corollary 3.

Corollary 3. *Let $k \in \mathbb{N}$ be a constant and Π be a* PCGS *of degree m with communication complexity k. Then there are constants p, t, $0 < t \leq m(m-1)^k$, such that any long enough word $w \in L(\Pi)$, $|w| > p$ can be written in the form $w = y_0 x_1 y_1 \cdots x_t y_t$, where $p \geq |x_1 \cdots x_t| > 0$, and $y_0 x_1^i y_1 \cdots x_t^i y_t \in L(\Pi)$ for any nonnegative integer i.*

Corollary 4. *Let $k \in \mathbb{N}$ be a constant, and Π be a* PCGS *with communication complexity k. Then the language $L(\Pi)$ is semi-linear.*

The assumption about the communication complexity in Corollary 4 is necessary. As the next example from [8] shows, the unconstrained PCGSs are able to generate also languages, that are *not* semi-linear.

Example 2. Bellow defined PCGS illustrates how communication cycle can be used to generate the language $L = \{a^{2^n} \mid n \in \mathbb{N}\}$ that is not semi-linear. Note also that the only generated symbol a is generated within the first generative section.

P_1 : $\{S_1 \to aB \mid Q_2,\ B_1 \to B \mid \epsilon\}$
P_2 : $\{S_2 \to Q_1,\ B \to Q_3\}$
P_3 : $\{S_3 \to Q_1,\ B \to B_1\}$

G_1 :	S_1	aB	S_1	Q_2	Q_2	aaB_1	aaB	S_1	Q_2	Q_2	$aaaaB_1$	$aaaa$
G_2 :	S_2	Q_1	aB	aQ_3	aaB_1	S_2	Q_1	aaB	aaQ_3	$aaaaB_1$	S_2	Q_1
G_3 :	S_3	Q_1	aB	aB_1	S_3	S_3	Q_1	aaB	aaB_1	S_3	S_3	Q_1

As follows from the next proposition, from the number of rewrites per cycle point of view the result from Theorem 1 is not far from the optimal.

Proposition 1. *Let* $L_{k-copy} = \{(w\#)^k\# \mid w \in \{a,b\}^+\}$. *Then*
$L_{k-copy} \in \mathcal{L}(\text{k-SLnRR}) - \mathcal{L}((\text{k-1})\text{-SLnRR})$, *while* $L_{k-copy} \in \mathsf{COM}(\mathsf{O}(\log k))$

Proof. First, we sketch an upper bound part for PCGS. Using quite a sophisticated construction that involves nondeterminism and timing we obtain PCGS Π_{copy} that generates L_{k-copy} with communication complexity $O(\log k)$.

PCGS generating L_{k-copy} consists of several groups of grammars:

$\mathbf{G_2}$	is used to generate w
$\mathbf{C_1, C_2, C_3}$	are used to prepare strings of the form $(w\#)^{2^i}W_i$, one-by-one $(w\#)^{2^i}W_i \quad \rightsquigarrow \quad (w\#)^{2^{i+1}}W_{i+1}$
$\mathbf{P_i}$	is used to store $(w\#)^{2^i}W_i, 0 \leq i \leq \lfloor \log k \rfloor + 1$
$\mathbf{Stop_3}$	is used to restart grammar C_3 whenever necessary

Let $b_m \dots b_0$ be the binary representation of k, $m = \lfloor \log k \rfloor$. Then, $\omega = (w\#)^k$ is contained in G_1 as a concatenation of "pieces" stored in those component grammars P_{i_j}'s for which the corresponding bit $b_{i_j} = 1$.

While w is generated in G_2 at the beginning of the derivation all other grammars are "idling", nondeterministically choosing $S_X \to S_X$ as long as necessary.

$\vdots$					
G_2	S_2	$w_1 S_2$		$w\#W_1$	S_2
C_1	S_{C_1}	S_{C_1}		Q_{G_2}	$w\#W_1$
C_2	S_{C_2}	S_{C_2}	$\cdots$	S_{C_2}	Q_{C_1} $\cdots$
C_3	S_{C_3}	S_{C_3}		S_{C_3}	Q_{C_1}
$\vdots$					

Then, G_2 decides to idle and all the other start working in a cycle.

$\vdots$					
C_1	$(w\#)^{2^i}W_i$	S_{C_1}	Q_{C_2}	Q_{C_2}	$(w\#)^{2^{i+1}}W'_{i+1}$ $(w\#)^{2^{i+1}}W_{i+1}$
C_2	Q_{C_1}	$(w\#)^{2^i}W_i$	$(w\#)^{2^i}Q_{C_3}$	$(w\#)^{2^{i+1}}W'_{i+1}$	S_{C_2} Q_{C_1}
C_3	Q_{C_1}	$(w\#)^{2^i}W_i$	$(w\#)^{2^i}W'_{i+1}$	S_{C_3}	S_{C_3} Q_{C_1}
P_i			Q_{C_2}	Q_{C_2}	$(w\#)^{2^{i+1}}W'_{i+1}$ $(w\#)^{2^{i+1}}W_{i+1}$
$Stop_3$				Q_{C_3}	

At the end G_1 nondeterministically wakes up and finishes generation by concatenating parts that are necessary to be concatenated. For illustration let $k = 5$; written in binary $k = 101$

G_1	Q_{P_1}	$(w\#)^{2^0}W_0$	$(w\#)^{2^0}Q_{P_2}$	$(w\#)^{2^0}(w\#)^{2^2}W_2$	$(w\#)^{2^0}(w\#)^{2^2}\# = \boxed{(w\#)^5\#}$
$\vdots$					
P_0	$(w\#)^{2^0}W_0$	S_{P_1}	S_{P_1}	S_{P_1}	S_{P_1}
P_1	$(w\#)^{2^1}W_1$				
P_2	$(w\#)^{2^2}W_2$	$(w\#)^{2^2}W_2$	$(w\#)^{2^2}W_2$	S_{P_2}	S_{P_2}
$\vdots$					

Now, let us analyze the recognition of L_{k-copy} with FRR automata. It is not hard to construct an k-SLnRR-automaton M_{cp} such that $L_P(M_{cp}) = L_{k-copy}$. In fact, a very similar construction was presented by [6].

For the lower-bound part let $k > 1$. Assume that M is a $(k-1)$-SLnRR-automaton on Γ such that $L_{\mathrm{P}}(M) = L_{k-copy}$. Let $A(m,n,k) := ((a^m b^m)^n \#)^k \#$ where $m, n \in \mathbb{N}_+$ are sufficiently large numbers. Obviously, $A(m,n,k) \in L_{k-copy}$. Hence, there exists an expanded version $w \in \Gamma^*$ of $A(m,n,k)$ such that $w \in L_{\mathrm{C}}(M)$. Assume that w is a shortest expanded version of $A(m,n,k)$ in $L_{\mathrm{C}}(M)$ and consider an accepting computation C of M on input w. Based on the Pumping Lemma it is easily seen that this computation cannot just consist of an accepting tail computation. Thus, it begins with a cycle of the form $w \vdash_M^c x$. From the Correctness Preserving Property and from the assumption that C is accepting it follows that $x \in L_{\mathrm{C}}(M)$, which in turn implies that $\mathsf{Pr}^{\Sigma}(x) \in L_{k-copy}$. As all rewrite steps of M are length-reducing, $|x| < |w|$ follows. Thus, our choice of $w \in L_{\mathrm{C}}(M)$ as a shortest expanded version of $A(m,n,k)$ implies that x is not an expanded version of $A(m,n,k)$. Since C executes at most $k-1$ rewrite steps in the above cycle, it follows that $\mathsf{Pr}^{\Sigma}(x) \notin L_{k-copy}$. Hence, $L_{\mathrm{P}}(M) \neq L_{k-copy}$, which implies that $L_{k-copy} \notin \mathcal{L}_{\mathrm{P}}((k-1)\text{-SLnRR})$. □

5 Conclusion

Motivated by the goals of computational linguistic we have studied two computational models: strongly linearized restarting automata and returning PCGS with regular components. While the restarting automata have already been used in the context of computational linguistic, we have also assumed the usefulness of PCGSs in the field. We believe that the obtained results have confirmed our intuition. Through the analysis by reduction we have separated a non-trivial subclass of languages generated by PCGSs, which is semi-linear. We also believe that the membership-problem for this class of languages is polynomial.

Based on the observations from this paper we will try to make the similar separations for restarting automata as well. Some effort in this direction has already been initiated in [9], where new measures for PCGS and a notion of skeleton for t-FRR-automata have been introduced and studied.

References

1. E.Czuhaj-Varjú, J. Dassow, J. Kelemen and G.Paun (eds.): Grammar Systems: A Gramatical Approach to Distribution and Cooperation. Gordon and Breach Science Publishers, 1994, ISBN 2-88124-957-4
2. J. Hromkovič, J. Kari, L. Kari, and D. Pardubská. Two lower bounds on distributive generation of languages. In *Proc. 19th International Symposium on Mathematical Foundations of Computer Science 1994*, *LNCS* vol. 841, Springer-Verlag, London, 423–432.
3. M. Lopatková, M. Plátek, and P. Sgall. Towards a formal model for functional generative description: Analysis by reduction and restarting automata. *The Prague Bulletin of Mathematical Linguistics* 87 (2007) 7–26.
4. V. Kuboň, M. Lopatková, M. Plátek, and P. Pognan. Segmentation of Complex Sentence. In:Lecture Notes In Computer Science 4188, 2006, 151-158.
5. F. Otto. Restarting automata. In: Z. Ésik, C. Martin-Vide, and V. Mitrana (eds.), *Recent Advances in Formal Languages and Applications*, Studies in Computational Intelligence, Vol. 25, Springer, Berlin, 2006, 269–303.
6. F. Otto and M. Plátek. A two-dimensional taxonomy of proper languages of lexicalized FRR-automata. *Pre-proc. LATA 2008*, S.Z. Fazekas, C. Martin-Vide, and C. Tirnăucă (eds.), Tarragona 2008, 419 – 430.
7. D. Pardubská. Communication complexity hierarchy of parallel communicating grammar system. In: *Developments in Theoretical Computer Science*. Yverdon: Gordon and Breach Science Publishers, 1994. - 115–122. - ISBN 2-88124-961-2.
8. D. Pardubská. Communication complexity hierarchy of parallel communicating grammar system. TR University Paderborn, No. 133, 1993.
9. Dana Pardubská, Martin Plátek, and Friedrich Otto. On PCGS and FRR-automata. Accepted to ITAT 2008.
10. Gh. Păun and L. Santean. Parallel communicating grammar systems: the regular case. *Ann. Univ. Buc. Ser. Mat.-Inform.* 37 vol.2 (1989) 55–63.