Volume 12 Number 4

ACTA CYBERNETICA

Selected papers of the workshop
Grammar Systems: Receut Results on Perspectives,
Budapest, July 1996

Guest Editor: E. Csuhaj Varjú (Hungary)

Szeged, 1996

Grammar Systems: Recent Results and Perspectives (Foreword)

Erzsébet CSUHAJ-VARJÚ *

On July 26-27, 1996, a workshop with the title *Grammar Systems: Recent Results and Perspectives* was held in Budapest, at the Computer and Automation Research Institute of the Hungarian Academy of Sciences[1]. The aim of the meeting was to provide a forum for exchanging ideas about the state of the art of research in the area of grammar systems and preview the trends and perspectives. The presented talks and the fruitful informal discussions resulted in, among other things, the present volume.

" *Grammar systems*" is a recent active field of formal language theory, providing syntactic models, frameworks and tools for describing and studying (the behaviour of) multi-agent systems at the symbolic level. Several scientific areas have motivated and influenced the developments in this theory: *distributed and decentralized artificial intelligence, distributed and parallel computing, artificial life, molecular computing, robotics, ecology, sociology, etc.* Computer networks, parallel and distributed computer architectures, distributed and cooperative text processing, natural language processing are candidates for possible applications.

Roughly speaking, a *grammar system* (the term "grammar" is used here in a general sense) consists of several language identifying devices (language processors or linguistic agents) that jointly develop a common symbolic environment (usually, a string or a finite set of strings) by applying string manipulating operations to it. The symbolic environment can be shared by the components of the system, or it can be given in the form of a collection of separated sub-environments, each belonging to a language processor. At any moment in time, the state of the system is represented by the current string describing the environment (collection of strings of the sub-environments). The functioning of the system is realized by changes in its states. Depending on the variant of multi-agent systems that the actual grammar system represents, in addition to performing derivation steps, the language processors are allowed to communicate with each other. Usually, this is done by exchange of strings that can be data (for example, sentential forms in derivation)

*Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende u. 13-17, H-1111 Budapest, Hungary. E-mail: csuhaj@sztaki.hu

[1]The workshop was supported by Hungarian Scientific Research Fund OTKA no. T 017 105.

or programs (productions or coded form of some operation). The behaviour of the grammar system can be characterized by the set of sequences of environmental states following each other, starting from an initial state, or by the set of all environmental states originating from an initial state and satisfying some criteria (final states).

Grammar systems are both computational and language identifying devices, capturing several phenomena characteristic for multi-agent systems: *cooperation, distribution, communication, parallelism, emergent behaviour, etc.*

To give a picture about the research directions in the area, without the aim of completeness, we list some important frameworks and models. (The interested reader can find detailed information in [6], [21], [14]).

The theory started in 1988 by introducing *cooperating/distributed grammar systems* (CD grammar systems) for modelling syntactic aspects of the blackboard model of problem solving ([4],[5]). We should note, however, that the first appearance of the term *"cooperating grammars"* was in [20] as a notion for extending two-level substitution mechanism of grammars to a multi-level concept. A concept, based on modularity and related to cooperation of grammars, motivated by regulated rewriting, was introduced in [1].

In the basic form, a *CD grammar system* is a finite set of generative (usually context-free) grammars that cooperate in deriving words of a common language. At any moment in time there is exactly one sentential form in generation. The component grammars generate the string in turns, under some cooperation protocol. In this model the cooperating grammars represent independent cooperating problem solving agents that jointly solve a problem by modifying contents of a global database, called blackboard, that is for storing information on the problem solving process. In this architecture the agents communicate with each other only through the blackboard.

The main research directions in the field of CD grammar systems concentrate, among other things, on studying the question whether cooperation adds power to the derivational capacity of the individual grammars or not, and, if the answer is positive, how simple presentation of the components and the protocol is sufficient to reach this power. While the original model was introduced for generative mechanisms, the framework has been extended and applied also to other computational devices (accepting grammars ([2]), automata ([13]), tree processing devices ([16]), etc). Parallel with this kind of enhancement, properties characterizing the model have been studied in details: determinism in cooperation, comparison of systems with components using hybrid and homogenous cooperation strategies, variants of competence-based cooperation, systems with time limited activity of grammars, hierarchies among the components, similarity, uniformity, etc. The achieved results demonstrate the power of cooperation. Large language classes (ET0L, programmed with apperance checking, context-sensitive) can be described in terms of systems of a limited number of very simple cooperating language identifying devices.

While CD grammar systems realize sequential computing devices, *team grammar systems* with simultaneous actions of some grammars (teams) on the sentential form, introduce parallelism in the model ([19],[23]). These systems demonstrate an equivalence between programming the sequence of actions and computation under some kind of competence-based cooperation of freely chosen grammar teams with a very limited number of components (pairs of grammars).

Colonies, motivated by subsumption architectures of R. Brooks, describe language classes in terms of behaviour of collections of very simple, purely reactive, situated agents with emergent behaviour ([17],[18]). In this model the agents are represented by very simple regular grammars (each grammar generates a finite language) that operate on a common sentential form. The basic variant of colonies determines the context-free language class, while the more sophisticated models (competition among the agents, timing, etc.) lead to considerably enhanced computational power ([12],[22]).

Eco-grammar systems form a language theoretic framework for modelling ecosystems: developing linguistic agents, represented by L systems, in a dynamically changing population, interact with each other and with their shared evolving symbolic environment ([9],[10]). Eco-grammar systems provide tools for describing life-like phenomena (birth, death, hybernation, overpopulation, pollution, etc.) in terms of formal grammars and languages.

Networks of language processors (this general term was introduced in [11] and [3]) form an essential part of the area. In this case language processors are located in nodes of a network (a virtual graph). Each processor works on its own sentential form (on its own collection of sentential forms) and informs the others about its activity by communicating strings that can be data and/or programs. Rewriting and communication take place alternately, the system is functioning (usually) in a synchronized manner. *Parallel communicating grammar systems,* a highly elaborated field, with Chomsky and Lindenmayer grammars at the nodes, studies networks with components communicating data strings by request ([24],[25]). *Test tube distributed systems based on splicing* and on *cutting and recombination* are particular cases of the model with components using variants of DNA recombination and realize computationally complete and universal machines (in some cases with a limited number of components) ([8],[15]). Ideas of the *WAVE paradigm* of active knowledge networks are implemented in [7] and *networks of parallel language processors,* where the nodes are represented by L systems, are studied in [11].

In addition to the above mentioned areas, investigations have been started for applications in related scientific areas, as natural language processing.

Recent developments that can be observed in grammar systems theory, are trends

- from *cooperating/distributed grammar systems* to **cooperating/distributed systems of language processors,**
- from *parallel communicating grammar systems* to **networks of language processors,**
- from *simple communities (colonies) of grammars* to **societies of linguistic agents,**
- from *computing* to **nature-motivated computing,**

- from (applications in) *natural language processing* to **natural processing of languages**.

Contributions to the present volume fit into the above trends, by providing a better understanding of cooperation and distribution through a deeper study of the existing models and enhancing the concept to further computational phenomena. The papers are clustered in the volume according to the subfield of grammar systems they represent.

The first three articles are devoted to the study of cooperating/ distributed language processors. Jürgen Dassow and Victor Mitrana studied fairness in CD grammar systems, Henning Bordihn and Erzsébet Csuhaj-Varjú dealt with competence and completeness of component grammars both in the case of generative CD grammar systems and in the case of accepting ones. Henning Fernau and Markus Holzer discussed accepting CD grammar systems in details, in comparison with the generating ones, taking new variants of cooperation protocols into account. They also considered team behaviour of accepting CD grammar systems. Networks of language processors are investigated in the next three papers. The first is the survey of Gheorghe Păun on parallel communicating grammar systems. It also studies new variants and raises several open problems in the area. Valeria Mihalache examined parallel communicating grammar systems with components having own nonterminal alphabets and terminal alphabets, Lucian Ilie and Arto Salomaa provided important characterizations of recursively enumerable languages in terms of parallel communicating grammar systems with WAVE-like communication. Societies of grammars, nature-motivated computing and natural language processing are represented by the last three papers. Maurice H. ter Beek investigated team grammar systems with teams using different cooperation strategies and a new variant of derivation mode, called weak rewriting. A framework motivated by DNA computing, a generalization of test tube systems, is introduced and studied in the paper of Rudolf Freund and Franziska Freund. The last paper in the volume is about the challenge that natural language understanding means for grammar systems, by Carlos Martin-Vide.

References

[1] Atanasiu, A., Mitrana, V. (1989): The modular grammars. Intern. J. Computer Math. 30, 17-35.

[2] Fernau, H., Holzer, M., Bordihn, H. (1996): Accepting multi-agent systems: the case of cooperating distributed grammar systems. Computers and Artif. Intell. 15(2-3), 123-139.

[3] Csuhaj-Varjú, E. (1996): Networks of language processors. A survey. In: Lenguajes Naturales Y Lenguajes Formales XII, (C. Martin-Vide, Ed.), PPU, Barcelona, 169-189.

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

[5] Csuhaj-Varjú, E., Kelemen, J. (1989): Cooperating grammar systems: a syntactical framework for the blackboard model of problem solving. In: Proc. AICSR'89, (I. Plander, Ed.), North Holland, Amsterdam, 121-127.

[6] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, Gh. (1994): Grammar Systems - A Grammatical Approach to Distribution and Cooperation. Gordon and Breach Science Publishers, London

[7] Csuhaj-Varjú, E., Kelemen, J., Păun, Gh. (1996): Grammar Systems with WAVE-like Communication. Computers and Artif. Intell. 15(5), 419-436.

[8] Csuhaj-Varjú, E., Kari, L., Păun, Gh. (1996): Test Tube Distributed Systems Based on Splicing. Computers and Artif. Intell. 15(2-3), 211-232.

[9] Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh. (1994): Eco(grammar)systems - a preview. In: Cybernetics and Systems '94, (R.Trappl, Ed.), World Scientific, Singapore, 941–948.

[10] Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh. (1996): Ecogrammar systems: a grammatical framework for studying lifelike interactions. Artificial Life, to appear.

[11] Csuhaj-Varjú, E., Salomaa, A. (1997): Networks of language processors. Developments in Regulated Rewriting and Grammar Systems, LNCS, to appear.

[12] Dassow, J., Kelemen, J., Păun, Gh. (1993): On Parallelism in Colonies. Cybernetics and Systems 24, 37-49.

[13] Dassow, J., Mitrana, V. (1995): Pushdown Automata Systems. Submitted.

[14] Dassow, J., Păun, Gh., Rozenberg, G. (1997): Grammar systems. In: Handbook of Formal Languages, (G. Rozenberg, A. Salomaa, Eds.), Springer Verlag, Berlin, to appear.

[15] Freund, R., Csuhaj-Varjú, E., Wachtler, F. (1997): Test Tube Systems with Cutting/Recombination Operations. In: Proc. PSB'97, to appear.

[16] Fülöp, Z. (1996): Distributed Tree Processing Devices. Submitted.

[17] Kelemen, J., Kelemenová, A. (1992): A subsumption architecture for generative symbol systems. In: Cybernetics and System Research'92 (R. Trappl, Ed.), World Scientific, Singapore, 1529-1536.

[18] Kelemen, J., Kelemenová, A. (1992): A grammar-theoretic treatment of multiagent systems. Cybernetics and Systems 23, 621–633.

[19] Kari, L., Mateescu, A., Păun, Gh., Salomaa, A. (1995): Teams in cooperating grammar systems. J. Exper. and Theoretical AI, 7, 347-359.

[20] Meersman, R., Rozenberg, G. (1978): Cooperating grammar systems. In: Proc. MFCS'78, LNCS 64, Springer Verlag, Berlin, 1978, 364-374.

[21] Păun, Gh. (1995): Grammar Systems: a grammatical approach to distribution and cooperation. In: ICALP'95, (Z. Fülöp and F. Gécseg, eds.), LNCS 995, Springer Verlag, Berlin, 429-443.

[22] Păun, Gh. (1995): On the generative capacity of colonies. Kybernetika, 83-97.

[23] Păun, Gh., Rozenberg, G. (1994): Prescribed teams of grammars, Acta Informatica, 31, 525-537.

[24] Păun, Gh., Santean, L. (1989): Parallel Communicating grammar systems: the regular case. Annales. Univ. Bucuresti., Ser. Matem.-Inform. 38, 55-63.

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

Fairness in Grammar Systems

Jürgen DASSOW [*] Victor MITRANA [†] [‡]

Abstract

The paper deals with two fairness concepts in cooperating distributed grammar systems. The effect of this restriction on the protocol of cooperation among the components of a grammar system is investigated. In all modes of derivation, the fairness restrictions lead to an increase in the generative power. Surprinsingly, even in the regular case.

1 Introduction

In modern computer science such notions as distribution, cooperation, parallelism, communication, synchronization are more and more vividly investigated. As practical materializations one can mention computer networks, parallel computing, distributed data bases, etc. There are several approaches to these ideas. In this paper we deal with grammar systems which form a grammatical approach.

A grammar system is a construct consisting of several usual grammars, working together, in a specified way, for generating a language. If the grammars work together on the same sentential form, then the system is called *cooperating/distributed* (CD for short) grammar system. If the grammars work on their own sentential forms and, from time to time, send the result of their work to other components, then the system is called *parallel communicating* grammar system.

This paper concerns CD grammar systems. Intuitively, such systems and their work can be described as follows: Initially, the axiom is the common sentential form. At each moment, one grammar is active, that means it rewrites the common string, and the others are inactive. The conditions under which a component can become active or it is disabled and leave the sentential form to other components are specified by the cooperation protocol. The language of terminal strings generated in this way is the language generated by the system. As basic stop conditions which will also be considered in this paper we mention: each component, when active, has to work for exactly k, at least k, at most k, or the maximal number of steps (a

[*]Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, P.O.Box 4120, D-39016 Magdeburg, Germany

[†]Faculty of Mathematics, University of Bucharest, Str. Academiei 14, R-70109 Bucharest, Romania

[‡]Research supported by the Alexander von Humboldt Foundation

step means the application of a rewriting rule). Many other starting and stopping conditions were considered in the literature (see [3]).

Such systems were introduced by different motivations:

1. The generalization of the two-level substitution grammars was the main purpose of the paper [5] where the sintagm "cooperating grammar system" was proposed.

2. Modular grammars as an alternative for the time-varying grammars were presented in [1].

3. In the architecture of a CD grammar system one can recognize the structure of a blackboard model, as used in problem-solving area [6]: the common sentential form is the blackboard (the common data structure containing the current state of the problem to be solved), the component grammars are the knowledge sources contributing to solving the problem, the protocol of cooperation encodes the control on the work of the knowledge sources ([4]). This was the explicit motivation in [2], the paper where CD grammar systems in the form we consider here were introduced.

4. The increase of the computational power of components by cooperation.

5. The decrease of the complexity of different tasks by distribution.

In some sense, the theory of grammar systems is the theory of cooperation protocols; the focus is not on the generative capacity, but on the functioning of the system, and on its influence on the generative capacity and on other specific properties.

The aim of this paper is the investigation of a quite natural feature of the strategy of cooperation: fairness. We require that all components of the system have approximately the same contribution to the common work, concerning the time spent by each of them during the derivation process. The first attempt in this direction, called weak fairness, asks for that each component has to be activated almost of the same number of times (the difference between the number of times for which any two components are activated is bounded). But this concept says nothing about the period of time in which a component is working. So, if we want to have more precisely a fair behaviour of the system, called strong fairness, then it is necessary to measure also this time, i.e. to count the number of applications of rules of a component during the whole derivation.

The requirement that a system has to be "fair" increases, even for systems with regular components (this situation contrasts the "unfair" case), the generative power of the system.

2 Definitions

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 we set $V^+ =$

$V^* \setminus \{\lambda\}$. The length of $x \in V^*$ is denoted by $|x|$. If $x \in V^*$ and $U \subseteq V$, then $|x|_U$ is the number of occurrences of symbols of in x (the length of the string obtained by erasing from x all symbols in $V \setminus U$). By REG, CF and $ETOL$ we denote the families of regular, context-free and ETOL languages, respectively (see [7], [8]).

A *CD grammar system of degree* $n, n \geq 1$, is a construct

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

where N, T are disjoint alphabets, $S \in N$, and $P_1, \ldots, P_n$ are finite sets of rewriting rules over $N \cup T$.

The elements of N are nonterminals, those of T are terminals; $P_1, \ldots, P_n$ are called *components* of the system. Here we work with CD grammar systems having only regular rules, i.e. rules of the form $A \to aB$ or $A \to a$ with $A, b \in N$, $a \in T$, or context-free rules, i.e. rules of the form $A \to w$ with $A \in N$, $w \in (N \cup T)^*$.

The domain of the i^{th} component denoted by $dom(P_i)$ is defined as $dom(P_i) = \{A | A \to x \in P_i\}$.

On $(N \cup T)^*$ one can define the usual one step derivation with respect to P_i, denoted by $\Longrightarrow_{P_i}$. The derivations consisting of exactly k, at most k (but at least one), at least k such steps $\Longrightarrow_{P_i}$ are denoted by $\Longrightarrow_{P_i}^{=k}$, $\Longrightarrow_{P_i}^{\leq k}$, $\Longrightarrow_{P_i}^{\geq k}$, respectively. Furthermore, we write $x \Longrightarrow_{P_i}^{t} y$ iff $x \Longrightarrow_{P_i}^{\geq 1} y$ and there is no $z \in (N \cup T)^*$ such that $y \Longrightarrow_{P_i} z$.

Let

$$M = \{t\} \cup \bigcup_{k \geq 1} \{\leq k, = k, \geq k\}.$$

The language generated by the system Γ in the derivation mode $f \in M$ is

$$\begin{aligned} L_f(\Gamma) \;=\; & \{w \mid w \in T^*,\ S \Longrightarrow_{P_{i_1}}^{f} w_1 \Longrightarrow_{P_{i_2}}^{f} \cdots \Longrightarrow_{P_{i_m}}^{f} w_m = w, \\ & m \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq m\}. \end{aligned}$$

The respective classes of languages are denoted by $CDL_n(X, f)$, where n is the degree of the grammar system, $X \in \{REG, CF\}$ indicates the type of the components (regular or context-free) and $f \in M$.

Let

$$D \; : \; S \Longrightarrow_{P_{i_1}}^{=m_1} w_1 \Longrightarrow_{P_{i_2}}^{=m_2} \cdots \Longrightarrow_{P_{i_t}}^{=m_t} w_k$$

be a derivation in the f-mode, $f \in M$ (i.e. m_j gives the number of derivation steps performed by the component P_{i_j} in D; especially, if $f \in \{= k\}$, then $m_i = k$ holds for $1 \leq j \leq t$, etc.) For any $1 \leq p \leq n$, we write

$$\psi_D(p) = \sum_{i_j = p} 1 \quad \text{and} \quad \varphi_D(p) = \sum_{i_j = p} m_j$$

(i.e by $\psi_D(p)$ we count the number of applications of P_p, and by $\varphi_D(p)$ we count the number of applications of rules of P_p). Conventionally, the empty sum delivers zero.

Let Γ be a CD grammar system with at least two components. Then we set

$$dw(D) = max\{|\psi_D(i) - \psi_D(j)| \mid 1 \leq i, j \leq n\}$$

and

$$ds(D) = max\{|\varphi_D(i) - \varphi_D(j)| \mid 1 \leq i, j \leq n\}.$$

By these two numbers we measure the maximal difference between the contribution of components involved in the derivation D. The contribution of a component may be expressed as the number of its applications and the number of rules applications in the considered component, respectively. Moreover, for $u \in \{w, s\}$, $x \in (N \cup T)^*$ and $f \in M$, we define

$$du(x, f) = min\{du(D) \mid D \text{ is a derivation in the } f-\text{mode for } x\}.$$

In order to get a concept of fairness, we now restrict the numbers $du(x, f)$ for the words x which belong to the language. If $u = w$, then we get a weaker notion since we only require that the components are used almost equally often, whereas $u = s$ gives a stronger notion where the times which the components work are approximately equal. Formally, this leads to the following definitions.

For a CD grammar system Γ of degree $n \geq 2$, $f \in M$ and a natural number $q \geq 0$, we define the *weakly q-fair* language generated by Γ in the f-mode as

$$L_f(\Gamma, w - q) = \{x \mid x \in L_f(\Gamma) \text{ and } dw(x, f) \leq q\}$$

and the *strongly q-fair* language of Γ as

$$L_f(\Gamma, s - q) = \{x \mid x \in L_f(\Gamma) \text{ and } ds(x, f) \leq q\}.$$

For $X \in \{REG, CF\}$ and integers $n \geq 2$ and $q \geq 0$, by $CDL_n(X, f, w - q)$ and $CDL_n(X, f, s - q)$ we denote the families of weakly and strongly q-fair languages, respectively, generated by CD grammar systems with n components.

Let us illustrate the concepts of fairness by two examples. We shall give just the components of the systems, the other components can easily be deduced under the assumption that S is the axiom.

Example 1 We consider the grammar system Γ_1 with the components

$$P_1 = \{S \to aA', A \to aA'\}, \quad P_2 = \{A' \to aA\},$$
$$P_3 = \{A \to bB', B \to bB'\}, \quad P_4 = \{B' \to bB, B' \to b\}.$$

Then, for $q \geq 0$ and $f \in \{t, = 1, \geq 1\} \cup \{\leq k \mid k \geq 1\}$, we obtain

$$L_f(\Gamma_1) = \{a^{2n}b^{2m} \mid n \geq 1, m \geq 1\}$$

and

$$L_f(\Gamma_1, w - q) = L_f(\Gamma_1, s - q) = \{a^{2n}b^{2m} \mid n \geq 1, m \geq 1, |n - m| \leq q\}.$$

Note that each component of Γ_1 is regular whereas the q-fair languages generated by Γ_1 are not regular.

Example 2 Let Γ_2 be the grammar system having the components

$$\begin{aligned} P_1 &= \{S \to AB, A \to aAb, A \to ab\}, \\ P_2 &= \{B \to C, C \to cC, C \to c\}. \end{aligned}$$

Clearly, for all $f \in \{t, =1, \geq 1\} \cup \{\leq k \mid k \geq 1\}$ and $q \geq 0$,

$$L_f(\Gamma_2) = L_f(\Gamma_2, w-q) = \{a^n b^n c^m \mid n, m \geq 1\}$$

holds (use each component exactly once) whereas

$$L_f(\Gamma_2, s-q) = \{a^n b^n c^m \mid n, m \geq 1, n = m + p \text{ or } m = n + p, 0 \leq p \leq q\}.$$

We mention that the languages $L_f(\Gamma_2, s-q)$ of the context-free grammar system Γ_2 are not context-free. Moreover, for all $k \geq 2$, we have

$$\begin{aligned} L_{=k}(\Gamma_2) &= \{a^n b^n c^m \mid n = r_1 \cdot k - 1, m = r_2 \cdot k - 1, r_1 \geq 1, r_2 \geq 1\}, \\ L_{=k}(\Gamma_2, w-q) &= \{a^{r_1 \cdot k - 1} b^{r_1 \cdot k - 1} c^{r_2 \cdot k - 1} \mid r_1,\ r_2 \geq 1,\ |r_1 - r_2| \leq q\}, \\ L_{=k}(\Gamma_2, s-q) &= \{a^{r_1 \cdot k - 1} b^{r_1 \cdot k - 1} c^{r_2 \cdot k - 1} \mid r_1,\ r_2 \geq 1,\ |(r_1 - r_2) \cdot k| \leq q\}. \end{aligned}$$

We add some remarks to the definitions.

1. The above definitions assume that the grammar system has at least two components.

2. If f is the mode $= k$, then the weak and strong concepts of fairness are nearly related to each other because

   $$L_{=k}(\Gamma, w-q) = L_{=k}(\Gamma, s-q')$$

   holds for $k \cdot q \leq q' < k \cdot (q+1)$. Particularly,

   $$L_{=k}(\Gamma, w-0) = L_{=k}(\Gamma, s-0).$$

3. It is also possible to allow the value ∞ for q. Thus, we get the equalities

   $$CDL_n(X, f) = CDL_n(X, F, w-\infty) = CDL_n(X, f, s-\infty).$$

In the sequel, we are going to investigate the influence of the fairness limitation on the generative power.

3 The regular case

Let

$$M_1 = \{t, =1, \geq 1\} \cup \{\leq k \mid k \geq 1\}.$$

First we recall that

$$CDL_n(REG, f) = REG$$

for all $n \geq 1, f \in M$. We now show that the situation is very different if we require a fair behaviour of the systems.

Theorem 1 *(i) REG and $CDL_n(REG, f, s-0)$ are incomparable, for all $f \in M$ and $n \geq 2$.*

(ii) $REG \subset CDL_n(REG, f, s-q)$, for all $f \in M_1$, $q \geq 1$, $n \geq 2$.

(iii) REG and $CDL_n(REG, f, s-q)$ are incomparable, for all $f \in M \setminus M_1$, $q \geq 1$, and $n \geq 2$.

Proof. It is easy to observe that any language in $CDL_n(REG, f, s-0)$, $f \in M$, contains only words of length divisible by n. Thus the regular language $L = \{a^m | m \geq 1\}$ does not belong to the class $CDL_n(REG, f, s-0)$.

On the other hand, the grammar system Γ consisting of the following components:

$$\begin{aligned} P_1 &= \{S \to aS, S \to aS_2\}, \\ P_i &= \begin{cases} \{S_i \to bS_i, S_i \to bS_{i+1}\}, & 2 \leq i \leq n-1, \ i \text{ is even} \\ \{S_i \to aS_i, S_i \to aS_{i+1}\}, & 2 \leq i \leq n-1, \ i \text{ is odd} \end{cases} \\ P_n &= \begin{cases} \{S_n \to bS_n, S_n \to b\}, & \text{if } n \text{ is even} \\ \{S_n \to aS_n, S_n \to a\}, & \text{if } n \text{ is odd} \end{cases} \end{aligned}$$

generates the 0-fair languages

$$L_{f_1}(\Gamma, s-0) = \begin{cases} \{(a^m b^m)^{\frac{n}{2}} \mid m \geq 1\}, & \text{if } n \text{ is even} \\ \{(a^m b^m)^{\frac{n-1}{2}} a^m \mid m \geq 1\}, & \text{if } n \text{ is odd} \end{cases}$$
$$\text{for} \quad f_1 \in M_1,$$

$$L_{=k}(\Gamma, s-0) = \begin{cases} \{(a^{mk} b^{mk})^{\frac{n}{2}} \mid m \geq 1\}, & \text{if } n \text{ is even} \\ \{(a^{mk} b^{mk})^{\frac{n-1}{2}} a^{mk} \mid m \geq 1\}, & \text{if } n \text{ is odd} \end{cases}$$
$$\text{for} \quad k \geq 1,$$

$$L_{\geq k}(\Gamma, s-0) = \begin{cases} \{(a^m b^m)^{\frac{n}{2}} \mid m \geq k\}, & \text{if } n \text{ is even} \\ \{(a^m b^m)^{\frac{n-1}{2}} a^m \mid m \geq k\}, & \text{if } n \text{ is odd} \end{cases}$$
$$\text{for} \quad k \geq 1,$$

which are not regular.

We prove now that the family of regular languages is contained in the families of q-fair languages generated in the f-mode, $f \in M_1$, by regular grammar systems as soon as $q \geq 1$.

For a regular grammar $G = (N, T, S, P)$ we construct the grammar system with regular components

$$\Gamma = (N', T, S_1, P_1, P_2, \ldots, P_n),$$

with

$$N' = \{A_i \mid A \in N, 1 \leq i \leq n\}$$

and the sets of productions

$$\begin{aligned} P_i &= \{A_i \to aB_{i+1} \mid A \to aB \in P\} \cup \{A_i \to a \mid A \to a \in P\}, \text{ for } 1 \le i \le n-1, \\ P_n &= \{A_n \to aB_1 \mid A \to aB \in P\} \cup \{A_n \to a \mid A \to a \in P\} \end{aligned}$$

By this construction any component performs exactly one step and the i-th, $1 \le i \le n-1$, and n-th component are followed by the $(i+1)$-st and first component, respectively. Thus, $L(G) = L_f(\Gamma, s-q)$, for all $q \ge 1$ and $f \in M_1$.

Now, it suffices to note that

$$L_f(\Gamma, s-q) = \begin{cases} \{a^{m_1}b^{m_2}a^{m_3}b^{m_4}\dots b^{m_n} \mid m_i \ge 1, |m_i - m_j| \le q, 1 \le i,j \le n\}, \\ \text{if } n \text{ is even} \\ \\ \{a^{m_1}b^{m_2}a^{m_3}b^{m_4}\dots a^{m_n} \mid m_i \ge 1, |m_i - m_j| \le q, 1 \le i,j \le n\}, \\ \text{if } n \text{ is odd} \end{cases}$$

for $f \in M_1$, are not regular languages.

In order to prove the last assertion let us remark that for any $k \ge 2$, every language in $CDL_n(REG, = k, s-q) \cup CDL_n(REG, \ge k, s-q)$ contains words of length greater than k, only. Therefore, $REG \setminus (CDL_n(REG, = k, s-q) \cup CDL_n(REG, \ge k, s-q)) \ne \emptyset$, for all $n \ge 2, q \ge 1$. The grammar system considered in the proof of the first statement provides languages for the converse part. Consequently, the proof is complete. □

Theorem 2 *(i) $CDL_2(REG, f, w-0) \subset REG$, for $f \in \{t, \ge 1\}$.*

(ii) The families $CDL_n(REG, f, w-0)$ and REG are incomparable, for all $n \ge 2$ and $f \in \bigcup_{k \ge 1}\{\le k, = k\}$.

(iii) For $f \in \{t, \ge 1\}$ and $n \ge 4$, REG and $CDL_n(REG, f, w-0)$ are incomparable.

Proof. (i) For $\Gamma = (N, T, S, P_1, P_2)$ consider the right linear grammars

$$G_1 = (\{S\} \cup \{A', A'' | A \in N\}, T, S, P)$$

where

$$\begin{aligned} P &= \{A' \to aB' | A \to aB \in P_1\} \cup \{A' \to aB' | A \to aB \in P_2, A \ne S\} \cup \\ &\cup \{A' \to a | A \to a \in P_2\} \cup \{A'' \to aB'' | A \to aB \in P_2\} \cup \{S \to S', S \to S''\} \\ &\cup \{A'' \to aB'' | A \to aB \in P_1, A \ne S\} \cup \{A'' \to a | A \to a \in P_1\} \end{aligned}$$

and

$$G_2 = (\{S\} \cup \{A_i, A'_i | A \in N, i = 1, 2\}, T, S, P \cup \{S \to S_1, S \to S'_2\})$$

where

$$\begin{array}{rcl} P & = & \{A_1 \to aB_1 | A \to aB \in P_1, B \in dom(P_1)\} \cup \\ & \cup & \{A_1 \to aB_2 | A \to aB \in P_1, B \notin dom(P_1)\} \cup \\ & \cup & \{A_2 \to aB_2 | A \to aB \in P_2, B \in dom(P_2)\} \cup \\ & \cup & \{A_2 \to aB_1 | A \to aB \in P_2, B \notin dom(P_2)\} \cup \\ & \cup & \{A_2 \to a | A \to a \in P_2\} \cup \\ & \cup & \{A'_2 \to aB'_2 | A \to aB \in P_2, B \in dom(P_2)\} \cup \\ & \cup & \{A'_2 \to aB'_1 | A \to aB \in P_2, B \notin dom(P_2)\} \cup \\ & \cup & \{A'_1 \to aB'_1 | A \to aB \in P_1, B \in dom(P_1)\} \cup \\ & \cup & \{A'_1 \to aB'_2 | A \to aB \in P_1, B \notin dom(P_1)\} \cup \\ & \cup & \{A'_1 \to a | A \to a \in P_1\} \end{array}$$

The equalities

$$L_{\geq 1}(\Gamma, w-0) = L(G_1) \quad \text{and} \quad L_t(\Gamma, w-0) = L(G_2)$$

follow immediately. The inclusions are proper since any language in $CDL_2(REG, f, w-0)$, $f \in \{t, \geq 1\}$, contains no word of length 1.

(ii) The second statement is completely proved if we provide, for all $k \geq 1, n \geq 2$, a non-regular language in $CDL_n(REG, \leq k, w-0)$. To this end, let us consider two cases.

• $n = 2$. The grammar system Γ identified by the following regular components

$$P_1 = \{S \to aS, S \to aB\}$$

$$P_2 = \{B \to bB, B \to b\}$$

generates in the $\leq k$-mode the non-regular language

$$L_{\leq k}(\Gamma, w-0) = \{a^t b^m | 1 \leq t \leq m \leq kt \text{ or } 1 \leq m \leq t \leq km\}$$

• $n > 2$. The grammar system Γ identified by the following regular components

$$\begin{array}{rcl} P_1 & = & \{S \to aS_2\} \\ P_i & = & \{S_i \to aS_{i+1}\}, 2 \leq i \leq n-2, \\ P_{n-1} & = & \{S_{n-1} \to aS, S_{n-1} \to aB\} \\ P_n & = & \{B \to bB, B \to b\} \end{array}$$

Observe that

$$L_{\leq k}(\Gamma, w-0) = \{a^{t(n-1)} b^m | 1 \leq t \leq m \leq kt \text{ or } 1 \leq m \leq t \leq km\}$$

which concludes the proof of the second item.

(iii) Any language in $CDL_n(REG, f, w-0)$, $f \in \{t, \geq 1\}$, contains only words of length at least n since any component has to be applied at least once. Thus the regular language $\{a, a^2, \ldots, a^n\}$ does not belong to the class $CDL_n(REG, f, w-0)$.

If $n = 4$, then the statement follows from Example 1.

If $n > 4$, then we subsitute the component P_2 of the grammar system Γ_1 in Example 1 by the components

$$\begin{aligned} P_{2,1} &= \{A' \to aA_2\}, \\ P_{2,i} &= \{A_i \to aA_{i+1}\} \quad \text{for} \quad 2 \leq i \leq n-4, \\ P_{2,n-3} &= \{A_{n-3} \to aA\} \end{aligned}$$

and obtain the grammar system Γ_1' which generates the non-regular 0-fair language

$$L_f(\Gamma_1', f, w-0) = \{a^{(n-2)m} b^{2m} \mid m \geq 1\}.$$

□

By using similar ideas as those involved in the previous proofs one can get:

Theorem 3 *For all $q \geq 1$ we have:*

$$\begin{aligned} (i)\ REG &= CDL_2(REG, f, w-q), f \in \{t, \geq 1\}. \\ (ii)\ REG &\subset CDL_n(REG, \leq k, w-q), n \geq 2, k \geq 1. \\ (iii)\ REG &\subset CDL_m(REG, f, w-q), m \geq 4, f \in \{t, \geq 1\}. \end{aligned}$$

At the end of this section we would like to mention that when considering grammar systems with right-linear components (i. e. containing rules of the forms $A \to xB, A \to x$, $x \in T^*$, $A, B \in N$) the results are similar to those given in the following section for the context-free case.

4 The context-free case

We start with a theorem which states a situation for context-free grammar systems which differs from that in the regular case.

Theorem 4 *For all $n \geq 2$, $q \geq 0$, $u \in \{w, s\}$ and $f \in M$,*

$$CDL_n(CF, f, u-q) \subseteq CDL_{n+1}(CF, f, u-q).$$

Proof. • $u = w$. For a CD grammar system $\Gamma = (N, T, S, P_1, P_2, \ldots, P_n)$, we construct the system

$$\Gamma' = (N', T, S, P_1', P_2, P_3, \ldots, P_n, P_{n+1}'),$$

with

$$\begin{aligned} N' &= N \cup \{X\} \quad \text{with } X \notin N, \\ P_1' &= P_1 \cup \{A \to wX \mid A \to w \in P_1\}, \\ P_{n+1}' &= \{X \to \lambda\} \end{aligned}$$

Obviously, $L_f(\Gamma) = L_f(\Gamma')$ and, because P_1' can be used as often as P_{n+1}', we obtain $L_f(\Gamma, w-q) = L_f(\Gamma', w-q)$.

• $u = s$. For a CD grammar system $\Gamma = (N, T, S, P_1, P_2, \ldots, P_n)$, we construct the system

$$\Gamma'' = (N'', T, S', P_1'', P_2'', \ldots, P_n'', P_{n+1}''),$$

with

$$\begin{aligned} N'' &= N \cup \{X, S'\} \quad \text{with } X, S' \notin N, \\ P_i'' &= P_i \cup \{S' \to wX \mid S \to w \in P_i\}, \quad 1 \leq i \leq n, \\ P_{n+1}'' &= \{X \to X, X \to \lambda\} \end{aligned}$$

Obviously, $L_f(\Gamma) = L_f(\Gamma')$ and, because the new introduced component P_{n+1}'' can work as long as we want, we infer $L_f(\Gamma, s-q) = L_f(\Gamma'', s-q)$. □

Theorem 5 *i) For $n \geq 2$, $q \geq 0$, $u \in \{w, s\}$ and $f \in M_1$,*

$$CDL_n(CF, f) \subseteq CDL_n(CF, f, u-q).$$

ii) The aforementioned inclusion is proper in the following cases:

$$\begin{array}{lllll} a) & u = w, & n \geq 4, & f \neq t, & q \geq 0, \\ b) & u = s, & n \geq 2, & f \neq t, & q \geq 0, \\ c) & u = w, & n \geq 6, & f = t, & q = 0, \\ d) & u = s, & n = 2 \text{ or } n \geq 7, & f = t, & q = 0. \end{array}$$

Proof. First we recall that, for $n \geq 2$, $m \geq 3$ and $f \in M_1 \setminus \{t\}$,

$$CDL_2(CF, t) = CDL_n(CF, f) = CF \quad \text{and} \quad CDL_m(CF, t) = ETOL. \qquad (1)$$

i) First we consider the case $u = w$ and $f \neq t$.

For a context-free grammar $G = (N, T, S, P)$, we construct the CD grammar system Γ with the following two components

$$\begin{aligned} P_1 &= \{A \to wX \mid A \to w \in P\}, \\ P_2 &= \{X \to \lambda\} \end{aligned}$$

where X is an additional nonterminal. Obviously, G and Γ generate the same language and, moreover, any word can be derived in Γ by using each component exactly once. This proves $L = L(G) = L_f(\Gamma, w-q)$.

Therefore, from (1) and Theorem 4, it follows that

$$CDL_n(CF,f) = CF \subseteq CDL_2(CF,f,w-q) \subseteq CDL_n(CF,f,w-q).$$

This proof can be carried over the cases $u = w$, $f = t$ and $n = 2$.

We now consider the case $u = w$, $f = t$ and $n \geq 3$. Let $L \in CDL_n(CF,t)$. By (1), L is generated by some ETOL system

$$G = (V,T,S,T_1,T_2,\ldots,T_m)$$

with the alphabet V, containing the set T of terminals, the start word S which can be assumed without loss of generality as an element of $V \setminus T$ and the tables $T_1, T_2, \ldots T_m$.

For $a \in V$, $0 \leq i \leq m$ and $1 \leq k \leq 5$, we introduce the new letters a_i^k and extend this inductively to words by

$$w_i^k = \begin{cases} \lambda & \text{for } w = \lambda \\ a_i^k & \text{for } w = a \\ v_i^k a_i^k & \text{for } w = va,\ v \in V^*,\ a \in V \end{cases} .$$

We now construct the CD grammar system

$$\Gamma = (N,T,S_0^1,P_1,P_2,P_3\}$$

with

$$\begin{aligned}
N &= \{F\} \cup \{a_i^k \mid a \in V, 0 \leq i \leq m, 1 \leq k \leq 5\}, \\
P_1 &= \{a_i^1 \to a_i^2 \mid 0 \leq i \leq m, a \in V\} \cup \{a_i^4 \to a_i^5 \mid 0 \leq i \leq m, a \in V\}, \\
P_2 &= \{a_i^2 \to a_i^3 \mid 0 \leq i \leq m, a \in V\} \cup \{a_i^1 \to a_i^4 \mid 0 \leq i \leq m, a \in V\}, \\
P_3 &= \bigcup_{i=1}^{m} \{a_i^3 \to w_0^1 \mid a \to w \in T_i\} \cup \{a_0^3 \to a \mid a \in T\} \\
&\quad \cup \{a_0^3 \to F \mid a \in V \setminus T\} \cup \{a_i^5 \to a_{i+1}^1 \mid 1 \leq i \leq m, a \in V\}.
\end{aligned}$$

Let us consider a word of the form x_i^1. Note that the axiom is of this form.

If we apply the component P_1, we obtain x_i^2 and we have to apply P_2 and P_3 in succession. If $i \geq 1$, then this yields y_0^1 where $x \Longrightarrow_{T_i} y$ is a derivation step in the ETOL system G, i.e. we have simulated the application of table T_i to x. If $i = 0$ then we obtain x or a word containing F according whether $x \in T^+$ or not and the derivation is finished.

If we apply the component P_2, we have to apply P_1 and P_3 and obtain x_{i+1}^1.

From this explanation it is easy to see that we can simulate the application of an arbitrary sequence of tables and finish the derivation by using any of the three components exactly once for the simulation of one step. Hence

$$L_t(\Gamma, w-q) = L_t(\Gamma, w-0) = L(G) = L$$

holds for any q which proves the statement.

Now let $u = s$ and $f \in M_1$. Let $\Gamma = (N, T, S, P_1, \dots, P_n)$, be a CD grammar system. The grammar system

$$\Gamma' = (N', T, S, P_1', \dots, P_n'),$$

with

$$\begin{aligned} N' &= N \cup \{X\} \quad \text{with } X \notin N, \\ P_i' &= P_i \cup \{A \to wX \mid A \to w \in P_i\} \cup \{X \to X, X \to \lambda\} \quad \text{for } 1 \leq i \leq n \end{aligned}$$

generates the same language as Γ. Therefore

$$L_f(\Gamma', f, s - q) \subseteq L_f(\Gamma)$$

for all $q \geq 0$.

By using the rules $X \to X$ and $X \to \lambda$, for any $w \in L_f(\Gamma')$, we can find a derivation D such that $ds(D) = 0$. Hence

$$L_f(\Gamma) = L_f(\Gamma') \subseteq L_f(\Gamma', s - q).$$

Consequently, for any $q \geq 0$,

$$L_f(\Gamma) = L_f(\Gamma', s - q)$$

which concludes the inclusions $CDL_n(CF, f) \subseteq CDL_n(CF, f)$ for all $n \geq 2$.

ii) a) The CD grammar system Γ with the components

$$\begin{aligned} P_1 &= \{S \to aA_1c, A \to aA_1c\}, \\ P_i &= \{A_i \to A_{i+1}\}, \quad 2 \leq i \leq n-3, \\ P_{n-2} &= \{A_{n-3} \to A\}, \\ P_{n-1} &= \{A \to bB', B \to bB'\}, \\ P_n &= \{B' \to B, B' \to \lambda\} \end{aligned}$$

generates in any mode $f \in M_1 \setminus \{t\}$ the weakly q-fair language

$$L_f(\Gamma, w - q) = \{a^n b^m c^n \mid n \geq 1, m \geq 1, |n - m| \leq q\}$$

which is not context-free. Now the statement follows from (1).

b) The statement follows from Example 2, Theorem 4 and (1).

c) We consider the CD grammar system

$$\Gamma = (\{A, A', A'', B, B', D, D', F\}, \{a, b, d\}, A, P_1, P_2, P_3, P_4, P_5, P_6)$$

with

$$\begin{aligned}
P_1 &= \{A \rightarrow BA', A \rightarrow BA''\},\\
P_2 &= \{A' \rightarrow A, A'' \rightarrow D\},\\
P_3 &= \{A \rightarrow F, A' \rightarrow F, A'' \rightarrow F, B \rightarrow B'b, D \rightarrow dD'\},\\
P_4 &= \{B' \rightarrow B, D' \rightarrow D, D' \rightarrow d\},\\
P_5 &= \{B' \rightarrow a, D' \rightarrow D\},\\
P_6 &= \{B \rightarrow F, B' \rightarrow F, D \rightarrow dD'\}.
\end{aligned}$$

It is easy to see that any derivation where any component works exactly n times is given by the sequence

$$(P_1P_2)^n(P_3P_4)^m P_3P_5(P_3P_4)^{n-m-1}(P_6P_5)^{n-1}P_6P_4$$

of components where $m < n$. Hence Γ generates in the t-mode the weakly 0-fair language

$$L' = \{(ab^{m+1})^n d^{2n+1} \mid n \geq m+1 \geq 1\}.$$

Using the closure properties of the family of ETOL languages and Corollary 2.2 of Part V in [7] we obtain that $L' \notin ETOL$. By i), (1) and Theorem 4, for $n \geq 6$,

$$CD_n(CF,t) = ETOL \subset CDL_6(CF,t,0-q) \subseteq CDL_n(CF,t,0-q)$$

follows.

d) The strict inclusion for $n = 2$ follows from Example 2.

We shall prove the strict inclusion for $n = 7$, hence all inclusions for $n \geq 7$ are consequences of the previous theorem.

Let us consider the CD grammar system

$$\begin{aligned}
P_1 &= \{S \rightarrow CAZXY, Y \rightarrow Y\},\\
P_2 &= \{C \rightarrow BC', X \rightarrow C'C'AZX', Y \rightarrow Y\},\\
P_3 &= \{C' \rightarrow C, X' \rightarrow X, Y \rightarrow Y\},\\
P_4 &= \{B \rightarrow \lambda, C \rightarrow \lambda, Z \rightarrow \lambda\},\\
P_5 &= \{A \rightarrow aA', A' \rightarrow bD', D \rightarrow bD', Y \rightarrow Y\},\\
P_6 &= \{D' \rightarrow D, Y \rightarrow Y, X \rightarrow \lambda\},\\
P_7 &= \{D \rightarrow \lambda, Y \rightarrow Y, Y \rightarrow \lambda\}.
\end{aligned}$$

Here are some explanations about the working mode of the above system. The sets P_1, P_2, P_3 are used in order to obtain strings α with

$$|\alpha|_{\{B,C\}} = m^2 \quad \text{and} \quad |\alpha|_A = |\alpha|_Z = m$$

for all $m \geq 1$. Every terminal derivation has to use the component P_4 only once but for $m(m+1)$ steps. On the other hand, each component P_i, $i \in \{1,2,3\}$ can be used either once or several times for a total amount of $m(m+1)$ steps.

The component P_5 is used first time for at least $2m$ steps and all the other times for at least m steps.

The component P_6 is used each time for at least m steps. Because, P_5 and P_6 are used together for introducing b's, the total number of b's in the terminal words of $L_t(\Gamma, s-0)$ is m^2. In conclusion,

$$L_t(\Gamma, s-0) = \{(ab^n)^m \mid m \geq n \geq 1\}$$

which is not an $ET0L$ language (see [7]). Now the result follows due to (1). □

As one can see there remain plenty of open problems in this area. We do not list them since the reader can easily identify them or can invent others.

Finally let us mention that there also are some other concepts of fairness which can be introduced.

- For a given CD grammar system $\Gamma = (N, T, S, P_1, P_2, \ldots, P_n)$, we fix some integers $r_1, r_2, \ldots, r_n$ and require that, for $1 \leq i \leq n$, the component P_i is applied at most or exactly r_i times. However, since the application of a component in one of the modes is equivalent to a finite or context-free substitution we obtain only finite or context-free languages and, obviously, we obtain all languages of this type.

- To each component P_i, $1 \leq i \leq n$, and to any moment l of time, $l \geq 0$ (this corresponds to the number of applications of components), we associate an integer $t_i(l)$ in the following way: Let $t_i(0) = 0$. If we apply the component P_j in the moment l, then we set $t_j(l+1) = 0$ and increase the number of all the other components, i.e. $t_i(l+1) = t_i(l) + 1$ for $i \neq j$. The fairness now consists in the requirement that in each moment l we can only apply a component P_i such that $t_i(l) = \max_{1 \leq p \leq n} t_p(l)$. The number $t_i(l)$ can be interpreted as the period during that the component P_i was not active, i.e. it is the waiting time of the component, and we can apply a component only if it has been waiting a maximal amount of time.

  Clearly, after the first activation, any component has to work after waiting $n-1$ steps. Thus this concept is nearly related to the weak fairness. We only mention that – by using the same proofs – one can show that similar statements as for weak fairness hold.

References

[1] A. Atanasiu, V. Mitrana, The modular grammars. *Intern. J. Computer Math.* 30 (1989) 17–35.

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

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

[4] J. Dassow, J. Kelemen, Cooperating distributed grammar systems: a link between formal languages and artificial intelligence. *Bulletin of EATCS* 45 (1991) 131–145.

[5] R. Meersman, G. Rozenberg, Cooperating grammar systems. In: *Proc. MFCS 78*, LNCS 64, Springer-Verlag, Berlin, 1978, 364–374.

[6] P. H. Nii, Blackboard systems. In: *The Handbook of AI*, vol. 4 (Eds.: A. Barr, P. R. Cohen, E. A. Feigenbaum), Addison-Wesley, Reading, Mass., 1989.

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

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

Acta Cybernetica **12** (1996) 347-360.

On Competence and Completeness in CD Grammar Systems*

Henning BORDIHN † Erzsébet CSUHAJ-VARJÚ ‡ §

Abstract

In this paper, different concepts of t-mode derivations in CD grammar systems which can be encountered in the literature and generalizations thereof are considered both in generating and in accepting case. Moreover, the influence of completeness of the components as an additional requirement to the derivational capacity of CD grammar systems is investigated.

1 Introduction

The theory of grammar systems is a recent vivid field of formal language theory describing multi-agent symbol systems by tools of formal grammars and languages ([4]). Cooperating/distributed grammar systems, (CD grammar systems, for short) is one the important subfields of the area, launched for syntactic modelling distributed problem solving systems based on blackboard architectures ([3]). We note, however, that the term "cooperating grammars" was introduced first in [9], as a generalization of two-level substitution grammars to a multi-level concept.

A CD grammar system consists of a finite set of grammars that cooperate in deriving words of a common language. At any moment in time there is exactly one sentential form in derivation and the grammars work on this string in turns, according to some cooperation protocol. In this model, the cooperating grammars correspond to the cooperating independent problem solving agents, the sentential form in derivation represents information on the current state of the problem solving stored in a global database, the blackboard, and the obtained language describes the set of problem solutions.

*Research supported by the German-Hungarian Research Project "Formal Languages, Automata and Petri-Nets" (1995-1997), No. D/102 (formerly: No. OMFB-NPI-102) of the TÉT Foundation, Budapest, Hungary, and No. 233.6. of Forschungszentrum Karlsruhe, Germany

†Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, P.O.Box 4120, D-39016 Magdeburg, Germany. E-mail:bordihn@irb.cs.uni-magdeburg.de

‡Research supported by Hungarian Scientific Research Fund OTKA T 017105

§Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende u. 13-17, H-1111 Budapest, Hungary. E-mail: csuhaj@sztaki.hu

Turning to the original motivation, it is a sensible question whether some important properties and features of agents that influence the behaviour of blackboard-type problem solving systems can be formalized and interpreted in the syntactic framework provided by CD grammar systems. In this paper we deal with two of these properties: competence and completeness of components, moreover, we study them both in the case of generating CD grammar systems and in the case of accepting ones.

The idea of accepting grammars and systems ([11]) is the following: Starting from a "terminal" word, the system tries to derive a given goal word (the axiom) where, according to [1, 2, 6], the yield relation is defined by textually the same words as in generating case. Possible restrictions to production sets are turned "around", e.g., coming to productions of the form $v \rightarrow a$ in the context-free case, where v is a (possibly empty) word and a is a symbol.

CD grammar systems as accepting devices (CD grammar systems consisting of accepting grammars), corresponding to backward deduction systems in contrast to generating CD grammar systems which correspond to forward deduction systems, were considered in [8].

Both competence and completeness can form a basis of the cooperation protocol. According to our approach, an agent is competent in a current state of the problem solving if it is able to contribute to the problem solution. In grammatical terms, the component grammar is competent in the derivation of the current sentential form if it is able to apply at least one of its productions to it.

So far there have been two kinds of cooperation protocols (t-modes of derivation) based on competence/incompetence of grammars introduced and examined: in the first case (hard t-mode), a grammar can start with the derivation if it is competent in the sentential form and stops with the derivation if it is no longer competent in the actual string (it has no production to apply; the agent is not able to contribute to the problem solving). In the second case, the start condition is the same but the stop condition differs: the grammar finishes the derivation if it is not able to derive a word different from the actual one (the competence of the grammar is not enough to change the state of the problem solving). We generalize the latter concept to a cooperation protocol called stagnation derivation mode (s-mode), where a component has to continue its work until and unless a word is derived from which no new word can be rewritten, i.e., it is impossible to derive a word which does not appear in the derivation before. Thus, the competence of the agent (of the component grammar) is not enough to leave a stagnating phase of the problem solving (the derivation).

In this paper we compare the power of context-free CD grammar systems working on the basis of the above cooperation protocols. We show that the three variants are equally powerful. In the case of generating CD grammar systems they identify the class of ET0L languages and in the case of accepting CD grammar systems they provide a description of the class of context-sensitive languages (supposing that λ-free productions are taken into account). These results, in the case of weak t-mode and stagnation mode of derivation do not change if we incorporate some requirement concerning completeness of the components.

The notion of completeness is well-known from Lindenmayer systems: For a set of productions of a usual L system it is required that, for any symbol a of the alphabet, there is at least one production rule replacing a. In [6], accepting L systems were investigated, where this concept of completeness does not apply any more.

Since a production set of an accepting (ET)0L system is seen as an inverse finite substitution it is required that, for any symbol a, there is at least one rule of the form $v \longrightarrow a$. One might wish to replace this "right-completeness" condition by a "left-completeness" condition as one is used to have in generating case, but such a condition must be in accord with the finiteness of the set of productions.

One sensible approach to define completeness can be inherited from [9], where the derivation strategy of the cooperating generative context-free grammars is defined as follows: every component grammar can start with the derivation if it is full competent in the generation of the current sentential form (if it has a production for any nonterminal symbol appearing in the actual string) and stops with it if it is no longer satisfies this criteria (there is at least one nonterminal in the string for which the grammar has no rewriting rule).

Clearly, this idea can be transferred to the accepting case (also applying to accepting Lindenmayer systems): a grammar component / set of productions is complete (thus full competent) for the current sentential form iff this sentential form can be partitioned into non-overlapping subwords each of which can be rewritten by a rule in the production set. We call this concept, that exhibits both completeness and competence, sentential-form-completeness (sf-completeness for short). Obviously, in generating case this concept coincides with the usual "left-completeness" condition known from L systems.

In [9] it is shown that context-free CD grammar systems with components working in sf-mode of derivation are equally powerful to the class of context-free programmed grammars with appearance checking. We show that in the case of accepting context-free CD grammar systems this protocol leads to the power of context-sensitive grammars, morevover, to reach this capacity at most five cooperating grammars are sufficient.

2 Basic definitions

We assume that the reader is familiar with the basic notions of formal language theory, for further details we refer to [11], and [5]. With our notations, we mostly follow [5]. Especially, we use $\subseteq$ to denote inclusion, $\subset$ to denote strict inclusion, and λ to denote the empty word. The length of a word w is denoted by $|w|$, **N** denotes the set of positive integers. Two languages L_1 and L_2 are considered to be equal iff $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$.

The family of languages generated by regular, context-free, context-sensitive, type-0 Chomsky grammars, ET0L systems, context-free programmed grammars, and context-free programmed grammars with appearance checking are denoted by $\mathcal{L}^{\text{gen}}(\text{REG})$, $\mathcal{L}^{\text{gen}}(\text{CF})$, $\mathcal{L}^{\text{gen}}(\text{CS})$, $\mathcal{L}^{\text{gen}}(\text{RE})$, $\mathcal{L}^{\text{gen}}(\text{ET0L})$, $\mathcal{L}^{\text{gen}}(\text{P,CF})$, and

$\mathcal{L}^{\text{gen}}$(P,CF,ac), respectively. In order to denote the family of languages accepted by a device of the corresponding type, we write the superscript *acc* instead of *gen*. If we want to exclude λ-productions, we add $-\lambda$ in our notations. Whenever we use bracket notations like $\mathcal{L}^{\text{gen}}(\text{P,CF}[-\lambda]) \subset \mathcal{L}^{\text{gen}}(\text{P,CF}[-\lambda],\text{ac})$ we mean that the statement is true both in the case of neglecting the bracket contents and in the case of ignoring the brackets themselves.

We define CD grammar systems in a way suitable for the interpretation of both generating and accepting systems.

A *CD grammar system* of degree n, with $n \geq 1$, is an $(n+3)$-tuple

$$G = (N, T, S, P_1, \ldots, P_n),$$

where N and T are two disjoint alphabets, the set of nonterminal and terminal symbols, respectively, $V = N \cup T$ is the total alphabet of G, $S \in N$ is the axiom, and $P_1, \ldots, P_n$ are finite sets of rewriting rules of the form $\alpha \rightarrow \beta$, $\alpha, \beta \in (N \cup T)^+$. In addition, we allow λ-rules of the form $\alpha \rightarrow \lambda$ in the generating case and $\lambda \rightarrow \beta$ in the accepting case. For $x, y \in (N \cup T)^*$, we write $x \underset{i}{\Longrightarrow} y$ iff $x = x_1\alpha x_2$, $y = x_1\beta x_2$ for some $\alpha \rightarrow \beta \in P_i$. Hence, subscript i refers to the production set (component) to be used. Furthermore, we denote by $\overset{f}{\underset{i}{\Longrightarrow}}$ a derivation executed by the i-th component according to some cooperation protocol f. For example, $\overset{\leq k}{\underset{i}{\Longrightarrow}}$ ($\overset{\geq k}{\underset{i}{\Longrightarrow}}$, $\overset{=k}{\underset{i}{\Longrightarrow}}$, or $\overset{*}{\underset{i}{\Longrightarrow}}$, respectively) denotes a derivation of at most k (at least k, exactly k, or an arbitrary number of) derivation steps as above.

For some cooperation protocol f, the language *generated* in f-mode (e.g., in $\leq k$-mode) by a CD grammar system G of degree n is

$$\mathcal{L}_f^{\text{gen}}(G) = \{w \in T^* \mid S = w_0 \overset{f}{\underset{i_1}{\Longrightarrow}} w_1 \overset{f}{\underset{i_2}{\Longrightarrow}} \cdots \overset{f}{\underset{i_{m-1}}{\Longrightarrow}} w_{m-1} \overset{f}{\underset{i_m}{\Longrightarrow}} w_m = w \text{ with } m \geq 1,\ 1 \leq i_j \leq n,\ 1 \leq j \leq m\}.$$

The language *accepted* in f-mode by G is defined by

$$\mathcal{L}_f^{\text{acc}}(G) = \{w \in T^* \mid w = w_0 \overset{f}{\underset{i_1}{\Longrightarrow}} w_1 \overset{f}{\underset{i_2}{\Longrightarrow}} \cdots \overset{f}{\underset{i_{m-1}}{\Longrightarrow}} w_{m-1} \overset{f}{\underset{i_m}{\Longrightarrow}} w_m = S \text{ with } m \geq 1,\ 1 \leq i_j \leq n,\ 1 \leq j \leq m\}.$$

The families of languages generated (accepted, respectively) in the f-mode by CD grammar systems with at most n [λ – free] context-free components are denoted by $\mathcal{L}^{\text{gen}}(\text{CD}_n, \text{CF}[-\lambda], f)$ (or $\mathcal{L}^{\text{acc}}(\text{CD}_n, \text{CF}[-\lambda], f)$). If the number of components is not restricted then we write $\mathcal{L}^{\text{gen}}(\text{CD}_\infty, \text{CF}[-\lambda], f)$ ($\mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}[-\lambda], f)$).

3 Cooperation and Competence

In this section we compare the derivational capacity of CD grammar systems working under cooperation protocols based on competence/incompetence of the cooperating grammars. Since incompetence (disability of rewriting) realizes a terminating condition for the component grammar, these kind of cooperation protocols are called t-modes of derivations.

Let us have formal definitions.

A derivation

$$D:\ x = x_0 \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \cdots \Rightarrow x_{n-1} \Rightarrow x_n = y$$

from x to y is said to be

(i) of type *hard-t* (t_h) iff there is no z such that $y \Rightarrow z$,

(ii) of type *weak-t* (t_w) iff there is no $z \neq y$ such that $y \Rightarrow z$,

(iii) *stagnating* (s) iff there is no $z \notin \{x_i \mid 0 \leq i \leq n\}$ such that $y \Rightarrow^* z$.

Let $G = (N, T, S, P_1, \ldots, P_n)$ be a CD grammar system and let $f \in \{t_h, t_w, s\}$. For $x, y \in (N \cup T)^*$, we write $x \underset{i}{\overset{f}{\Longrightarrow}} y$ iff $x \underset{i}{\overset{*}{\Longrightarrow}} y$ and $x \underset{i}{\overset{*}{\Longrightarrow}} y$ is a derivation of type f.

Then, $\mathcal{L}^{\text{gen}}(\text{CD}_\infty, \text{CF}, t_w)$ is exactly the family of languages generated by CD grammar systems with an arbitrary number of context-free components working in t-mode as defined, e.g., in [4], whereas $\mathcal{L}^{\text{gen}}(\text{CD}_\infty, \text{CF}, t_h)$ equals the corresponding family as defined, e.g., in [8].

Observation 3.1 *By definition, if a derivation is of type t_h then it is of type t_w, and if it is of type t_w then it is stagnating.*
Hence, for a CD grammar system G, we have

(i) $L^{\text{gen}}_{t_h}(G) \subseteq L^{\text{gen}}_{t_w}(G) \subseteq L^{\text{gen}}_{s}(G)$ and

(ii) $L^{\text{acc}}_{t_h}(G) \subseteq L^{\text{acc}}_{t_w}(G) \subseteq L^{\text{acc}}_{s}(G)$. □

Note that productions of the form $A \rightarrow A$ block derivations in hard-t-mode whereas they can be neglected in CD grammar systems working in weak-t-mode. Nevertheless, we find the following lemma.

Lemma 3.2 *For $n \in \mathbf{N} \cup \{\infty\}$, we have*

$$\begin{aligned}\mathcal{L}^{\text{gen}}(CD_n, CF[-\lambda], t_h) &= \mathcal{L}^{\text{gen}}(CD_n, CF[-\lambda], t_w) \text{ and}\\ \mathcal{L}^{\text{acc}}(CD_n, CF[-\lambda], t_h) &= \mathcal{L}^{\text{acc}}(CD_n, CF[-\lambda], t_w)\end{aligned}$$

Proof. Suppose we have a CD grammar system working in t_w-mode. A simulating CD grammar system working in t_h-mode can be obtained by cancelling all rules of the form $A \rightarrow A$ from the productions sets of the original one. Conversely, if we replace all rules of the form $A \rightarrow A$ by $A \rightarrow F$, where F is a trap symbol, in a CD grammar system working in t_h-mode, we obtain a simulating grammar system working in the t_w-mode of derivations. □

In order to compare these families of languages with that one generated (accepted) by CD grammar systems working in s-mode, we take a better look at stagnating derivations.

Let

$$D: x = x_0 \Rightarrow x_1 \Rightarrow \cdots \Rightarrow x_n \Rightarrow \cdots$$

be a stagnating derivation which is not terminating. Then there is a finite language $L_{\text{stag}}(D)$ which consists of all words appearing in D such that, for any pair $(u, v) \in L_{\text{stag}}(D) \times L_{\text{stag}}(D)$, we have $u \overset{*}{\Rightarrow} v$ and $v \overset{*}{\Rightarrow} u$ in D.

Lemma 3.3 *Let G be a generating CD grammar system with context-free components. For any derivation D of G which is stagnating, all words in $L_{\text{stag}}(D)$ have one and the same length.*

Proof. Assume the contrary. Let u be a word of maximum length in $L_{\text{stag}}(D)$, i.e., if a rule is applicable to u then this rule has the form $A \to B$ or $A \to \lambda$, where A and B are symbols. Here, renaming rules $(A \to B)$ cannot introduce symbols B to which productions $B \to \beta$ with $|\beta| > 1$ can be applied, since otherwise u is not the word of maximum length in $L_{\text{stag}}(D)$. Thus, we can apply only rules of the form $A \to B$ and $A \to \lambda$ also to any word derived from u. In particular, this holds for any word $u' \in L_{\text{stag}}(D)$ with $|u'| < |u|$. This contradicts $u' \overset{*}{\Rightarrow} u$. □

In conclusion, if a derivation D is stagnating then there is a word in D such that only renaming rules can be applied in the sequel or it is a terminating derivation.

Lemma 3.4 $\mathcal{L}^{\text{gen}}(CD_\infty, CF[-\lambda], s) = \mathcal{L}^{\text{gen}}(ET0L)$.

Proof. The inclusion $\mathcal{L}^{\text{gen}}(\text{ET0L}) \subseteq \mathcal{L}^{\text{gen}}(\text{CD}_\infty, \text{CF}[-\lambda], s)$ follows by the construction given in [4, pp. 40–42] for the t_h-mode case.

Thus, it is left to prove $\mathcal{L}^{\text{gen}}(\text{CD}_\infty, \text{CF}[-\lambda], s) \subseteq \mathcal{L}^{\text{gen}}(\text{ET0L})$. Let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$ be a generating CD grammar system. For any nonterminal A and $1 \leq i \leq n$, we set $L^i_A = SF(G^i_A)$ where G^i_A is the context-free grammar $G^i_A = (N, T, P_i, A)$, i.e., L^i_A is the set of all sentential forms which can be generated by the i-th component of Γ starting with A. Furthermore, let

$$\begin{aligned} C^i_A &= \begin{cases} L^i_A & \text{iff } L^i_A \subseteq N \text{ and, for all } B \in L^i_A, \text{ we have } L^i_B = L^i_A \\ \emptyset & \text{otherwise} \end{cases}, \\ M^i &= \{B \in N \mid \text{there is no } \beta \text{ such that } B \to \beta \in P_i\}, \end{aligned}$$

and

$$N^i_{\text{stag}} = M^i \cup \bigcup_{A \in N} C^i_A .$$

Obviously, N^i_{stag} is the set of nonterminal symbols which induce stagnation of the i-th component, more precisely, if the i-th component is active, stagnation appears iff a sentential form $w \in (N^i_{\text{stag}} \cup T)^*$ has been derived. Now, construct the ET0L system $G = (V, T, \mathcal{P}, S)$, with $V = N \cup T \cup \{A_i \mid A \in N,\ 1 \leq i \leq n\} \cup \{F\}$ and, for $1 \leq i \leq n$, $\mathcal{P}$ contains the following tables:

$$\begin{aligned} P_{i,1} &= \{A \to A_i \mid A \in N\} \cup \{a \to a \mid a \in T\} \cup \{X \to F \mid X \notin N \cup T\} \\ P_{i,2} &= \{A_i \to h_i(w) \mid A \to w \in P_i\} \cup \{a \to a \mid a \in T\} \cup \{A_i \to A_i \mid A \in N\} \cup \\ &\quad \{A_j \to F \mid A \in N,\ j \neq i\} \cup \{A \to F \mid A \in N\} \cup \{F \to F\}, \end{aligned}$$

where h_i is the morphism defined by $h_i(A) = A_i$ for $A \in N$ and $h_i(a) = a$ for $a \in T$,

$$P_{i,3} \;=\; \{A_i \to A \mid A \in N^i_{\mathrm{stag}}\} \cup \{a \to a \mid a \in T\} \cup \{X \to F \mid X \notin N^i_{\mathrm{stag}} \cup T\}.$$

Here, table $P_{i,1}$ simulates the selection of component P_i of Γ by "colouring" the nonterminals in the sentential form (replacing them by their corresponding subscripted version), $P_{i,2}$ simulates the application of the chosen component, and $P_{i,3}$ allows the system to leave the i-th component iff stagnation appears. The rules with the trap symbol F on their right-hand sides forbid shortcuts. Thus, we have $L^{\mathrm{gen}}(G) = L^{\mathrm{gen}}(\Gamma)$. □

The proof for $\mathcal{L}^{\mathrm{gen}}(\mathrm{ET0L}) \subseteq \mathcal{L}^{\mathrm{gen}}(\mathrm{CD}_3, \mathrm{CF}[-\lambda], t)$ in [4, pp. 40–42] is given by a construction of a CD grammar system working in hard-t-mode with three components. Together with Observation 3.1, we can summarize the results as follows.

Corollary 3.5 *For $f \in \{t_h, t_w, s\}$, we have*

$$\begin{aligned}\mathcal{L}^{\mathrm{gen}}(ET0L) &= \mathcal{L}^{\mathrm{gen}}(CD_\infty, CF[-\lambda], t_h)\\ &= \mathcal{L}^{\mathrm{gen}}(CD_\infty, CF[-\lambda], t_w)\\ &= \mathcal{L}^{\mathrm{gen}}(CD_\infty, CF[-\lambda], s)\\ &= \mathcal{L}^{\mathrm{gen}}(CD_3, CF[-\lambda], f).\end{aligned}$$

Let us turn to the accepting case. Clearly, it is impossible to get any terminating or stagnating derivation of a component of an accepting CD grammar system if λ-rules, i.e., rules of the form $\lambda \to v$, are present. Hence, we restrict ourselves to the λ-free case. Moreover, it is a direct consequence that as in the case of generating CD grammar systems, also in the case of accepting CD grammar systems with context-free components it holds that all words in $L_{\mathrm{stag}}(D)$ have one and the same length for any stagnating derivation D.

In [8] it is shown how to construct an accepting CD grammar system with two context-free λ-free components working in hard-t-mode in order to simulate a given context-senstive grammar. On the other hand, it is obvious that any accepting CD grammar system with non-erasing context-free components in t_h- or t_w-mode can be simulated by a linear-bounded automaton. But even such a system working in s-mode can be simulated by a linear-bounded automaton. This fact is obvious if we take into consideration that a derivation is stagnating only if it is terminating or after deriving a certain sentential form the rules that can be applied to the sentential form are only certain renaming rules, that is, productions of the form $A \to B$, with $A, B \in N$. Thus, we easily find the next theorem.

Theorem 3.6 *For $f \in \{t_h, t_w, s\}$, we have*

$$\begin{aligned}\mathcal{L}^{\mathrm{gen}}(CF) = \mathcal{L}^{\mathrm{acc}}(CF) &= \mathcal{L}^{\mathrm{acc}}(CD_1, CF-\lambda, f)\\ &\subset \mathcal{L}^{\mathrm{acc}}(CD_2, CF-\lambda, f)\\ &= \mathcal{L}^{\mathrm{acc}}(CD_\infty, CF-\lambda, f)\\ &= \mathcal{L}^{\mathrm{gen}}(CS) = \mathcal{L}^{\mathrm{acc}}(CS).\end{aligned}$$

4 Cooperation and Completeness

In this section we investigate how the derivational capacity of CD grammar systems changes when the components satisfy some completeness criteria. We show that these additional conditions do not necessarily alter the derivational power, even in some t-mode cases, where, actually, the communication protocol is determined by the incompleteness of the components.

First, let us discuss the concept of completeness.

From the theory of (generating) (ET)0L systems we know a condition called *left-completeness*: For each symbol a of the alphabet V of the system there is at least one production $a \longrightarrow v$, $v \in V^*$. Analogously, *right-completeness* (originally for accepting (ET)0L systems) is defined: For each symbol a of the alphabet V there is at least one production $v \longrightarrow a$, $v \in V^*$. These conditions can be modified for CD grammar systems as follows: A set P of *generating* context-free productions (a component of the CD grammar system) is said to be left-complete (right-complete) iff there is at least one production of the form $A \longrightarrow \beta$ for each nonterminal A (at least one production of the form $A \longrightarrow x$ for each symbol x of the total alphabet, respectively) in P. A set P of *accepting* context-free productions is called left-complete (right-complete) iff there is at least one production of the form $x \longrightarrow A$ for each symbol x of the total alphabet (at least one production of the form $\alpha \longrightarrow A$ for each nonterminal A, respectively) in P.

We concede that the above concept of right-completeness for the generating case and that of left-completeness for the case of accepting sets of productions are not very satisfying. For, e.g., the accepting case, one might take into account the possibility to require the following: If $w \longrightarrow a$, with $|w| = k$, is in the set of productions P then $P \subseteq V^* \times V$ is surjective from $\bigcup_{i=1}^{k} V^i$ into V. This requirement is no restriction, since we can simply add rules of the form $v \longrightarrow F$ for such words $v \in \bigcup_{i=1}^{k} V^i$ which do not appear on left-hand sides in the given production set, and those "dummy rules" are out of any influence to the rewriting process. Moreover, we get no genuine completeness at all.

Another concept of completeness which is appropriate both for generating and for accepting devices can be defined, based on the cooperation protocol used in [9], as follows:

Let V be an alphabet and let $L \subseteq V^*$ be a language over V. A set of production rules P is said to be *sentential-form-complete* (*sf-complete*, for short) *with respect to* L iff every word $w \in L \setminus \{\lambda\}$ has a factorization $w = x_1 x_2 \cdots x_n$ such that, for each i, $1 \leq i \leq n$, there is a rule $x_i \longrightarrow \beta \in P$, with $\beta \in V^*$.

Then, e.g., in a (generating) E0L system $G = (V, \Sigma, P, \omega)$, P has to be *sf*-complete with respect to V^*.

Observation 4.1 *If a set of production rules P is left-complete then it is sf-complete with respect to $(domP)^*$, where $domP$ denotes the set of all symbols ap-*

pearing on left-hand sides of the rules in P.

For a generating context-free set P, we even find equivalence between these two properties. □

A CD grammar system $G = (N, T, S, P_1, \dots, P_n)$ is called left-complete or right-complete iff each P_i, $1 \leq i \leq n$, is left-complete or right-complete, respectively. G is said to be sf-complete iff each P_i, $1 \leq i \leq n$, is sf-complete with respect to N^* if G has generating context-free components, and with respect to $(N \cup T)^*$ if G has accepting context-free components.

Note that it is sensible to differentiate between generating and accepting mode in this definition since in generating mode only nonterminal symbols can be rewritten whereas in accepting case there must also be rules for rewriting terminals.

Theorem 4.2 *The requirement of left-completeness, right-completeness, and / or sf-completeness does not alter the derivational power in the case of the following grammars and systems:*

(i) generating and accepting [E][T]0L systems,

(ii) generating and accepting context-free grammars,

(iii) generating and accepting CD grammar systems with [λ-free] context-free components working in $*$-, in weak-t-, or in stagnating mode,

(iv) generating and accepting CD grammar systems with [λ-free] context-free components working in $\leq k$-, $\geq k$-, or $= k$-mode, with $k \geq 1$.

Proof. Let N be the set of nonterminals and T be the set of terminals of the system under consideration.

(i) Add rules $x \longrightarrow x$ for any $x \in V$ to the set of productions (or to each production table, respectively) both in the generating and in the accepting case.

(ii)-(iii) In the generating case, to the set of productions (or to each component, respectively) add rules $A \longrightarrow A$ for any $A \in N \cup \{F\}$, where F is a new nonterminal symbol and add $F \longrightarrow a$ for any $a \in T$ (in order to get right-completeness). In the accepting case, add $T' = \{a' \mid a \in T\}$ to the set of nonterminal symbols. Then, to the set of productions (or to each component, respectively), add rules $A \longrightarrow A$ for each $A \in N \cup T'$, and $a \longrightarrow a'$ for each $a \in T$. Moreover, in the case of CD grammar systems working in weak-t or stagnating mode, we have to add the rules in $\{v \longrightarrow \beta \mid v \in s(\alpha),\ \alpha \longrightarrow \beta \in P_i\}$ to component P_i, where s is the finite substitution defined by $s(A) = \{A\}$, for $A \in N$, and $s(a) = \{a, a'\}$, for $a \in T$.

(iv) Let F_1 and F_2 be two new nonterminal symbols. Add $F_1 \longrightarrow F_1$ and $F_2 \longrightarrow F_2$ to each component. Moreover, add rules $x \longrightarrow F_1$ for $x \in N$ in the generating case and for $x \in N \cup T$ in the accepting case, respectively. This guarantees left- and sf-completeness. Furthermore, add rules $F_2 \longrightarrow x$ for $x \in N \cup T$ in the generating case and for $x \in$ in the accepting case, respectively, in order to get right-completeness, too. □

The next theorem deals with the hard-t-mode of derivation where the situation is different.

Theorem 4.3 *(i) Right-completeness is no restriction for generating as well as for accepting CD grammar systems with [λ-free] context-free components working in hard-t-mode.*

(ii) Both the family of languages generated by left-complete CD grammar systems with [λ-free] context-free components working in hard-t-mode and the family of languages generated by sf-complete CD grammmar systems with [λ-free] context-free components working in hard-t-mode is equal to $\mathcal{L}(CF)$.
The corresponding families of languages in the accepting case are empty.

Proof. (i) Add rules $F \to x$ for all symbols x of the total alphabet in the generating case and for all nonterminal symbols in the accepting case, respectively, and the rule $F \to F$, where F is a new nonterminal symbol, again. Statement (ii) is obvious. □

5 Cooperation and sf-completeness

In [9], cooperating (generating) grammar systems were defined in such a way that the concept of sf-completeness was used as the basis of the cooperation protocol. In this section we present results about the derivational capacity and size complexity of accepting CD grammar systems with components cooperating in the above manner.

Using our notation, we first give the following definition which is appropriate both for generating and accepting CD grammar systems.

Let $G = (N, T, S, P_1, \ldots, P_n)$ be a context-free CD grammar system and let h be a morphism defined by $h(A) = A$, for $A \in N$ and $h(a) = \lambda$, for $a \in T$. In *sf-mode of derivation* the rewriting has to be performed by one and the same component P_i until and unless it is *disabled*, i.e., a sentential form w has been derived such that P_i is not sf-complete any more with respect to $h(w)$ in the generating case and with respect to w in the accepting case.

The family of languages generated by CD grammar systems with at most n [λ-free] context-free components working in the sf-mode is denoted by $\mathcal{L}^{\text{gen}}(\text{CD}_n, \text{CF}[-\lambda], sf)$. If there is no limit for the number of components then we write the subscript ∞ instead of n. The language families defined by the corresponding accepting devices are denoted analogously.

In contrast to the results about generating CD grammar systems with context-free components working in (weak-)t-mode, for the systems defined by Meersman and Rozenberg the following result is shown ([9]):

Theorem 5.1 $\mathcal{L}^{\text{gen}}(CD_\infty, CF[-\lambda], sf) = \mathcal{L}^{\text{gen}}(P, CF[-\lambda], ac)$. □

As it concerns the λ-free accepting case, we find the same hierarchical relationship as for (hard-)t-mode proved in [8], but here, we have also nonempty families of languages accepted by devices containing λ-rules.

Theorem 5.2 (i) $\mathcal{L}^{\mathrm{acc}}(\mathrm{CD}_\infty, \mathrm{CF} - \lambda, sf) = \mathcal{L}^{\mathrm{gen}}(\mathrm{CS}) = \mathcal{L}^{\mathrm{acc}}(\mathrm{CS})$

(ii) $\mathcal{L}^{\mathrm{acc}}(\mathrm{CD}_\infty, \mathrm{CF}, sf) = \mathcal{L}^{\mathrm{gen}}(\mathrm{RE}) = \mathcal{L}^{\mathrm{acc}}(\mathrm{RE})$

Proof. (i) Since any CD grammar system from $(\mathrm{CD}_\infty, \mathrm{CF} - \lambda, sf)$ can be simulated by a linear-bounded automaton (that is, by a context-sensitive grammar), we only show that the reverse inclusion holds. For, our proof is based on the underlying idea of the proof of [5, Theorem 3.3]. Let us have a generating context-sensitive grammar $G = (N, T, P, S)$ in Kuroda normal form, without λ productions. Assume that a unique label r is attached to any context-sensitive rule, of the form $XU \to YZ$ with $X, U, Y, Z \in N$, in P. Let us denote the set of labels by $\mathrm{Lab}(P) = \{r_1, r_2, \ldots, r_R\}$. Let $\bar{T} = \{\bar{a} \mid a \in T\}$, $\{\bar{\bar{T}} = \{\bar{\bar{a}} \mid a \in T\}$ and let h be a morphism defined by $h(a) = A$ for $A \in N$ and $h(a) = \bar{a}$ for $a \in T$. For a string $w \in (N \cup T)^*$ let us denote by $\bar{w} = h(w)$.

We construct an accepting CD grammar system

$$\Gamma = (N', T, S', P_{\mathrm{init}}, P_{\mathrm{CF}}, P_{1,1}, P_{1,2}, P_{1,3}, P'_{1,1}, P'_{1,2}, P'_{1,3} \ldots, P'_{R,1}, P'_{R,2}, P'_{R,3})$$

such that $L^{acc}_{sf}(\Gamma) = L(G)$ holds. Let Γ be defined with

$$\begin{aligned} N' = N \cup \bar{T} \cup \bar{\bar{T}} \cup \{S', F\} \cup \{[A, r], (A, r) \mid A \in N \text{ and } r \in \mathrm{Lab}(P)\} \\ \cup \{x' \mid x \in N \cup \bar{T}\} \cup \{< x, r > \mid x \in N \cup \bar{T} \text{ and } r \in \mathrm{Lab}(P)\} \end{aligned}$$

(the unions being disjoint). The components of Γ are constructed as follows:

$$\begin{aligned} P_{\mathrm{init}} &= \{a \to \bar{a}, \bar{a} \to \bar{a}, \bar{a} \to \bar{\bar{a}} \mid a \in T\}, \\ P_{\mathrm{CF}} &= \{S \to S'\} \cup \{x \to x \mid x \in N \cup \bar{T}\} \cup \{\bar{w} \to C \mid C \to w \in P\} \cup \\ &\quad \{Y \to [Y, r] \mid r : XU \to YZ \in P\} \cup \{x' \to x \mid x \in N \cup \bar{T}\} \cup \\ &\quad \{\bar{\bar{a}} \to \bar{a} \mid a \in T\}, \end{aligned}$$

and, for $1 \leq r \leq R$, $r : XU \to YZ$:

$$\begin{aligned} P_{r,1} &= \{[Y, r] \to [Y, r], Z \to (Z, r)\} \cup \{x \to x \mid x \in N \cup \bar{T}\} \cup \{x' \to x \mid x \in N \cup \bar{T}\} \\ P_{r,2} &= \{[Y, r](Z, r) \to F\} \cup \{x \to x \mid x \in N \cup \bar{T}\} \cup \{x \to < x, r > \mid x \in N \cup \bar{T}\} \\ P_{r,3} &= \{[Y, r] \to X, (Z, r) \to U\} \cup \{x \to x \mid x \in N \cup \bar{T}\} \cup \\ &\quad \{< x, r > \to x' \mid x \in N \cup \bar{T}\} \\ P'_{r,1} &= \{[Y, r] \to [Y, r], Z \to Z'\} \\ P'_{r,2} &= \{[Y, r]Z' \to F, [Y, r] \to Y'\} \\ P'_{r,3} &= \{Y' \to X, Z' \to U, X \to X\} \end{aligned}$$

Production set P_{init} is for starting the derivation process. Obviously, by P_{CF} context-free derivation steps of G are simulated whereas the components

$P_{r,1}, P_{r,2}, P_{r,3}$ and $P'_{r,1}, P'_{r,2}, P'_{r,3}$ simulate applications done by the rule with label r after replacing exactly one appearance of symbol Y in the sentential form by $[Y, r]$. The first group of production sets handles the situation when the sentential form is of the form $u[Y, r]Zv$, with $uv \in (N \cup \bar{T})^+$, and the second is for the situation when the sentential form is $[Y, r]Z$. In the first case it is necessary to replace a symbol $< x, r >$ in order to leave $P_{r,3}$ which can only be introduced by application of $P_{r,2}$. But $P_{r,2}$ can be active only if the symbols $[Y, r]$ and (Z, r) are neighbouring in the "correct" manner or they do not appear at all. In the latter case, the application of $P_{r,2}$ and $P_{r,3}$ remain without any effect. Shortcuts are impossible since a component must be fully competent when applied. Similarly, it is easy to see that production sets $P'_{r,1}, P'_{r,2}, P'_{r,3}$ can be successfully applied only if the sentential form is of the form $[Y, r]Z$. Hence, $L^{\text{acc}}(\Gamma) = L^{\text{gen}}(G)$.

(ii) Without loss of generality we can assume the given type-0 grammar to have only rules as a grammar in Kuroda normal form only having rules of the form $A \rightarrow \lambda$, with $A \in N$, in addition. Thus, we can use the same construction as in (i) only giving additional rules $\lambda \rightarrow A$ to component P_{CF} if needed. The other direction of the proof follows by the Church theses or it can be shown by construction of a Turing machine. □

Finally, we investigate the question if the number of components can be restricted for devices working in sf-mode. Indeed, we find that 5 components are sufficient in order to describe the whole language family.

Theorem 5.3 (i) $\mathcal{L}^{\text{gen}}(\text{CD}_\infty, \text{CF}[-\lambda], sf) = \mathcal{L}^{\text{gen}}(\text{CD}_5, \text{CF}[-\lambda], sf)$

(ii) $\mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}[-\lambda], sf) = \mathcal{L}^{\text{acc}}(\text{CD}_5, \text{CF}[-\lambda], sf)$

Proof. Let $G = (N, T, S, P_1, \ldots, P_n)$ be a CD grammar system of degree $n > 5$ working in sf-mode. Construct a CD grammar system working in sf-mode of the same type with 5 components according to the following basic idea:
Component 1 contains all rules of the given system, but the symbols in the rules carry the number of the component of G, where they originally belong to, as subscript. For simulating work of component P_i, the symbols in the sentential form must carry subscript i, too. Then, this component becomes disabled iff there is no rule stemming from P_i which is applicable to the current sentential form, and one can continue with component 2 or 3.
In *component 2* to even subscripts i of the symbols of the sentential form, one is added (modulo n) in order to change the production set of G which shall be simulated.
Component 2' is the only component which can get active after applying component 2. Here, it is checked whether all symbols have changed their subscript. If yes, a continuation with component 1 or 3 is possible, otherwise the derivation is blocked.
Components 3 and 3' do the analogous job as components 2 and 2' for odd subscripts.

Now, we give the formal description of that system for the accepting case. The construction for the generating case can be given analogously.

Let $G' = (N', T, S', P_1, P_2, P_2', P_3, P_3')$, where

$$N' = N \cup \{S'\} \cup \{A_i, A_i', A_i'' \mid A \in N \cup T \text{ and } 1 \le i \le n\}$$

(the unions being disjoint). Furthermore, let h_i be the morphism defined by $h_i(A) = A_i$ for $A \in N \cup T$, $1 \le i \le n$. Then, the components of G' are constructed as follows:

$$\begin{aligned}
P_1 &= \{h_i(\alpha) \to h_i(\beta) \mid \alpha \to \beta \in P_i,\ 1 \le i \le n\} \cup \{A_i'' \to A_i \mid 1 \le i \le n\} \cup \\
&\quad \{S_i \to S' \mid 1 \le i \le n\}, \\
P_2 &= \{A_i \to A_{i+1} \mid i \equiv 0 \bmod 2\} \cup \{A_{i+1} \to A_{i+1} \mid i \equiv 0 \bmod 2\} \cup \\
&\quad \{A_i \to A_{i+1}' \mid i \equiv 0 \bmod 2\} \cup \{A_i'' \to A_i \mid i \equiv 0 \bmod 2\} \cup \\
&\quad \{A \to A_1 \mid A \in N \cup T\} \cup \{A \to A_1' \mid A \in N \cup T\} \cup Q_2, \\
P_2' &= \{A_{i+1} \to A_{i+1} \mid i \equiv 0 \bmod 2\} \cup \{A_{i+1}' \to A_{i+1}'' \mid i \equiv 0 \bmod 2\}, \\
P_3 &= \{A_i \to A_{i+1} \mid i \equiv 1 \bmod 2\} \cup \{A_{i+1} \to A_{i+1} \mid i \equiv 1 \bmod 2\} \cup \\
&\quad \{A_i \to A_{i+1}' \mid i \equiv 1 \bmod 2\} \cup \{A_i'' \to A_i \mid i \equiv 1 \bmod 2\} \cup Q_3, \\
P_3' &= \{A_{i+1} \to A_{i+1} \mid i \equiv 1 \bmod 2\} \cup \{A_{i+1}' \to A_{i+1}'' \mid i \equiv 1 \bmod 2\},
\end{aligned}$$

where

$$Q_2 = \begin{cases} \{A_n \to A_1,\ A_n \to A_1' \mid A \in N \cup T\} \cup \\ \{A_1 \to A_1 \mid A \in N \cup T\} & \textit{if} \quad n \equiv 0 \bmod 2 \\ \emptyset & \textit{if} \quad n \equiv 1 \bmod 2 \end{cases} \qquad \text{and}$$

$$Q_3 = \begin{cases} \{A_n \to A_1,\ A_n \to A_1' \mid A \in N \cup T\} \cup \\ \{A_1 \to A_1 \mid A \in N \cup T\} & \textit{if} \quad n \equiv 1 \bmod 2 \\ \emptyset & \textit{if} \quad n \equiv 0 \bmod 2 \end{cases}.$$

Clearly, the system has to start with component P_2 by rewriting any terminal a by a_1. The axiom can be derived after yielding S_i for some i with P_1. Note that, by technical reasons, after applying the rules in Q_3 the system has to continue with component P_2' and then with P_3. Moreover, each component is a set of accepting context-free productions; in this connection the modifications are necessary for proving the statement in the generating case. □

Acknowledgements: The authors are grateful to Henning Fernau and Markus Holzer for discussions on the topic.

References

[1] H. Bordihn and H. Fernau. Accepting grammars with regulation. *International Journal of Computer Mathematics*, 53:1–18, 1994.

[2] H. Bordihn and H. Fernau. Accepting grammars and systems via context condition grammars. *Journal of Automata, Languages and Combinatorics*, 1(2):97–112, 1996.

[3] E. Csuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 26(1/2):49–63, 1990.

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

[5] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science.* Berlin: Springer, 1989.

[6] H. Fernau and H. Bordihn. Remarks on accepting parallel systems. *International Journal of Computer Mathematics*, 56:51–67, 1995.

[7] H. Fernau and M. Holzer. Accepting multi-agent systems II. *Acta Cybernetica*, 1996. In this volume.

[8] H. Fernau, M. Holzer, and H. Bordihn. Accepting multi-agent systems: The case of cooperating distributed grammar systems. *Computers and Artificial Intelligence*, 15(2–3):123–139, 1996.

[9] R. Meersman and G. Rozenberg. Cooperating grammar systems. In *Proc. MFCS'78*, volume 64 of *LNCS*, pages 364–373. Berlin: Springer, 1978.

[10] Gh. Păun. On the generative capacity of hybrid CD grammar systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 30(4):231–244, 1994.

[11] A. Salomaa. *Formal Languages.* Academic Press, 1973.

Acta Cybernetica 12 (1996) 361-379.

Accepting Multi-Agent Systems II

Henning FERNAU [*] [†] [‡] Markus HOLZER [*] [‡]

Abstract

We continue our previous research on cooperating distributed grammar systems (CDGS) and variants thereof as language acceptors [16]. Here, we classify the accepting capacity of CDGS working in the modes recently introduced by the authors together with Freund [14]. Moreover, we study (prescribed) teams as language accepting mechanisms.

In this way, we solve an open problem from the area of accepting grammars: there exists a grammar family such that its generating capacity is strictly more powerful that its accepting capacity, see [6] for a recent survey.

1 Introduction

In Artificial Intelligence (AI), a common methodology in order to achieve a goal, which can hardly be done by a single expert or agent, is to create a so-called multi-agent system which solves the task using distributed and cooperating agents, see [29] for a survey on this area. Blackboard architecture models [10] can be seen as an approach to model the communication aspects of the agents.

On the other hand, one general concept in problem solving methods in AI are production systems [23], which arise from a computational formalism proposed by Post [27] that was based on string replacement. Many generalizations of production systems are proposed in AI, e.g., rule-based systems, blackboard systems, or pattern-directed-inference systems. Production systems are closely connected to string rewriting, one of the backbones of formal language theory. In order to understand the nature of production systems, it is therefore natural to study them on well-known and traditional formal language theoretical devices. Based on blackboard systems, Csuhaj-Varjú and Dassow [7] introduced cooperating distributed grammar systems (CDGS), where each component (grammar) corresponds to the particular knowledge source of the system (this is an expert or agent), and the global database—the blackboard—is modelled by a common sentential form, where the

[*]Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Germany. E-mail:fernau/holzer@informatik.uni-tuebingen.de

[†]Supported by Deutsche Forschungsgemeinschaft grant DFG La 618/3-1.

[‡]Supported by German-Hungarian Research Project "Formal Languages, Automata and Petri-Nets" (1995-1997), No. D/102 (formerly: No. OMFB-NPI-102) of the TÉT Foundation, Budapest, Hungary, and No. 233.6. of Forschungszentrum Karlsruhe, Germany

components perform their rewritings. Independently and sometimes with different motivation, also other authors introduced similar computational models, see, e.g., [1, 20, 21].

In the systems considered by Csuhaj-Varjú and Dassow and in a series of subsequent papers, all components work according to the same strategy; more precisely, the rewriting by another component can be done, e.g., after a given number of steps. For an overview on this topic, refer to [8, 25]. This model of multi-agent systems is not very realistic, because usually such agents have different capabilities. Therefore, a generalization of CDGS called hybrid CDGS has been investigated [22, 24, 25]. Another idea is to allow the formation of teams of agents, as proposed in [18, 19, 26].

In this paper, we take the original above-sketched idea of CDGS, but contrary to the approach of Csuhaj-Varjú *et al.* [8], we take, instead of generating grammars, accepting grammars as agents. They try to derive a given goal word (axiom) in a cooperating and distributive manner. In this way, we pursue our studies on accepting grammars and systems [3, 4, 5, 6, 11]. Accepting CDGS have been investigated in the works of the authors together with Bordihn [16] and Freund [12, 13]. For an intermediate approach, combining generating and accepting grammars in CDGS, we refer to Fernau and Holzer [15].

Depending on the mode in which the grammars cooperate, we obtain, on the one hand, equivalences between the accepting and generating case, but also, on the other hand, results which are fundamentally different, this means that accepting devices are much more powerful than generating ones or vice versa, thereby solving an open problem in the theory of accepting grammars [3].

Observe that, when defining accepting counterparts of existing generating devices, we want to carry over the original idea and motivation of the generating mechanism in order to define the corresponding accepting mechanism. Formally, such accepting grammars look like their generating counterparts, just turning the core productions "around" and keeping the control mechanism ruling the application of these productions textually the same. In the case of CDGS, this procedure is very well motivated by the original ideas stemming from AI: while generating devices correspond to forward deduction system, accepting devices correspond to backward deduction systems.

The present paper extends our studies on accepting CDGS in three directions: (1) We consider other working modes of the components which have been introduced in [14] and (2) we consider (external) hybridizations of these new modes (with the new and old ones). Finally (3) we briefly consider accepting CDGS with (prescribed) teams.

This is reflected in the organization of our paper. In the next section, we introduce the necessary notions. Section 3 lists the easy cases (the interval mode and t combined with greater than or equal to k) where we can profit from our previous results [16] on the t-mode. Section 4 contains the more interesting case incorporating the t-mode combined with either equal than or less equal than k. Strange things can be observed here, e.g., no unary infinite language can be accepted by CDGS working in $(t \wedge \leq k)$-mode. This leads to the first examples of grammar mechanisms where the generating power of such CDGS is greater than their accepting power. In Section 5, we we consider accepting hybrid CDGS, where the (external) hybridization incorporates also the modes $(t \wedge \leq k)$, $(t \wedge = k)$, and $(t \wedge \geq k)$. Furthermore, we consider the number of components in a CDGS as a natural measure of descriptional complexity for CDGS. Since each component corresponds to one expert according to the original AI motivation, we may paraphrase our results as follows: in backward deduction systems, two experts are enough. This contrasts to the situation found in forward deduction systems where it is only known that three or four experts are enough. Finally, we consider in Section 6 CDGS with (prescribed) teams as language acceptors. Observe that similar systems have been studied in the context of array grammars from a quite practical viewpoint, see [12, 13].

2 Definitions

We assume the reader to be familiar with some basic notions of formal language theory, as contained in Dassow and Păun [9]. Especially, we consider two languages L_1, L_2 to be equal if and only if $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$, where λ denotes the empty word. We use $\subseteq$ to denote inclusion, while $\subset$ denotes strict inclusion. The set of positive integers is denoted by $\mathbf{N}$. The end of a proof or of a proved statement is marked by $\Box$. Let $\mathcal{L}(\mathrm{FIN})$ be the family of finite languages. The families of languages generated by regular, linear context-free, context-sensitive, type-0 Chomsky grammars, ET0L systems, context-free ordered, context-free programmed, and context-free programmed grammars with appearance checking are denoted by $\mathcal{L}^{gen}(\mathrm{REG})$, $\mathcal{L}^{gen}(\mathrm{LIN})$, $\mathcal{L}^{gen}(\mathrm{CF})$, $\mathcal{L}^{gen}(\mathrm{CS})$, $\mathcal{L}^{gen}(\mathrm{RE})$, $\mathcal{L}^{gen}(\mathrm{ET0L})$, $\mathcal{L}^{gen}(\mathrm{O,CF})$, $\mathcal{L}^{gen}(\mathrm{P,CF})$, and $\mathcal{L}^{gen}(\mathrm{P,CF,ac})$, respectively. A subscript *fin* (k, or 1, respectively) denotes the family of languages generated by the appropriate device restricting the derivation to be of finite index (finite index k, finite index 1, respectively). For a definition of the finite index property we refer to [9].

A superscript *acc* instead of *gen* is used to denote the family of languages accepted by the appropriate device. If we want to exclude λ-rules, we add $-\lambda$ in our notations.

We use bracket notations like $\mathcal{L}^{gen}(\mathrm{P,CF}[-\lambda]) \subset \mathcal{L}^{gen}(\mathrm{P,CF}[-\lambda],\mathrm{ac})$ in order to say that the equation holds both in the case of forbidding λ-rules and in the case of admitting λ-rules (neglecting the bracket contents).

For the convenience of the reader, we repeat the basic definitions of CDGS, hybrid CDGS, respectively, adapted from Păun [24], in a way suitable for the interpretation both as generating and accepting systems.

A *cooperating distributed grammar system* (CDGS for short) of degree n, with $n \geq 1$, is a $(n+3)$-tuple $G = (N, T, S, P_1, \ldots, P_n)$, where N, T are disjoint alphabets of nonterminal and terminal symbols, respectively, $S \in N$ is the axiom, and $P_1, \ldots, P_n$ are finite sets of rewriting rules over $N \cup T$.

Throughout this paper, we consider only context-free rewriting rules. Since we are interested in generating and accepting systems, we further distinguish between so-called *generating rules*, which have the form $A \to w$, with $A \in N$ and $w \in (N \cup T)^*$, and *accepting rules*, which are of the form $w \to A$, with $A \in N$ and $w \in (N \cup T)^*$.

Let G be a CDGS with only generating rules. For $x, y \in (N \cup T)^*$ and $1 \leq i \leq n$, we write $x \Rightarrow_i y$ if and only if $x = x_1 A x_2$, $y = x_1 z x_2$ for some $A \to z \in P_i$. Hence, subscript i refers to the component to be used. By $\Rightarrow_i^{\leq k}$, $\Rightarrow_i^{=k}$, $\Rightarrow_i^{\geq k}$, $\Rightarrow_i^*$ we denote a derivation consisting of at most k steps, exactly k steps, at least k steps, an arbitrary number of steps, respectively. We also write $x \Rightarrow_i^t y$ if and only if $x \Rightarrow_i^* y$ and there is no z such that $y \Rightarrow_i z$. Combining the former three modes with the t-mode requirement we obtain the modes $(t \wedge \leq k)$, $(t \wedge = k)$, and $(t \wedge \geq k)$ which are defined as follows: there exists a derivation which satisfies both properties e.g., $x \Rightarrow_i^{(t \wedge \leq k)} y$ if and only if there exists an m-step derivation from x to y using P_i such that $m \leq k$ and there is no z such that $y \Rightarrow_i z$.

Let $D := \{*, t\} \cup \{\leq k, = k, \geq k \mid k \in \mathrm{N}\} \cup \{(\geq k_1 \wedge \leq k_2), (t \wedge \leq k), (t \wedge = k), (t \wedge \geq k) \mid k_1, k_2, k \in \mathrm{N} \text{ and } k_1 \leq k_2\}$.

The language *generated* in the f-mode, $f \in D$, by a CDGS G with only generating rules is defined as:

$$L_f^{gen}(G) \;:=\; \{w \in T^* \mid S \Rightarrow_{i_1}^f \alpha_1 \Rightarrow_{i_2}^f \ldots \Rightarrow_{i_{m-1}}^f \alpha_{m-1} \Rightarrow_{i_m}^f \alpha_m = w \text{ with } m \geq 1,\ 1 \leq i_j \leq n, \text{ and } 1 \leq j \leq m\}.$$

If $f \in D$, the families of languages generated in f-mode by [λ-free] CDGS with at most n components are denoted by $\mathcal{L}^{gen}(\mathrm{CD}_n, \mathrm{CF}[-\lambda], f)$. If the number of components is not restricted, we write $\mathcal{L}^{gen}(\mathrm{CD}_\infty, \mathrm{CF}[-\lambda], f)$.

For CDGS with only accepting rules, we define the relations $x \Rightarrow_i y$ and $x \Rightarrow_i^f y$ accordingly. Hence, we define the language *accepted* in f-mode, $f \in D$, by a CDGS G with only accepting rules as follows:

$$L_f^{acc}(G) \;:=\; \{w \in T^* \mid w \Rightarrow_{i_1}^f \alpha_1 \Rightarrow_{i_2}^f \ldots \Rightarrow_{i_{m-1}}^f \alpha_{m-1} \Rightarrow_{i_m}^f \alpha_m = S \text{ with } m \geq 1,\ 1 \leq i_j \leq n, \text{ and } 1 \leq j \leq m\}$$

If $f \in D$, the families of languages accepted in f-mode by [λ-free] CDGS with at most n components are denoted by $\mathcal{L}^{acc}(\mathrm{CD}_n, \mathrm{CF}[-\lambda], f)$. If the number of components is not restricted, we write $\mathcal{L}^{acc}(\mathrm{CD}_\infty, \mathrm{CF}[-\lambda], f)$.

If each component of a CDGS may work in a different mode, then we get the notion of *(externally) hybrid CDGS* of degree n, with $n \geq 1$, which is a $(n+3)$-tuple $G = (N, T, S, (P_1, f_1), \ldots, (P_n, f_n))$, where $N, T, S, P_1, \ldots, P_n$ are as in CDGS, and $f_i \in D$, for $1 \leq i \leq n$. Thus, we can define the language *generated* by a hybrid CDGS with only generating rules as:

$$L^{gen}(G) \;:=\; \{w \in T^* \mid S \Rightarrow_{i_1}^{f_{i_1}} w_1 \Rightarrow_{i_2}^{f_{i_2}} \ldots \Rightarrow_{i_{m-1}}^{f_{i_{m-1}}} w_{m-1} \Rightarrow_{i_m}^{f_{i_m}} w_m = w \text{ with } m \geq 1,\ 1 \leq i_j \leq n, \text{ and } 1 \leq j \leq m\}$$

Accordingly, accepting hybrid CDGS can be defined. If $F \subseteq D$, the family of languages generated (accepted, respectively) by [λ-free] CDGS with at most n

components, each component working in one of the modes contained in F, are denoted by $\mathcal{L}^{gen}(\mathrm{HCD}_n, \mathrm{CF}[-\lambda], F)$ ($\mathcal{L}^{acc}(\mathrm{HCD}_n, \mathrm{CF}[-\lambda], F)$, respectively). Similarly, $\mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F)$, and $\mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F)$, respectively, is written when the number of components is not restricted.

3 When generating is weaker than accepting

The easiest case from the new modes is the interval mode. Fortunately, the observations given in [16, page 128, Theorem 3.2] showing that in case of classical modes, except for the t-mode, equivalence between generating and accepting devices prevails, readily transfers to the $(\geq k_1 \wedge \leq k_2)$-mode. Hence, we get:

Theorem 3.1 *If* $N \in \mathbf{N} \cup \{\infty\}$ *and* $f \in \{(\geq k_1 \wedge \leq k_2) \mid k_1, k_2 \in \mathbf{N}, k_1 \leq k_2\}$, *then* $\mathcal{L}^{gen}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], f) = \mathcal{L}^{acc}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], f)$. □

We turn our attention to grammar systems working in $(t \wedge \geq k)$-mode. Note that $(t \wedge \geq 1)$- and the classical t-mode trivially coincide, which leads us to:

Theorem 3.2 *If* $N \in \mathbf{N} \cup \{\infty\}$, *then, for any* $N \geq 3$,

$$\begin{aligned} \mathcal{L}^{gen}(\mathrm{CF}) &= \mathcal{L}^{gen}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], (t \wedge \geq 1)) &&= \mathcal{L}^{gen}(\mathrm{CD}_2, \mathrm{CF}[-\lambda], (t \wedge \geq 1)) \\ &\subset \mathcal{L}^{gen}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], (t \wedge \geq 1)) &&= \mathcal{L}^{gen}(\mathrm{ET0L}). \end{aligned}$$

For k in general, the situation is a little bit different from the previous one.

Theorem 3.3 *If* $N \in \mathbf{N} \cup \{\infty\}$, *then, for any* $N \geq 3$ *and for each* $k \geq 2$,

$$\begin{aligned} \mathcal{L}^{gen}(\mathrm{CF}) &= \mathcal{L}^{gen}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], (t \wedge \geq k)) &&\subset \mathcal{L}^{gen}(\mathrm{CD}_2, \mathrm{CF}[-\lambda], (t \wedge \geq k)) \\ &\subseteq \mathcal{L}^{gen}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], (t \wedge \geq k)) &&= \mathcal{L}^{gen}(\mathrm{CD}_\infty, \mathrm{CF}[-\lambda], (t \wedge \geq k)). \end{aligned}$$

The latter class coincides with the family of E[P]T0L languages with random context conditions [9, 28], as shown in [14], a special case of $\mathcal{L}^{gen}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac})$.

The case is totally different for accepting CDGS. Nevertheless, as in the case of t-mode, the admittance of λ-productions does not enhance the accepting power of CDGS working in $(t \wedge \geq k)$-mode.

Theorem 3.4 *If* $N \in \mathbf{N} \cup \{\infty\}$, *then, for any* $N \geq 2$ *and for each* $k \geq 1$,

$$\begin{aligned} \mathcal{L}^{gen}(\mathrm{CF}) &= \mathcal{L}^{acc}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], (t \wedge \geq k)) &&\subset \mathcal{L}^{acc}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], (t \wedge \geq k)) \\ &= \mathcal{L}^{acc}(\mathrm{CD}_\infty, \mathrm{CF}[-\lambda], (t \wedge \geq k)) &&= \mathcal{L}^{gen}(\mathrm{CS}). \end{aligned}$$

Proof. The first relation is obvious. By [16, Theorem 4.5], we know that, for each $N \geq 2$, $\mathcal{L}^{acc}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], t) = \mathcal{L}^{acc}(\mathrm{CD}_\infty, \mathrm{CF}[-\lambda], t) = \mathcal{L}^{gen}(\mathrm{CS})$. Since t- and $(t \wedge \geq 1)$-mode trivially coincide, the results carry over to the $(t \wedge \geq 1)$-mode. By introducing prolongating rules, we obtain the desired result for $(t \wedge \geq k)$-mode for k in general. □

Corollary 3.5 *Let $k \in \mathbf{N}$. Then, we have*

$$\mathcal{L}^{gen}(\mathrm{CF}) = \mathcal{L}^{gen}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], (t \wedge \geq k)) = \mathcal{L}^{acc}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], (t \wedge \geq k)).$$

Moreover, if $N \in \mathbf{N} \cup \{\infty\}$ with $N \geq 2$, we get

$$\mathcal{L}^{gen}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], (t \wedge \geq k)) \subseteq \mathcal{L}^{acc}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], (t \wedge \geq k)),$$

where the inclusion is known to be strict only in the absence of λ-rules. □

4 When accepting is weaker than generating

In the present section, we deal with CDGS working in $(t \wedge \leq k)$- and $(t \wedge = k)$-mode with context-free components. Again, at first we mention the known results in the generating case [14].

Theorem 4.1 *If $f \in \{(t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N}\}$, then there exists a function $s_f : \mathbf{N} \to \mathbf{N}$ so that for each $n \in \mathbf{N}$,*

$$\begin{aligned}\mathcal{L}(\mathrm{FIN}) &= \mathcal{L}^{gen}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], f) \subseteq \mathcal{L}^{gen}(\mathrm{CD}_n, \mathrm{CF}[-\lambda], f) \\ &\subset \mathcal{L}^{gen}(\mathrm{CD}_{s_f(n)}, \mathrm{CF}[-\lambda], f) \subset \mathcal{L}^{gen}(\mathrm{CD}_\infty, \mathrm{CF}[-\lambda], f) = \mathcal{L}^{gen}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac}).\end{aligned}$$

The preceding theorem demonstrates that both the $(t \wedge \leq k)$- and the $(t \wedge = k)$-mode nicely fit into the known framework of formal language families. On the other hand, the accepting counterparts behave very strange in comparison to earlier results on accepting CDGS. This is shown in the next lemma.

Lemma 4.2 *For every $k \in \mathbf{N}$, no infinite one-letter language can be accepted by a CDGS working either in $(t \wedge \leq k)$ or $(t \wedge = k)$-mode.*

Proof. We only prove the statement for the $(t \wedge = k)$-mode. Assume that the CDGS $G = (N, T, S, P_1, \ldots, P_n)$ accepts an infinite one-letter language $L \subseteq \{a\}^*$ in $(t \wedge = k)$-mode. Set $M = \max\{\, m \mid a^m \to A \in P_i \text{ for } 1 \leq i \leq n \,\}$. Since L is infinite, there exists a word a^m in L such that $m \geq 3k \cdot M$. On this word, there is no way to start the (accepting) derivation process, since every component is only able to handle at most $k \cdot M$ symbols a due to the $(t \wedge = k)$-mode. Thus, a^m does not belong L, which contradicts our assumption that L is infinite. □

Thus, $\{a\}^* \in \mathcal{L}^{gen}(\mathrm{REG}) \setminus \mathcal{L}^{acc}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], f)$, if $N \in \mathbf{N} \cup \{\infty\}$ and $f \in \{\,(t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N}\,\}$, but $\{\#\}\{a\}^* \in \mathcal{L}^{acc}(\mathrm{CD}_2, \mathrm{CF} - \lambda, (t \wedge = 1))$, which is shown in the following example.

Example. Let $G = (\{S, A, A', B\}, \{\#, a\}, S, P_1, P_2)$ be a CDGS with the sets $P_1 = \{\# \to B, Aa \to A', Aa \to S\}$ and $P_2 = \{B \to A, A' \to A\}$. It is easy to see that G working in $(t \wedge = 1)$-mode accepts $\{\#\}\{a\}^*$.

The idea of a special marking symbol generalizes to arbitrary regular languages. This shows that every marked regular language belongs to, e.g., the language family $\mathcal{L}^{acc}(\mathrm{CD}_2, \mathrm{CF} - \lambda, (t \wedge = 1))$.

Before we consider CDGS with an arbitrary number of components, we study the case where two components are working in $(t \wedge \leq k)$- or $(t \wedge = k)$-mode together. Again, we recall what is known for these language families [14].

Theorem 4.3 *If* $f \in \{ (t \wedge \leq k), (t \wedge = k) \mid k \in \mathrm{N} \}$, *then*

$$\mathcal{L}^{gen}(\mathrm{LIN}) \subseteq \mathcal{L}^{gen}(\mathrm{CD}_2, \mathrm{CF}[-\lambda], f) \subseteq \mathcal{L}_k^{gen}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac}).$$

The latter inclusion is known to be strict only in case $k = 1$.

First, consider the trivially coinciding modes $(t \wedge \leq 1)$ and $(t \wedge = 1)$ for accepting CDGS with two components. Surprisingly, we find:

Theorem 4.4 $\mathcal{L}^{acc}(\mathrm{CD}_2, \mathrm{CF}[-\lambda], (t \wedge = 1)) \subset \mathcal{L}^{gen}(\mathrm{LIN})$.

Before we prove this theorem, let us mention that the preceding theorem answers an open question stated in [3]: is there a grammar family such that the generating mode is strictly more powerful than the accepting mode? Combining the previous two theorems, we obtain:

Corollary 4.5 $\mathcal{L}^{acc}(\mathrm{CD}_2, \mathrm{CF}[-\lambda], (t \wedge = 1)) \subset \mathcal{L}^{gen}(\mathrm{CD}_2, \mathrm{CF}[-\lambda], (t \wedge = 1))$. □

For the proof of the theorem, we need a detailed analysis of the accepting $(t\wedge = 1)$ derivation of the grammar system. Assume that we are given a grammar system $G = (N, T, S, P_1, P_2)$. On input w the system can mainly behave as follows. By the $(t \wedge = 1)$-mode (like in the t-mode), the only way to accept a word is by an "interplay" of the two components, i.e., the sequence of production sets applied looks like $\ldots, P_1, P_2, P_1, P_2, \ldots$. W.l.o.g. assume that P_1 starts the derivation process, reducing a subword of w to some nonterminal, say A. The only way to continue is an application of a rule of P_2. At this point of derivation, we have to distinguish two cases, as illustrated in Figure 1.

1. The rule chosen from P_2 contains the previously introduced nonterminal A on the left-hand side. Hence, A with a left- and right-context is replaced by another nonterminal again. The only way to successively continue the derivation is to apply P_1 again, reducing a sub-word that contains the previously introduced nonterminal. Otherwise, the previous application of a rule of P_1 would have not been possible. Further analysis of the derivation process shows that in this case the derivation has a "fish-bone" structure like in a linear grammar.
2. The rule chosen from P_2 does not contain the previously introduced nonterminal A. Hence, after its application the derived sentential form contains exactly two nonterminals, say A and B. P_1 is not able to handle a derivation where A is involved, as long as P_2 has not changed the left- or right-context of A properly. Otherwise, the previous application of a rule of P_1 would have not been possible.

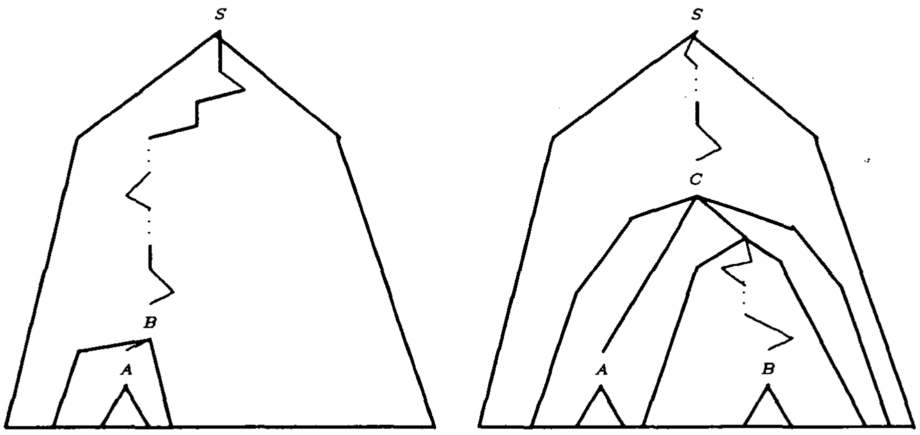

Figure 1: The cases 1. (left) and 2. (right; the case where A and B change their rôles are symmetric) of a $(t \wedge = 1)$ derivation of a CDGS with two components.

Thus, the applicable rule from P_1 must contain nonterminal B on the left-hand side. This interplay between P_1 and P_2 goes on while successively reducing the word at the point where P_2 has made the first application of a rule at all. The derivation is nothing else than a fish-bone again. Then at some point P_i, for $1 \leq i \leq 2$, has prepared the context for the other to make an application of a rule where the nonterminal A is involved; this rule application breaks the fish-bone into two parts, when looking at the derivation as a whole. Afterwards, the interplay of the components continues, leading to a linear derivation structure.

We sketch the construction of a linear grammar that simulates the accepting derivation of G in a generative manner. Let $LS(P_i)$ denote the left-hand sides of the productions in P_i, i.e., $LS(P_i) = \{ w \mid w \to B \in P_i \}$.

The nonterminal of the linear grammar contains the following (finite amount of) information: (1) the actual nonterminal, (2) which component and rule starts the original derivation, (3) the control for the interplay, (4) the information which left-hand sides of the productions in both components are contained as sub-words in the sentential form derived so far, and (5) additional information to compute (4).

Thus, the generating linear grammar first guesses the component that starts and ends the original derivation process. Additionally, the first rule ever applied in the original derivation is guessed, say this is $\alpha \to A$. Further, also the form of the derivation (case 1. or 2.) is guessed. These cases are treated separately:

1. A linear derivation has to be performed, starting with the rule set that ends the original derivation. The interplay as well as the application of a rule is controlled by the information stored in the current nonterminal. In this situation, a rule $u \to B$ (a linear one!) from P_i, for $1 \leq i \leq 2$, is applicable if and only if the sentential form γ contains one occurrence of B, and no word from $LS(P_i)$ occurs in γ. The latter can

be tested with information (4). During the derivation simulation, the information (1), (3), (4), and (5) is updated. When terminating (simulating $\alpha \rightarrow A$) in addition it is checked (using information (2) and (3)) whether we actually simulate the component that starts the original derivation process.

2. The second case consists of two symmetric cases (see Figure 1). Assume $\beta \rightarrow B$ is the rule applied in the second step of the original derivation. In the first part of the derivation a linear derivation is done (see above). At some point, the grammar guesses to apply the rule (which is member of the component P_i, for $1 \leq i \leq 2$) that breaks the linear structure of the original derivation. This rule is *not* linear anymore, i.e., it looks like, e.g., $C \rightarrow uAvBw$ (the symmetric case $C \rightarrow uBvAw$ is similar). Then the derivation is continued as follows: apply the rule $C \rightarrow u\alpha vBw$ under the condition that no word of $LS(P_i)$ is stored as information (4), and then update information (4) and (5) according to $C \rightarrow uAvBw$. Thereafter, the derivation process is continued in a linear manner like in case 1. The only difference lies in the termination, because we already applied $\alpha \rightarrow A$. Therefore, we must test that $\beta \rightarrow B$ is member of the set P_j, for $1 \leq j \leq 2$, which does not start the original derivation and that no word from $\{\alpha\} \cup LS(P_j)$ is stored as information (4).

This completes our construction for $\mathcal{L}^{acc}(\mathrm{CD}_2, \mathrm{CF}[-\lambda], (t \wedge = 1)) \subsetneq \mathcal{L}^{gen}(\mathrm{LIN})$. The strictness of the inclusion follows from the lemma on one-letter languages. □

The question arises whether Theorem 4.4 generalizes to CDGS with more than two components and for derivation modes $(t \wedge \leq k)$ or $(t \wedge = k)$ in general. First, let us analyze the derivation trees obtained by such systems. We only discuss the $(t \wedge = k)$-mode, but it generalizes to the other mode as well.

Let G be a two component system working in $(t \wedge = 2)$-mode. Again, we must have an interplay between the two grammars to accept a word successfully. The number of possible tree structures, compared to the $(t \wedge = 1)$-case, increases significantly, but remains finite. Why this? The start component can introduce at most two nonterminals. Then, the application of the other grammar increases the number of occurrences of nonterminals again at most by two. Now, there are only three possibilities to continue the accepting derivation: either we reduce sub-words in a linear manner (sequential rule application), or we replace two nonterminals with some context in an application of a production set (parallel rule application), or we combine several nonterminals into one (union step). Moreover, the first and second step of the derivation can be only a sequential or a parallel one, but from then on, a sequence of parallel steps can only be followed by a sequence of sequential steps, and afterwards a union step. Finally, only a sequence of sequential steps mixed with a finite number of union steps can be performed until the axiom is reached. At this point, one observes that the number of nonterminals occurring in a sentential form is bounded. Obviously, with a similar construction as in the preceding $(t \wedge = 1)$ case, a programmed grammar with appearance checking can do the simulation job. Note that the whole derivation is of finite index, too.

With a much more detailed analysis also the case of CDGS with three grammars working in $(t \wedge = 1)$ mode can be done. In general, for arbitrary number n and k, the number of possible structures for the derivation trees is bounded, so that a programmed grammar with appearance checking fulfilling the finite index restriction is able to simulate the original grammar system in a generating way. Thus, together with the lemma on one-letter languages and Theorem 4.1) we obtain:

Theorem 4.6 *If $f \in \{ (t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N} \}$, then, for any $N \in \mathbf{N} \cup \{\infty\}$,*

$$\mathcal{L}^{acc}(\mathrm{CD}_N, \mathrm{CF}[-\lambda], f) \subset \mathcal{L}^{gen}(\mathrm{CD}_\infty, \mathrm{CF}[-\lambda], f) = \mathcal{L}^{gen}_{fin}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac}).$$

5 More on hybrid CDGS

As already observed in the previous section (see also [16]), accepting and generating modes coincide when only considering the $*$-, $\leq k$-, $= k$-, $\geq k$, and $(\geq k_1 \wedge \leq k_2)$-modes. We summarize these facts in the following theorem without proof.

Theorem 5.1 *If $F \subseteq \{*\} \cup \{ \leq k, = k, \geq k \mid k \in \mathbf{N} \} \cup \{ (\geq k_1 \wedge \leq k_2) \mid k_1, k_2 \in \mathbf{N}, k_1 \leq k_2 \}$, then, for any $N \in \mathbf{N} \cup \{\infty\}$,*

$$\mathcal{L}^{gen}(\mathrm{HCD}_N, \mathrm{CF}[-\lambda], F) = \mathcal{L}^{acc}(\mathrm{HCD}_N, \mathrm{CF}[-\lambda], F).$$

To exhibit the relations between other mode combinations, we have to explore their generating and accepting power in more detail. From [14], we summarize:

Theorem 5.2 *1. If $\emptyset \neq F \subseteq \{*, t\} \cup \{ \leq k \mid k \in \mathbf{N} \} \cup \{= 1, \geq 1\}$, then*

$$\mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F \cup \{(t \wedge = 1)\}) = \mathcal{L}^{gen}(\mathrm{O}, \mathrm{CF}[-\lambda]).$$

2. If $\emptyset \neq F \subseteq \{ = k, \geq k \mid k \geq 2 \} \cup \{ (t \wedge \geq k) \mid k \geq 2 \}$, then

$$\mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F \cup \{(t \wedge = 1)\}) = \mathcal{L}^{gen}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac}).$$

3. If $\emptyset \neq F \subseteq \{*, t\} \cup \{ \leq k, = k, \geq k, (t \wedge \geq k) \mid k \in \mathbf{N}\} \cup \{(\geq k_1 \wedge \leq k_2) \mid k_1, k_2 \in \mathbf{N}$ and $k_1 \leq k_2 \}$, then

$$\mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F \cup \{(t \wedge = 2)\}) = \mathcal{L}^{gen}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac}).$$

4. Let $\emptyset \neq F \subseteq \{*, t\} \cup \{ \leq k \mid k \in \mathbf{N} \}$. For every $k \in \mathbf{N}$, $k \geq 2$,

$$\mathcal{L}^{gen}(\mathrm{O}, \mathrm{CF}[-\lambda]) \subseteq \mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F \cup \{(t \wedge \leq k)\}) \subseteq \mathcal{L}^{gen}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ut}).$$

Since ordered languages are strictly included in programmed languages with unconditional transfer, at least one of the inclusions is strict.

Theorem 5.3 *Let $F \subseteq D$ contain one mode from $\{*\} \cup \{ \leq k, = k, \geq k \mid k \in \mathbf{N} \} \cup \{ (\geq k_1 \wedge \leq k_2) \mid k_1, k_2 \in \mathbf{N}$ and $k_1 \leq k_2 \}$ and one from $\{ (t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N} \}$. Then, we have:*

1. $\mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}-\lambda, F) = \mathcal{L}^{gen}(\mathrm{CS})$, *and*

2. $\mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}, F) = \mathcal{L}^{gen}(\mathrm{RE})$.

Proof. Our proof is very similar to the t-mode case shown in [16, Theorem 4.2]. We give only technical details here. First, we consider the λ-free case. It is easy to construct a simulating linear bounded automaton accepting $L^{acc}(G)$ in case not admitting λ-rules. Therefore, the inclusion $\mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F) \subseteq \mathcal{L}^{gen}(\mathrm{CS})$ is clear. We have to show the other inclusion.

By a standard argument, it can be shown that (*) $\mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F)$ is closed under union and embraces the context-free languages. Let $L \in \mathcal{L}^{gen}(\mathrm{CS})$, $L \subseteq T^*$. Then, $L = \bigcup_{a,b,c\in T}(\{a\}T^+\{bc\} \cap L) \cup (L \cap T) \cup (L \cap T^2) \cup (L \cap T^3)$. Since L is context-sensitive, $L_{abc} = \{\, w \in T^+ \mid awbc \in L \,\}$ is context-sensitive due to the closure of $\mathcal{L}^{gen}(\mathrm{CS})$ under derivatives. By (*), it is sufficient to show that $\{a\}M\{bc\} \in \mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}-\lambda, F)$ provided that $M \subseteq T^+$ is context-sensitive.

By simple prolongation arguments, it suffices to show that $\{a\}M\{bc\}$ belongs to $\mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}-\lambda, \{*\} \cup \{(t \wedge = 1)\})$ provided that $M \subseteq T^+$ is context-sensitive.

Let $G = (N, T, S, P)$ be a context-sensitive grammar without λ-productions in Kuroda normal form generating M. Let us assume a unique label r being attached to any genuine context-sensitive rule of the form $XU \to YZ$ with $X, U, Y, Z \in N$; the set of labels is denoted by $\mathrm{Lab}_{\mathrm{cs}} = \{r_1, \ldots, r_R\}$.

We construct a $(\mathrm{HCD}_\infty, \mathrm{CF}-\lambda, \{*\} \cup \{(t \wedge = 1)\})$ system

$$G' = (N', T, S', P_0, P_{1,1}, P_{1,2}, P_{1,3}, P_{1,4} \ldots, P_{R,1}, P_{R,2}, P_{R,3}, P_{R,4})$$

accepting $\{a\}M\{bc\}$. The common terminal alphabet of these grammars is T, and their nonterminal alphabet is

$$N' \quad = \quad N \cup \{C_1, \ldots C_R\} \cup \{\, D', D'' \mid D \in N \cup T \,\} \cup \{\, A, B, C, S', F \,\}$$

(the unions being disjoint).

The component P_0 working in $*$-mode equals $\{\, a \to A, b \to B, c \to C \,\} \cup \{ASBC \to S'\} \cup \{\, w \to D \mid D \in N \text{ and } D \to w \in P,\, w \in T \,\} \cup \{\, D \to D', D \to D'' \mid D \in N \,\}$ and is used for four purposes: (1) It turns the left delimiter a into A and the right delimiters b and c into B and C (initialization). (2) The check of the correctness of this initialization application is postponed until the last applicable production $ASBC \to S'$ does its work (termination). (3) Context-free rules can be simulated here. (4) Colouring of nonterminals into (double-)primed counterparts prepares the simulation of a genuine context-sensitive rule.

Finally, we introduce four production sets working in $(t \wedge = 1)$-mode for the simulation of a genuine context-sensitive production $r_\rho : XU \to YZ \in P$:

$$\begin{aligned} P_{\rho,1} &= \{C \to C_\rho\} \cup \{\, Y'z \to F \mid z \neq Z'' \,\} \cup \{\, yZ'' \to F \mid y \neq Y' \,\}, \\ P_{\rho,2} &= \{\, C \to F \,\} \cup \{\, C_\sigma \to F \mid \sigma \neq \rho \,\} \cup \\ &\quad\ \{Y' \to X\} \cup \{\, D' \to F, E'' \to F \mid D, E \in N \wedge E'' \neq Z'' \},\\ P_{\rho,3} &= \{\, C \to F \,\} \cup \{\, C_\sigma \to F \mid \sigma \neq \rho \,\} \cup \end{aligned}$$

$$\begin{aligned} & \{Z'' \to U\} \cup \{ D' \to F, D'' \to F \mid D \in N\}, \quad \text{and} \\ P_{\rho,4} &= \{ C_\rho \to C \} \cup \{ C_\sigma \to F \mid \sigma \neq \rho \} \cup \{ D' \to F, D'' \to F \mid D \in N\}. \end{aligned}$$

Observe that the second production set only serves for checking whether the first production set has correctly nondeterministically selected two adjacent marked occurrences Y and Z. These checks are always possible, since we introduced left and right end-markers A and B, respectively.

If we allow λ-productions, these can be put in the P_0-component, too. □

Corollary 5.4 *Let $F \subseteq \{t\} \cup \{ (t \wedge \leq k), (t \wedge = k), (t \wedge \geq k) \mid k \in \mathbf{N} \}$. If F contain one of the modes $\{t\} \cup \{ (t \wedge \geq k) \mid k \in \mathbf{N} \}$ and one of the modes $\{ (t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N} \}$, then we have $\mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F) = \mathcal{L}^{gen}(\mathrm{CS})$.*

Proof. By our result [16, Theorem 4.2] regarding t-components only, it is only to prove that additional components working in one of the $\{ (t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N} \}$-modes do not enhance the accepting power. Again, λ-rules do not add to the power, and an easy simulation by a linear bounded automaton shows the assertion. □

When contrasting generating and accepting grammars, we obtain:

Theorem 5.5 *1. Let $F \subseteq D$ contain one of the modes $\{*, t\} \cup \{ \leq k, = k, \geq k, (t \wedge \geq k) \mid k \in \mathbf{N} \} \cup \{ (\geq k_1 \wedge \leq k_2) \mid k_1, k_2 \in \mathbf{N}$ and $k_1 \leq k_2 \}$ and one of the modes $\{ (t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N} \}$. Then, we have:*

$$\mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF} - \lambda, F) \subset \mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF} - \lambda, F).$$

2. Assume that $F \subseteq D$ contains one of the modes $\{*\} \cup \{ \leq k, = k, \geq k \mid k \in \mathbf{N} \} \cup \{ (\geq k_1 \wedge \leq k_2) \mid k_1, k_2 \in \mathbf{N},\ k_1 \leq k_2 \}$ and one of the modes $\{ (t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N} \}$. Then, we have:

$$\mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF}, F) \subseteq \mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}, F).$$

(a) This inclusion is known to be strict in case F contains none of the modes $\{ (t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N}, k \geq 2 \}$.

(b) This inclusion is known to be non-strict in case F contains one of the modes $\{ (t \wedge = k) \mid k \in \mathbf{N}, k \geq 2 \}$.

3. Let $F \subseteq \{t\} \cup \{ (t \wedge \geq k), (t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N} \}$ and let F contain one of the modes $\{t\} \cup \{ (t \wedge \geq k) \mid k \in \mathbf{N} \}$ and one of the modes $\{ (t \wedge \leq k), (t \wedge = k) \mid k \in \mathbf{N} \}$.

(a) If $F \cap (\{ (t \wedge = k) \mid k \in \mathbf{N}, k \geq 2 \} \cup \{ (t \wedge \geq k) \mid k \in \mathbf{N}, k \geq 2 \}) \neq \emptyset$, we have

$$\mathcal{L}^{gen}(\mathrm{CS}) = \mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}, F) \subset \mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF}, F) = \mathcal{L}^{gen}(\mathrm{RE}).$$

(b) *Otherwise,* $\mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}, F)$ *is not contained in* $\mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF}, F)$*, since* $\mathcal{L}^{gen}(\mathrm{CS})$ *is not contained in* $\mathcal{L}^{gen}(\mathrm{O}, \mathrm{CF})$. □

Theorem 5.6 *Let* $F \subseteq D$ *contain one of the modes* $\{*, t\} \cup \{\leq k, = k, \geq k \mid k \in \mathrm{N}\} \cup \{(\geq k_1 \wedge \leq k_2) \mid k_1, k_2 \in \mathrm{N}$ *and* $k_1 \leq k_2\} \cup \{(t \wedge \geq k) \mid k \in \mathrm{N}\}$ *and one of the modes* $\{(t \wedge \leq k), (t \wedge = k) \mid k \in \mathrm{N}\}$*. Then, we have:*

$$\mathcal{L}^{acc}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F) = \mathcal{L}^{acc}(\mathrm{HCD}_2, \mathrm{CF}[-\lambda], F).$$

Proof. We can distinguish two cases:

1. If $F \cap (\{t\} \cup \{(t \wedge \geq k) \mid k \in \mathrm{N}\}) \neq \emptyset$, then [16, Theorem 4.5] is applied.

2. If $F \cap (\{*\} \cup \{\leq k, = k, \geq k \mid k \in \mathrm{N}\} \cup \{(\geq k_1 \wedge \leq k_2) \mid k_1, k_2 \in \mathrm{N} \wedge k_1 \leq k_2\}) \neq \emptyset$, then following idea is helpful: First, due to our last theorem, it is sufficient to consider $*$- and $(t \wedge = 1)$-components, because every context-sensitive (or even recursively enumerable, if λ-rules are admitted) language can be accepted in such a way. Then, possibly a prolongation argument is applied. It is possible to colour each symbol of the originally given hybrid CDGS with a special colour indicating the $(t \wedge = 1)$-component we are going to apply next. Furthermore, we assume only coloured versions of terminals appearing in the sentential form. All original $(t \wedge = 1)$-components are put together, where each set of productions only works on its private alphabet. In addition, mixtures are excluded introducing productions of the form $XY \to F$ (where X and Y are from different colours or terminal symbols). □

Naturally, Theorem 5.6 cannot be improved, since (hybrid) CDGS with context-free rules having one component can accept at most the context-free languages. Just as an aside, we remark in this place:

Theorem 5.7 *For every* $k \in \mathrm{N}$,

$$\begin{aligned} \mathcal{L}(\mathrm{FIN}) &= \mathcal{L}^{acc}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], (t \wedge = k)) = \mathcal{L}^{gen}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], (t \wedge = k)) \\ &= \mathcal{L}^{acc}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], (t \wedge \leq k)) = \mathcal{L}^{gen}(\mathrm{CD}_1, \mathrm{CF}[-\lambda], (t \wedge \leq k)). \quad \square \end{aligned}$$

Theorem 5.6 contrasts sharply with our results in the generating case. Unfortunately, many points are still open here. Therefore, we only quote two preliminary results which prove that accepting hybrid systems are much more powerful than their generating counterparts in many cases.

Theorem 5.8 *1. If* $f \in \{(t \wedge \leq 1), (t \wedge = 1)\}$*, then, for* $n \in \{1, 2\}$*, we have*

$$\mathcal{L}^{gen}(\mathrm{HCD}_n, \mathrm{CF}[-\lambda], \{*, t\} \cup \{\leq k \mid k \in \mathrm{N}\} \cup \{f\}) = \mathcal{L}^{gen}(\mathrm{CF}).$$

2. Let $\emptyset \neq F \subseteq \{t\} \cup \{(t \wedge \geq k) \mid k \in \mathrm{N}\}$*. For every* $f \in \{(t \wedge \leq k), (t \wedge = k) \mid k \in \mathrm{N}\}$*, we have*

$$\mathcal{L}^{gen}(\mathrm{HCD}_\infty, \mathrm{CF}[-\lambda], F \cup \{f\}) = \mathcal{L}^{gen}(\mathrm{HCD}_4, \mathrm{CF}[-\lambda], F \cup \{f\}).$$

Especially, observe that

$$\begin{array}{rcl}\mathcal{L}^{gen}(\mathrm{CF}) & = & \mathcal{L}^{gen}(\mathrm{HCD}_2, \mathrm{CF}[-\lambda], \{*\} \cup \{(t \wedge = 1)\}) \\ & \subset & \mathcal{L}^{acc}(\mathrm{HCD}_2, \mathrm{CF}[-\lambda], \{*\} \cup \{(t \wedge = 1)\}) = \mathcal{L}^{gen}(\mathrm{RE})\end{array}$$

is an amazing jump from the generating to the accepting power.

Moreover, it is always interesting to see for what hybridization is really good. In this respect, we want to contrast $\mathcal{L}^{gen}(\mathrm{CF}) = \mathcal{L}^{acc}(\mathrm{CD}_2, \mathrm{CF}, *)$ and $\mathcal{L}^{gen}(\mathrm{LIN}) = \mathcal{L}^{acc}(\mathrm{CD}_2, \mathrm{CF}, (t \wedge = 1))$ with $\mathcal{L}^{gen}(\mathrm{RE}) = \mathcal{L}^{acc}(\mathrm{HCD}_2, \mathrm{CF}, \{*\} \cup \{(t \wedge = 1)\})$.

6 (Prescribed) Teams as acceptors

A *cooperating distributed grammar system with prescribed teams* (PTCDGS for short), confer [17, 18, 19, 26], is a construct $G = (N, T, S, P_1, \ldots, P_n, Q_1, \ldots, Q_m)$, with $n, m \in \mathbf{N}$, where $(N, T, S, P_1, \ldots, P_n)$ is a usual CDGS and $Q_1, \ldots, Q_m$ are *teams*, i.e., subsets of $\{P_1, \ldots, P_n\}$. If each subset of $\{P_1, \ldots, P_n\}$ can be a team, then we say that G has *free teams*, G is called a *cooperating distributed grammar system with teams* (TCDGS for short).

For $x, y \in (N \cup T)^*$, we write $x \Rightarrow_{Q_i} y$ for some team $Q_i = \{P_{j_1}, \ldots, P_{j_s}\}$ if and only if $x = x_1 A_1 x_2 A_2 \ldots x_s A_s x_{s+1}$, $y = x_1 y_1 x_2 y_2 \ldots x_s y_s x_{s+1}$, where $x_\ell \in (N \cup T)^*$, $1 \leq \ell \leq s+1$, $A_r \to y_r \in P_{j_r}$, $1 \leq r \leq s$. Having defined the one-step derivation, we can easily define derivations in Q_i of k steps, at most k steps or at least k steps, and of any number of steps, denoted again by $\Rightarrow_{Q_i}^{=k}$, $\Rightarrow_{Q_i}^{\leq k}$, $\Rightarrow_{Q_i}^{\geq k}$, $\Rightarrow_{Q_i}^{*}$, respectively. For maximal derivations in a team Q_i, we can consider three variants:

1. $x \Rightarrow_{Q_i}^{t_0} y$ if and only if $x \Rightarrow_{Q_i}^{*} y$ and there is no z such that $y \Rightarrow_{Q_i} z$ [18].
2. $x \Rightarrow_{Q_i}^{t_1} y$ if and only if $x \Rightarrow_{Q_i}^{*} y$ and for no component $P_{j_r} \in Q_i$ and no z there is a derivation $y \Rightarrow_{P_{j_r}} z$ [19].
3. $x \Rightarrow_{Q_i}^{t_2} y$ if and only if $x \Rightarrow_{Q_i}^{*} y$ and there is a component $P_{j_r} \in Q_i$ such that for no z there is a derivation $y \Rightarrow_{P_{j_r}} z$ [26].

Given a (P)TCDGS G working in mode $f \in \{*, t_0, t_1, t_2\} \cup \{\leq k, = k, \geq k \mid k \in \mathbf{N}\}$, we define the derivation relation $x \Rightarrow y$ if and only if there exists a team Q_i such that $x \Rightarrow_{Q_i}^{f} y$. As usual, the language generated in f-mode by G is defined as

$$\begin{array}{rcl}L_f^{gen}(G) & := & \{ w \in T^* \mid S \Rightarrow_{Q_{i_1}}^{f} \alpha_1 \Rightarrow_{Q_{i_2}}^{f} \ldots \Rightarrow_{Q_{i_{\ell-1}}}^{f} \alpha_{\ell-1} \Rightarrow_{Q_{i_\ell}}^{f} \alpha_\ell = w \text{ with} \\ & & \ell \geq 1,\ 1 \leq i_j \leq m, \text{ and } 1 \leq j \leq \ell \}.\end{array}$$

We denote by $\mathcal{L}^{gen}((\mathrm{P})\mathrm{TCD}, \mathrm{CF}[-\lambda], f)$ the family of languages generated by [λ-free] (P)TCDGS. Correspondingly, accepted languages and language classes are defined. We summarize the known results on generating (P)TCDGS.

Theorem 6.1 *1. For all $f \in \{*\} \cup \{\leq k, = k, \geq k \mid k \in \mathbf{N}\}$,*

$$\mathcal{L}^{gen}(\mathrm{PTCD}, \mathrm{CF}[-\lambda], f) = \mathcal{L}^{gen}(\mathrm{P}, \mathrm{CF}[-\lambda]).$$

2. *For all $f \in \{t_0, t_1, t_2\}$,*

$$\mathcal{L}^{gen}(\mathrm{TCD}, \mathrm{CF}[-\lambda], f) = \mathcal{L}^{gen}(\mathrm{PTCD}, \mathrm{CF}[-\lambda], f) = \mathcal{L}^{gen}(\mathrm{P}, \mathrm{CF}[-\lambda], \mathrm{ac}).$$

The strictness of the inclusion $\mathcal{L}^{gen}(\mathrm{TCD}, \mathrm{CF}[-\lambda], f) \subseteq \mathcal{L}^{gen}(\mathrm{P}, \mathrm{CF}[-\lambda])$, for $f \in \{*\} \cup \{\leq k, =k, \geq k \mid k \in \mathrm{N}\}$ is open. However, it is quite clear that the generating and accepting capacity of (P)TCDGS working in one of the modes $f \in \{*\} \cup \{\leq k, =k, \geq k \mid k \in \mathrm{N}\}$ coincides.

Theorem 6.2 *For all $f \in \{*\} \cup \{\leq k, =k, \geq k \mid k \in \mathrm{N}\}$, we have:*

1. $\mathcal{L}^{acc}(\mathrm{PTCD}, \mathrm{CF}[-\lambda], f) = \mathcal{L}^{gen}(\mathrm{PTCD}, \mathrm{CF}[-\lambda], f) = \mathcal{L}^{gen}(\mathrm{P}, \mathrm{CF}[-\lambda])$,
2. $\mathcal{L}^{acc}(\mathrm{TCD}, \mathrm{CF}[-\lambda], f) = \mathcal{L}^{gen}(\mathrm{TCD}, \mathrm{CF}[-\lambda], f)$. □

As regards the different t-modes, the observations sketched in the following allow us to carry over our result contained in [16, Theorems 4.2, and 4.5]:

1. If we have only one-element teams, all t-modes introduced for PTCDGS coincide (with the classical t-mode). Hence, such PTCDGS accept all context-sensitive languages.

2. The simulation also works when permitting larger (arbitrary) teams, since it is possible to use different colours in the simulation of different genuine context-sensitive rules (simulating a context-sensitive grammar in Kuroda normal form). "Wrong" colours are alway sent to the failure symbol. Possibly, if we choose teams containing more than one component, two or more rules of the originally given context-sensitive grammar are simulated in parallel, but this does no harm, since a sequentialization choosing singleton teams is always possible. Further observe that a simulation of a genuine context-sensitive rule by a t-mode component as given in [16, Theorems 4.2, and 4.5] always takes the same number of steps, so that no garbage can be derived employing the t_2-mode. Hence, such TCDGS can also accept all context-sensitive languages.

3. As regards λ-rules, they are simply useless in case of classical CDGS working in t-mode, since such a rule is always applicable. A similar argument is applied to (P)TCDGS working in t_1-mode. The situation is different for CDGS working in t_0-mode or in t_2-mode. Why? First, we can assume that the type-0-grammar we are going to simulate has only one production of the form $E \to \lambda$, where E is a special nonterminal symbol serving as a place-holder for the empty word. Moreover, due to the closure properties of $\mathcal{L}^{gen}(\mathrm{RE})$, we can assume an additional left-marker symbol $\#$. Now, λ-productions can be simulated by three components, $P_{\lambda,1} = \{\# \to \#, \# \to \#'\}$, $P_{\lambda,2} = \{\#' \to \#\}$, and $P_{\lambda,3} = \{\lambda \to E\}$. When combining $P_{\lambda,1}$ and $P_{\lambda,3}$ into one team, arbitrarily many E's can be introduced. When using free teams, other combinations are now possible which may block a t-mode derivation prematurely. Therefore, this case remains as an open question.

We collect our observations in the following.

Theorem 6.3 *For each $f \in \{t_0, t_1, t_2\}$, we find in general*

1. $\mathcal{L}^{acc}(\mathrm{PTCD}, \mathrm{CF}-\lambda, f) = \mathcal{L}^{acc}(\mathrm{TCD}, \mathrm{CF}-\lambda, f) = \mathcal{L}^{gen}(\mathrm{CS})$, *and*

2. $\mathcal{L}^{gen}(\mathrm{CS}) \subseteq \mathcal{L}^{acc}(\mathrm{TCD}, \mathrm{CF}, f) \subseteq \mathcal{L}^{acc}(\mathrm{PTCD}, \mathrm{CF}, f) \subseteq \mathcal{L}^{gen}(\mathrm{RE})$.

More specifically, we obtain by our third observation:

3. $\mathcal{L}^{acc}(\mathrm{PTCD}, \mathrm{CF}, t_1) = \mathcal{L}^{acc}(\mathrm{TCD}, \mathrm{CF}, t_1) = \mathcal{L}^{gen}(\mathrm{CS})$, *and*

4. $\mathcal{L}^{acc}(\mathrm{PTCD}, \mathrm{CF}, t_0) = \mathcal{L}^{acc}(\mathrm{PTCD}, \mathrm{CF}, t_2) = \mathcal{L}^{gen}(\mathrm{RE})$. □

Comparing the generating versus the accepting capacity, we get:

Corollary 6.4 *For each $f \in \{t_0, t_1, t_2\}$, we find*

1. $\mathcal{L}^{gen}((\mathrm{P})\mathrm{TCD}, \mathrm{CF}-\lambda, f) \subset \mathcal{L}^{acc}((\mathrm{P})\mathrm{TCD}, \mathrm{CF}-\lambda, f)$;

2. $\mathcal{L}^{acc}((\mathrm{P})\mathrm{TCD}, \mathrm{CF}, f) \subseteq \mathcal{L}^{gen}((\mathrm{P})\mathrm{TCD}, \mathrm{CF}, f)$;

3. $\mathcal{L}^{acc}((\mathrm{P})\mathrm{TCD}, \mathrm{CF}, t_1) \subset \mathcal{L}^{gen}((\mathrm{P})\mathrm{TCD}, \mathrm{CF}, t_1)$;

4. $\mathcal{L}^{acc}(\mathrm{PTCD}, \mathrm{CF}, g) = \mathcal{L}^{gen}(\mathrm{PTCD}, \mathrm{CF}, g)$, *for $g \in \{t_0, t_2\}$*. □

7 Conclusions

We continued our studies on accepting systems of grammars, paying special attention towards internally hybrid modes and teams. In this way, we also found first examples of grammar mechanisms whose generating power is greater than its accepting power.

In [2], two variants of the t-mode, namely weak t and stagnation, have been introduced: In weak t-mode a component P_j works on a string up to the point a sentential form w is obtained with $w \Rightarrow_{P_j} v$ implies $w = v$. This corresponds to the adult mechanism known from the theory of Lindenmayer systems. Now, another component may start its work with w.

The stagnation-mode is defined as follows: a component P_j works on a string deriving subsequently $w_1 \Rightarrow_{P_j} w_2 \Rightarrow_{P_j} w_3 \Rightarrow_{P_j} \cdots \Rightarrow_{P_j} w_n$, and $w_n \Rightarrow_{P_j} v$ implies $w_i = v$ for some $1 \leq i \leq n$. Now, another component may start its work with w_n.

Analyzing the proof of [16, Theorem 4.2], we see that both variants also characterize the context-sensitive languages when seen as language acceptors. These results on the accepting capacity of these modes have been independently obtained from [2].

References

[1] A. Atanasiu and V. Mitrana. The modular grammars. *International Journal of Computer Mathematics*, 30:101–122, 1989.

[2] H. Bordihn and E. Csuhaj-Varjú. On Competence and Completeness in CD Grammar Systems. In this volume.

[3] H. Bordihn and H. Fernau. Accepting grammars with regulation. *International Journal of Computer Mathematics*, 53:1–18, 1994.

[4] H. Bordihn and H. Fernau. Accepting programmed grammars without nonterminals. In *5. GI Theorietag "Automaten und Formale Sprachen", Technical Report 9503, Universität Gießen, Arbeitsgruppe Informatik*, pages 4–16, 1995.

[5] H. Bordihn and H. Fernau. Accepting grammars and systems: an overview. In J. Dassow, G. Rozenberg, and A. Salomaa, editors, *Developments in Language Theory II; at the crossroads of mathematics, computer science and biology*, pages 199–208. Singapore: World Scientific, 1996.

[6] H. Bordihn and H. Fernau. Accepting grammars and systems via context condition grammars. *Journal of Automata, Languages and Combinatorics*, 1(2):97–112, 1996.

[7] E. Csuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 26(1/2):49–63, 1990.

[8] E. Csuhaj-Varjú et al. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation.* London: Gordon and Breach, 1994.

[9] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science.* Berlin: Springer, 1989.

[10] R. Engelmore and T. Morgan. *Blackboard Systems.* Addison-Wesley, 1988.

[11] H. Fernau and H. Bordihn. Remarks on accepting parallel systems. *International Journal of Computer Mathematics*, 56:51–67, 1995.

[12] H. Fernau and R. Freund. Accepting array grammars with control mechanisms. Unpublished manuscript, 1996.

[13] H. Fernau and R. Freund. Bounded parallelism in array grammars used for character recognition. In P. Perner, P. Wang, and A. Rosenfeld, editors, *Advances in Structural and Syntactical Pattern Recognition (Proceedings of the SSPR'96)*, volume 1121 of *LNCS*, pages 40–49. Berlin: Springer, 1996.

[14] H. Fernau, R. Freund, and M. Holzer. External versus internal hybridization for cooperating distributed grammar systems. Technical Report TR 185–2/FR–1/96, Technische Universität Wien (Austria), 1996.

[15] H. Fernau and M. Holzer. Bidirectional cooperating distributed grammar systems. Technical Report WSI–96–1, Universität Tübingen (Germany), Wilhelm-Schickard-Institut für Informatik, 1996.

[16] H. Fernau, M. Holzer, and H. Bordihn. Accepting multi-agent systems: the case of cooperating distributed grammar systems. *Computers and Artificial Intelligence*, 15(2-3):123–139, 1996.

[17] R. Freund. Array grammars with prescribed teams of array productions. In *Developments in Language Theory II; at the crossroads of mathematics, computer science and biology*, pages 220–229. London: Gordon and Breach, 1996.

[18] R. Freund and Gh. Păun. A variant of team cooperation in grammar systems. *Journal of Universal Computer Science*, 1(2):105–130, 1995.

[19] L. Kari, A. Mateescu, Gh. Păun, and A. Salomaa. Teams in cooperating distributed grammar systems. *Journal of Experimental and Theoretical AI*, 7:347–359, 1995.

[20] R. Meersman and G. Rozenberg. Cooperating grammar systems. In *Proceedings of Mathematical Foundations of Computer Science MFCS'78*, volume 64 of *LNCS*, pages 364–374. Berlin: Springer, 1978.

[21] R. Meersman, G. Rozenberg, and D. Vermeir. Persistent ET0L systems. *Information Sciences*, 18:189–212, 1979.

[22] V. Mitrana. Hybrid cooperating/distributed grammar systems. *Computers and Artificial Intelligence*, 12(1):83–88, 1993.

[23] N. J. Nilsson. *Principles of Artificial Intelligence*. Berlin: Springer, 1982.

[24] Gh. Păun. On the generative capacity of hybrid CD grammar systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 30(4):231–244, 1994.

[25] Gh. Păun. Grammar systems: a grammatical approach to distribution and cooperation. In *Automata, Languages and Programming; 22nd International Colloquium, ICALP'95, Szeged, Hungary*, volume 944 of *LNCS*, pages 429–443. Berlin: Springer, 1995.

[26] Gh. Păun and G. Rozenberg. Prescribed teams of grammars. *Acta Informatica*, 31:525–537, 1994.

[27] E. L. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–215, 1943.

[28] S. H. von Solms. Some notes on ET0L-languages. *International Journal of Computer Mathematics*, 5(A):285–296, 1976.

[29] M. J. Wooldridge and N. R Jennings. Agent theories, architectures and languages: a survey. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents; ECAI-94 Workshop on Agent Theories, Architectures, and Languages (Amsterdam 1994)*, volume 890 of *LNCS (LNAI)*, pages 1–39. Berlin: Springer, 1994.

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

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

where N, T, K 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 (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 $(x_1, \ldots, x_n), (y_1, \ldots, y_n)$, 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 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 $(S_1, \ldots, S_n)$, that is

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

(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.)

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

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

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:

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

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]).

$$\begin{tikzcd}
 & & & RE & & & \\
 & & & CS \arrow[u] & & & \\
 & & PC(CF) \arrow[uur] & & NPC(CF) \arrow[ll] & & \\
PC(RL) \arrow[urr] & & & & & & NPC(RL) \arrow[llllll] \arrow[ull] \\
 & & & MAT \arrow[uuu] \arrow[uul] & & & \\
 & & CPC(CF) \arrow[uuu] \arrow[uuuur] & & NCPC(CF) \arrow[uuu] \arrow[uuuul] & & \\
 & & & CF \arrow[ul] \arrow[uu] \arrow[ur] & & & \\
 & CPC(RL) \arrow[uuuul] \arrow[uur] \arrow[uuurr] & & & & NCPC(RL) \arrow[uul] \arrow[uuuur] & \\
 & & & LIN \arrow[uu] \arrow[uuuuulll] & & & \\
 & & & REG \arrow[u] \arrow[uull] \arrow[uurr] & & &
\end{tikzcd}$$

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 $G = (N_1, \ldots, N_n, T, S, M)$, where

1. $N_1, \ldots, N_n, T$ 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) $(S \rightarrow A_1 A_2 \ldots A_n),\ A_i \in N_i, 1 \leq i \leq n,$;

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

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

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

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

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

$$x \sqcup\!\sqcup\, y = \{x_1y_1x_2y_2\dots x_ny_n \mid x = x_1x_2\dots x_n, y = y_1y_2\dots y_n,$$
$$x_i, y_i \in V^*, 1 \leq i \leq n, n \geq 1\}.$$

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

$$\begin{aligned}
N &= \{S_1, S_2, S_3, S_4, X\} \cup \{X_a \mid a \in V\},\\
P_1 &= \{S_1 \to S_1,\ S_1 \to Q_2S_1,\ S_1 \to Q_3S_1,\ S_1 \to Q_2Q_3,\ S_1 \to Q_3Q_2\},\\
P_2 &= \{S_2 \to Q_4,\ X \to c\} \cup \{X_a \to aS_2 \mid a \in V\},\\
P_3 &= \{S_3 \to Q_4,\ X \to \bar{c}\} \cup \{X_a \to \bar{a}S_3 \mid a \in V\},\\
P_4 &= \{S_4 \to X_a,\ X_a \to X_a \mid a \in V\} \cup \{S_4 \to X\}.
\end{aligned}$$

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.

Acta Cybernetica **12** (1996) 397-409.

Parallel Communicating Grammar Systems with Separated Alphabets*

Valeria MIHALACHE †

Abstract

The generative capacity of parallel communicating grammar systems is considered in the context that the component grammars have distinct terminal and nonterminal sets. In the regular case, this results in strictly more powerful systems in comparison to the classical ones. In the context-free case, characterization of recursively enumerable languages is obtained when λ-rules are allowed in non-centralized returning systems, deriving in the synchronized mode. Unsynchronized context-free systems with separated alphabets have the same power as the corresponding usual systems.

1 Introduction

One of the main trends of our days in several fields of computer science is to solve a complex problem by dividing it into subproblems, and then having it solved in a cooperative mode by several "processors". The concretization of this trend in grammar theory are the so-called *grammar systems.*

There are two basic models of grammar systems: *cooperating distributed* (CD, for short) grammar systems, which have been introduced in [2] (a former variant can be found in [7]; a particular case appears also in [1]), and *parallel communicating* (PC, for short) grammar systems, which have been introduced in [10].

Roughly speaking, a grammar system consists of several (Chomsky) grammars (called *components*) working together, towards generating a common language. In a CD grammar system the component grammars work in turn, on the same sentential form, only one being active at a given moment, according to a predefined protocol. In a PC grammar system the components work simultaneously, in a synchronized manner, each having its own sentential form and cooperating with the others by communication, which is done by request. The Artificial Intelligence counterpart

*Research supported by the Academy of Finland, project 11281

†Faculty of Mathematics, University of Bucharest, Str. Academiei 17, 70109 Bucharest, Romania. Current address: Turku Centre for Computer Science (TUCS), Lemminkäisenkatu 14 A, 4th Floor, 20520, Turku, Finland

of a CD grammar system is the *blackboard model* in problem solving, whereas to PC grammar system the *classroom model* corresponds (see [3]).

In the original definition, for parallel communicating grammar systems it is assumed that all the grammars have the same terminal and nonterminal sets. This is very convenient in terms of the classroom model. Thus, it is rather natural to assume that all the pupils in a classroom have similar background and similar abilities (that is, the associated grammars in the system share the same nonterminal set), and also that they are asked to perform similar tasks (in the corresponding formal modelisation, this implies the same terminal set for all the grammars). However, if one considers that not pupils are working towards solving a problem, but agents, instead, the working protocol being the same as in the classroom model, such assumptions are not natural anymore. Agents can have to perform totally different tasks, and they can have different skills.

Such a set-up can be modeled in the grammar systems framework by a slightly modification of the original variant of PC grammar systems. One can consider that any of the component grammars of the system has its own terminal and nonterminal sets ([3], [9]). The main difference between these systems and the usual ones is that here the same letter can act as terminal symbol in one grammar and nonterminal in another one. In the regular case, PC grammar systems modified like that are proved to be more powerful than systems of the initial form. Furthermore, in the context-free case, characterization of recursively enumerable languages is obtained.

2 Preliminary definitions

Throughout this paper, we use the notation and basic results of formal language theory from [4], [11] ; for grammar systems notions we refer to [3], [5]. We specify here only some notation.

For an alphabet V, V^* denotes the free monoid generated by V; the empty string is denoted by λ, $|x|$ is the length of $x \in V^*$ and $|x|_U$ is the number of occurrences in $x \in V^*$ of symbols of $U \subseteq V$. The classes of regular, context-free, type-0 grammars and matrix grammars with appearance checking are denoted by REG, CF, RE, MAT_{ac}, respectively. Unless otherwise specified, we consider in this paper only generative tools without λ-productions.

For a class X of generative mechanisms, the family of languages generated by elements of X is denoted by $\mathbf{L}(X)$.

Definition 1 *Let $n \geq 1$ be a natural number. A* parallel communicating grammar system of degree n with separated alphabets *(PC grammar system* of type s, *for short) is an $(n+1)$-tuple*

$$\Gamma = (K, G_1, \ldots, G_n),$$

where $K = \{Q_1, Q_2, \ldots, Q_n\}$ and

$$G_i = (N_i \cup K, T_i, P_i, S_i), 1 \leq i \leq n,$$

are usual Chomsky grammars (the sets N_i, T_i, K being mutually disjoint, for any $i, 1 \leq i \leq n$).

We write $V_i = N_i \cup T_i \cup K$ and $V_\Gamma = \bigcup_{i=1}^{n}(N_i \cup T_i) \cup K$. The grammars $G_i, 1 \leq i \leq n$, are called *components* of the system, and the elements of K are called *query symbols*; their indices, $1, \ldots, n$, point to the components $G_1, \ldots, G_n$, respectively.

Remark that the definiton of PC grammar systems of type s does not require $N_i \cap T_j = \emptyset$ for $1 \leq i, j \leq n, i \neq j$.

The convention throughout this paper is to denote the start symbol and the production set of a component of a system with the same indices as the grammar component is denoted. This convention holds for query symbols, too, so we do not need to specify in details the set of query symbols for a given system.

The derivation in PC grammar systems of type s is defined in a similar manner as for usual PC grammar systems, that is

Definition 2 *Given a PC grammar system $\Gamma = (K, G_1, \ldots, G_n)$ as above, for two n-tuples $(x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n), x_i, y_i \in V_i^*, 1 \leq i \leq n$, $x_1 \notin T_1^*$, we write $(x_1, \ldots, x_n) \Longrightarrow (y_1, \ldots, y_n)$ if one of the next two cases holds:*

(i) $|x_i|_K = 0, 1 \leq i \leq n$, and for each i, $1 \leq i \leq n$, we have $x_i \Longrightarrow y_i$ in the grammar G_i, or $x_i \in T_i^*$ and $x_i = y_i$;

(ii) there is an i, $1 \leq i \leq n$, such that $|x_i|_K > 0$; then for each such i, we write $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}, t \geq 1$, for $z_j \in V_\Gamma^*, |z_j|_K = 0, 1 \leq j \leq t+1$; if $|x_{i_j}|_K = 0, 1 \leq j \leq t$, then $y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$, providing that $y_i \in V_i^*$, [and $y_{i_j} = S_{i_j}, 1 \leq j \leq t$]; when, for some j, $1 \leq j \leq t, |x_{i_j}|_K \neq 0$, then $y_i = x_i$; for all i, $1 \leq i \leq n$, for which y_i is not specified above, we have $y_i = x_i$.

Point (i) defines (componentwise) *derivation steps*, whereas point (ii) defines *communication steps*. In a communication operation, when the communicated string x_j replaces the query symbol Q_j, we say that Q_j is satisfied. The communication has priority over the effective rewriting. If some query symbols are not satisfied at a given communication step, then they will have to be satisfied at a next one. No rewriting is possible when at least one query symbol is present.

The work of a PC grammar system with separated alphabets is blocked in three cases: (1) when a component x_i of the current n-tuple $(x_1, \ldots, x_n)$ (sometimes we shall call it a *configuration*) is not terminal with respect to G_i, but no rule of G_i can be applied to x_i, or (2) when a *circular query* appears, that is G_{i_1} introduces Q_{i_2}, G_{i_2} introduces Q_{i_3}, and so on, until $G_{i_{k-1}}$ introduces Q_{i_k} and G_{i_k} introduces Q_{i_1} (because only strings without query symbols can be communicated), or (3) when after satisfying a query, the sentential form of a grammar is not a string over the alphabet of that grammar, that is, in case *(ii)* we have $z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1} \notin V_i^*$ (we recall that for usual PC grammar systems case (3) does not appear).

Definition 3 *The language generated by a PC grammar system Γ as above is*

$$L(\Gamma) = \{x \in T_1^* \mid (S_1, S_2, \ldots, S_n) \stackrel{*}{\Longrightarrow} (x, \alpha_2, \ldots, \alpha_n), \alpha_i \in V_\Gamma^*, 2 \leq i \leq n\}.$$

Observe that due to the definition of the language generated by a PC grammar system of type s we have that the terminal set of the system is actually the same as the terminal set of the first component of the system. From the definition of the derivation relation it follows that once the sentential form of G_1 has become a terminal string, the derivation cannot continue anymore in the system.

Just as in the case of usual PC grammar systems, one distinguishes several variants.

Definition 4 *If in Definition 2 only grammar G_1 is allowed to introduce query symbols, then we say that Γ is a* centralized *PC grammar system of type s; in contrast, the unrestricted case is called* non-centralized.

A PC grammar system of type s is said to be returning *(to axiom) if, after communicating, each component returns to axiom. A PC grammar system of type s is* non-returning *if in point (ii) of Definition 2, the brackets,*

$$[and\, y_{i_j} = S_{i_j}, 1 \leq j \leq t],$$

are omitted.

A PC grammar system is said to be regular, context-free, λ-free, etc., when the rules of its components are of these types.

For $n \geq 1$ and $X \in \{REG, CF\}$, we shall denote the families of languages generated by *non-returning centralized, non-returning non-centralized, returning centralized*, and *returning non-centralized*, respectively, PC grammar systems of type s, of degree at most n and with components of type X by

$$\mathbf{L}(NCPC_nX, s)\ ;\ \mathbf{L}(NPC_nX, s)\ ;\ \mathbf{L}(CPC_nX, s)\ ;\ \mathbf{L}(PC_nX, s).$$

When an arbitrary number of components is considered, we shall use $*$ instead of n.

If we still require in point (ii) of Definition 2 that those x_{i_j} which are to be communicated must be terminal strings in the grammars whose sentential forms they are, that is $x_{i_j} \in T_{i_j}^*$, then we say that Γ derives in the *terminal* mode. The language generated by Γ in this way is denoted by $L_T(\Gamma)$ and the family corresponding to $\mathbf{L}(Y_nX, s), \mathbf{L}(Y_*X, s)$ as above is denoted by $\mathbf{L}_T(Y_nX, s), \mathbf{L}_T(Y_*X, s)$. Here and in the sequel Y ranges over $\{NCPC, NPC, CPC, PC\}$ or some specified subset of it.

If we replace point (i) in Definition 2 by

$(i')|x_i|_K = 0, 1 \leq i \leq n$ *and, for each $i, 1 \leq i \leq n$, we have either $x_i \Rightarrow y_i$ in grammar G_i, or $x_i = y_i$*, then, just as in the case of usual PC grammar systems, we get a *PC grammar system of type s deriving in an unsynchronized manner.*

Denote by $L_U(\Gamma)$ the language generated by Γ in this way. The family of languages generated by unsynchronized PC grammar systems of type s corresponding to a family $\mathbf{L}(Y_nX, s), \mathbf{L}(Y_*X, s)$ as above is denoted by $\mathbf{L}_U(Y_nX, s), \mathbf{L}_U(Y_*X, s)$.

In order to illustrate the difference between usual PC grammar systems and the ones studied here, let us consider an example.

Example: Let $\Gamma = (K, G_1, G_2)$, be the returning non-centralized PC grammar system with

$$\begin{aligned} G_1 &= (\{S_1, A, B, Y\}, \{X, a, b, c\}, \\ &\quad \{S_1 \to XA, A \to XA, S_1 \to aB, B \to aB, B \to aQ_2, Y \to b\}, S_1) \\ G_2 &= (\{S_2, X, A\}, \{Y, c\}, \{S_2 \to YS_2, S_2 \to YQ_1, X \to c, A \to c\}, S_2) \end{aligned}$$

If we start a derivation by using in G_1 the production $S_1 \to aB$, then we have
either $(S_1, S_2) \overset{*}{\Rightarrow} (a^{k+1}Q_2, Y^{k+1}Z)$, $k \geq 1, Z \in \{Q_1, S_2\}$,
and then the derivation is blocked due to circular query, if Z is Q_1, or due to $S_2 \notin N_1 \cup T_1$, when attempting to satisfy the query in G_1, if Z is S_2,
or $(S_1, S_2) \overset{*}{\Rightarrow} (a^{k+1}B, Y^{k+1}Q_1)$, $k \geq 0$,
and then the derivation is blocked because $B \notin N_2 \cup T_2$, and hence we cannot satisfy the query in G_2.

The only successful derivation is the one which starts as
$(S_1, S_2) \overset{*}{\Rightarrow} (X^{k+1}A, Y^{k+1}Q_1) \Rightarrow (S_1, Y^{k+1}X^{k+1}A) \overset{*}{\Rightarrow} (a^lQ_2, Y^{k+1}\alpha) \Rightarrow (a^lY^{k+1}\alpha, S_2)$,
where $k \geq 1$ and α is obtained from the string $X^{k+1}A$ by replacing some of the X-s and/or A with c, and $l = |\alpha|_c$, if $\alpha \notin T_2^*$, or $l \geq k+2$, if $\alpha \in T_2^*$ (i.e. $\alpha = c^{k+2}$). If $\alpha \notin T_2$, in order to rewrite it as a terminal string, after a number of steps G_2 must ask for the sentential form of G_1. But this sentential form contains symbols a, which are not in the alphabet of G_2, and hence the query would not be satisfied. This implies that it must be the case $\alpha \in T_2$, and then the configuration above is

$$(a^lY^{k+1}c^{k+2}, S_2), l \geq k+2.$$

Because, as we have made the observation, G_2 cannot accept a sentential form of G_1 containing symbol a, then the derivation has to end as

$$(a^lY^{k+1}c^{k+2}, S_2) \overset{*}{\Rightarrow} (a^lb^{k+1}c^{k+2}, Y^{k+1}Z).$$

Thus we have that

$$L(\Gamma) = \{a^{k+1+s}b^kc^{k+1} \mid s \geq 0, k \geq 1\},$$

a language which is not context-free. Note that what increases the power of the system (for the usual PC grammar systems we have $\mathbf{L}(PC_2REG) \subseteq \mathbf{L}(CF)$, see [13]) was the possibility of a component to rewrite symbols which are considered terminals in another one, as well as the restriction that a communicated string has to be a string over the alphabet of the grammar which required it.

3 Generative Capacity

We first present some general properties for parallel communicating grammar systems of type s, which are true also in case of usual parallel communicating grammar systems.

Lemma 1 *For any $Y \in \{PC, CPC, NPC, NCPC\}$ and for any class X of grammars, we have*

(i) $\mathbf{L}(Y_1X, s) = \mathbf{L}(X)$;
$\mathbf{L}_U(Y_1X, s) = \mathbf{L}(X)$;
$\mathbf{L}_T(Y_1X, s) = \mathbf{L}(X)$;

(ii) $\mathbf{L}(Y_nX, s) \subseteq \mathbf{L}(Y_{n+1}X, s), n \geq 1$,
$\mathbf{L}_U(Y_nX, s) \subseteq \mathbf{L}_U(Y_{n+1}X, s), n \geq 1$,
$\mathbf{L}_T(Y_nX, s) \subseteq \mathbf{L}_T(Y_{n+1}X, s), n \geq 1$,

(iii) $\mathbf{L}(CPC_nX, s) \subseteq \mathbf{L}(PC_nX, s)$; $\mathbf{L}(NCPC_nX, s) \subseteq \mathbf{L}(NPC_nX, s), n \geq 1$;
$\mathbf{L}_U(CPC_nX, s) \subseteq \mathbf{L}_U(PC_nX, s)$; $\mathbf{L}_U(NCPC_nX, s) \subseteq \mathbf{L}_U(NPC_nX, s)$, $n \geq 1$;
$\mathbf{L}_T(CPC_nX, s) \subseteq \mathbf{L}_T(PC_nX, s)$; $\mathbf{L}_T(NCPC_nX, s) \subseteq \mathbf{L}_T(NPC_nX, s)$, $n \geq 1$;

(iv) $\mathbf{L}(CPC_*X, s) \subseteq \mathbf{L}(PC_*X, s)$; $\mathbf{L}(NCPC_*X, s) \subseteq \mathbf{L}(NPC_*X, s)$;
$\mathbf{L}_U(CPC_*X, s) \subseteq \mathbf{L}_U(PC_*X, s)$; $\mathbf{L}_U(NCPC_*X, s) \subseteq \mathbf{L}_U(NPC_*X, s)$;
$\mathbf{L}_T(CPC_*X, s) \subseteq \mathbf{L}_T(PC_*X, s)$; $\mathbf{L}_T(NCPC_*X, s) \subseteq \mathbf{L}_T(NPC_*X, s)$.

Proof: Directly from definitions. □

Each usual PC grammar system can be considered a PC grammar system of type s (we simply skip N, T when writing $\Gamma = (N, K, T, G_1, \ldots, G_n)$), hence we also have

Lemma 2 *For any $Y \in \{PC, CPC, NPC, NCPC\}$ and for any class X of grammars,*

(i) $\mathbf{L}(Y_nX) \subseteq \mathbf{L}(Y_nX, s), n \geq 1$;

(ii) $\mathbf{L}_U(Y_nX) \subseteq \mathbf{L}_U(Y_nX, s), n \geq 1$;

(iii) $\mathbf{L}_T(Y_nX) \subseteq \mathbf{L}_T(Y_nX, s), n \geq 1$;

Just as in the case of usual PC grammar systems, we have relations between the families of generated languages, when considering various modes of derivation.

Lemma 3 *(i)* $\mathbf{L}_U(Y_nX, s) \subseteq \mathbf{L}(Y_nX, s)$, *for any class X of grammars allowing chain rules (that is rules of the form $A \rightarrow B$) and for any $Y \in \{CPC, PC, NCPC, NPC\}$*;

(ii) $\mathbf{L}_T(Y_nX, s) \subseteq \mathbf{L}(Y_nX, s)$, *for any class X of grammars and for any $Y \in \{CPC, NCPC\}$*.

Proof: The proofs are entirely the same as for usual PC grammar systems, [3]. □

We next survey the properties known so far about regular PC grammar systems with separated alphabets. For the proofs we refer to [9].

Proposition 1 *(i) The family* $\mathbf{L}(PC_3REG, s)$ *contains one-letter non-regular languages.*

(ii) The families $\mathbf{L}(NCPC_2REG, s), \mathbf{L}(NPC_2REG, s)$ *contain one-letter non-regular languages.*

(iii) The families $\mathbf{L}_U(NCPC_2REG, s), \mathbf{L}_U(NPC_2REG, s)$ *contain non-semi-linear languages.*

(iv) The family $\mathbf{L}_T(CPC_2REG, s)$ *contains non-semi-linear languages.*

Corollary 1 $\mathbf{L}(CPC_nREG) \subset \mathbf{L}(CPC_nREG, s)$, *strict inclusion, for any* $n \geq 2$.

So just as in the case of cooperating distributed grammar systems, by considering distinct terminal sets for the grammar components of the system, also in the case of PC grammar systems, the generative power is increased (at least in the regular centralized returning case).

But even if centralized returning PC grammar systems with regular components of type s are able to generate non-finite index matrix languages, we still can find an upper-bound for the languages generated by them among the regulated rewriting tools with context-free rules. More exactly, we have

Theorem 1 *(i)* $\mathbf{L}(CPC_*REG, s) \subseteq \mathbf{L}(MAT_{ac})$.

(ii) $\mathbf{L}_U(CPC_*REG, s) \subseteq \mathbf{L}(MAT_{ac})$.

(iii) $\mathbf{L}_T(CPC_*REG, s) \subseteq \mathbf{L}(MAT_{ac})$.

One can observe that although the preceding theorem is for the regular PC grammar systems, it is true as well for the right linear case, and the proof is entirely the same.

As we shall prove in the following, for unsynchronized derivation we can actually find a more specific relation. First, we have the theorem

Theorem 2 $\mathbf{L}_U(CPC_2REG, s) - \mathbf{L}(REG) \neq \emptyset$.

Proof: Consider the following PC grammar system of type s with regular components

$$\Gamma = (K, G_1, G_2),$$

where

$$\begin{aligned} G_1 &= (\{S_1, A, B\}, \{a, b\}, \{S_1 \to aQ_2, A \to aQ_2, B \to b, A \to a\}, S_1), \\ G_2 &= (\{S_2, B\}, \{A\}, \{S_2 \to AB\}, S_2), \end{aligned}$$

and consider the derivation mode to be the unsynchronized one.

Then a terminal derivation has to proceed as follows:

$$\begin{aligned}(S_1, S_2) &\overset{*}{\Rightarrow} (aQ_2, AB) \Rightarrow (aAB, S_2) \overset{*}{\Rightarrow} (a^2Q_2X, AB) \Rightarrow (a^2ABX, S_2)\\ &\overset{*}{\Rightarrow} \dots \overset{*}{\Rightarrow} (a^kQ_2\alpha, AB) \Rightarrow (a^kAB\alpha, S_2) \overset{*}{\Rightarrow} (a^{k+1}b^k, \beta)\end{aligned}$$

where $X \in \{b, B\}$, $\alpha \in \{b, B\}^*, |\alpha| = k - 1$ and $\beta \in \{S_2, AB\}$.

We then have

$$L(\Gamma) = \{a^{k+1}b^k \mid k \geq 1\},$$

which is not a regular language. □

As a corollary, we obtain that also in the unsynchronized derivation, centralized returning PC grammar systems are more powerful when considering distinct sets of terminal and non-terminal symbols then in the case when we do not.

Corollary 2 $\mathbf{L}_U(CPC_nREG) \subset \mathbf{L}_U(CPC_nREG, s)$, *strict inclusion, for any* $n \geq 2$.

Proof: The inclusion is by Lemma 2, and the strictness of it follows from the above theorem and from $\mathbf{L}_U(CPC_*REG) = \mathbf{L}(REG)$, which is known from [3]. □

Our intention in the following is to present other properties concerning PC grammar systems of type s deriving in the unsynchronized mode. We need to recall the following definition.

Definition 5 *Let* $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$ *be a usual parallel communicating grammar system. We say that* Γ *is* with multiple queries *if there is a component of* Γ *with a production* $A \rightarrow \alpha Q_i \beta Q_i \gamma, \alpha, \beta, \gamma \in (N \cup K \cup T)^*, i \in \{1, \dots, n\}$. *Otherwise, we say that* Γ *is* without multiple queries.

The class of such grammar systems is denoted by WY_nX, for $Y \in \{PC, CPC, NPC, NCPC\}$, $n \geq 1$, X a class of grammars.

Theorem 3 *For any* $Y \in \{CPC, PC, NCPC, NPC\}$ *and for any* $n \geq 1$,

(i) $\mathbf{L}_U(Y_nREG, s) \subseteq \mathbf{L}_U(WY_nCF)$;

(ii) $\mathbf{L}_U(Y_nCF, s) = \mathbf{L}_U(Y_nCF)$.

(iii) $\mathbf{L}_U(WY_nCF, s) = \mathbf{L}_U(WY_nCF)$.

Proof: To prove point (i), take a PC grammar system of type s with regular components

$$\Gamma = (K, G_1, G_2, \dots, G_n),$$

with $G_i = (N_i, T_i, P_i, S_i)$, for any $i, 1 \leq i \leq n$.

Denote

$$V = \bigcup_{i=1}^{n}(N_i \cup T_i).$$

For each symbol $\alpha \in V$, consider a new symbol α', denote by V' their set and define the substitution h as

$$\begin{aligned} h(\alpha) &= \{\alpha, \alpha'\}, \text{ for } \alpha \in T_1, \\ h(\alpha) &= \{\alpha'\}, \text{ for } \alpha \in V - T_1, \\ h(Q_i) &= \{Q_i\}, \text{ for } 1 \leq i \leq n. \end{aligned}$$

Construct the PC grammar system

$$\Gamma' = (V', K, T_1, (P_1', S_1'), \ldots, (P_n', S_n')),$$

where

$$P_i' = \{A' \to y \mid A \to x \in P_i, y \in h(x)\}, \text{ for any } 1 \leq i \leq n.$$

Note that even if we have started from a PC grammar system Γ with regular components, because this is of type s, the resulted system, Γ', is with context-free components and not with regular. This happens because we can have in a component $G_i, i \geq 1$ a production $A \to BC$, where B is a terminal symbol with respect to G_i but is not a terminal symbol with respect a grammar to which it will communicate a string containing that B.

Moreover, note that if Γ is returning, centralized, non-returning or non-centralized, then Γ' is of the same type.

Because in any production of any grammar at most one query symbol can appear, we have that Γ' is without multiple queries.

One can see that $L_U(\Gamma) = L_U(\Gamma')$, and thus point (i) follows.

For point (ii), we have $\mathbf{L}_U(Y_nCF) \subseteq \mathbf{L}_U(Y_nCF, s)$ by Lemma 2. To prove the reverse inclusion, we only need to observe that the same construction that we have considered for the proof of point (i) transforms a context-free PC grammar system of type s into a corresponding usual context-free PC grammar system.

Point (iii) is a consequence of the relation in point (ii), by the observation that the construction we have considered does not introduce multiple queries. □

Corollary 3 $\mathbf{L}_U(WCPC_*CF, s) = \mathbf{L}(CF)$.

Proof: It is simply a consequence of the preceding theorem, point (iii), by Theorem 1 of [8], which states that $\mathbf{L}_U(WCPC_*CF) = \mathbf{L}(CF)$. □

Now we can improve the relation obtained in Theorem 1, for the case of unsynchronized derivation. That is, we have

Theorem 4 $\mathbf{L}(REG) \subset \mathbf{L}_U(CPC_*REG, s) \subseteq \mathbf{L}(CF)$.

Proof: The first inclusion is from Lemma 1 and from Theorem 2. The second inclusion is a consequence of the preceding theorem, point (i), by Theorem 1 of [8]. □

Note that once again parallel communicating grammar systems of type s are more powerful, but still not "too powerful" (from the generative capacity point of view) than usual PC grammar systems, in case of regular components, because we have $\mathbf{L}_U(CPC_*REG) = \mathbf{L}(REG)$. But in case of context-free components, for unsynchronized derivation, systems of type s are only as powerful as usual systems are.

It is known, [8], that matrix languages can be generated by usual returning non-centralized PC grammar systems with context-free components. When separated terminal and nonterminal alphabets are considered for the components of the system, one can simulate matrix grammars with appearance checking.

Theorem 5 $\mathbf{L}(MAT_{ac}) \subseteq \mathbf{L}(PC_*CF, s)$.

Proof: The proof bears resemblance with the corresponding one in [8]. Let

$$G = (N, T, S, M, F)$$

be a context-free matrix grammar with the appearance checking set F. It is known ([4]) that for each matrix grammar there is an equivalent matrix grammar, of the same type, in the 2-normal form, that is with

$$N = \{S\} \cup N_1 \cup N_2, \quad N_1 \cap N_2 = \emptyset, \quad S \notin N_1 \cup N_2,$$

and each matrix of M has one of the following forms:

$$\begin{array}{ll}
(i) & (S \to AX),\ A \in N_1,\ X \in N_2 \\
(ii) & (A \to \alpha, X \to Y),\ A \in N_1,\ \alpha \in (N_1 \cup T)^+, X, Y \in N_2, \\
(iii) & (A \to \alpha, X \to a),\ A \in N_1,\ \alpha \in (N_1 \cup T)^+, X \in N_2, a \in T, \\
(iv) & (S \to x), x \in T^*.
\end{array}$$

Moreover, the productions of F are only of the form $A \to \alpha$, with $A \in N_1, \alpha \in (N \cup T)^*$.

Let $P_1(M)$ be the set of matrices of type *(i)*, let $P_2(M)$ be the set of matrices of types *(ii)*, *(iii)* and let r be the cardinality of $P_2(M)$. A matrix of $P_2(M)$ will be denoted in the following by

$$m_k : \ (A_k \to \alpha_k, B_k \to C_k)\ ,\ 1 \leq k \leq r.$$

Denote

$$N' = N \cup \{S', W, V, Z, L_1, L_2, L_3\} \cup \{S_s, S_{a1}, S_{a2}\} \cup \{S_{1k}, S_{2k} \mid k = 1, \ldots, n\}$$

$(S', W, V, Z, L_1, L_2, L_3$ are new symbols). We construct the PC grammar system

$$\Gamma = (K, G_s, G_{11}, G_{21}, G_{12}, G_{22}, \ldots, G_{1r}, G_{2r}, G_{a1}, G_{a2})$$

as follows:

$$\begin{aligned}
P_s &= \{S_s \to x \mid (S_s \to x) \in M, x \in T^*\} \cup \\
&\cup \{S_s \to S', S' \to Q_{2k} \mid k = 1, \ldots, r\} \cup \\
&\cup \{S_s \to AB \mid (S \to AB) \in P_1(M)\} \cup \\
&\cup \{X \to X \mid X \in N_2\}, \\
P_{1k} &= \{S_{1k} \to W, W \to Q_s\} \cup \\
&\cup \{A_k \to \alpha_k \mid m_k : (A_k \to \alpha_k, B_k \to C_k)\} \cup \\
&\cup \{X \to Z \mid X \in N_1 \cup N_2\} \cup \\
&\cup \{V \to Q_{1k}\}, \ \text{ for each } k = 1, 2, \ldots, r, \\
P_{2k} &= \{S_{2k} \to V, V \to Q_{1k}\} \cup \\
&\cup \{B_k \to C_k \mid m_k : (A_k \to \alpha_k, B_k \to C_k)\} \cup \\
&\cup \{X \to Z \mid X \in N_1 \cup N_2\}, \ \text{ for each } k = 1, 2, \ldots, r, \\
P_{a1} &= \{S_{a1} \to Q_{21}Q_{22}...Q_{2r}\} \cup \{V \to V\}, \\
P_{a2} &= \{S_{a2} \to L_1, \, L_1 \to L_2, \, L_2 \to L_3, \, L_3 \to L_1Q_{21}Q_{22}...Q_{2r}\}.
\end{aligned}$$

The terminal sets of the components grammars of Γ are defined as

$$T_s = T_{a1} = T_{a2} = T_{2k} = T, \text{ for any } k = 1, 2, \ldots, r,$$

while for any $k = 1, 2, \ldots, r$,

$$T_{1k} = \begin{cases} T \cup N_2 \cup N_1 - & \{A_k \mid m_k : (A_k \to \alpha_k, B_k \to C_k)\}, \\ & \quad \text{if this occurrence of} \\ & \quad \text{the production } A_k \to \alpha_k \in F \\ T, & \quad \text{otherwise.} \end{cases}$$

As for the nonterminal sets of the components of Γ, they are defined as

$$\begin{aligned}
N_s &= N' - T_s, \\
N_{1k} &= N' - T_{1k}, \text{ for any } k = 1, \ldots, r, \\
N_{2k} &= N' - T_{2k}, \text{ for any } k = 1, \ldots, r, \\
N_{a1} &= N' - T_{a1}, \\
N_{a2} &= N' - T_{a2}.
\end{aligned}$$

One can verify that $L(\Gamma) = L(G)$, and therefore the theorem follows. □

As an immediate corollary of the above theorem, characterization of recursively enumerable languages results when λ-productions are allowed in the system.

Corollary 4 $\mathbf{L}(PC_*CF^\lambda, s) = \mathbf{L}(RE)$, *where the notation* PC_*CF^λ *stands for returning non-centralized PC grammar systems with* λ*-rules.*

The similarity between PC grammar systems with separated alphabets and usual PC grammar systems let us think that any proof of a relation between two classes of usual PC grammar systems can be adapted as to result in a relation between the corresponding classes of PC systems of type s. More precisely, we conjecture that if $\mathbf{L}(Y_nCF) \subseteq \mathbf{L}(Y'_mCF)$ for two classes Y, Y' of PC grammar systems, $m, n, \geq 1$, then $\mathbf{L}(Y_nCF, s) \subseteq \mathbf{L}(Y'_mCF, s)$.

In particular, by [6], [12],

$$\mathbf{L}(NPC_*CF, s) \subseteq \mathbf{L}(PC_*CF, s)$$

would result.

References

[1] A. Atanasiu, V. Mitrana, The Modular Grammars, *Internat. J. Comp. Math.* 30 (1989), 101-122

[2] E. Csuhaj-Varjú, J. Dassow, On Cooperating Distributed Grammar Systems, *J. Inf. Processing and Cybern. EIK*, 26 (1990), 49-63

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

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

[5] J. Dassow, Gh. Păun, G. Rozenberg, Grammar Systems, in G. Rozenberg, A. Salomaa (eds.), *The Handbook of Formal Languages*, Springer-Verlag, to appear

[6] S. Dumitrescu, Non-returning parallel communicating grammar systems can be simulated by returning systems, *Theoretical Computer Science*, 165 (1996), 463-474.

[7] R. Meersman, G. Rozenberg, Cooperating Grammar Systems, in *Proc. MFCS'78 Symp. LNCS 64*, Springer-Verlag, Berlin, 1978, 364-374

[8] V. Mihalache, Matrix Grammars versus Parallel Communicating Grammar Systems, in Gh. Păun (ed.), *Mathematical Aspects of Natural and Formal Languages*, World Scientific, 1994, 293-318

[9] V. Mihalache, Terminal versus Non-terminal Symbols in Parallel Communicating Grammar Systems, *Revue Roumaine de Mathématiques Pures et Appliquées*, to appear

[10] Gh. Păun, L. Sântean, Parallel Communicating Grammar Systems: the Regular Case, *Ann. Univ. Buc., Ser. Matem.-Inform.*, 38 (1989), 55-63

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

[12] Gy. Vaszil, On Simulating Non-Returning PC Grammar Systems with Returning Systems, *submitted for publication*

[13] S. Vicolov, Non-centralized Parallel Grammar Systems, *Stud. Cerc. Mat.*, 44 (1992), 455-462

On Regular Characterizations of Languages by Grammar Systems*

Lucian ILIE [†] Arto SALOMAA[‡]

Abstract

We show that grammar systems with communication by command and with extremely simple rewriting rules are able to generate all recursively enumerable languages. The result settles several open problems in the area of grammar systems.

1 Introduction

The purpose of this paper is to investigate the power of cooperation in rewriting systems. This is done using the abstract model of a grammar system, [3]. We show that grammar systems with the most simple components, all rewriting rules being letter-to-letter, possess the power of generating all recursively enumerable languages. This result and its corollaries settle several open problems in the area of grammar systems. We now describe the contents of the paper in non-technical terms.

A parallel communicating grammar system, as introduced in [12], consists of several grammars which work synchronously, each of them rewriting its own sentential form, the communication being made by *request*: when a component introduces a query symbol (from a special set) for another component, then the latter one sends its current sentential form to the former which rewrites it in place of the query symbol. The language generated by the system is the set of terminal strings generated (using communication or not) by a distinguished component called master. (For results and references see [3].)

Another kind of parallel communicating grammar systems, with communication by *command*, is introduced in [4] with suggestions from the WAVE paradigm for data flow in highly parallel machines ([5], [6], [14]), Boltzmann machine ([7]), the Connection Machine ([8], [15]), and other well-known parallel machines.

*Research supported by the Academy of Finland, Project 11281

†Turku Centre for Computer Science, FIN-20520 Turku, Finland

‡Academy of Finland and Mathematics Department, University of Turku, FIN-20014 Turku, Finland

The communication by command means that when the current sentential form derived in a component coresponds to another component, i.e., belongs to the regular language associated to the respective component or fits the pattern (in the sense of [1], [11]) associated to that component, then the sentential form is sent to the other component. The language generated by the system is also the set of terminal strings generated by a component designed as master. Here we investigate only the case when each component has associated a regular language.

In [4] it is proved that any context-sensitive language can be generated by a grammar system with communication by command with context-free components while in [10] it is shown that the grammar systems with context-sensitive components and the same type of communication can generate only context-sensitive languages. The characterization of the family of context-sensitive languages as the family of languages generated by grammar systems with context-free components and communication by command follows. We shall strengthen this result by showing that the family of context-sensitive languages is exactly the family of languages generated by the grammar systems with *regular* components and communication by command.

We consider also the case when the splitting is allowed in communication, that is, if the current sentential form of a component is a concatenation of strings each belonging to the regular language associated to another component, then the communication can still be performed: each factor of the sentential form can be sent to the respective component, with the restriction that only one factor can be sent to one component.

As already mentioned in [4], this type of communication is natural: following the *logic flow paradigm* proposed in [6] as a basic architecture for parallel symbolic processing, we deal with a symbolic process which develops in a virtually complete graph having processors which are able to handle data, in its nodes. The process starts by injecting data in a node and each node having data can perform a local processing; under well defined conditions, the local data are spread to other nodes by *replication* or by *splitting*.

In this case we prove a characterization of recursively enumerable languages by grammar systems with (non-erasing) regular rules. In fact, the rules have a particularly simple form: a letter (nonterminal) always goes to a letter (terminal or nonterminal).

2 Grammar systems

We shall denote by V^* the set of all finite strings over the alphabet V, the empty string is denoted by λ, and $V^+ = V^* - \{\lambda\}$. The set of regular, context-free, context-sensitive, and recursively enumerable languages will be denoted by REG, CF, CS, and RE, respectively. For further elements of formal language theory we refer to [9] and [13].

A *parallel communicating grammar system with communication by command* of degree $n \geq 1$ is a construct of the form

$$\Gamma = (N, T, (S_1, P_1, R_1), \ldots, (S_n, P_n, R_n)),$$

where N is the *nonterminal* alphabet, T is the *terminal* alphabet, and (S_i, P_i, R_i), $1 \leq i \leq n$, are the *components* of the system: S_i is the *axiom*, P_i is the (finite) *set of rules*, (note that we do not allow λ-rules, that is rules in which the right-hand member is empty), and $R_i \in REG$ is the *selector* language for the component i.

Such a system works as follows:

– start from the initial configuration $(S_1, S_2, \ldots, S_n)$,

– at each step, the configuration of the system will be described by an n-tuple $(x_1, x_2, \ldots, x_n) \in ((N \cup T)^*)^n$,

– the configuration of the system can be modified either by *rewriting* steps or by *communication* steps,

– rewriting steps are performed componentwise and the derivation must be maximal in each component (that is the component can not rewrite its sentential form any longer),

– communication steps are performed as follows:
(i) communication *without splitting*: when (after maximal derivations) some components $S_{i_1}, S_{i_2}, \ldots, S_{i_k}, 1 \leq i_1 < i_2 < \cdots < i_k \leq n$, have derived the strings $w_1, w_2, \ldots, w_k \in R_i$, for some $1 \leq i \leq n, i \notin \{i_1, i_2, \ldots, i_k\}$ (a component may not communicate with itself) and these are all the components, at that moment, able to communicate their sentential forms to the component i, then the string $w_1 w_2 \ldots w_k$ will *replace* the sentential form of the component i becoming the current sentential form of this component; the components which send their sentential forms will *restart* from the initial symbol,
(ii) communication *with splitting*: similar to the one without splitting, the difference being that if the sentential form of a component is a catenation of strings each of them belonging to the regular set associated to another component, then each factor of the current string can be sent to the respective component with the following restrictions:

1. only one string can be sent to one component,
2. a component cannot send a factor of its current sentential form to itself (also not the entire string),
3. the catenation of the factors of the current string which are sent must be the entire string (nothing is lost).

– if, after a sequence of rewriting/communication steps, the string on the first position in the current configuration is a terminal one, then it belongs to the generated language (so the master is always the first component).

Formally, a *rewriting step* is

$$(x_1, \ldots, x_n) \Longrightarrow (y_1, \ldots, y_n) \text{ iff } x_i \Longrightarrow^* y_i \text{ in } P_i \text{ and there is no } z_i \in (N \cup T)^* \text{ with } y_i \Longrightarrow z_i \text{ in } P_i.$$

In order to define a communication step without splitting, let us denote

$$\delta(x_i, j) = \begin{cases} \lambda, & \text{if } x_i \notin R_j \text{ or } i = j, \\ x_i, & \text{if } x_i \in R_j \text{ and } i \neq j, \end{cases}$$

for $1 \le i, j \le n$,

$$\Delta(i) = \delta(x_1, i)\delta(x_2, i)\dots\delta(x_n, i),$$
$$\delta(i) = \delta(x_i, 1)\delta(x_i, 2)\dots\delta(x_i, n),$$

for $1 \le i \le n$.

A *communication step without splitting* is:

$$(x_1,\dots,x_n) \vdash (y_1,\dots,y_n) \text{ iff } y_i = \begin{cases} \Delta(i), & \text{if } \Delta(i) \ne \lambda, \\ x_i, & \text{if } \Delta(i) = \lambda \text{ and } \delta(i) = \lambda, \\ S_i, & \text{if } \Delta(i) = \lambda \text{ and } \delta(i) \ne \lambda. \end{cases}$$

Because the splitting will not be used very much, we define it rather informally. A *communication step with splitting* is

$$(x_1, x_2, \dots, x_n) \vdash_S (y_1, y_2, \dots, y_n)$$

if and only if there is a set $\emptyset \ne M \subseteq \{1, 2, \dots, n\}$ (M is the set of indices of those components which send their sentential forms) such that

(i) for any $i \in M$ there is a permutation of n elements $\pi_i \in S_n$ and a decomposition $x_i = x_{i,\pi_i(1)}x_{i,\pi_i(2)}\dots x_{i,\pi_i(n)}$ such that $x_{i,i} = \lambda$ and, for any $1 \le k \le n, k \ne \pi_i^{-1}(i)$, $x_{i,\pi_i(k)} \in R_{\pi_i(k)}$ or $x_{i,\pi_i(k)} = \lambda$

(ii) for any $i \in \{1, 2, \dots, n\} - M$ and any $1 \le j \le n, x_{i,j} = \lambda$,

(iii) for any $1 \le j \le n$, if $\lambda \notin R_j$, then

$$y_j = \begin{cases} x_{1,j}x_{2,j}\dots x_{n,j}, & \text{if } x_{1,j}x_{2,j}\dots x_{n,j} \ne \lambda \\ x_j, & \text{if (there is no } i \in M \text{ with } x_{i,j} \in R_j) \text{ and } j \notin M, \\ S_j, & \text{if (there is no } i \in M \text{ with } x_{i,j} \in R_j) \text{ and } j \in M. \end{cases}$$

If $\lambda \in R_j$, then, nondeterministically, the component j can receive λ or can work as in the case when $\lambda \notin R_j$.

Note that the communication without splitting is a particular case of the communication with splitting and also that the empty string can be sent.

The generated language is

$$\begin{aligned} L_c(\Gamma) = \{w \in T^* \mid\ & (S_1,\dots,S_n) \Longrightarrow (x_1^{(1)},\dots,x_n^{(1)}) \vdash_c (y_1^{(1)},\dots,y_n^{(1)}) \Longrightarrow \\ & (x_1^{(2)},\dots,x_n^{(2)}) \vdash_c (y_1^{(2)},\dots,y_n^{(2)}) \Longrightarrow \dots \Longrightarrow (x_1^{(k)},\dots,x_n^{(k)}), \\ & \text{for some } k \ge 1 \text{ such that } w = x_1^{(k)}\}, \end{aligned}$$

where, for $c = \lambda$, we identify $L_\lambda(\Gamma)$ with $L(\Gamma)$ and $\vdash_\lambda$ with $\vdash$ and, for $c = S$, we have $L_S(\Gamma)$ and $\vdash_S$.

We denote by $CCPC_nX$ the family of languages $L(\Gamma)$, generated by grammar systems of degree at most $n, n \ge 1$, with components of type $X \in \{REG, CF, CS\}$, working with communication without splitting, and by $SCCPC_nX$ the family of languages $L_S(\Gamma)$, generated by grammar systems of degree at most $n, n \ge 1$, with components of type X, working with communication with splitting. When the number of components is arbitrary, we write $CCPC_\infty X$ and, respectively, $SCCPC_\infty X$.

3 The characterization results

We begin with the following simple observation. Because in the case when the system has only two components no communication by splitting can be done, we have

Lemma 1 *For any family X, $CCPC_2X = SCCPC_2X$.*

Our first theorem shows that, in the case of communication with splitting, any recursively enumerable language can be generated using a system with *four* regular components. Because the languages associated to the components are regular too, we can say that this is a fully regular characterization of recursively enumerable languages. (Note that we do not allow λ-rules and also not the rewriting of the terminal symbols. The sets of nonterminals and terminals are defined at the level of the system.)

Actually, *three* regular components suffice, as seen in Theorem 2 below. From the point of view of exposition, it is convenient to consider first the weaker version. A further reduction to *two* components is not possible because of Lemma 1 and a result in [4] which shows that in the case of communication without splitting, using two components, only regular languges can be produced.

Theorem 1 *$SCCPC_4REG = RE$.*

Proof. Let L be a recursively enumerable language over the alphabet T. Then, by a slight modification of Theorem 9.9 in [13], there is a context-sensitive language L_1 and two symbols $a_1, a_2 \notin T$, such that:

(i) L_1 consists of words of the form $wa_2a_1^n, n \geq 0, w \in L$, and

(ii) for every $w \in L$, there is a $n \geq 0$ such that $wa_2a_1^n \in L_1$.

The main idea of our proof is: we construct a system (with four regular components) which generates in one component (which is not the master) any string $wa_2a_1^n \in L_1$ and then, by splitting, the string w is communicated to the master and the garbage $a_2a_1^n$ is communicated to another component. (In fact this is the only moment when the splitting is used, the entire derivation, excepting this, being as in a usual grammar system with communication by command.)

So let $G = (N, T \cup \{a_1, a_2\}, S, P)$ be a context-sensitive grammar generating L_1. Suppose that G is in Kuroda normal form, that is, all productions in G are of the form $AB \longrightarrow CD, A \longrightarrow BC$, and $A \longrightarrow a$ where A, B, C, D are nonterminals and a is a terminal symbol. By introducing, whenever needed, productions of the form $A \longrightarrow B$, A, B nonterminals, we may suppose that if a production of the form $AB \longrightarrow CD$ appears in P, then $A \neq B$.

For a reason that will be seen later, we introduce also the production $S \longrightarrow S$. We label all productions in P by natural numbers $r, 1 \leq r \leq card(P)$. (We construct a bijection between P and the set $\{1, 2, \ldots, card(P)\}$, each production being uniquely identified by its associated number.)

Let S_1', S_2', S_3', X, and Y be symbols not in $N \cup T \cup \{a_1, a_2\}$ and let us put

$$N' = \{A' \mid A \in N\} \cup \{X'\}$$

and

$$\begin{aligned} V = \ & \{A_r \mid A \in N, r : AB \longrightarrow CD \in P \text{ or } r : BA \longrightarrow CD \in P\} \cup \\ & \cup \{A_r \mid A \in N, r : A \longrightarrow \alpha \in P\} \cup \\ & \cup \{X_r \mid r : A \longrightarrow BC \in P\} \cup \\ & \cup \{Z_A, W_A \mid A \in N\}. \end{aligned}$$

We consider the system

$$\begin{aligned} \Gamma = \ & (N \cup N' \cup \{S_1', S_2', S_3', X, Y\} \cup V, T \cup \{a_1, a_2\}, \\ & \qquad (S_1', P_1, R_1), (S_2', P_2, R_2), (S_3', P_3, R_3), (S_4, P_4, R_4)) \end{aligned}$$

where $S_4 = S$ and

$$\begin{aligned} P_1 &= \emptyset, \\ R_1 &= T^* \cup \{Y\}, \end{aligned}$$

$$\begin{aligned} P_2 &= \{S_2' \longrightarrow X, S_2' \longrightarrow Y\}, \\ R_2 &= a_2 a_1^*, \end{aligned}$$

$$\begin{aligned} P_3 = \ & \{A' \longrightarrow A \mid A \in N \cup \{X\}\} \cup \\ & \cup \{A_r \longrightarrow a \mid r : A \longrightarrow a \in P\} \cup \\ & \cup \{A_r \longrightarrow B \mid r : A \longrightarrow B \in P\} \cup \\ & \cup \{A_r \longrightarrow C, B_r \longrightarrow D \mid r : AB \longrightarrow CD \in P\} \cup \\ & \cup \{X_r \longrightarrow B, A_r \longrightarrow C \mid r : A \longrightarrow BC \in P\} \cup \\ & \cup \{Z_A \longrightarrow A, W_A \longrightarrow X \mid A \in N\}, \\ R_3 = \ & \{\alpha_1 A_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, r : A \longrightarrow a \in P\} \cup \\ & \cup \{\alpha_1 A_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, r : A \longrightarrow B \in P\} \cup \\ & \cup \{\alpha_1 A_r B_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, r : AB \longrightarrow CD \in P\} \cup \\ & \cup \{\alpha_1 X_r A_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, r : A \longrightarrow BC \in P\} \cup \\ & \cup \{\alpha_1 Z_A W_A \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, A \in N\}, \end{aligned}$$

$$\begin{aligned} P_4 = \ & \{A \longrightarrow A' \mid A \in N \cup \{X\}\} \cup \\ & \cup \{A \longrightarrow A_r \mid r : A \longrightarrow a \in P \text{ or } r : A \longrightarrow B \in P\} \cup \\ & \cup \{X \longrightarrow X_r, A \longrightarrow A_r \mid r : A \longrightarrow BC \in P\} \cup \\ & \cup \{A \longrightarrow A_r, B \longrightarrow B_r \mid r : AB \longrightarrow CD \in P\} \cup \\ & \cup \{X \longrightarrow Z_A, A \longrightarrow W_A \mid A \in N\}, \\ R_4 = \ & (N \cup \{X\} \cup T \cup \{a_1, a_2\})^+. \end{aligned}$$

(Note that $\lambda \notin R_4$, hence the fourth component cannot be restarted by receiving λ in a communication with splitting.)

Let us prove that the construction is correct, that is $L_S(\Gamma) = L$. We shall do this by showing inclusion in both directions.

Claim 1. If $wa_2a_1^n \in L_1$, then the system Γ can reach a configuration which has in the third position the string $wa_2a_1^n$.

Remark. If Claim 1 holds, then $L \subseteq L_S(\Gamma)$. Indeed, for any $w \in L$, there is an $n \geq 0$ such that $wa_2a_1^n \in L_1$. But, by Claim 1, Γ can reach a configuration with $wa_2a_1^n$ as the current sentential form of the third component. In this case, as $w \in T^* \subset R_1$ and $a_2a_1^n \in R_2$, by splitting, w is communicated to the first component and $a_2a_1^n$ to the second one. Consequently, w is a terminal string and it is the current sentential form of the master, hence $w \in L_S(\Gamma)$.

Proof of Claim 1. Let $wa_2a_1^n$ be a string in L_1. It follows that there is a derivation in G generating it. We show that if α and β are two sentential forms of G such that $\alpha \Longrightarrow_G \beta$, then, having α as the current sentential form of the third component of Γ, we can obtain also β as the sentential form of the third component of Γ.

Because the case when $\alpha = S$ requires some additional explanations, we shall investigate it separately. (In fact, in the first case it will be shown that the derivation in Γ can simulate any beginning of a derivation in G, that is, we can obtain any β with $S \Longrightarrow_G \beta$ as the sentential form of the third component of Γ.)

Case 1 $\alpha = S$. Depending on the form of β, we have three cases:

(i) $\beta = a \in T \cup \{a_1, a_2\}$ and $r : S \longrightarrow a \in P$. (As an observation, because $L_1 \subseteq La_2a_1^*$, a cannot be a_1.) We simulate this in Γ by

$$(S_1', S_2', S_3', S) \Longrightarrow_\Gamma (S_1', Y, S_3', S_r) \vdash_\Gamma (Y, S_2', S_r, S) \Longrightarrow_\Gamma (Y, Y, a, S_r).$$

(ii) $\beta = A \in N$ and $r : S \longrightarrow A$. In Γ we have

$$(S_1', S_2', S_3', S) \Longrightarrow_\Gamma (S_1', Y, S_3', S_r) \vdash_\Gamma (Y, S_2', S_r, S) \Longrightarrow_\Gamma (Y, Y, A, S_r).$$

(iii) $\beta = AB, A, B \in N$ and $r : S \longrightarrow AB$. Supposing that $p : S \longrightarrow S \in P$, we perform in Γ

$$\begin{aligned}(S_1', S_2', S_3', S) &\Longrightarrow_\Gamma (S_1', Y, S_3', S_p) \vdash_\Gamma (Y, S_2', S_p, S) \Longrightarrow_\Gamma (Y, X, S, S_p) \vdash_\Gamma \\ \vdash_\Gamma (Y, S_2', S_p, XS) &\Longrightarrow_\Gamma (Y, Y, S, X_rS_r) \vdash_\Gamma (Y, S_2, X_rS_r, S) \Longrightarrow_\Gamma (Y, Y, AB, S_p).\end{aligned}$$

In words, we have added the rule $S \longrightarrow S$ to P in order to be able to perform this type of rule ($S \longrightarrow AB$) with S on the left-hand side. If the rule $S \longrightarrow S$ is not provided, then we are forced to apply in the fourth component another rule instead of $S \longrightarrow S_p$ ($p : S \longrightarrow S$) and, as at this moment we did not yet get an X in the sentential form of the fourth component, after sending the current string of the last component to the third one, only rules of the form $S \longrightarrow a, a \in T$ or $S \longrightarrow A, A \in N$, can be applied. Consequently, we would not be able to apply a rule of the form $S \longrightarrow AB, A, B \in N$, in this case.

Case 2 $\alpha \in (N \cup T \cup \{a_1, a_2\})^+ - \{S\}$. Depending on the form of the applied production, we have four cases here.

(i) $\alpha = \alpha_1 A \alpha_2, A \in N, \beta = \alpha_1 a \alpha_2, a \in T \cup \{a_1, a_2\}$, $r : A \longrightarrow a \in P$. We simulate this in Γ as follows. If the current sentential form of a component is not important at some moment, we shall replace it by $-$.

$$\begin{gathered}(-, Y, \alpha_1 A \alpha_2, -) \vdash_\Gamma (Y, S_2', -, \alpha_1 A \alpha_2) \Longrightarrow_\Gamma (Y, Y, -, \alpha_1' A_r \alpha_2') \vdash_\Gamma \\ \vdash_\Gamma (Y, S_2', \alpha_1' A_r \alpha_2', -) \Longrightarrow_\Gamma (Y, Y, \alpha_1 a \alpha_2, -)\end{gathered}$$

(where for $\alpha \in (N \cup \{X\} \cup T \cup \{a_1, a_2\})^*$ we have denoted by α' the string $h(\alpha)$ where $h : (N \cup \{X\} \cup T \cup \{a_1 a_2\})^* \longrightarrow (N' \cup T \cup \{a_1 a_2\})^*$ is the homomorphism defined by $h(A) = A'$, for any $A \in N \cup \{X\}$, $h(a) = a$, for any $a \in T \cup \{a_1, a_2\}$).

(ii) $\alpha = \alpha_1 A \alpha_2, \beta = \alpha_1 B \alpha_2, A, B \in N, r : A \longrightarrow B \in P$. This is handled as Case 2 (i).

(iii) $\alpha = \alpha_1 AB \alpha_2, \beta = \alpha_1 CD \alpha_2, A, B, C, D \in N, r : AB \longrightarrow CD \in P$. This rule is simulated in Γ by

$$\begin{gathered}(-, Y, \alpha_1 AB \alpha_2, -) \vdash_\Gamma (Y, S_2', -, \alpha_1 AB \alpha_2) \Longrightarrow_\Gamma (Y, Y, -, \alpha_1' A_r B_r \alpha_2') \vdash_\Gamma \\ \vdash_\Gamma (Y, S_2', \alpha_1' A_r B_r \alpha_2', -) \Longrightarrow_\Gamma (Y, Y, \alpha_1 CD \alpha_2, -).\end{gathered}$$

(iv) $\alpha = \alpha_1 A \alpha_2, \beta = \alpha_1 BC \alpha_2, A, B, C \in N, r : A \longrightarrow BC \in P$. Because the string generated by P_2 (X or Y) is communicated by the second component (to the fourth component or to the first one, respectively) at each communication step, the derivation in the second component is restarted after each communication performed in the system. Therefore, after each communication step, the second component is able to produce a new X, if needed. (It can also produce a Y if an X is not needed.) As $\alpha \neq S$, there exists a sentential form γ of G such that $\gamma \Longrightarrow_G \alpha$ and we can suppose that (*) when the current sentential form of the third component of Γ is α, then the current string in the second component is X. (We can suppose, for instance, that the second component has introduced an X when γ was obtained in the third one. It is essential here that $\alpha \neq S$; we have seen in Case 1 (iii) how the alternative $\alpha = S$ is handled.)

We may also suppose that the string α contains only nonterminal symbols. (We may obviously suppose that, in a derivation in G, we can apply first only productions of the form $A \longrightarrow B$ or $A \longrightarrow BC$ or $AB \longrightarrow CD$, $A, B, C, D \in N$, and, after that, only productions of the form $A \longrightarrow a$, $A \in N, a \in T \cup \{a_1, a_2\}$.) Consequently, we can put $\alpha_1 = A_1 A_2 \ldots A_k, A_1, A_2, \ldots, A_k \in N, k \geq 0$ ($k = 0$ implies $\alpha_1 = \lambda$) and we can write (using (*))

$$\begin{gathered}(-, X, \alpha, -) = (-, X, A_1 A_2 \ldots A_k A \alpha_2, -) \vdash_\Gamma (-, S_2', -, X A_1 A_2 \ldots A_k A \alpha_2) \Longrightarrow_\Gamma \\ \Longrightarrow_\Gamma (-, Y, -, Z_{A_1} W_{A_1} A_2' \ldots A_k' A' \alpha_2') \vdash_\Gamma (Y, S_2', Z_{A_1} W_{A_1} A_2' \ldots A_k' A' \alpha_2', -) \Longrightarrow_\Gamma \\ \Longrightarrow_\Gamma (Y, Y, A_1 X A_2 \ldots A_k A \alpha_2, -) \vdash_\Gamma (Y, S_2', -, A_1 X A_2 \ldots A_k A \alpha_2) \Longrightarrow_\Gamma \\ \Longrightarrow_\Gamma \cdots\cdots\cdots\cdots\cdots\cdots \Longrightarrow_\Gamma \\ \Longrightarrow_\Gamma (Y, Y, A_1 \ldots A_{k-1} X A_k A \alpha_2, -) \vdash_\Gamma (Y, S_2', -, A_1 \ldots A_{k-1} X A_k A \alpha_2) \Longrightarrow_\Gamma \\ \Longrightarrow_\Gamma (Y, Y, -, A_1' \ldots A_{k-1}' Z_{A_k} W_{A_k} A' \alpha_2') \vdash_\Gamma (Y, S_2', A_1' \ldots A_{k-1}' Z_{A_k} W_{A_k} A' \alpha_2', -) \Longrightarrow_\Gamma \\ \Longrightarrow_\Gamma (Y, Y, A_1 \ldots A_{k-1} A_k X A \alpha_2, -) \vdash_\Gamma (Y, S_2', -, A_1 \ldots A_{k-1} A_k X A \alpha_2) \Longrightarrow_\Gamma \\ \Longrightarrow_\Gamma (Y, Y, -, A_1' \ldots A_k' X_r A_r \alpha_2') \vdash_\Gamma (Y, S_2', A_1' \ldots A_k' X_r A_r \alpha_2', -) \Longrightarrow_\Gamma \\ \Longrightarrow_\Gamma (Y, Y, A_1 A_2 \ldots A_k BC \alpha_2, -) = (Y, Y, \beta, -).\end{gathered} \tag{1}$$

Thus Claim 1 is proved.

Claim 2. If $w \in T^*$ was communicated to the master component in Γ (by the third one – this is the only possibility), then, at the moment of communication, the current sentential form of the third component was $wa_2a_1^n \in L_1$ and , by splitting, w was communicated to the master and $a_2a_1^n$ to the second component.

Remark. Obviously, Claim 2 implies $L_S(\Gamma) \subseteq L$.

Proof of Claim 2. Observe that the only possible communications among the components of Γ are represented by the following graph. (An arrow from i to j is present if and only if it is possible that the component i communicates, at some moment, its sentential form to the component j; some arrows are labeled by the regular sets which control the communication.)

$$\begin{tikzcd}
& 1 & \\
& 3 \arrow[u, "T^*"'] \arrow[dl, "a_2a_1^*"] \arrow[dr, shift left] & \\
2 \arrow[uur, "Y"] \arrow[rr, "X"] & & 4 \arrow[ul, shift left]
\end{tikzcd}$$

We make the following further observations:

1. The second component can communicate to the first one only the string Y which is not terminal. (This communication takes place in order to restart the second component, making it able to produce an X at any moment.)

2. The second component can communicate to the fourth one only the string X.

3. The communication from the third component to the first and the second ones can be done only in the same time by splitting and only when the sentential form of the third component is of the form $wa_2a_1^n, n \geq 0$, w being communicated to the master and $a_2a_1^n$ to the second component. (Note that the string communicated to the first component can be empty.)

4. Always, after a maximal derivation in the third component, its current sentential form can be communicated to the fourth component.

5. Due to the form of R_3, if the current sentential form of the fourth component is communicated to the third one (and only to the third one) then a production in P will be correctly applied at the next step in the third component. Indeed, everything should be clear in what concerns the productions of the form $A \longrightarrow a$ or $A \longrightarrow B, A, B \in N, a \in T \cup \{a_1, a_2\}$. A discussion is needed only for the other two types of productions.

(i) For $r : AB \longrightarrow CD \in P; A, B, C, D \in N$. In order to apply this production, in the fourth component one performs $A \longrightarrow A_r$ and $B \longrightarrow B_r$ (providing, of course, that these productions can be applied). After that, the current sentential form is communicated to the third component if and only if the occurrences of A_r and B_r appear consecutively and in this order (i.e., A_rB_r). In the third component, using the rules $A_r \longrightarrow C$ and $B_r \longrightarrow D$, the string CD is obtained. Because we have supposed that $A \neq B$, there is no danger to apply the production $AB \longrightarrow DC$ instead of $AB \longrightarrow CD$.

(ii) For $r : A \longrightarrow BC \in P; A, B, C \in N$. As it was already seen, for applying a production of this type an occurrence of an X in needed. Without it, the fourth component applies $A \longrightarrow A_r$ but the current sentential form cannot be communicated to the third component because an occurrence of the string A_rX_r is asked by R_3.

Because an occurrence of the symbol X can be communicated by the second component to the fourth one at each communication step (we can apply in P_2 only $S_2' \longrightarrow X$) there is only the danger that too many X's are contained in the sentential form communicated between the last two components. But if the number of X's communicated by the second component to the last one is strictly greater than the number of productions of the form $A \longrightarrow BC$ applied, then the string can be never communicated to the master (no string in R_1 contains X). Hence nothing will be produced in this case.

From the observations above, it should be clear that no parasitic string can be obtained in Γ. Consequently, Claim 2 is proved so we have $L_S(\Gamma) = L$.

□

As said before, the number of the components can be reduced to three.

Theorem 2 *$SCCPC_3REG = RE$.*

Proof. We use the same notations as in Theorem 1 with the only difference that we consider here one new nonterminal symbol, Z, which is added to the set of nonterminals of the system Γ (with three components)

$$\Gamma = (N \cup N' \cup \{S_1', S_2', X, Y, Z\} \cup V, T \cup \{a_1, a_2\}, (S_1', P_1, R_1), (S_2', P_2, R_2), (S_3, P_3, R_3))$$

where $S_3 = S$ and, supposing that $p : S \longrightarrow S \in P$,

$$\begin{aligned} P_1 &= \{Y \longrightarrow X, Y \longrightarrow Z\}, \\ R_1 &= T^* \cup \{Y\}, \end{aligned}$$

$$\begin{aligned} P_2 = \; & \{S_p \longrightarrow Y\} \cup \\ & \cup \{A' \longrightarrow A \mid A \in N \cup \{X\}\} \cup \\ & \cup \{A_r \longrightarrow a \mid r : A \longrightarrow a \in P\} \cup \\ & \cup \{A_r \longrightarrow B \mid r : A \longrightarrow B \in P\} \cup \\ & \cup \{A_r \longrightarrow C, B_r \longrightarrow D \mid r : AB \longrightarrow CD \in P\} \cup \\ & \cup \{X_r \longrightarrow B, A_r \longrightarrow C \mid r : A \longrightarrow BC \in P\} \cup \\ & \cup \{Z_A \longrightarrow A, W_A \longrightarrow X \mid A \in N\}, \end{aligned}$$

$$\begin{aligned}
R_2 &= \{\alpha_1 A_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, r : A \longrightarrow a \in P\} \cup \\
&\quad \cup \{\alpha_1 A_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, r : A \longrightarrow B \in P\} \cup \\
&\quad \cup \{\alpha_1 A_r B_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, r : AB \longrightarrow CD \in P\} \cup \\
&\quad \cup \{\alpha_1 X_r A_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, r : A \longrightarrow BC \in P\} \cup \\
&\quad \cup \{\alpha_1 Z_A W_A \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T \cup \{a_1, a_2\})^*, A \in N\},
\end{aligned}$$

$$\begin{aligned}
P_3 &= \cup \{A \longrightarrow A' \mid A \in N \cup \{X\}\} \cup \\
&\quad \cup \{A \longrightarrow A_r \mid r : A \longrightarrow a \in P \text{ or } r : A \longrightarrow B \in P\} \cup \\
&\quad \cup \{X \longrightarrow X_r, A \longrightarrow A_r \mid r : A \longrightarrow BC \in P\} \cup \\
&\quad \cup \{A \longrightarrow A_r, B \longrightarrow B_r \mid r : AB \longrightarrow CD \in P\} \cup \\
&\quad \cup \{X \longrightarrow Z_A, A \longrightarrow W_A \mid A \in N\}, \\
R_3 &= (N \cup \{X\} \cup T \cup \{a_1, a_2\})^+ \cup a_2 a_1^*.
\end{aligned}$$

The system is working similarly to the one in the proof of Theorem 1. The only differences are the following two:

1. Any string $w a_2 a_1^n \in L_1$ is produced here in the second component (instead of the third) and, by splitting, w is sent to the master and $a_2 a_1^n$ to the third component (instead of the fourth one). But, because the communication by splitting from the second component to the other two is made only in the case when the sentential form of the second component is of the form $w a_2 a_1^n$, w being necessarily sent to the master and $a_2 a_1^n$ to the last component, this step is correctly performed.

2. The way in which the occurrences of X are handled in order to help us to use the productions of the form $A \longrightarrow BC, A, B, C \in N$, is slightly different. However, if the number of X's is too big, then no string will be produced (see observation 5 (ii) in the proof of Claim 2 above). We have only to show that indeed we can get sufficiently many X's to be able to apply a rule of the form $A \longrightarrow BC$ anytime it is needed. Supposing that the derivation in G is $\alpha_1 A \alpha_2 \Longrightarrow_G \alpha_1 BC \alpha_2$, we have two cases:

(i) $\alpha_1 = \alpha_2 = \lambda, A = S, r : S \longrightarrow BC \in P; B, C \in N$. We have in Γ (with $p : S \longrightarrow S \in P$)

$$\begin{gathered}
(S_1', S_2', S) \Longrightarrow_\Gamma (S_1', S_2', S_p) \vdash_\Gamma (S_1', S_p, S) \Longrightarrow_\Gamma (S_1', Y, S_p) \vdash_\Gamma \\
\vdash_\Gamma (Y, S_p, S) \Longrightarrow_\Gamma (X, S, S_p) \vdash_\Gamma (S_1', S_p, XS) \Longrightarrow_\Gamma (S_1', S, X_r S_r) \vdash_\Gamma \\
\vdash_\Gamma (S_1', X_r S_r, S) \Longrightarrow_\Gamma (S_1', BC, S_p).
\end{gathered} \tag{2}$$

(ii) $\alpha_1 A \alpha_2 \neq S, r : A \longrightarrow BC \in P, A, B, C \in N$. Let us prove first that we can have an X as the current sentential form of the first component anytime needed.

Any simulation in Γ of a derivation in one step, say $\alpha \Longrightarrow_G \beta$, consists of one or several iterations of the following sequence of steps: being the current sentential form of the second component, α is sent to the third one, is rewritten there, is sent back to the second component, and again rewritten. Because $p : S \longrightarrow S \in P$, we have $S_p \longrightarrow S \in P_2$ and $S \longrightarrow S_p \in P_3$. Thus, we can suppose that when the main string (that is the string which is at the beginning α and, rewritten and communicated between the last two components, will be β) is communicated from component 2 to the component 3 (or from 3 to 2), then the string S_p is communicated from the component 3 to the component 2 (or from 2 to 3, respectively). That can be also seen in (2).

Because we can perform in Γ

$$(-,-,S_p) \vdash_\Gamma (-,S_p,-) \Longrightarrow_\Gamma (-,Y,-) \vdash_\Gamma (Y,-,S) \Longrightarrow_\Gamma (X,-,S_p),$$

using the observations above, it should be clear that we can get an X as the current sentential form of the first component whenever we need one. (It is also seen that the role of the production $S \longrightarrow S$ introduced in P is much more important here.)

Going back to our case, we can suppose (as in the proof of Claim 1, Case 2 (iv)) that when the current sentential form of the second component is $\alpha_1 A \alpha_2$, then the current string in the first component is X. We can also suppose (also as in the proof of Claim 1, Case 2 (iv)) that $\alpha_1 = A_1 A_2 \ldots A_k \in N^*$. The derivation goes now similarly to (1).

Consequently, the system constructed here generates the same language as the one in the proof of Theorem 1. It follows that $L_S(\Gamma) = L$ and the proof is over. $\square$

We notice that in the system Γ in Theorem 2, the splitting communication is used only at the end when the string $w \in L$ is sent to the master and it will be the output of the system and the garbage $a_2 a_1^n$ is sent to the third component. In fact, the splitting communication is done in order to allow a workspace as big as needed.

If the splitting communication is not allowed, we can still obtain (using only regular rules) any context-sensitive language. The following result is a strengthening of Theorem 1 in [4] or of Corollary 3.4 in [10] (which establish that $CCPC_\infty CF = CS$.) It solves also the problem, open so far, of the hierarchy $(CCPC_n REG)_{n \geq 0}$.

Theorem 3 $CCPC_3 REG = CS$.

Proof. The construction is very similar to the one in Theorem 2. The difference is that the second component there is the master one here because we do not need any communication after obtaining the terminal string in the given language.

Let L be a context-sensitive language and let $G = (N, T, S, P)$ be a context-sensitive grammar generating L. We have seen in the proof of Theorem 1 that any production of G can be supposed to be of one of the following forms: $AB \longrightarrow CD$ with $A \neq B$, $A \longrightarrow BC$, $A \longrightarrow B$, or $A \longrightarrow a$ for some $A, B, C, D \in N, a \in T$.

Let S_2', S_3', X, and Y be symbols not in $N \cup T$ and

$$N' = \{A \mid A \in N\} \cup \{X'\},$$

$$\begin{aligned} V = \ & \{A_r \mid A \in N, r : AB \longrightarrow CD \in P \text{ or } r : BA \longrightarrow CD \in P\} \cup \\ & \cup \{A_r \mid A \in N, r : A \longrightarrow \alpha \in P\} \cup \\ & \cup \{X_r \mid r : A \longrightarrow BC \in P\} \cup \\ & \cup \{Z_A, W_A \mid A \in N\}. \end{aligned}$$

The system is here:

$$\Gamma = (N \cup N' \cup \{S_2', S_3', X, Y\} \cup V, T, (S_1, P_1, R_1), (S_2', P_2, R_2), (S_3', P_3, R_3))$$

where $S_1 = S$ and

$$\begin{aligned}
P_1 &= \{S_p \longrightarrow Y\} \cup \\
&\cup \{A' \longrightarrow A \mid A \in N \cup \{X\}\} \cup \\
&\cup \{A_r \longrightarrow a \mid r : A \longrightarrow a \in P\} \cup \\
&\cup \{A_r \longrightarrow B \mid r : A \longrightarrow B \in P\} \cup \\
&\cup \{A_r \longrightarrow C, B_r \longrightarrow D \mid r : AB \longrightarrow CD \in P\} \cup \\
&\cup \{X_r \longrightarrow B, A_r \longrightarrow C \mid R : A \longrightarrow BC \in P\} \cup \\
&\cup \{Z_A \longrightarrow A, W_A \longrightarrow X \mid A \in N\}, \\
R_1 &= \{\alpha_1 A_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T)^*, r : A \longrightarrow a \in P\} \cup \\
&\cup \{\alpha_1 A_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T)^*, r : A \longrightarrow B \in P\} \cup \\
&\cup \{\alpha_1 A_r B_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T)^*, r : AB \longrightarrow CD \in P\} \cup \\
&\cup \{\alpha_1 X_r A_r \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T)^*, r : A \longrightarrow BC \in P\} \cup \\
&\cup \{\alpha_1 Z_A W_A \alpha_2 \mid \alpha_1, \alpha_2 \in (N' \cup T)^*, A \in N\},
\end{aligned}$$

$$\begin{aligned}
P_2 &= \{A \longrightarrow A' \mid A \in N \cup \{X\}\} \cup \\
&\cup \{A \longrightarrow A_r \mid r : A \longrightarrow a \in P \text{ or } r : A \longrightarrow B \in P\} \cup \\
&\cup \{X \longrightarrow X_r, A \longrightarrow A_r \mid r : A \longrightarrow BC \in P\} \cup \\
&\cup \{A \longrightarrow A_r, B \longrightarrow B_r \mid r : AB \longrightarrow CD \in P\} \cup \\
&\cup \{X \longrightarrow Z_A, A \longrightarrow W_A \mid A \in N\}, \\
R_2 &= (N \cup \{X\} \cup T)^+.
\end{aligned}$$

$$\begin{aligned}
P_3 &= \{Y \longrightarrow X, Y \longrightarrow Z\}, \\
R_3 &= \{Y\},
\end{aligned}$$

The proof for $L(\Gamma) = L$ is very similar to the proof of Theorem 1 and therefore omitted.

□

It is proved in [4] that $CCPC_2REG = REG$ hence, using Lemma 1, we obtain that the results in Theorem 2 and Theorem 3 are optimal. Using also the results $CCPC_\infty CS = CS$ from [10] and $CS \subseteq CCPC_2CF$ from [4], we can draw the following diagram which shows the generative power of all types of systems with communication by command investigated so far by comparing them with the families in Chomsky hierarchy. (The place of the families $SCCPC_nX, CCPC_nX$ not mentioned in the diagram is obvious.)

$$\begin{tikzcd}
\left.\begin{array}{r} SCCPC_3REG \\ SCCPC_3CF \\ SCCPC_3CS \end{array}\right\} = & \mathbf{RE} & \\
\left.\begin{array}{r} SCCPC_2CF \\ SCCPC_2CS \end{array}\right\} = & \mathbf{CS} \arrow[u] & = \left\{\begin{array}{l} CCPC_3REG = \cdots = CCPC_\infty REG \\ CCPC_2CF = \cdots = CCPC_\infty CF \\ CCPC_1CS = \cdots = CCPC_\infty CS \end{array}\right. \\
SCCPC_1CF = & \mathbf{CF} \arrow[u] & = CCPC_1CF \\
SCCPC_2REG = & \mathbf{REG} \arrow[u] & = CCPC_2REG
\end{tikzcd}$$

References

[1] D. Angluin, Finding patterns common to a set of strings, *J. Comput. Syst. Sci.*, **21**(1980), 46 - 62.

[2] K. Culik II, A purely homomorphic characterization of recursively enumerable sets, *Journal of the Association for Computing Machinery*, **26**(1979), 345–350.

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

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

[5] L. Errico, "WAVE: An Overview of the Model and the Language", CSRG, Dept. Electronic and Electr. Eng., Univ. of Surrey, UK, 1993.

[6] L. Errico, C. Jesshope, Towards a new architecture for symbolic processing, *in* "Artificial Intelligence and Information-Control Systems of Robots '94" (I. Plander, ed.), World Sci. Publ., Singapore, 1994, 31 - 40.

[7] S. E. Fahlman, G. E. Hinton, T. J. Seijnowski, Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines, *in* "Proc. AAAI National Conf. on AI", William Kaufman, Los Altos, 1983, 109 - 113.

[8] W. D. Hillis, "The Connection Machine", MIT Press, Cambridge, 1985.

[9] J. E. Hopcroft, J. D. Ullman, "Introduction to automata theory, languages, and computation", Addison-Wesley, Reading, Mass., 1979.

[10] L. Ilie, Collapsing hierarchies in parallel communicating grammar systems with communication by command, *Computers and AI*, **15**, **2-3**(1996), 173 - 184.

[11] T. Jiang, E. Kinber, A Salomaa, K. Salomaa, S. Yu, Pattern languages with and without erasing, *Intern. J. Computer Math.*, **50**(1994), 147 - 163.

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

[13] A. Salomaa, "Formal Languages", Academic Press, New York, 1973.

[14] P. S. Sapaty, "The WAVE Paradigm", Internal Report 17/92, Dept. Informatics, Univ. of Karlsruhe, Germany, 1992.

[15] ***, "Connection Machine, Model CM-2. Tehnical Summary", Thinking Machines T. R. HA 87 - 4, MIT, Cambridge, USA, 1987.

Acta Cybernetica **12** (1996) 427-444.

Teams in Grammar Systems: Hybridity and Weak Rewriting [*]

Maurice H. ter BEEK [†]

Abstract

Some new ideas in the theory of teams in grammar systems are introduced and studied. Traditionally, a team is formed from a finite number of sets of productions and in every derivation step, one production from each component is used to rewrite a symbol of the sentential form. Hence rewriting is done in parallel. Several derivation modes are considered, varying from using a team exactly one time to using it a maximal amount of times. Here, the possibility of different teams having different modes of derivation is defined, as is a weaker restriction on the application of a team. The generative power of such mechanisms is investigated.

1 Introduction

In [4], *cooperating distributed grammar systems* (CD grammar sytems for short) were introduced to formalize a link, recognized in [6], between the so-called *multi-agent systems* theory in *Artificial Intelligence* and the theory of formal languages. Since then these systems have been studied intensively and this has already resulted in the monograph [5], which contains an exhaustive survey of the state of the art in the so-called theory of grammar systems until ca. 1992.

By now, many well-motivated enhancements have been introduced, resulting in *hybrid* CD grammar systems (allowing the grammars to have different capabilities, [22]) and *team* CD grammar systems (grouping the grammars in teams and rewrite in parallel, [20]), to name but a few.

Here hybrid (prescribed) team CD grammar systems are defined, thus allowing work to be done in teams while at the same time assuming these teams to have different capabilities. Two basically different versions can be defined. One can consider a hybrid CD grammar system and automatically form teams of its components according to some strategy or one can consider a CD grammar system

[*]This research was supported by a scholarship from the Hungarian Ministry of Culture and Education. Moreover, the facilities provided by the Department of General Computer Science of the Eötvös Loránd University and in particular by the Computer and Automation Research Institute of the Hungarian Academy of Sciences were essential.

[†]Department of Computer Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands. E-mail: mtbeek@wi.leidenuniv.nl

with prescribed teams and simply associate a (possibly different) so-called mode of derivation with each team. Concerning the latter one it will be shown that this hybridity does not enlarge the generative power any further. However, every recursively enumerable language can be generated by a hybrid prescribed team CD grammar system with teams of two members. The question whether the automatic forming of teams enlarges the generative power of hybrid CD grammar systems remains an open problem.

Furthermore, a variant of the way teams work in the literature so far is presented. The motivation to introduce a different concept of rewriting is twofold. Not only is the strict requirement that every component of the team must participate in every step often bothering in generating languages but, perhaps more important, it is definitely too restrictive in the most recent application of grammar systems as a framework for natural language generation (see, e.g., [8] and [10]).

This new way of rewriting is called *weak* rewriting and it is investigated in the case of teams in eco-grammar systems in [2]. It resembles the well-known concept of appearance checking in regulated rewriting: every component of a team which contains a production that can rewrite the sentential form must be used, but a component which does not contain any production with a left-hand side that is contained in the sentential form does not need to be used. The generative power of CD grammar systems with prescribed teams of variable size operating in the weak rewriting step will be shown to equal that of the class of programmed grammars with unconditional transfer. This implies that these families and those of the prescribed team CD grammar systems operating in the traditional rewriting step and the same modes of derivation do not coincide.

Finally, in the special case of prescribed team CD grammar systems with only one production per component and teams of variable size, an equality with the class of unordered scattered context grammars is presented. This leads to the fact that there are several cases when only one production per component suffices for prescribed team CD grammar systems with teams of variable size.

2 Preliminaries

In this section, some prerequisites necessary for understanding the sequel are defined. For details and unexplained notions, the reader is referred to [28] for formal languages, [13] for regulated rewriting, [27] for Lindenmayer systems and [5], [9], [11], [24] and [3] for (variants of) grammar systems.

The set of all non-empty strings over an *alphabet* V is denoted by V^+. If the *empty string*, λ, is included, the notation becomes V^*. The *length* of a string x is denoted by $|x|$.

An *inclusion* is denoted by $\subseteq$, whereas a *proper inclusion* is denoted by $\subset$.

Sometimes, the notation for a family of languages contains a λ between the brackets [and]. This means that the statement holds in the case of allowing λ-productions (indicated by the λ inbetween brackets) as well as in the case of a restriction to λ-free productions (thus neglecting the λ inbetween brackets). Also other symbols between brackets must now be understood.

Without definition, the family of context-free languages (CF) is used in the sequel. Its definition can be found in, e.g., [13]. The same holds for the family of languages generated by ET0L systems ($ET0L$). Finally, also the family of languages generated by [hybrid] CD grammar systems ($[H]CD$) shall not be defined here. However, their definitions can be found in [5] and will become clear in the sequel.

None of the above families of languages will be used in any construction in the proofs. Those families of languages that are used in (some of) the proofs below, are defined next.

An *unordered scattered context grammar with appearance checking* ([21]) is a construct $G = (N, T, S, P, F)$, where N is the set of nonterminals, T is the set of terminals, $S \in N$ is the axiom, $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of *rules* (rules are of the form $p_i : (\alpha_1, \alpha_2, \ldots, \alpha_{m_i}) \to (\beta_1, \beta_2, \ldots, \beta_{m_i})$, where $\alpha_j \to \beta_j$ are productions over $N \cup T$) and F is a set of occurrences of productions in P, $1 \leq i \leq n$. For $w, w' \in (N \cup T)^*$ and $1 \leq i \leq n$ it is said that w directly derives w', written as

$$\begin{array}{lll} w \Longrightarrow w' & \text{iff} & w = w_1\alpha_{i_1}w_2\alpha_{i_2}\ldots w_m\alpha_{i_m}w_{m+1},\ w' = w_1\beta_{i_1}w_2\beta_{i_2}\ldots w_m\beta_{i_m}w_{m+1}, \\ & & p_i : (\alpha_1, \alpha_2, \ldots, \alpha_p) \to (\beta_1, \beta_2, \ldots, \beta_p) \in P,\ (\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_m}) \text{ is a} \\ & & \text{permutation of a subsequence of } (\alpha_1, \alpha_2, \ldots, \alpha_p),\ w_l \in (N \cup T)^* \\ & & \text{and } 1 \leq l \leq m+1 \\ & \text{and} & \alpha_j \text{ in } \{\alpha_1, \alpha_2, \ldots, \alpha_p\} \text{ and not in } \{\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_m}\} \text{ implies that} \\ & & \alpha_j \text{ is not contained in } w \text{ and } \alpha_j \to \beta_j \in F. \end{array}$$

If $F = \emptyset$, the unordered scattered context grammar is called an *unordered scattered context grammar without appearance checking* and F is omitted from the construct. Moreover, if F contains all occurrences of productions in P, the unordered scattered context grammar is called *with unconditional transfer*. The language generated by G is $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$, where $\Longrightarrow^*$ denotes the reflexive and transitive closure of $\Longrightarrow$.

The family of languages generated by unordered scattered context grammars with λ-free context-free productions in P is denoted by USC_{ac} in the case of grammars with appearance checking; when grammars without appearance checking are considered the subscript ac is omitted and when grammars with unconditional transfer are considered the subscript ac is replaced by ut.

A *matrix grammar with appearance checking* is a construct $G = (N, T, S, M, F)$, where N is the set of nonterminals, T is the set of terminals, $S \in N$ is the axiom, M is a finite set of *matrices* of the form $m : (r_1, r_2, \ldots, r_n)$, where $r_i : \alpha_i \to \beta_i$ are productions over $N \cup T$ and $|\alpha|_N \geq 1$, $1 \leq i \leq n$ and F, finally, is a set of occurrences of productions in M. For $w, w' \in (N \cup T)^*$ and $m : (\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_n \to \beta_n) \in M$ it is said that w directly derives w', written as

$$w \Longrightarrow w' \quad \text{iff} \quad \text{there exist } w_0, w_1, \ldots, w_n \in (N \cup T)^* \text{ such that}$$

$w_0 = w$ and $w_n = w'$ and for all $0 \leq i \leq n-1$

either $w_{i-1} = w'_{i-1}\alpha_i w''_{i-1}$ and $w_i = w'_{i-1}\beta_i w''_{i-1}$
for some $w'_{i-1}, w''_{i-1} \in (N \cup T)^*$

or the production $\alpha_i \to \beta_i$ cannot be applied to w_{i-1},
$\alpha_i \to \beta_i \in F$ and $w_i = w_{i-1}$.

If $F = \emptyset$, the matrix grammar is called a *matrix grammar without appearance checking* and F is omitted from the construct. Moreover, if F contains all occurrences of productions in M, the matrix grammar is called *with unconditional transfer*. The language generated by G is $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$, where $\Longrightarrow^*$ denotes the reflexive and transitive closure of $\Longrightarrow$.

The family of languages generated by matrix grammars with λ-free context-free productions in M is denoted by MAT_{ac} in the case of grammars with appearance checking; when grammars without appearance checking are considered the subscript *ac* is omitted and when grammars with unconditional transfer are considered the subscript *ac* is replaced by *ut*.

For all generative devices mentioned above, only the notation in the case of λ-free context-free productions was given. When there is no restriction to λ-free productions a superscript λ is added to the notation.

3 Teams in grammar systems

Definition 1 *Let N and T be two disjoint alphabets. A* production *over (N,T) is a pair $(A,x) \in N \times (N \cup T)^*$. Usually, $A \to x$ shall be written instead of (A,x). If $x \neq \lambda$, then $A \to x$ is called a* λ-free *production. A* team *over (N,T) is a multiset of sets of productions over (N,T). The sets of productions occurring in a team shall be referred to as* components.

Traditionally, a team rewrites a string in the following manner. Here, this original notion is renamed *strong* rewriting since another way of rewriting is introduced after this definition.

Definition 2 *Let N and T be two disjoint alphabets. Let Q be a team over (N,T) and $x,y \in (N \cup T)^*$. Then x is rewritten by Q, in the strong rewriting step, into y, written as*

$$x \overset{s}{\Longrightarrow}_Q y \quad \textit{iff} \quad \begin{array}{l} x = x_1A_1x_2A_2\ldots x_nA_nx_{n+1},\ y = x_1y_1x_2y_2\ldots x_ny_nx_{n+1}, \\ x_i \in (N \cup T)^*,\ 1 \leq i \leq n+1,\ A_j \to y_j \in P_j,\ 1 \leq j \leq n \textit{ and} \\ Q = \{P_1, P_2, \ldots, P_n\}. \end{array}$$

A derivation step of a team thus consists of choosing a production from each component of this team and applying these in parallel on the string to be rewritten.

Now the *weak* rewriting step for teams is introduced. It is loosely based on the so-called weakly competitive rewriting step for colonies as introduced in [12].

Definition 3 *Let N and T be two disjoint alphabets. Let Q be a team over (N,T) and $x, y \in (N \cup T)^*$. Then x is rewritten by Q, in the weak rewriting step, into y, written as*

$$x \overset{w}{\Longrightarrow}_Q y \quad \text{iff} \quad \begin{array}{l} x = x_1A_1x_2A_2\ldots x_nA_nx_{n+1},\ y = x_1y_1x_2y_2\ldots x_ny_nx_{n+1}, \\ x_i \in (N\cup T)^*,\ 1 \le i \le n+1,\ A_j \to y_j \in P_j,\ 1 \le j \le n \text{ and} \\ \{P_1, P_2, \ldots, P_n\} \subseteq \{P_1, P_2, \ldots, P_s\} = Q \text{ such that} \\ \text{for all } P_q \in Q \setminus \{P_1, P_2, \ldots, P_n\} \text{ there exists} \\ \text{no production } \alpha \to \beta \in P_q \text{ such that } \alpha \in x_1x_2\ldots x_{n+1}. \end{array}$$

The weak rewriting step of a team thus works in the same way as the strong rewriting step, as far as choosing a production from each component of this team and applying these in parallel on the current sentential form is concerned. However, a derivation according to the strong rewriting step is blocked (1) when a component of the team does not contain a production with a left-hand side that is contained in the current sentential form or (2) when two (or more) components can only rewrite a symbol of the current sentential form that appears only once in that sentential form. In the weak rewriting step neither case results in a blocked derivation, since only every component containing a production that can rewrite a symbol from the current sentential form, without clashing with another component for wanting to rewrite the same symbol, applies these productions in parallel on the current sentential form.

If Q is a singleton team, i.e. $Q = \{P\}$ for some set of productions P, then $x \overset{-}{\Longrightarrow}_P y$ shall be written instead of $x \overset{-}{\Longrightarrow}_{\{P\}} y$, for $- \in \{s, w\}$. It is clear that in that case only one symbol in x is rewritten, using a production from P.

So-called modes of derivation are used to prescribe halting requirements on the use of a team. These modes can be divided into three groups. Firstly, mode $*$ has *no restrictions* whatsoever. Any number of derivation steps is allowed. Secondly, modes $\le k$, $= k$ and $\ge k$ restrict the number of derivation steps to *at most*, *exactly* and *at least* k derivation steps, respectively. Thirdly, modes t_0, t_1 and t_2 are modes that represent a so-called *maximal* number of derivation steps. All three prescribe a slightly different condition which needs to be fulfilled before a team is considered to have successfully worked in that mode. In the case of mode t_0 the work of a team ends successfully when *no further derivation step can be done as a team*, in the case of mode t_1 the work ends when *no component of the team can apply one of its productions any longer* and in mode t_2, finally, the work of a team ends when *there is at least one component that can no longer apply one of its productions.* For these so-called maximal derivation modes, a distinction is made between the weak and the strong rewriting step.

Definition 4 *Let $Q = \{P_1, P_2, \ldots, P_n\}$ be a team over (N,T) and let $f \in \{\le k, = k, \ge k \mid k \ge 1\} \cup \{*, t_0, t_1, t_2\}$ be a* mode (of derivation). *Furthermore, let*

$x, y, z \in (N \cup T)^*$ and $k \in \mathbf{N}$. Then x is rewritten by Q, in the weak ($- = w$) or strong ($- = s$) rewriting step and working in mode f, into y, written as

$$
\begin{array}{llll}
x & \overset{-}{\Longrightarrow}{}_Q^{\leq k} & y & \textit{iff} \quad x \overset{-}{\Longrightarrow}{}_Q^{k'} y \textit{ for some } k' \leq k, \\
x & \overset{-}{\Longrightarrow}{}_Q^{=k} & y & \textit{iff} \quad x \overset{-}{\Longrightarrow}{}_Q^{k} y, \\
x & \overset{-}{\Longrightarrow}{}_Q^{\geq k} & y & \textit{iff} \quad x \overset{-}{\Longrightarrow}{}_Q^{k'} y \textit{ for some } k' \geq k, \\
x & \overset{-}{\Longrightarrow}{}_Q^{*} & y & \textit{iff} \quad x \overset{-}{\Longrightarrow}{}_Q^{k} y \textit{ for some } k, \\
x & \overset{-}{\Longrightarrow}{}_Q^{t_0} & y & \textit{iff} \quad x \overset{-}{\Longrightarrow}{}_Q^{*} y \textit{ and there is no } z \textit{ such that } y \overset{-}{\Longrightarrow}{}_Q z, \\
x & \overset{s}{\Longrightarrow}{}_Q^{t_1} & y & \textit{iff} \quad x \overset{s}{\Longrightarrow}{}_Q^{*} y \textit{ and for no component } P_i \in Q \textit{ and no } z \\
 & & & \qquad \textit{there is a derivation } y \overset{s}{\Longrightarrow}{}_{P_i} z \textit{ and} \\
x & \overset{s}{\Longrightarrow}{}_Q^{t_2} & y & \textit{iff} \quad x \overset{s}{\Longrightarrow}{}_Q^{*} y \textit{ and there is a component } P_i \in Q \\
 & & & \qquad \textit{for which there is no derivation } y \overset{s}{\Longrightarrow}{}_{P_i} z.
\end{array}
$$

The three variants of the t-mode of derivation first appeared in [17] (t_0), [20] (t_1) and [26] (t_2); the other modes of derivation are the natural extension of the modes in CD grammar systems (see [5]) to teams of grammars.

Now a more general definition of teams in the theory of grammar systems than the original one from [20] and its generalization from [26] can be introduced.

Definition 5 *A* hybrid prescribed team CD grammar system *is a construct*

$$\Gamma = (N, T, S, P_1, P_2, \ldots, P_n, (Q_1, f_1), (Q_2, f_2), \ldots, (Q_m, f_m)),$$

where N is the set of nonterminals, T is the set of terminals, with $N \cap T = \emptyset$, $S \in N$ is the axiom, $P_1, P_2, \ldots, P_n$ are sets of productions over (N, T), $Q_1, Q_2, \ldots, Q_m$ are teams with components from $P_1, P_2, \ldots, P_n$ and $f_1, f_2, \ldots, f_m$ are modes of derivation.

If, in this construct, $f_i = f_j$ for all $1 \leq i, j \leq m$, the definition of a prescribed team CD grammar system as in [26] is obtained.

Note that in this definition, there is no restriction on the size of a team. In the original definition of teams in [20], however, they are of constant size. A natural number $s \geq 1$ is given and the teams are formed such that the number of components of every team is exactly s; these teams are called of constant size s. Moreover, in that definition the teams are not prescribed, but each set of components can be a team (so-called *free* teams) as long as the size restriction is fulfilled.

It is now clear that one can differentiate between the following four variants of teams in the theory of grammar systems. For all four, hybridity is another possibility.

Free teams of constant size: this is the original definition of [20], as explained above.

Free teams of variable size: each subset of components can be a team.

Prescribed teams of constant size: all prescribed teams consist of the same number of components.

Prescribed teams of variable size: these are defined in Definition 5.

In the case of teams of constant size, whether prescribed or free, a finite set of axioms $W \subseteq (N \cup T)^*$, with only one string in it containing nonterminals, is allowed. This is done since otherwise in the case of λ-free productions no string shorter than s could be generated. In the case of free teams with teams of constant size, the construct thus becomes $\Gamma = (N, T, W, P_1, P_2, \ldots, P_n)$. The modifications in the other cases are obvious.

Definition 6 *Consider a hybrid prescribed team CD grammar system* Γ *as in Definition 5. Then the language generated by* Γ*, operating in the weak* $(- = w)$ *or strong* $(- = s)$ *rewriting step, is*

$$L^-(\Gamma) = \{z \in T^* \mid S \overset{-}{\Longrightarrow}{}^{f_{i_1}}_{Q_{i_1}} w_{i_1} \overset{-}{\Longrightarrow}{}^{f_{i_2}}_{Q_{i_2}} \cdots \overset{-}{\Longrightarrow}{}^{f_{i_p}}_{Q_{i_p}} w_{i_p} = z,\ 1 \leq i_j \leq m,\ 1 \leq j \leq p\}.$$

When dealing with a language generated by teams of constant size, the notation of Definition 6 is modified to $L^-(\Gamma, s)$. When the teams are not hybrid, the mode of derivation is added as a subscript to this notation.

The family of languages generated by CD grammar systems with hybrid prescribed teams of variable size, operating in the strong rewriting step and λ-free context-free productions is denoted by HPT_*CD. When teams are of constant size s, the $*$ in the notation is replaced by s and when there is no restriction to λ-free productions, λ is added to the notation as a superscript. When the teams are not hybrid (prescribed) the H (P) in the notation is omitted.

The weak rewriting step is only considered in the sequel for CD grammar systems with prescribed teams of variable size. The family of languages generated by such systems, working in derivation mode f and operating in the weak rewriting step, is denoted by $PT_wCD(f)$ in the case of λ-free context-free productions; when λ-productions are allowed the superscript λ is added.

Instead of prescribing the hybrid teams, another way to introduce hybrid teams is defined next. Consider a hybrid CD grammar system and automatically form teams by combining all components with a certain mode of derivation to form a team with that mode of derivation. Because the teams are formed automatically, they are not part of the system "hardware", but a way to define the work of the system.

Definition 7 *Consider a hybrid CD grammar system*

$$\Gamma = (N, T, S, (P_1, f_1), (P_2, f_2), \ldots, (P_n, f_n)),$$

where N is the set of nonterminals, T is the set of terminals, with $N \cap T = \emptyset$, $S \in N$ is the axiom, $P_1, P_2, \ldots, P_n$ are sets of productions over (N,T) and $f_1, f_2, \ldots, f_m$ are modes of derivation.

Then teams $(Q_i, g_i) \subseteq \{(P_1, f_1), (P_2, f_2), \ldots, (P_n, f_n)\}$ are automatically formed in the following way. For $g_i \in \{*, t_0, t_1, t_2\} \cup \{\leq k, = k, \geq k \mid k \geq 1\}$

$$(Q_i, g_i) = \{(P_k, f_k) \mid f_k = g_i,\ 1 \leq k \leq n\}.$$

Such a team $(Q_i, g_i) = \{(P_{j_1}, f_{j_1}), (P_{j_2}, f_{j_2}), \ldots, (P_{j_s}, f_{j_{s_i}})\}$, is called an automatically formed team working in mode g_i.

The language generated by Γ with automatically formed teams is

$$L^{aut}(\Gamma) = \{z \in T^* \mid S \Longrightarrow_{Q_{i_1}}^{g_{i_1}} w_{i_1} \Longrightarrow_{Q_{i_2}}^{g_{i_1}} \cdots \Longrightarrow_{Q_{i_m}}^{g_{i_m}} w_{i_m} = z,\ m \geq 1\}.$$

The family of languages generated by hybrid CD grammar systems with automatically formed teams of variable size and only λ-free context-free productions is denoted by HT_*CD; when λ-productions are allowed the notation becomes HT_*CD^λ. Note that due to the automatical construction from a hybrid CD grammar system (with a one-symbol axiom), the notion of teams of constant size is very restricted. Only teams of constant size 1 could be constructed, but they obviously have the same generative power as the underlying hybrid CD grammar system. Naturally, it is possible to consider hybrid CD grammar systems with a string axiom instead of a single nonterminal.

Some relations concerning the generative power of several of these grammar systems discussed above are given next. A more complete overview can be found in [1]. In the first paper on teams in grammar systems, [20], it was proved that, for $f \in \{=1, \geq 1, *\} \cup \{\leq k \mid k \geq 1\}$,

$$CF = T_1CD(f) \subset T_2CD(f) \text{ and}$$
$$ET0L = T_1CD(t) \subset T_2CD(t_1).$$

These relations prove that there are modes of derivation for which the forming of teams strictly increases the power of CD grammar systems, since $CD(t) = ET0L$ and $CF = CD(=1) = CD(\geq 1) = CD(*) = CD(\leq k)$ for a $k \geq 1$ were already known to hold (see, e.g., [5]). In [7] it was proved that teams of size two suffice, i.e. for $s \geq 2$

$$T_sCD(t_1) \subseteq T_2CD(t_1).$$

The main results of [26] are, for $s \geq 2$, $f \in \{*\} \cup \{\leq k, = k, \geq k \mid k \geq 1\}$ and $g \in \{t_1, t_2\}$,

$$PR^{[\lambda]} = PT_sCD^{[\lambda]}(f) = PT_*CD^{[\lambda]}(f) \text{ and}$$
$$PR_{ac}^{[\lambda]} = T_sCD^{[\lambda]}(g) = PT_sCD^{[\lambda]}(g) = PT_*CD^{[\lambda]}(g)$$

and the main result of [17] is, for $s \geq 2$ and $h \in \{t_0, t_1\}$,

$$MAT_{ac}^{[\lambda]} = T_sCD^{[\lambda]}(h) = PT_sCD^{[\lambda]}(h) = PT_*CD^{[\lambda]}(h) = T_*CD^{[\lambda]}(h).$$

4 Homogeneous versus heterogeneous teams

The next lemma follows immediately from the definitions stated in the previous section.

Lemma 1 *For* $s \geq 1$ *and* $f \in \{*, t_0, t_1, t_2\} \cup \{\leq k, =k, \geq k \mid k \geq 1\}$

(i) $T_sCD^{[\lambda]}(f) \subseteq PT_sCD^{[\lambda]}(f) \subseteq PT_*CD^{[\lambda]}(f)$,
$T_*CD^{[\lambda]}(f) \subseteq PT_*CD^{[\lambda]}(f) \subseteq HPT_*CD^{[\lambda]}$ *and*
$PT_sCD^{[\lambda]}(f) \subseteq HPT_sCD^{[\lambda]} \subseteq HPT_*CD^{[\lambda]}$,

(ii) $HCD^{[\lambda]} = HT_1CD^{[\lambda]} \subseteq HPT_sCD^{[\lambda]} \subseteq HPT_*CD^{[\lambda]}$ *and*
$HT_1CD^{[\lambda]} \subseteq HT_*CD^{[\lambda]} \subseteq HPT_*CD^{[\lambda]}$ *and*

(iii) $[H][P]T_sCD^{[\lambda]} \subseteq [H][P]T_{s+1}CD^{[\lambda]}$.

It is natural to ask whether results similar to those that were stated in the previous section, can be obtained for the new definitions concerning hybrid teams of grammars. Indeed, some similar results for the hybrid cases will be proved below, but some open problems remain.

To begin with, some results concerning hybrid prescribed team CD grammar systems are presented. The next corollary follows immediately from Lemma 1 and results stated in the previous section.

Corollary 1 *For* $s \geq 2$

$$PR_{ac}^{[\lambda]} \subseteq HPT_sCD^{[\lambda]}.$$

For the λ-free case the next lemma is necessary to conclude that hybrid prescribed team CD grammar systems cannot generate more than the non-hybrid ones.

Lemma 2

$$HPT_*CD^{[\lambda]} \subseteq MAT_{ac}^{[\lambda]}.$$

Proof Consider the hybrid prescribed team CD grammar system

$$\Gamma = (N, T, S, P_1, P_2, \ldots, P_n, (Q_1, f_1), (Q_2, f_2), \ldots, (Q_m, f_m)).$$

Define the homomorphism h from $(N \cup T)^*$ into $(\{A' \mid A \in N\} \cup T)^*$ by

$$h(a) = a \text{ for } a \in T \text{ and } h(A) = A' \text{ for } A \in N.$$

Moreover, associate to a team $Q_i = \{P_{i_1}, P_{i_2}, \ldots, P_{i_{s_i}}\}$, $1 \leq i \leq m$, all sequences of productions such that from each component P_{i_j}, $1 \leq j \leq s_i$, exactly one production is included in such a sequence. Denote such a sequence by $\sigma = (A_1 \to x_1, \ldots, A_s \to x_s)$ and all such sequences associated to a team Q_i by $Seq_i = \{\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_{l_i}}\}$, $1 \leq i \leq m$.

To simulate this hybrid prescribed team CD grammar system, construct the following matrix grammar

$$G' = (N', T', S', M', F'),$$

where

$$\begin{aligned}
N' &= N \cup \{A' \mid A \in N\} \cup \{T, F\} \cup \{\Sigma_{i_j}, \Sigma'_{i_j} \mid 1 \leq j \leq l_i, 1 \leq i \leq m\} \cup \\
&\quad \{[Q_i, f_i, j] \mid (Q_i, f_i) \in \Gamma, f_i \in \{\leq k, = k, \geq k\}, 1 \leq i \leq m, 0 \leq j \leq k\} \cup \\
&\quad \{[Q_i, g_i], [Q_i, t_0]' \mid (Q_i, g_i) \in \Gamma, g_i \in \{*, t_0, t_1, t_2\}, 1 \leq i \leq m\}, \\
T' &= T \cup \{z\}, \\
M' &= \{(S' \to ST)\} \cup \\
&\quad \{(T \to [Q_i, f_i, 0]) \mid f_i \in \{\leq k, = k, \geq k\}, 1 \leq i \leq m\} \cup \\
&\quad \{(T \to [Q_i, g_i]) \mid g_i \in \{*, t_0, t_1, t_2\}, 1 \leq i \leq m\} \cup \\
&\quad \{([Q_i, f_i, j] \to [Q_i, f_i, j+1], A_1 \to h(x_1), A_2 \to h(x_2), \ldots, A_s \to h(x_s)) \mid \\
&\qquad\qquad 0 \leq j \leq k-1, (Q_i, f_i) = \{P_{j_1}, P_{j_2}, \ldots, P_{j_s}\}, A_r \to x_r \in P_{j_r}, \\
&\qquad\qquad\qquad f_i \in \{\leq k, = k, \geq k\}, 1 \leq i \leq m, 1 \leq r \leq s\} \cup \\
&\quad \{([Q_i, \geq k, k] \to [Q_i, \geq k, k], A_1 \to h(x_1), A_2 \to h(x_2), \ldots, A_s \to h(x_s)) \mid \\
&\qquad (Q_i, \geq k) = \{P_{j_1}, P_{j_2}, \ldots, P_{j_s}\}, A_r \to x_r \in P_{j_r}, 1 \leq i \leq m, 1 \leq r \leq s\} \cup \\
&\quad \{([Q_i, g_i] \to [Q_i, g_i], A_1 \to h(x_1), A_2 \to h(x_2), \ldots, A_s \to h(x_s)) \mid \\
&\qquad (Q_i, g_i) = \{P_{j_1}, P_{j_2}, \ldots, P_{j_s}\}, A_r \to x_r \in P_{j_r}, g_i \in \{*, t_0, t_1, t_2\}, \\
&\qquad\qquad\qquad 1 \leq i \leq m, 1 \leq r \leq s\} \cup \\
&\quad \{([Q_i, t_0] \to \Sigma_{i_1}) \mid 1 \leq i \leq m\} \cup \\
&\quad \{(\Sigma_{i_j} \to \Sigma'_{i_{j+1}}, A_1 \to \varphi_1, A_2 \to \varphi_2, \ldots, A_s \to \varphi_s) \mid \\
&\qquad \sigma_{i_j} = (A_1 \to x_1, \ldots, A_s \to x_s), \varphi_r \in \{A'_r, F\}, \varphi_r = F \text{ must hold for} \\
&\qquad\qquad \text{at least one } r, 1 \leq r \leq s, 1 \leq j \leq l_i - 1, 1 \leq i \leq m\} \cup \\
&\quad \{(\Sigma'_{i_{l_i}} \to [Q_i, t_0]', A_1 \to \varphi_1, A_2 \to \varphi_2, \ldots, A_s \to \varphi_s) \mid \\
&\qquad \sigma_{i_{l_i}} = (A_1 \to x_1, \ldots, A_s \to x_s), \varphi_r \in \{A'_r, F\}, \varphi_r = F \text{ must hold for} \\
&\qquad\qquad \text{at least one } r, 1 \leq r \leq s, 1 \leq i \leq m\} \cup \\
&\quad \{(\Sigma'_{i_j} \to \Sigma_{i_j}, A'_1 \to F, A'_2 \to F, \ldots, A'_k \to F) \mid \\
&\qquad\qquad \{A_1, A_2, \ldots, A_k\} = N, 1 \leq j \leq l_i, 1 \leq i \leq m\} \cup \\
&\quad \{(A' \to A) \mid A \in N\} \cup \\
&\quad \{([Q_i, \leq k, j] \to T), ([Q_i, = k, k] \to T), ([Q_i, \geq k, k] \to T), ([Q_i, *] \to T) \mid
\end{aligned}$$

$$1 \le i \le m, 0 \le j \le k\} \cup$$
$$\{([Q_i, t_0]' \to T, A_1' \to F, A_2' \to F, \ldots, A_k' \to F) \mid$$
$$\{A_1, A_2, \ldots, A_k\} = N, 1 \le i \le m\} \cup$$
$$\{([Q_i, t_1] \to T, A_1 \to F, A_1' \to F, A_2 \to F, A_2' \to F, \ldots, A_r' \to F) \mid$$
$$\{A_1, A_2, \ldots, A_r\} = \bigcup_{P_j \in (Q_i, t_1)} dom(P_j), 1 \le i \le m\} \cup$$
$$\{([Q_i, t_2] \to T, A_1 \to F, A_1' \to F, A_2 \to F, A_2' \to F, \ldots, A_r' \to F) \mid$$
$$\{A_1, A_2, \ldots, A_r\} = dom(P_j) \text{ for some } P_j \in (Q_i, t_2), 1 \le i \le m\} \cup$$
$$\{(T \to z)\} \text{ and}$$

in F' are all the productions $A \to F$ appearing in M'.

The simulation of Γ starts with introducing the sentential form ST, in which S is the start-symbol of Γ and T is a marker. The marker will control the derivation and S will generate the language of the hybrid CD grammar system with prescribed teams. This marker is non-deterministically replaced by a control symbol of the form $[Q_i, f_i, j]$ or $[Q_i, g_i]$. In these nonterminals, Q_i is the team working in mode f_i or g_i and j is a counter, necessary for the modes $f_i \in \{\le k, = k, \ge k\}$. With teams working in mode $g_i \in \{*, t_0, t_1, t_2\}$ we do not need to count and the third component is omitted.

When the marker $[Q_i, f_i, j]$ ($[Q_i, g_i]$) is present in the sentential form a simulation by Q_i in mode f_i (g_i) is simulated. The homomorphism h priming all nonterminals in the matrices is necessary to guarantee that the productions are applied to nonterminals that were already existing in the sentential form before these matrices were applied and not to those introduced by a production from these matrices themselves. The counter in the case of modes $\le k$, $= k$ and $\ge k$ guarantees that a team rewrites the sentential form less than k, exactly k or at least k times, respectively. In case of mode $*$, t_0, t_1 and t_2 there is no counting at all.

In case of t_1 and t_2, however, the productions in the set F guarantee that a team does not stop rewriting until no more component or at least one component of the team can no longer be used, respectively. Finally, in mode t_0 the symbol $[Q_i, t_0]$ can be replaced only by Σ_{i_1}. This symbol can then be replaced by $\Sigma'_{i_{j+1}}$ and back to $\Sigma_{i_{j+1}}$ until $\Sigma'_{i_{k_i}}$ is reached. In this way the correct termination of Q_i in mode t_0 is checked, by the following restrictions.

Firstly, Σ_{i_j} can only be replaced by $\Sigma'_{i_{j+1}}$ if the corresponding sequence of productions indeed cannot be used anymore. An F is introduced otherwise, since each sequence must have at least one $\varphi_r = F$. Secondly, $\Sigma'_{i_{j+1}}$ is allowed to be replaced by $\Sigma_{i_{j+1}}$ only after all primed symbols have been replaced by their originals. Finally, $\Sigma'_{i_{k_i}}$ can only be replaced by $[Q_i, t_0]'$ after indeed none of the sequences Σ_{i_j}, $1 \le j \le l_i$, can be used and then eventually be replaced by T.

In every case, afterwards the primes are removed and another team can non-deterministically take the marker spot and start its simulation in its mode. Eventually a terminal string results from S followed by the marker T. This marker is then replaced by z thus yielding $L(G') = L(\Gamma)\{z\}$. This symbol z can be removed by a morphism and thus, since it is known from [13] that the family MAT_{ac} is closed under restricted morphisms, $L(\Gamma) \in MAT_{ac}$ and the first statement of the lemma is proved.

$HPT_*CD^{\lambda} \subseteq MAT_{ac}^{\lambda}$ can be proved directly by a similar construction, even simplified since the marker can eventually be replaced by λ, making the use of a morphism unnecessary. □

It is known that $PR_{ac}^{[\lambda]} = MAT_{ac}^{[\lambda]}$ (see, e.g., [13]), hence the following corollary follows directly from Lemma 2.

Corollary 2 $HPT_*CD^{[\lambda]} \subseteq PR_{ac}^{[\lambda]}$.

All these results for hybrid prescribed team CD grammar systems immediately lead to a result for hybrid CD grammar systems with automatically formed teams, presented next.

Corollary 3 *For* $s \geq 1$

$$HT_1CD^{[\lambda]} \subseteq HT_*CD^{[\lambda]} \subseteq PR_{ac}^{[\lambda]}.$$

Combining these lemmas and corollaries concerning the new definitions, the following theorem is obtained.

Theorem 1 *For* $s \geq 2$

$$HT_*CD^{[\lambda]} \subseteq HPT_*CD^{[\lambda]} = HPT_sCD^{[\lambda]} = PR_{ac}^{[\lambda]}.$$

5 Weak versus strong rewriting

It is not hard to see that the principle of weak rewriting, not having to apply productions if they cannot be applied, resembles the appearance checking feature in regulated rewriting. Therefore, the following lemma does not come as a surprise. In the sequel, a restriction to only one production per component will be indicated by a 1 added as subscript. To be even more precise, denote U_mSC_{ut} for the class of unordered scattered context grammars with unconditional transfer and m scattered context rules and denote $P_mT_wCD_1(f)$ for the class of prescribed team CD grammar systems with m teams of variable size, 1 production per component, working in mode f and operating in the weak rewriting step.

Lemma 3 *For* $m \geq 1$ *and* $f \in \{=1, \geq 1, *\} \cup \{\leq k \mid k \geq 1\}$

$$U_mSC_{ut}^{[\lambda]} = P_mT_wCD_1^{[\lambda]}(f) \text{ and } U_mSC^{[\lambda]} = P_mT_*CD_1^{[\lambda]}(f).$$

Proof Only the inclusion from left to right of the first statement is proved here, all other inclusions can be proved in a similar straigthforward way. Consider an unordered scattered context grammar

$$G = (N, T, S, P, F)$$

with unconditional transfer and m scattered context rules. Moreover, for $P = \{p_1, p_2, \ldots, p_m\}$, $p_i : (\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,k_i}) \to (\beta_{i,1}, \beta_{i,2}, \ldots, \beta_{i,k_i})$ and $1 \le i \le m$, denote

$$r_{i,j} = \alpha_{i,j} \to \beta_{i,j} \text{ for } 1 \le j \le k_i.$$

To simulate this unordered scattered context grammar, construct the prescribed team CD grammar system

$$\Gamma = (N, T, S, P_1, P_2, \ldots, P_n, Q_1, Q_2, \ldots, Q_m),$$

where

$P_1, P_2, \ldots, P_n$	are	the components $\{r_{i,j}\}$ for $1 \le j \le k_i$ and $1 \le i \le m$ and
$Q_1, Q_2, \ldots, Q_m$	are	the teams $\{\{r_{1,j}\}, \{r_{2,j}\}, \ldots, \{r_{m,j}\}\}$ for $1 \le j \le k_i$ and $1 \le i \le m$.

A parallel rewriting step of an unordered scattered context grammar is simulated by a parallel rewriting step of a team, with its components being exactly the same productions as in the scattered context rule. Every component contains exactly one such a production and the number of teams equals the number of scattered context rules. Any production in G as well as in Γ does not have to be applied, if it cannot be applied to the sentential form.

Note that the proof requires the unordered character of the scattered context grammar, for a component of a team can rewrite any occurrence of the left-hand side of its production in the current sentential form. Since a team has to simulate the use of a scattered context rule, its mode of derivation is restricted to the cases as stated in the lemma. Clearly, $L(\Gamma) = L(G)$ and the lemma is proved for the case with as well as for the case without λ-productions. □

This lemma has some interesting corollaries.

Corollary 4 *For $x \in \{s, *\}$, $f \in \{=1, \ge 1, *\} \cup \{\le k \mid k \ge 1\}$ and $g \in \{*\} \cup \{\le k, =k, \ge k \mid k \ge 1\}$*

$$PR_{ut}^{[\lambda]} = PT_wCD_1^{[\lambda]}(f) \not\subseteq PT_xCD^{[\lambda]}(g).$$

Proof The equalities $PR_{ut}^{[\lambda]} = USC_{ut}^{[\lambda]}$ can be found in [16] and Lemma 3 thus leads to the equality in the statement. In [19] it is proved that the language

$\{a^{2^n} \mid n \geq 1\}$ cannot be generated by $PR^{[\lambda]}$. However, the programmed grammar (with unconditional transfer)

$$G_1 = (\{S, A, F\}, \{a\}, S, P),$$

where

$$\begin{aligned} P \;=\; & \{(1 : S \rightarrow AA, \{1,2,5\}, \{1,2,5\}), \\ & (2 : S \rightarrow F, \{3\}, \{3\}), \\ & (3 : A \rightarrow S, \{3,4\}, \{3,4\}), \\ & (4 : A \rightarrow F, \{1\}, \{1\}), \\ & (5 : A \rightarrow a, \{5\}, \{5\})\} \end{aligned}$$

generates $L(G_1) = \{a^{2^n} \mid n \geq 1\} \in PR_{ut}^{[\lambda]}$ and thus $PR_{ut}^{[\lambda]} \not\subseteq PR^{[\lambda]}$ holds. Finally, $PR^{[\lambda]} = PT_xCD^{[\lambda]}(g)$, for $x \in \{s, *\}$ and $g \in \{*\} \cup \{\leq k, =k, \geq k \mid k \geq 1$, is stated in Section 3. □

Thus, for several modes of derivation, a prescribed team CD grammar system with only 1 production per component and operating in the weak rewriting mode cannot be simulated by a prescribed team CD grammar system operating in the strong rewriting step not even when there is no limit of 1 production per component.

Corollary 5 *For* $f \in \{=1, \geq 1, *\} \cup \{\leq k \mid k \geq 1\}$

$$CD(t) \subset PT_wCD_1(f) \subset PT_wCD_1^{\lambda}(f).$$

Proof The equality $CD(t) = ET0L$ can be found in [5]. The strict inclusion $ET0L \subset O$, where O denotes the family of languages generated by the ordered grammars (with context-free productions) as introduced in [18], can be found in [13]. Furthermore, $O \subset PR_{ut}$ can be found in [14]. In [16], $PR_{ut} = USC_{ut}$ is proved. Finally, in [15], it was proved that $PR_{ut} \subset PR_{ut}^{\lambda}$. Together with Lemma 3 these results lead to a proof of the statement. □

Hence, for several modes of derivation, already a prescribed team CD grammar system with only 1 production per component and operating in the weak rewriting step can generate more than a CD grammar system working in mode t can.

Corollary 6 *For* $f \in \{=1, \geq 1, *\} \cup \{\leq k \mid k \geq 1\}$

$$PT_*CD^{[\lambda]}(f) = PT_*CD_1^{[\lambda]}(f).$$

Proof These results follow from Lemma 3 and the fact that $USC^{[\lambda]} = PR^{[\lambda]}$ (see, e.g., [13]) and $PR^{[\lambda]} = PT_*CD^{[\lambda]}(f)$ for $f \in \{*\} \cup \{\leq k, =k, \geq k \mid k \geq 1\}$ (see

Section 3) hold. □

Hence teams with one production per component suffice for prescribed team CD grammar systems with teams of variable size operating in derivation mode $=1$, ≥ 1, $*$ or $\leq k$ (for a $k \geq 1$).

Remark 1 *Note that $CD(f) = CF$ (see Section 3), though $CF \subset PT_*CD_1(f)$ (see Section 3 and Corollary 6), for $f \in \{=1, \geq 1, *\} \cup \{\leq k \mid k \geq 1\}$. Hence even CD grammar systems with n components cannot generate all languages that can be generated by prescribed team CD grammar systems with teams of variable size and only* 1 *production per component, for modes $f \in \{=1, \geq 1, *\} \cup \{\leq k \mid k \geq 1\}$.*

6 Open problems

It is clear that many open problems remain, both in the field of homogeneous versus heterogeneous teams as in the case of weak versus strong rewriting. To start with the latter: is strong rewriting more powerful than weak rewriting, or is the class of programmed grammars with unconditional transfer equal to the class of programmed grammars with appearance checking? My conjecture is the former, since the latter would settle the conjecture $PR_{ut}^{[\lambda]} \subset PR_{ac}^{[\lambda]}$ in the negative and this very interesting open problem in the theory of formal languages is very widely conjectured to hold. In fact, in [29], the class of programmed grammars is claimed to be closed under intersection with regular sets (which would result in a proper inclusion indeed), but the proof is subject to disbelief (see, e.g., [15]).

A possible angle into solving this open problem is to investigate the generative power of prescribed team CD grammar systems operating in the weak rewriting step with a maximal derivation mode. This might help to fill or to definitely establish the gap between programmed grammars with unconditional transfer and those with appearance checking. More investigation into the weak rewriting step might also finally prove $PR^{[\lambda]} \not\subseteq PR_{ut}^{[\lambda]}$.

It is interesting to note that also for colonies (for a definition of colonies, see, e.g., [12]) and for teams in eco-grammar systems ([2]), the relation between weak and strong rewriting is unknown. An answer to those relations would not necessarily solve the case for teams in CD grammar systems, but it might shed light on some intrinsic characteristics of weak versus strong rewriting. However, in the case of colonies no relation between the two ways of rewriting is known yet, whereas in the case of eco-grammar systems it was proved in [2] that strong rewriting can be simulated by weak rewriting.

Concerning homogeneous and heterogeneous teams, the main open problem is whether automatic forming of teams strictly increases the generative power of hybrid CD grammar systems. The conjecture, at least for the λ-free case, is yes since this would result in confirmation of the conjecture, stated in [23], that the inclusion $HCD \subseteq MAT_{ac}$ is proper. This might be a difficult open problem to settle since several years after their introduction in [22] still many problems concerning hybrid CD grammar systems are open.

Especially the relation with matrix grammars is wide open, since in [23] also the relation between matrix grammars without appearance checking and hybrid CD grammar systems is posed as an open problem. However, several different angles have been provided so far. For example, in [1], graph controlled hybrid CD grammar systems ($GCHCD$) were defined and they were proved to be included in the matrix grammars with appearance checking and to include both the hybrid CD grammar systems and the matrix grammars without appearance checking. It is not known, however, whether these inclusions are proper or whether equalities can be proved, but one of the inclusions of $MAT \subseteq GCHCD \subseteq MAT_{ac}$ must be proper. A solution to (one of) these open problems could shed light on this relation between hybrid CD grammar systems and matrix grammars without appearance checking, or perhaps even solve this open problem.

Acknowledgements

This work has benefited from discussions with and comments and suggestions from E. Csuhaj-Varjú, H.C.M. Kleijn and Gh. Păun. This paper is an excerpt from Part III: *Teams in CD grammar systems* of my master's thesis ([1]).

References

[1] M. H. ter Beek, Teams in grammar systems, *IR-96-32 (master's thesis)*, Leiden University, 1996.

[2] M. H. ter Beek, Simple eco-grammar systems with prescribed teams. To appear in *Grammatical Models of Multi-Agent Systems*, Gordon and Breach, London, 1997.

[3] M. H. ter Beek, Teams in grammar systems: sub-context-free cases. To appear in *Developments in Regulated Rewriting and Grammar Systems, Lecture Notes in Computer Science* (1997).

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

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

[6] E. Csuhaj-Varjú and J. Kelemen, Cooperating grammar systems: a syntactical framework for the blackboard model of problem solving. In *Proc. AI and information-control systems of robots '89* (I. Plander, ed.), North-Holland Publ. , 1989, 121 - 127.

[7] E. Csuhaj-Varjú and Gh. Păun, Limiting the size of teams in cooperating grammar systems. *Bulletin EATCS* 51 (1993), 198 - 202.

[8] E. Csuhaj-Varjú, Grammar systems: a framework for natural language generation. In *Mathematical Aspects of Natural and Formal Languages (Gh. Păun, ed.), World Scientific Series in Computer Science* 43 (1994), World Scientific, Singapore, 63 - 78.

[9] E. Csuhaj-Varjú, Eco-grammar systems: recent results and perspectives. In [25] (1995), 79 - 103.

[10] E. Csuhaj-Varjú, Generalized eco-grammar systems: a framework for natural language generation. In *Lenguajes Naturales Y Lenguajes Formales XII (C. Martin-Vide,ed.)*, PPU, Barcelona, 1996, 13-27.

[11] J. Dassow, Cooperating grammar systems (definitions, basic results, open problems). In [25] (1995), 40 - 52.

[12] J. Dassow, J. Kelemen and Gh. Păun, On parallelism in colonies. *Cybernet. Systems* 24 (1993), 37 - 49.

[13] J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, 1989.

[14] H. Fernau, Membership for 1-limited ET0L languages is not decidable. *J. Inform. Process. Cybern. EIK* 30 (1994), 191 - 211.

[15] H. Fernau, On unconditional transfer. *Proceedings of the MFCS'96*, *Lecture Notes in Computer Science* 1113, Springer-Verlag, Berlin, 1996, 348 - 359.

[16] H. Fernau, Scattered context grammars with regulation. *Ann. Univ. Bucureşti, Math-Informatics Series* 45, 1 (1996), 41 - 50.

[17] R. Freund and Gh. Păun, A variant of team cooperation in grammar systems. *J. UCS* 1, 2 (1995), 105 - 130.
http://hyperg.iicm.tu-graz.ac.at

[18] I. Friš, Grammars with partial ordering of the rules. *Inform. Control* 12 (1968), 412 - 425. Correction in *Inform. Control* 14 (1969), 5.

[19] D. Hauschildt and M. Jantzen, Petri net algorithms in the theory of matrix grammars, *Acta Informatica* 31 (1994), 719 - 728.

[20] L. Kari, A. Mateescu, Gh. Păun and A. Salomaa, Teams in cooperating grammar systems, *J. Exper. Th. AI* 7 (1995), 347 - 359.

[21] O. Mayer, Some restricted devices for context-free languages. *Inform. Control* 20 (1972), 69 - 92.

[22] V. Mitrana, Hybrid cooperating distributed grammar systems. *Computers and AI* 2 (1993), 83 - 88.

[23] Gh. Păun, On the generative capacity of hybrid CD grammar systems, *J. Inform. Process. Cybern. EIK* 30, 4 (1994), 231 - 244.

[24] Gh. Păun, Grammar systems: a grammatical approach to distribution and cooperation. In *Automata, Languages and Programming; 22nd International Colloquium, ICALP'95, Szeged, Hungary, Lecture Notes in Computer Science* 944 (1995), 429 - 443.

[25] *Artificial Life: Grammatical Models* (Gh. Păun, ed.), Black Sea Univ. Press, Bucharest, Romania, 1995.

[26] Gh. Păun and G. Rozenberg, Prescribed teams of grammars. *Acta Informatica* 31 (1994), 525 - 537.

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

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

[29] E. D. Stotskii, Control of the conclusion in formal grammars. *Problems of Information Transmission* 7, 3 (1971, translated 1973), 257 - 270.

Test Tube Systems or How to Bake a DNA Cake

Rudolf FREUND [*] Franziska FREUND [†]

Abstract

We introduce various general models for test tube systems which not only are a theoretical basis for the different test tube systems used for practical applications (confer to [1], [2], [3], [12]), but also cover different theoretical models to be found in literature, e.g. the test tube systems based on the splicing operation introduced in [4] as well as test tube systems based on the operations of cutting and recombination introduced in [9]. In test tube systems specific operations are applied to the objects in their components (test tubes) in a distributed and parallel manner; the results of these computations are redistributed according to specific input and/or output filters. We investigate relations between the different models of test tube systems introduced in this paper and show how the results presented in [4] and [9] fit into our general framework. Moreover, we investigate the computational power of test tube systems with context-free productions and regular filters.

1 Introduction

Test tube systems were introduced as biological computer systems based on DNA molecules ([1], [2], [3], [12]), and the practical solution of various problems (e.g. even of NP complete problems like the Hamiltonian path problem in [1]) with such systems was described. The theoretical features of test tube systems based on the splicing operation were investigated in [4]; in [9] test tube systems based on the operations of cutting and recombination were explored; in both cases, these test tube systems were shown to have the computational power of Turing machines.

Most of the test tube systems to be found in literature have the following common features: in the components (test tubes) of the systems specific operations are applied to the objects in the tubes in a distributed and parallel manner; the results of these computations are redistributed according to specific output and/or input filters which only allow specific parts of the contents of a tube to pass over to other test tubes. As it was shown in the theoretical papers mentioned above ([4], [9]), even very restricted kinds of such filters testing for the existence respectively non-existence of specific symbols respectively markings allow for reaching the

[*] Institut für Computersprachen, Technische Universität Wien Resselgasse 3, A-1040 Wien, Austria. Email: rudi@logic.tuwien.ac.at

[†] Gymnasium der Schulbrüder, Strebersdorf, Anton Böck-Gasse 37, A-1215 Wien, Austria. Email: freund@strebersdorf.ac.at

computational power of Turing machines. The computational universality of these specific variants of test tube systems was proved in these papers, too.

In [5] and [6] a general framework for describing networks of language identifying devices (networks of language processors) was introduced and investigated. In this paper we shall restrict ourselves to introduce various general models for test tube systems. We investigate relations between these models of test tube systems and also consider their computational power with respect to the complexity of the output/input relations and the filters we use. Moreover, we show how the results presented in [4] and [9] fit into the general framework presented in this paper.

In the second section we start with defining the notions from formal language theory needed in this paper; we introduce the formal definitions for the general models of test tube systems to be investigated in this paper as well as the notions of test tube systems based on the splicing operation ([4]) and the test tube systems based on the operations of cutting and recombination ([9]); moreover, we give some examples illustrating the notions of test tube systems and we show how the test tube systems based on the splicing operation ([4]) and the test tube systems based on the operations of cutting and recombination ([9]) can be interpreted in the framework introduced in this paper. In the third section we investigate some characteristic features of the different general models of test tube systems and also elaborate some specific results for test tube systems which use context-free productions in the test tubes and regular filters for redistribution. A short summary of the results obtained in this paper and an overview of future research topics conclude the paper.

2 Definitions and Examples

In this section we define some notions from formal language theory and recall the definitions of splicing schemes (H-schemes; see [4], [7], [13]) and of cutting/recombination schemes (CR schemes; confer to [8]). Moreover, we introduce the general definitions of test tube systems and give some explanatory examples.

2.1 Formal language theory prerequisites

In this subsection we only define some notions from formal language theory that we shall need in this paper. For general formal language theory prerequisites we refer to [16].

The free monoid generated by the alphabet V is denoted by V^*, its elements are called *strings* or *words* over V; λ is the empty string, $V^+ = V^* \setminus \{\lambda\}$.

A *grammar scheme* γ is a triple (V_N, V_T, P), where V_N is a (finite) alphabet of *non-terminal symbols;* V_T is a (finite) alphabet of *terminal symbols* with $V_N \cap V_T = \emptyset$; P is a (finite) set of *productions* of the form (α, β), where $\alpha \in (V_N \cup V_T)^+$ and $\beta \in (V_N \cup V_T)^*$. For two words $x, y \in (V_N \cup V_T)^+$, the *derivation relation* $\vdash_\gamma$ is defined if and only if $x = u\alpha v$ and $y = u\beta v$ for some production $(\alpha, \beta) \in P$ and two strings $u, v \in (V_N \cup V_T)^*$; we then also write $x \vdash_\gamma y$. The reflexive and transitive closure of the relation $\vdash_\gamma$ is denoted by $\vdash_\gamma^*$.

A *grammar* G is a quadruple (V_N, V_T, P, S), where $\gamma = (V_N, V_T, P)$ is a grammar scheme and $S \in V_N$; in a more general way, we can also take $S \in (V_N \cup V_T)^+$. The λ-free language generated by G is $L(G) = \{w \in V_T^+ \mid S \vdash_\gamma^* w\}$.

The grammar G, $G = (V_N, V_T, P, S)$, as well as the corresponding grammar scheme (V_N, V_T, P) is called *context-free,* if every production in P is of the form (A, w), where $A \in V_N$ and $w \in (V_N \cup V_T)^*$; G is called *regular,* if every production in P is of the form (A, w), where $A \in V_N$ and $w \in V_T^* V_N \cup V_T^+$.

The family of (λ-free) languages over V_T generated by arbitrary, context-free, and regular grammars is denoted by $ENUM(V_T)$, $CF(V_T)$, and $REG(V_T)$, respectively, and the family of finite (λ-free) languages over V_T is denoted by $FIN(V_T)$. The corresponding families of languages over arbitrary terminal alphabets are denoted by $ENUM$, CF, REG, and FIN, respectively. By REG^+ we denote the family of regular languages of the form W^+ for some finite set W.

A *grammar scheme* $\gamma_U(V_T)$ with $\gamma_U(V_T) = (V_N, V_T, P)$ is called *universal (for V_T)* if for every $L \in ENUM(V_T)$ there exists a word A_L such that the grammar G_L with $G_L = (V_N, V_T, P, A_L)$ generates L. One of the important results of formal language theory is that for every V_T such a universal grammar $\gamma_U(V_T)$ exists.

2.2 Splicing schemes and cutting/recombination schemes

We now recall the definitions of splicing schemes (H-schemes; see [4], [7], [13]) and of cutting/recombination schemes (CR-schemes; confer to [8]).

As the empty word has no meaningful representation in nature, λ is not considered to be an object we have to deal with; as for grammars above, also in the following only mechanisms for generating λ-free languages will be considered (all the definitions we shall give have been adapted in a suitable manner).

Definition 1. A *splicing scheme (H-scheme)* is a pair σ, $\sigma = (V, R)$, where V is an alphabet and $R \subseteq V^*\#V^*\$V^*\#V^*$; $\#, \$$ are special symbols not in V. R is the set of *splicing rules.* For $x, y, z, w \in V^+$ and $r = u_1\#u_2\$u_3\#u_4$ in R we define $(x, y) \vdash_r (z, w)$ if and only if $x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$, and $z = x_1u_1u_4y_2$, $w = y_1u_3u_2x_2$, for some $x_1, x_2, y_1, y_2 \in V^*$.

For any language $L \subseteq V^+$, we write

$$\sigma(L) = \{z \in V^+ \mid (x,y) \vdash_r (z,w) \text{ or } (x,y) \vdash_r (w,z), \text{ for some } x,y \in L, r \in R\},$$

and we define $\sigma^*(L) = \bigcup_{i \geq 0} \sigma^i(L)$,where
$\sigma^0(L) = L$, $\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L))$ for $i \geq 0$.

An *extended H-system* (or *extended splicing system*) is a quadruple γ, $\gamma = (V, V_T, A, R)$, where $V_T \subseteq V$ is the set of terminal symbols and A is the set of axioms. The *language generated* by the extended H-system γ is defined by $L(\gamma) = \sigma^*(A) \cap V_T^+$.

Definition 2. A *cutting/recombination scheme* (or a *CR-scheme*) is a quadruple $\sigma = (V, M, C, R)$, where V is a finite alphabet; M is a finite set of *markings;* V and M are disjoint sets; C is a set of *cutting rules* of the form $u\#l\$m\#v$, where

$u \in V^* \cup MV^*$, $v \in V^* \cup V^*M$, and $m, l \in M$, and $\#, \$$ are special symbols not in $V \cup M$; $R \subseteq M \times M$ is the recombination relation representing the *recombination rules*. Cutting and recombination rules are applied to objects from $O(V, M)$, where we define

$$O(V, M) = V^+ \cup MV^* \cup V^*M \cup MV^*M.$$

For $x, y, z \in O(V, M)$ and a cutting rule $c = u\#l\$m\#v$ we define $x \vdash_c (y, z)$ if and only if for some $\alpha \in V^* \cup MV^*$ and $\beta \in V^* \cup V^*M$ we have $x = \alpha uv\beta$ and $y = \alpha ul$, $z = mv\beta$. For $x, y, z \in O(V, M)$ and a recombination rule $r = (l, m)$ from R we define $(x, y) \vdash_r z$ if and only if for some $\alpha \in V^* \cup MV^*$ and $\beta \in V^* \cup V^*M$ we have $x = \alpha l$, $y = m\beta$, and $z = \alpha\beta$. For a CR-scheme $\sigma = (V, M, C, R)$ and any language $L \subseteq O(V, M)$ we write

$$\begin{aligned}\sigma(L) &= \{y \mid x \vdash_c (y, z) \text{ or } x \vdash_c (z, y) \text{ for some } x \in L,\ c \in C\} \cup \\ &\quad \{z \mid (x, y) \vdash_r z \text{ for some } x, y \in L,\ r \in R\};\end{aligned}$$

$\sigma^*(L)$ and $\sigma^i(L)$ for $i \geq 0$ are defined in a similar way as for splicing schemes.

An *extended CR-system* is a sextuple γ, $\gamma = (V, M, V_T, A, C, R)$, where $V_T \subseteq V$ is the set of terminal symbols, A is the set of axioms, and (V, M, C, R) is the underlying CR-scheme. The *language generated* by the extended CR-system γ is defined by $L(\gamma) = \sigma^*(A) \cap V_T^+$. □

Thus $\sigma(L)$ contains all objects obtained by applying one cutting or one recombination rule to objects from L; $\sigma^*(L)$ is the smallest subset of $O(V, M)$ that contains L and is closed under the cutting and recombination rules of σ. $L(\gamma)$ is the set of all terminal words that can be obtained from the axioms by an arbitrary number of cuttings and recombinations.

There is a close relationship between CR-schemes and splicing schemes (H-schemes): For instance, applying the splicing rule $u_1\#u_2\$u_3\#u_4$ to two strings $x_1u_1u_2x_2$ and $y_1u_3u_4y_2$ yields the two strings $x_1u_1u_4y_2$ and $y_1u_3u_2x_2$ which corresponds to cutting the strings $x_1u_1u_2x_2$ and $y_1u_3u_4y_2$ into the strings $x_1u_1[m]^+$, $[m]^- u_2x_2$ and $y_1u_3[m]^+$, $[m]^- u_4y_2$ by using the cutting rules $u_1\#[m]^+\$[m]^-\#u_2$ and $u_3\#[m]^+\$[m]^-\#u_4$ and recombining them immediately by applying the recombination rule $\left([m]^+, [m]^-\right)$ in a crosswise way.

In [13] it was shown that H-systems with a finite set of axioms and a finite set of splicing rules characterize REG, whereas with a regular set of splicing rules we obtain $ENUM$. In [7] it was proved that by adding specific control mechanisms like multisets or context conditions (permitting respectively forbidden contexts) to extended H-systems with a finite number of axioms and a finite number of splicing rules again the computational power of Turing machines or arbitrary grammars can be obtained. Similar results for CR-systems were proved in [8].

2.3 Test tube systems

In this section we introduce several general models of test tube systems (confer to [2], [3], [4], [12] for practical implementations). The idea of test tube systems is to describe computational devices where the computations in each test tube are based on specific operations and where any computation is done in a distributed way. As a communication step the resulting contents of the test tubes then is redistributed according to specific constraints, i.e. the contents of each test tube is distributed to all test tubes according to specific output and input filters again, whereas the rest remains in the test tube. These ideas have already be formalized for the splicing operation in [4] and for the operations of cutting and recombination in [9].

Definition 3. A *test tube system with output and input filters* (a *TTSOI* for short) σ is a sextuple (B, n, A, ρ, O, I), where

1. B is a set of *objects;*
2. n, $n \geq 1$, is the number of test tubes in σ;
3. $A = (A_1, ..., A_n)$, where A_i is a set of *axioms,* which are elements from B, $1 \leq i \leq n$;
4. ρ is a sequence $(\rho_1, ..., \rho_n)$ of sets of *test tube operations,* where ρ_i contains specific operations for the test tube T_i, $1 \leq i \leq n$;
5. $O = (O_1, ..., O_n)$, where $O_i \subseteq B$ is the *output filter* for the test tube T_i, $1 \leq i \leq n$;
6. $I = (I_1, ..., I_n)$, where $I_i \subseteq B$ is the *input filter* for the test tube T_i, $1 \leq i \leq n$.

In order to indicate the number n of test tubes, we also call σ a TTSOI$_n$.

The computations in the system σ run as follows: At the beginning of the computation the axioms are distributed over the n test tubes according to A, i.e. test tube T_i starts with A_i. Now let L_i be the contents of test tube T_i at the beginning of a derivation step. Then in each test tube the rules of T_i operate on L_i, i.e. we obtain $\rho_i^* (L_i)$. The next substep is the redistribution of the $\rho_i^* (L_i)$ over all test tubes according to the corresponding output and input filters. From $\rho_i^* (L_i)$ only the part $(\rho_i^* (L_i) \cap O_i) \cap I_j$ that passes the output filter O_i as well as the input filter I_j is distributed to the test tube T_j, $1 \leq j \leq n$, whereas the rest

$$\rho_i^* (L_i) \setminus \left(\bigcup_{1 \leq j \leq n} ((\rho_i^* (L_i) \cap O_i) \cap I_j) \right)$$

remains in T_i. The final result of the computations in σ consists of all objects from B that can be extracted from the *final test tube* T_1 via the ouput filter O_1.

More formally, an *instantaneous description* (ID for short) of the system σ is an n-tuple $(L_1, ..., L_n)$ with $L_i \subseteq B$, $1 \leq i \leq n$, where L_i describes the contents of test tube T_i at the beginning of a derivation step. The initial ID is $(A_1, ..., A_n)$, i.e. at time $t = 0$ the test tubes T_i contain the axioms A_i. Let $(L_1 (t), ..., L_n (t))$

denote the ID at time t; then one derivation step with the system σ yields the ID $(L_1(t+1), ..., L_n(t+1))$, where

$$\begin{aligned} L_i(t+1) &= \left(\bigcup_{1\leq j\leq n}\left(\rho_j^*\left(L_j(t)\cap O_j\right)\cap I_i\right)\right)\cup \\ &\left(\rho_i^*\left(L_i(t)\right)\setminus\left(\bigcup_{1\leq j\leq n}\left(\left(\rho_i^*\left(L_i(t)\right)\cap O_i\right)\cap I_j\right)\right)\right). \end{aligned}$$

We also write $(L_1(t), ..., L_n(t)) \vdash_\sigma (L_1(t+1), ..., L_n(t+1))$. The language generated by the system σ, $L(\sigma)$, then is defined by $L(\sigma) = \bigcup_{t=0}^{\infty}(L_1(t)\cap O_1)$. Moreover, we say that σ is of type (F_1, F_2, F_3, F_4), if $A_i \in F_1$, $\rho_i \in F_2$, $O_i \in F_3$, and $I_i \in F_4$ for all i with $1 \leq i \leq n$. □

Definition 4. A *test tube system with input filters* (a *TTSI* for short) σ of type (F_1, F_2, F_4) is a quintuple (B, n, A, ρ, I), where $(B, n, A, \rho, (B, ..., B), I)$ is the corresponding TTSOI of type $(F_1, F_2, \{B\}, F_4)$. A *test tube system with output filters* (a *TTSO* for short) σ of type (F_1, F_2, F_3) is a quintuple (B, n, A, ρ, O), where $(B, n, A, \rho, O, (B, ..., B))$ is the corresponding TTSOI of type $(F_1, F_2, F_3, \{B\})$. In order to indicate the number n of test tubes, we also call σ a TTSI$_n$ and a TTSO$_n$, respectively. □

We should like to mention that in general the TTSOI $(B, n, A, \rho, (B, ..., B), I)$ corresponding with a TTSI (B, n, A, ρ, I) of type (F_1, F_2, F_4) need not be a TTSOI of type (F_1, F_2, F_4, F_4), because B need not be an element of F_4.

Remark 1. The reader should observe that we are not dealing with multisets in this paper; hence we assume that every object in any test tube is available in an unbounded number. Moreover, in the phase of redistribution every object x from the test tube T_i that passes the output filter O_i is distributed (in an unbounded number) to each test tube T_j the input filter of which allows x to pass. In some sense this corresponds to an intermediate step which in practice is called *amplification*, e.g. in test tubes working with DNA strands (and the operation of splicing) copies can be made by applying the polymerase chain reaction (see [3]). Moreover, it would be a more practical assumption that instead of $\rho_i^*(L_i)$ any arbitrary (finite) subset of $\rho_i^*(L_i)$ could evolve in the test tube T_i during a computation period. Then only this subset would be distributed to all test tubes according to the input filters. In fact, in most cases this would still allow us to generate all desired objects, although it would never be clear, when these objets would evolve. In a practical implementation the number and the size of objects that can be generated also depends on the amount of original material of axioms we take at the beginning. Moreover, if parts of (the subset of) $\rho_i^*(L_i)$ are to be redistributed over different test tubes it is only necessary to assume that any allowed distribution of the whole material will possibly happen; in practical implementations of test tube systems the intermediate amplification (see [2], [3], [12]) of the material may already guarantee that enough material is distributed to all the possible test tubes. □

A minimal requirement on the feasability of the input filters I_i and the output filters O_j is their recursiveness, i.e. we demand that it is decidable whether an object from B can pass a filter or not.

The following example shows how under these constraints every recursive language can be generated by a large class of TTSO_1:

Example 1. Let $L \subseteq V^+$ be an arbitrary recursive language and let σ_L be the TTSO_1 $\sigma_L = (B, 1, (A), (\rho), (L))$ such that $\rho^*(A) \supseteq L$. Then we obtain $(A) \vdash_{\sigma_L} (\rho^*(A) \cap L) = (L)$ and therefore $L(\sigma) = (A \cap L) \cup (\rho^*(A) \cap L) = L$.

Hence, for any family of languages F_3 with $F_3 \subseteq REC$, a language $L \subseteq F_3$ can be generated by a TTSO_1 of type (F_1, F_2, F_3) if F_1 contains a set A such that $\rho^*(A) \supseteq L$ for some $\rho \in F_2$. □

Definition 5. A *CR-TTSOI* σ is a TTSOI $(O(V,M), n, A, \rho, O, I)$, where $\rho = (\rho_1, ..., \rho_n)$, $\rho_i = C_i \cup R_i$, $1 \leq i \leq n$, and $\sigma_i = (V, M, C_i, R_i)$ is a CR-scheme. In order to emphasize that σ is a CR-TTSOI, we shall also write (C_i, R_i) for ρ_i instead of $C_i \cup R_i$. An H-TTSOI σ is a TTSOI (V^+, n, A, ρ, O, I), where $\sigma_i = (V, \rho_i)$, $1 \leq i \leq n$, is an H-scheme. A G-TTSOI σ is a TTSOI $(W(V_N, V_T), n, A, \rho, O, I)$, where V_N and V_T are disjoint alphabets, $W(V_N, V_T)$ denotes $(V_N \cup V_T)^+$, and $\sigma_i = (V_N, V_T, \rho_i)$, $1 \leq i \leq n$, is a grammar scheme; if every grammar scheme σ_i, $1 \leq i \leq n$, is context-free (regular), then also σ is called context-free (regular). □

Remark 2. The notation $W(V_N, V_T)$ in a G-TTSOI σ, $\sigma = (W(V_N, V_T), n, A, \rho, O, I)$, allows us to distinguish the non-terminal symbols in V_N and the terminal symbols in V_T; σ is considered to work "correctly" only if $L(\sigma) \subseteq V_T^+$. In a similar manner for a CR-TTSOI σ, $\sigma = (O(V,M), n, A, \rho, O, I)$, we demand $L(\sigma) \subseteq V^+$. □

We now exhibit an example of a regular $\mathrm{G\text{-}TTSO}_7$ of type (FIN, FIN, REG^+) which generates a non-context-free language:

Example 2. Let $\sigma = (W(V_N, V_T), 7, A, \rho, O)$ be the $\mathrm{G\text{-}TTSO}_7$ with

$V_N = \{X, Y\}$, $V_T = \{a, b\}$,
$A = (\emptyset, \{XX\}, \{XX\}, \{XX\}, \{XX\}, \emptyset, \emptyset)$,
$\rho = (\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7)$, $O = (O_1, O_2, O_3, O_4, O_5, O_6, O_7)$.
$\rho_1 = \emptyset$, $\rho_2 = \{(X, a), (Y, a)\}$, $\rho_3 = \{(X, b), (Y, b)\}$,
$\rho_4 = \{(X, aY)\}$, $\rho_5 = \{(X, bY)\}$, $\rho_6 = \{(Y, aX)\}$, $\rho_7 = \{(Y, bX)\}$,
$O_1 = O_2 = O_3 = \{a, b\}^+$, $O_4 = O_5 = \{a, b, Y\}^+$, $O_6 = O_7 = \{a, b, X\}^+$.

The generation of the words ww in this G-TTSO briefly can be described in the following way:

From arbitrary words of the forms $uXuX$, $u \in \{a, b\}^*$, and $vYvY$, $v \in \{a, b\}^+$, respectively, by the productions in ρ_2 we obtain $uaua$ and $vava$, respectively, whereas by the productions in ρ_3 we obtain $ubub$ and $vbvb$, respectively, i.e. we obtain terminal words from $\{a, b\}^+$, which then can be extracted from T_1 as terminal results of the computations in σ. By the corresponding productions in ρ_4, ρ_5, ρ_6, ρ_7, the length of the current strings is prolongued by one more symbol in a synchronized way, because only the strings of the forms $uaYuaY$, $ubYubY$, $vaXvaX$, and $vbXvbX$, respectively, can pass the corresponding output filters O_i of the test tubes T_i, $4 \leq i \leq 7$. These observations show that $L(\sigma) = \left\{ ww \mid w \in \{a, b\}^+ \right\}$. □

For CR-TTSOI the following types of filters are suitable:

Definition 6. A subset of $O(V,M)$ is called a *simple* $(V,M)_2$*-filter* if it equals

1. V^+ or
2. $\{m\}\, V^*$ for some $m \in M$ or
3. $V^*\, \{m\}$ for some $m \in M$ or
4. $\{m\}\, V^*\, \{n\}$ for some $m, n \in M$.

A simple $(V,M)_2$-filter is called a *simple* $(V,M)_1$*-filter*, if it is not of the form $\{m\}\, V^*\, \{n\}$. Any finite union of simple $(V,M)_i$-filters, $i \in \{1,2\}$, is called a $(V,M)_i$*-filter*; the families of $(V,M)_i$-filters and simple $(V,M)_i$-filters for arbitrary V, M are denoted by CRF_i and $CRSF_i$, respectively. □

The proof of the following result is obvious from the definitions and therefore omitted:

Lemma 1. The union and the intersection of two $(V,M)_i$-filters again is a $(V,M)_i$-filter, $i \in \{1,2\}$. Moreover, $O(V,M)$ is a $(V,M)_2$-filter, but not a $(V,M)_1$-filter.

The distribution of the contents of a test tube over all test tubes of the system not only gives rise to theoretical problems (for obtaining filters of a complexity as low as possible) but also to practical problems ("waste" of the material that is put into test tubes where on one hand it cannot be processed or used any more and on the other hand it nonetheless has to remain forever). Hence, a more natural and realistic scenario is to assume that the contents of the test tubes is only distributed to selected test tubes that are prescribed from the beginning. In fact, most of the test tube systems to be found in literature work in such a manner, i.e. *programs* how to redistribute the contents of test tubes are described (see [1], [3], [12]).

A formalization of these ideas discussed above leads to the following definition:

Definition 7. A *test tube system with prescribed output/input relations* (a *TTSPOI* for short) σ is a quintuple (B, n, A, ρ, D), where

1. B is a set of *objects;*
2. n, $n \geq 1$, is the number of test tubes in σ;
3. $A = (A_1, ..., A_n)$ is a sequence of sets of *axioms,* where $A_i \subseteq B$, $1 \leq i \leq n$;
4. ρ is a sequence $(\rho_1, ..., \rho_n)$ of sets of *test tube operations*, where ρ_i contains specific operations for the test tube T_i, $1 \leq i \leq n$;
5. D is a (finite) set of *prescribed output/input relations* between the test tubes in σ of the form (i, F, j), where $1 \leq i \leq n$, $1 \leq j \leq n$ and F is a (recursive) subset of B; F is called a filter between the test tubes T_i and T_j.

In order to indicate the number of test tubes, we also say that σ is a TTSPOI$_n$.

The computations in the system σ run as follows: At the beginning of the computation the axioms are distributed over the n test tubes according to A, i.e. test tube T_i starts with A_i. Now let L_i be the contents of test tube T_i at the beginning of a derivation step. Then in each test tube the rules of ρ_i operate on L_i, i.e. we obtain $\rho_i^*(L_i)$. The next substep is the redistribution of the $\rho_i^*(L_i)$ over all test tubes according to the corresponding output/input relations from D, i.e. if $(i,F,j) \in D$ then the test tube T_j from $\rho_i^*(L_i)$ gets $\rho_i^*(L_i) \cap F$, whereas the rest of $\rho_i^*(L_i)$ that cannot be distributed to other test tubes remains in T_i. The final result of the computations in σ consists of all objects from B that can be extracted from the *final test tube* T_1 (hence usually we shall assume $F = \emptyset$ for all $(1,F,j) \in D$).

More formally, an *instantaneous description* (ID for short) of the system σ is an n-tuple $(L_1, ..., L_n)$ with $L_i \subseteq B$, $1 \leq i \leq n$, where L_i describes the contents of test tube T_i at the beginning of a derivation step. The initial ID is $(A_1, ..., A_n)$, i.e. at time $t = 0$ test tubes T_i contain the axioms A_i. Let $(L_1(t), ..., L_n(t))$ denote the ID at time t; then one derivation step with the system σ yields the ID $(L_1(t+1), ..., L_n(t+1))$, where

$$L_i(t+1) = \left(\bigcup_{(j,F,i) \in D} (\rho_j^*(L_j(t)) \cap F) \right) \cup \left(\rho_i^*(L_i(t)) \setminus \bigcup_{(i,F,j) \in D} (\rho_i^*(L_i(t)) \cap F) \right).$$

We also write $(L_1(t), ..., L_n(t)) \vdash_\sigma (L_1(t+1), ..., L_n(t+1))$. The language generated by the system σ, $L(\sigma)$, is defined by $L(\sigma) = \bigcup_{t=0}^{\infty} L_1(t)$. Moreover, we say that σ is of type (F_1, F_2, F_3), if $A_i \in F_1$, $\rho_i \in F_2$ for all i with $1 \leq i \leq n$, and $F \in F_3$ for all F with $(i,F,j) \in D$ for some i, j with $1 \leq i \leq n$, $1 \leq j \leq n$. □

Definition 8. A *CR-TTSPOI* σ is a TTSPOI $(O(V,M), n, A, \rho, D)$, where $\rho = (\rho_1, ..., \rho_n)$, $\rho_i = (C_i, R_i)$, $1 \leq i \leq n$, and $\sigma_i = (V, M, C_i, R_i)$ is a CR-scheme. An H-TTSPOI σ is a TTSPOI (V^+, n, A, ρ, D) where $\sigma_i = (V, \rho_i)$, $1 \leq i \leq n$, is an H-scheme. A G-TTSPOI σ is a TTSPOI $(W(V_N, V_T), n, A, \rho, D)$, where $\sigma_i = (V_N, V_T, \rho_i)$, $1 \leq i \leq n$, is a grammar scheme; if every grammar scheme σ_i, $1 \leq i \leq n$, is context-free (regular), then also σ is called context-free (regular). □

Remark 3. As already stated in Remark 2 for CR-TTSOI and G-TTSOI, respectively, also for a CR-TTSPOI σ, $\sigma = (O(V,M), n, A, \rho, D)$, we demand $L(\sigma) \subseteq V^+$ and for a G-TTSPOI σ, $\sigma = (W(V_N, V_T), n, A, \rho, D)$, we demand $L(\sigma) \subseteq V_T^+$.

The following results were established in [9]:

Proposition 1. For every $L \in ENUM$ we can construct a CR-TTSI of type (FIN, FIN, CRF_1) which generates L.

Proposition 2. For every $L \in ENUM$ we can construct a CR-TTSPOI$_4$ of type (FIN, FIN, CRF_2) which generates L.

For the CR TTSI in Proposition 1 it is an open question whether the number of test tubes needed for generating arbitrary recursively enumerable languages can be bounded or not.

The following result was proved in [4]:

Proposition 3. For every $L \in ENUM(V_T)$ we can construct an H-TTSI$_{8+card(V_T)}$ of type (FIN, FIN, REG^+) which generates L.

3 Results

In the first part of this section we elaborate some general relations between the different models of test tube systems we introduced in the previous section; these results even hold true for arbitrary objects and for arbitrary operations used in the test tubes. In the second part of this section we shall prove some specific results for test tube systems working on strings, e.g. we shall show how every recursively enumerable string language can be generated by test tube systems using context-free productions and a very restricted form of regular filters only.

3.1 General results for test tube systems

In this subsection we show some general results for the different models of test tube systems introduced in the previous section; these results neither depend on the operations used in the test tubes nor on the objects we consider. Moreover, we give some applications of these general results for CR test tube systems.

For every TTSOI we can easily construct an equivalent TTSPOI generating the same language:

Lemma 2. Let $\sigma = (B, n, A, \rho, O, I)$ be a TTSOI$_n$ of type (F_1, F_2, F_3, F_4) and let F_0 be a set containing $L(\sigma)$. Then the TTSPOI$_{n+1}$ $\sigma' = (B, n+1, (\emptyset, \rho_1, ..., \rho_n), (\emptyset, A_1, ..., A_n), D)$ with

$$D = \{(i+1, O_i \cap I_j, j+1) \mid 1 \leq i, j \leq n\} \cup \{(2, F_0 \cap O_1, 1)\}$$

generates the same language as σ, i.e. $L(\sigma') = L(\sigma)$. Let F_1 and F_2 contain the empty set and denote $\sqcap(F_3, F_4) = \{X \cap Y \mid X \in F_3 \wedge Y \in F_4\}$; then σ' is a TTSPOI$_{n+1}$ of type (F_1, F_2, F_5) for every family F_5 with $F_5 \supseteq \{F_0 \cap O_1\} \cup \sqcap(F_3, F_4)$.

Proof. The components (test tubes) T'_{i+1}, $1 \leq i \leq n$, in σ' work in the same way as the corresponding test tubes T_i in σ, because by definition they contain the same rules, i.e. $\rho'_{i+1} = \rho_i$. We also start with the desired axioms A_j in each test tube T'_{j+1}, $1 \leq j \leq n$. The output/input relations $(i+1, O_i \cap I_j, j+1)$, $1 \leq i \leq n$, $1 \leq j \leq n$, guarantee that the test tubes T'_{i+1} in σ' are distributed in the same way as the test tubes T_i in σ after each computation step. The test tube T'_1 is only needed to extract the final objects in σ' in a similar way as by extracting these strings from T_1 in σ. In sum we obtain

$$\begin{aligned}(A_1, ..., A_n) &\vdash_\sigma^* (L_1(t), ..., L_n(t)) \\ &\vdash_\sigma (L_1(t+1), ..., L_n(t+1))\end{aligned}$$

if and only if

$$\begin{array}{lll}(\emptyset, A_1, ..., A_n) & \vdash_{\sigma'}^{*} & (L_0(t), L_1(t), ..., L_n(t)) \\ & \vdash_{\sigma'} & (L_0(t) \cup (L_1(t) \cap (F_0 \cap O_1)), L_1(t+1), ..., L_n(t+1)),\end{array}$$

which immediately yields

$$\begin{array}{rl} L(\sigma') = & \bigcup_{t=0}^{\infty} L_0(t) = \emptyset \cup \bigcup_{t=0}^{\infty} (L_1(t) \cap (F_0 \cap O_1)) = \\ & (\bigcup_{t=0}^{\infty} L_1(t) \cap O_1) \cap F_0 = L(\sigma) \cap F_0 = L(\sigma). \end{array}$$

□

Corollary 1. For every $L \in ENUM(V_T)$ we can construct a CR-TTSPOI of type (FIN, FIN, CRF_1) which generates L.

Proof. From Proposition 1 we know that for L we can construct a CR-TTSOI $\sigma = (O(V,M), n, \rho, (O(V,M), ..., O(V,M)), I)$ of type $(FIN, FIN, \{O(V,M)\}, CRF_1)$ with $L(\sigma) = L$. Now take $F_0 = V^+$ (observe that $V^+ \in CRF_1$); obviously, for any $F \in CRF_1$ we have $F \cap O(V,M) = F$ and therefore $\{F_0 \cap O_1\} \cup (\sqcap(\{O(V,M)\}, CRF_1)) \subset CRF_1$; hence, we can apply Lemma 2. □

Because of Lemma 1, the following result, for instance, holds true for CR-TTSPOI of type (F_1, F_2, CRF_i), $i \in \{1,2\}$:

Lemma 3. Let $\sigma = (B, n, A, \rho, D)$ be a TTSPOI$_n$ of type (F_1, F_2, F_3). If the family of filters F_3 contains the empty set and is closed under union, then we can construct an equivalent TTSPOI$_n$ $\sigma' = (B, n, A, \rho, D')$ of the same type (F_1, F_2, F_3) such that for any two test tubes there is exactly one output/input relation (by a filter from F_3), and moreover, the derivation relations $\vdash_\sigma$ and $\vdash_{\sigma'}$ are identical.

Proof. The result is obvious by defining

$$D' = \bigcup_{1 \le i,j \le n} \left\{ (i, F_{i,j}, j) \mid F_{i,j} = \bigcup_{(i,F,j) \in D} F \right\}.$$

Observe that $F_{i,j}$ is empty if in D no output/input relation between T_i and T_j exists. □

Remark 4. If we want to use the filters $F_{i,j,1}, ..., F_{i,j,k}$ only instead of the union filter $\bigcup_{m=1}^{k} F_{i,j,m}$ between the test tubes T_i and T_j, but still do not want to have more than one connection between two test tubes, we have to add k additional test tubes $T_{i,j,1}, ..., T_{i,j,k}$, which with (time) delay one contain the same strings as T_i, and then from $T_{i,j,m}$ by the filter $F_{i,j,m}$ distribute this filtered part of T_i to T_j, i.e. $\rho_{i,j,m} = \emptyset$, and instead of the output/input relations $(i, F_{i,j,m}, j)$, $1 \le m \le k$, we have the relations $(i, F_{i,j,m}, (i,j,m))$ and $((i,j,m), F_{i,j,m}, j)$ for all m with $1 \le m \le k$. □

As an immediate consequence of these considerations, $(V, M)_i$-filters, $i \in \{1,2\}$, which by definition are finite unions of simple $(V, M)_i$-filters, in the same way can be split up into their components. Hence, Corollary 1 now can be sharpened to the following result which shows that in CR-TTSPOI we only need simple $(V, M)_i$-filters in order to obtain full generative power:

Corollary 2. For every $L \in ENUM$ we can construct a CR-TTSPOI of type $(FIN, FIN, CRSF_i)$, $i \in \{1,2\}$, which generates L.

Under specific constraints, a TTSPOI$_n$ can even be simulated by a TTSOI$_n$:

Lemma 4. Let $\sigma = (B, n, A, \rho, D)$ be a TTSPOI$_n$ of type (F_1, F_2, F_3) such that

1. the family of filters F_3 contains the empty set and is closed under union;
2. for any two test tubes T_i and T_k with $1 \leq i \leq n$, $1 \leq k \leq n$, there is exactly one output/input relation $(i, F_{i,k}, k)$ with $F_{i,k} \in F_3$;
3. for all $(i, F_{i,j}, j)$ and $(i, F_{i,k}, k)$ in D with $j \neq k$ we have $F_{i,j} \cap F_{i,k} = \emptyset$;
4. for all k with $1 \leq k \leq n$ we have $F_{1,k} = \emptyset$;

then we can construct an equivalent TTSOI$_n$ $\sigma' = (B, n, A, \rho, O, I)$ of the type (F_1, F_2, F_3, F_3) such that $L(\sigma') = L(\sigma)$.

Proof. The desired result is obvious by defining $O_i = \bigcup_{1 \leq j \leq n} F_{i,j}$ and $I_i = \bigcup_{1 \leq j \leq n} F_{j,i}$ for $2 \leq i \leq n$ as well as $O_1 = I_1 = \bigcup_{1 \leq j \leq n} F_{j,1}$. □

3.2 Some specific results for test tube systems

In this subsection we shall show how any recursively enumerable language can be generated by test tube systems with context-free productions and regular filters:

Theorem 1. For every $L \in ENUM(V_T)$ we can construct a context-free G-TTSPOI$_3$ of type (FIN, FIN, REG) which generates L.

Proof. Without loss of generality we may assume that L is given by a grammar G in Geffert normal form (see [10]), i.e. $G = (\{S, A, B, C\}, V_T, P_{cf} \cup \{ABC \to \lambda\}, S)$, where P_{cf} contains only context-free productions of the form (S, w). Now we define

$$
\begin{array}{lcl}
\sigma & = & (W(V_N', V_T), 3, (\emptyset, \{S\}, \emptyset), (\emptyset, \rho_2, \rho_3), D), \\
V_N' & = & \{S, A, B, C, A', B', C'\}, \\
V & = & \{S, A, B, C\} \cup V_T, \\
\rho_2 & = & P_{cf} \cup \{A \to A', B \to B', C \to C'\}, \\
\rho_3 & = & \{A' \to \lambda, B' \to \lambda, C' \to \lambda\}, \\
D & = & \{(2, V_T^+, 1), (2, V^* \{A'B'C'\} V^*, 3), (3, V_T^+, 1), (3, V^+ \setminus V_T^+, 2)\} \cup \\
 & & \{(1, \emptyset, 1), (1, \emptyset, 2), (1, \emptyset, 3), (2, \emptyset, 2), (3, \emptyset, 3)\}.
\end{array}
$$

Whenever an application of the only non-context-free rule $ABC \to \lambda$ has to be simulated in σ, we have to apply the productions $A \to A', B \to B', C \to C'$ in T_2 in such a manner that the resulting word can pass the filter $V^* \{A'B'C'\} V^*$, which checks the context condition; the final *execution* of the simulation is carried out by the productions $A' \to \lambda, B' \to \lambda, C' \to \lambda$ in T_3. Only terminal words are passed from T_2 and T_3 to T_1. Hence we conclude $L(\sigma) = L$. □

As the construction elaborated in the preceding proof fulfills the necessary assumptions, we immediately can apply Lemma 4, which shows that we can construct a context-free G-TTSOI$_3$ of type (FIN, FIN, REG) which generates L; yet we can even get more, i.e. we only need a context-free G-TTSI$_3$ or a G-TTSO$_3$ of type (FIN, FIN, REG) for generating L :

Corollary 3. For every $L \in ENUM(V_T)$ we can construct a context-free G-TTSI$_3$ which generates L as well as a context-free G-TTSO$_3$ of type (FIN, FIN, REG) which generates L.

Proof. In a similar way as in the proof of Theorem 1 we define the context-free G-TTSI$_3$

$$\sigma_I = (W(V'_N, V_T), 3, (\emptyset, \{S\}, \emptyset), (\emptyset, \rho_2, \rho_3), I)$$

and the context-free G-TTSO$_3$

$$\sigma_O = (W(V'_N, V_T), 3, (\emptyset, \{S\}, \emptyset), (\emptyset, \rho_2, \rho_3), O)$$

where V', ρ_2, ρ_3 are defined as in the proof of Theorem 1 as well as

$$I_1 = V_T^+, I_2 = V^+, I_3 = V^* \{A'B'C'\} V^*, \text{ and}$$
$$O_1 = V_T^+, O_2 = V_T^+ \cup V^* \{A'B'C'\} V^*, O_3 = V^+.$$

It is easy to verify that $L(\sigma_I) = L(\sigma_O) = L(\sigma) = L$. □

The results in Theorem 1 and Corollary 3 are optimal in the sense that a context-free G-TTSPOI$_2$ of type (FIN, FIN, REG) can only generate context-free languages:

Theorem 2. For every context-free G-TTSPOI$_2$ σ of type (FIN, FIN, REG), $L(\sigma) \in CF$.

Proof. Let $\sigma = (W(V_N, V_T), 2, (A_1, A_2), (\rho_1, \rho_2), D)$ be a context-free G-TTSPOI of type (FIN, FIN, REG), i.e. ρ_1 and ρ_2 contain only context-free productions, and the filters in D are regular. The elements of the first test tube must be terminal words, hence no context-free productions can be applied any more to these words, neither in the first test tube nor, after distribution according to an output/input relation $(1, F_{1,2}, 2)$, in the second test tube, hence we can assume $\rho_1 = \emptyset$ as well as $D = \{(2, F_{2,1}, 1), (2, F_{2,2}, 2), (1, \emptyset, 1), (1, \emptyset, 2)\}$, where $F_{2,1}$ and $F_{2,2}$ are regular languages. $F_{2,1}$ only has the task to extract terminal strings from the contents of the second test tube, i.e. the relation $(2, F_{2,1}, 1)$ only works as a final intersection with the regular set $F_{2,1}$. During the first derivation step, in T_2 from A_2 we obtain

$\rho_2^*(A_2)$. As $\rho_2^*(A_2) \cap F_{2,2} \subseteq \rho_2^*(A_2)$ and $\rho_2^*(\rho_2^*(A_2)) = \rho_2^*(A_2)$, in further derivation steps no additional strings can evolve in T_2. Hence, as the family of context-free languages is closed under union as well as under intersection with regular sets, we obtain $\rho_2^*(A_2) \in CF$, $L(\sigma) = \rho_2^*(A_2) \cap F_{2,1}$, and therefore $L(\sigma) \in CF$. □

Remark 5. In a similar way as above it is easy to show that for every regular G-TTSPOI$_2$ σ of type (FIN, FIN, REG), $L(\sigma) \in REG$. Obviously, for any context-free G-TTSPOI$_1$ σ of type (FIN, FIN, REG) with $\sigma = (W(V_N, V_T), (A_1), (\rho_1), \{(1, F_{1,1}, 1)\})$ we have $L(\sigma) = A_1$, because the words in A_1 must not contain non-terminal symbols; hence, only finite languages can be generated. On the other hand, the language generated by a regular respectively by a context-free G-TTSO$_1$ with $\sigma = (W(V_N, V_T), (A_1), (\rho_1), (F_1))$ is $\rho_1^*(A_1) \cap F_1$, i.e. as $REG(V_T)$ and $CF(V_T)$ are closed under union as well as under intersection with regular sets, such G-TTSO$_1$ exactly characterize $REG(V_T)$ and $CF(V_T)$, respectively. □

Remark 6. The existence of a universal grammar scheme $\gamma_U(V_T)$ for V_T and the results shown above also imply the existence of a universal context-free G-TTSI$_3$ $\sigma_U(V_T)$ for V_T, $\sigma_U(V_T) = (W(V_N', V_T), 3, (\emptyset, \emptyset, \emptyset), (\emptyset, \rho_2, \rho_3), I)$, where ρ_2 and ρ_3 contain context-free productions and the filters in I are regular, such that for every $L \in ENUM(V_T)$ the context-free G-TTSI$_3$ σ_L, $\sigma_L = (W(V_N', V_T), 3, (\emptyset, \{A_L\}, \emptyset), (\emptyset, \rho_2, \rho_3), I)$, generates L, where A_L denotes the initial word used for $\gamma_U(V_T)$ to obtain a grammar generating L. □

4 Summary and Future Research

In this paper we introduced various general models of test tube systems. We investigated several general relations between different kinds of these models and also showed some specific results, e.g. how to generate any arbitrary recursively enumerable language by a test tube system with context-free productions and a restricted type of regular filters.

Special practical variants have already been described in literature for solving very specific problems in the area of DNA computing and the construction of molecular computers based on test tubes has been considered by using different operations on the test tubes (e.g. see [1], [2], [3], [12]). In [4], test tube systems based on the splicing operation were shown to allow the construction of universal mechanisms; a similar result was shown for test tube systems based on the operations of cutting and recombination in [9]. Various other types of test tube systems based on context-free productions can also be shown to be computationally universal as we have exhibited in the previous section.

The general results proved in the first part of the preceding section also hold true for test tube systems working on other objects than strings, e.g. for circular strings, graphs, and arrays. Hence, there is a wide field of interesting problems to be considered in the future.

Acknowledgements

We gratefully appreciate fruitful discussions with Erzsébet Csuhaj-Varjú and Gheorghe Păun on some of the topics considered in this paper.

References

[1] L. M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, 226 (Nov. 1994), 1021 - 1024.

[2] L. M. Adleman, On constructing a molecular computer, manuscript, January 1995.

[3] D. Boneh, C. Dunworth, R.J. Lipton, J. Sgall, On the computational Power of DNA, *to appear.*

[4] E. Csuhaj-Varjú, L. Kari, and Gh. Păun, Test tube distributed systems based on splicing, *Computers and Artificial Intelligence,* Vol. 15 (2) (1996), 211-232

[5] E. Csuhaj-Varjú and A. Salomaa, Networks of parallel language processors, submitted.

[6] E. Csuhaj-Varjú, Networks of language processors. A survey. In: *Lenguajes Naturales Y Lenguajes Formales XII,* C. Martin-Vide, ed., PPU, Barcelona, 1996, pp. 169-189.

[7] R. Freund, L. Kari, and Gh. Păun, DNA computing based on splicing: The existence of universal computers, Techn. Report 185-2/FR-2/95, TU Wien, 1995.

[8] R. Freund and F. Wachtler, Universal systems with operations related to splicing, *Computers and Artificial Intelligence,* Vol. 15 (4) (1996), 273-294.

[9] R. Freund, E. Csuhaj-Varjú, and F. Wachtler, Test tube systems with cutting/recombination operations, Proceedings PSB'97, 1997.

[10] V. Geffert, Context-free-like forms for the phrase-structure grammars, *Proceedings MFCS'88,* Lecture Notes in Computer Science, Vol. 324, Springer-Verlag, Berlin (1988), 309 - 317.

[11] T. Head, Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology,* 49 (1987), 737 - 759.

[12] R. J. Lipton, Speeding up computations via molecular biology, *manuscript,* December 1994.

[13] Gh. Păun, Regular extended H systems are computationally universal, *J. of Automata, Languages and Combinatorics,* Vol. 1, Nr. 1 (1996), 27 - 37.

[14] P. W. K. Rothemund, A DNA and restriction enzyme implementation of Turing machines, *manuscript,* 1995.

[15] R. Siromoney, K. G. Subramanian, and V. R. Dare, Circular DNA and splicing systems, Lecture Notes in Computer Science, Vol. 654, Springer-Verlag, Berlin (1992), 260 - 273.

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

Acta Cybernetica 12 (1996) 461-472.

Natural Language Understanding: a New Challenge for Grammar Systems

Carlos MARTÍN-VIDE *

Abstract

We show the basic architecture of a natural language understanding system. Given the well- known difficulties other simple grammar formalisms find when attempting to model such an architecture, as well as the plausibility of the modular hypothesis, we advocate the suitability of complex and modular constructs like grammar systems for giving account of human language.

"Sentence processing is most plausibly modeled as a fully interactive parallel process: each word, as it is heard in the context of normal discourse, is immediately entered into the processing system at all levels of description, and is simultaneously analysed at all these levels in the light of whatever information is available at each level at that point in the processing of the sentence".

(W.D. Marslen-Wilson (1975), "Sentence perception as an interactive parallel process", *Science*, 189: 226-228)

1 Postulates

Let us begin stating some postulates in order to contextualize our paper:

1. There exists a certain undesirable gap between the communities of linguists and computer scientists, more specifically between the communities of computational linguists and formal language theoreticians. Often, linguists ignore all that is strictly beyond/outside the Chomsky hierarchy, and computer scientists don't know precisely the kind of problems linguists are interested in.

2. Formal language theoreticians show an understandable obsession to design mechanisms able to generate recursively enumerable languages. However, no natural language is so large as recursively enumerable. Linguists need formal tools endowed with a very rich internal structure, rather than with an impressive generative power.

*Research Group in Mathematical Linguistics and Language Engineering (GRLMC), Rovira i Virgili University, Pl. Imperial Tàrraco, 1, 43005 Tarragona, Spain, E-mail: cmv@astor.urv.es

3. For theories which try to become rich in applications (and we guess this is the case with grammar systems), empirical well-foundedness is as important as completeness.

4. In an initial step, one of the most essential features of a scientific theory is its metaphorical content, as different from its technical content. Grammar systems seem quite rich in this respect, and quite flexible too.

5. Grammar systems theory needs to assume and face all the complexities of natural language if it wants to be accepted as a good candidate for the solution of natural language processing problems.

6. Linguists are not as much interested in generative capacity aspects of language-theoretical models as in other basic matters like descriptive adequacy, expressiveness, naturality or computational easiness.

We are going to offer an introductory overview of natural language understanding area for non-specialists, which perhaps will help somebody to bring his/her research closer to natural language as it is regarded by current theoretical linguistics. We'll show a picture of natural language from a computational viewpoint. If one wishes the own work will become relevant for linguists, I'm sure one will share the opinion that linguists have something to say about.

2 Language and the computer

It is generally accepted that computers serve not only to process numbers but language too. Even it is a matter of public concern the idea of a machine that could communicate with people in their own language to take commands or to answer questions. In fact, many linguistic tasks, such as translation, improve if performed by a machine with a knowledge of natural language.

We are going to survey the problem of giving a computer comprehension of language. The focus will be on tasks that involve language carrying meaning, rather than those, such as speech processing, that involve only the superficial form of text without regard to its content.

Research in computational linguistics is generally taken as a branch of artificial intelligence, that part of computer science concerned with the computational simulation of intelligent human behaviour (which surely includes language understanding). Most of the natural language research in artificial intelligence has been directed implicitly or explicitly to the problem of computer understanding of language. The converse of language understanding is language production or generation. For the computer, to produce text has proved to be an even harder problem than comprehension.

Although linguists agree that human language is essentially oral, natural language understanding deals almost exclusively with a simple form of language: written text. The additional problems that arise with spoken input and output have been traditionally taken to be primarily matters of physics and engineering.

3 Understanding

3.1 What is understanding

A preliminary question needs to be posed: what means to say that a computer understands? As one believes that understanding is a somewhat subjective state that admits of degrees, it would not seem appropriate to attribute understanding to a machine. We'll agree, however, with the idea that behaviour is the key fact: if the machine always responds to sentences just as a human would in the same situation, then it can meaningfully be said that it understands the sentences.

3.2 The illusion of understanding

Real understanding is hard to achieve in practice. It is not difficult for a computer program to give us the illusion that it understands. One of the most famous examples is ELIZA system, built by Joseph Weizenbaum on 1966, which was not taken as a serious model of understanding, because it used simple scripts and tricks for the computer to keep up a conversation. It's doctor script simulated a psychotherapist. It looked out for certain key words and sentence forms, and answered with one of a few predetermined phrases for each. If the user of the program typed:

(1) I am depressed,

it might answer:

(2) I'm sorry to hear that you are depressed.

If the user's sentence included the word *mother*, the computer might say:

(3) Tell me more about your family.

If the user's sentence matched nothing at all in the script, it would either respond:

(4) What else does that bring to mind?, or

(5) Earlier, you mentioned your mother.

The illusion of understanding cannot be sustained for a long time. The computer only encourages the user to continue, but it understands nothing.

3.3 Levels of understanding

Four levels of understanding can be identified:

a) The most superficial level is that involved in message passing. If a computer is asked to:

   (6) Tell George that I'll meet him next Monday in Salgótarján,

   it needs not understand the message itself, or where Salgótarján is and so on, in order to be able to pass on the message; but it does need to determine that *him* here refers to George.

b) The second level is almost literal understanding within a very limited domain of discourse. This level is a characteristic feature of many natural language systems of the early 1990s, such as interfaces to databases.

c) The third level might be called complete understanding: a full apprehension of all aspects and nuances of the sentence. It allows to read texts and integrate the knowledge gained from them with its previous knowledge from other linguistic sources. This depth of understanding, for instance, seems necessary for unassisted machine translation.

d) The fourth and deepest level is emotional understanding, the level at which people may understand poetry. Today computers are far from this sophisticated level of comprehension.

3.4 Why language understanding is difficult for computers

Language understanding is difficult for computers because both language and the world itself to which language refers are extremely complex, much more complex than expected. But native speakers' facility with their own language is so great and early in their lives that it is hard to see why it is difficult to design computer programs to perform the same task. We only notice the difficulty of language in the special situation of learning a second language, and the problems encountered there –learning things like vocabulary, morphology, conjugations, genders, and irregularities– become memorization tasks which seem to be straightforward to computerize. But these tasks are not as simple as they seem at first sight. The syntax and morphology of natural languages are objects of high complexity. Words and idioms may convey complicated meanings. Native speakers may make quite subtle distinctions at any level of the structure of language. This big body of knowledge is the main topic of theoretical linguistics, and the progress of computational processing of language has been, is and will be closely linked to it.

Complexities of language are compounded by other minor problems that are easy to handle for humans but can be extremely difficult for computers. It is the case of ambiguity. Ambiguity appears at several levels of language:

a) at the lexical level: few dictionary entries list just one meaning for a word,

b) at the syntactic level: most sentences admit more than one parse tree,

c) at the pragmatic level: most sentences allow more than one analysis of the pragmatic role they play in the context of discourse where they are being uttered.

However, in spite of such potential multiplicity of choices, just a single interpretation of the sentence is intended by the speaker: it is the task of the listener (and of the computer) to recover it in order to achieve full understanding.

3.5 Knowledge of the world

The difficulty of language understanding is also a reflection of the complexity of the world, for one cannot understand language without becoming involved in the speaker's knowledge of the world. Let we read a text in our native language on a topic that we know nothing about, but written for an audience that does know the topic. We may identify all the words and parse all the sentences, but have little idea as to what the author is saying. Without the particular knowledge that the author assumes of the reader, one cannot understand at more than a superficial level. Knowledge of the language is not enough; knowledge of the world is required.

Knowledge of the world is particularly important in the resolution of ambiguity and anaphors. Frequently, only one reading of an ambiguous sentence will make sense, or one will be more plausible or more likely than the others, given the appropriate knowledge. For example:

(7) George drank a glass of port.

(8) George went to the library to get a book.

The word *port* can refer to a drink or to a certain place besides the sea. In order to interpret (7) correctly, we need to know from the world that only *port* as a drink can be put in a glass. Sentence (8) admits two plausible readings of *a book*: it could refer to a specific book that George is looking for or to any book. Both readings are plausible; thus, our knowledge of the world will help us to decide which to choose as probably intended by the speaker.

Although Bar-Hillel was the first, in 1960, to point out the need for knowledge of the world, its importance for natural language understanding has been only recently fully recognized. In the early days of computational linguistics, it was naively thought, for instance, that machine translation would require little more than a bilingual dictionary and a bit of morphological and structural analysis. Initial failures of such an approach were attributed to underestimating the complexities of syntax, and were unsuccessfully tried to solve by means of different kinds of syntax of increasing complexity.

4 The architecture of a natural language understanding system

The architecture of a standard natural language understanding system is as follows:

$$\begin{tikzcd}
 & \text{natural language sentence} \arrow[d, Rightarrow] & \\
\textbf{grammar} \arrow[r] & \text{PARSER} \arrow[r, Leftrightarrow] \arrow[dd, Rightarrow, "\textit{parse tree}"'] & \text{MORPHOLOGICAL ANALYZER} \arrow[d] \\
 & & \textbf{lexicon} \arrow[dl] \\
 & \text{SEMANTIC INTERPRETER} \arrow[dd, Leftrightarrow, "\textit{semantic representations}"'] & \\
 & & \textbf{general knowledge base} \arrow[ul] \\
 & \text{PRAGMATICS MODULE} \arrow[ur] \arrow[d, Leftrightarrow, "\textit{semantic representations}"'] & \\
 & \text{APPLICATION PROGRAM} \arrow[d, Rightarrow] & \\
 & \text{answer} &
\end{tikzcd}$$

(Capital letters stand for processes and bold types represent knowledge sources. Arrows show the flow of information.)

The purpose of such a system depends on the application program, which could be, for example, a database system or a travel reservation system. It allows the user to easily ask in natural language, instead of having to learn some special formalism in order to interact with the computer.

A natural language understanding system shows a modular architecture, consisting of four major subsystems:

a) a morphological analyzer,

b) a parser,

c) a semantic interpreter, and

d) a module for discourse pragmatics.

Furthermore, the system has three main sources of knowledge:

a) a grammar,

b) a lexicon, and

c) a general knowledge base.

Finally, the application program uses to have its own specific knowledge base. The arrows are representative of the dynamic character of the system: the movement of information through it. The input is a natural language sentence, and the output is any form of answer from the application program. The answer could be in natural language, by means of a natural language generator. In between, the sentence is processed by each of the subsystems one after another, and then passed to the application program. As the arrows suggest, the subsystems do not act in an isolated way, but may interact to produce the final result.

Now, we are going to describe briefly each one of the subsystems and associated knowledge sources.

4.1 Morphological analyzer and lexicon

The lexicon is the list of words that the system has to recognize. The information listed for each word will typically include:

a) part of speech,

b) syntactic irregularities, and

c) representation of the meaning.

Irregular forms are usually listed as a cross-reference to the base form. If a word has more than one meaning or belongs to more than one part of speech, all are included. For instance:

(9) port = noun, regular; drink.
port = noun, regular; harbour.
port = verb, regular; to present (arms).

(10) men = plural, man.

The meanings shown are the names of knowledge representation structures where the detailed semantics of the words can be found.

The first thing that a natural language understanding system must do with each word it sees is to check that the word is in the lexicon. Normally, the lexicon will contain only the root forms of regular words, not plurals or inflected forms of verbs. If the word found is not in the lexicon, a morphological analyzer will try to determine the uninflected form. For many languages, this is a relatively simple matter of removing affixes and adjusting the spelling to see if the resulting word is included in the lexicon. In the case of agglutinative languages and others with complex morphology, however, the task may be complex and require a lot of interaction with the parser. The goal of this stage of the system is to discover all possible analyses of the input word. If it finds the word *drinks*, for example, it should report the possibility of a plural noun or a third-person singular verb.

If the system sees a word that it cannot find in the lexicon nor analyze morphologically, it must consider several possibilities:

a) that the unknown word is actually a known word mistyped: in this case, spelling correction techniques have to be attempted;

b) that it is a special word such as a bank account number, which could be the subject of queries to a business application: in principle, this kind of words can easily be recognized;

c) that the word is a name: the parser will have to determine whether a name could occur at this point of the sentence;

d) that it is a word which has been unconsciously omitted from the vocabulary of the system: then, if the system is interactive, the user will be asked to either rephrase the sentence without it or add it to the lexicon.

4.2 Parser and grammar

The parser is the component of the system that determines the syntactic structure of the sentence. The input to the parser is the sentence, and the output is a parse tree or phrase marker.

As it is building the tree, the parser draws upon information from the lexicon as well as from the morphological analyzer; if offered more than one possible morphological analysis of a word, it takes the one that best fits the context. And, of course, the parser needs to know the grammar of the language that it is parsing. Usually, the grammar is represented separately, in such a way that the parser can draw upon as it needs to. In theory, the parser can analyze any language whose grammar, morphology, and lexicon are given; it contains the universal and general principles of syntax, independent of any particular language. In practice, however, most real parsers that have been developed so far have been limited to at most a few typologically related languages (belonging to the same family).

There are many different kinds of parser, and many different ways of representing the grammar of a language. The two most common types in computational linguistics are:

a) chart parsers, and

b) augmented transition network parsers (ATN's).

A chart parser attempts to find a combination of words allowed by the grammar that matches the input sentence. This may involve much trial and error. A chart parser maintains a chart of the alternatives tried and the hypotheses tentatively accepted.

An ATN represents a grammar as a network; to parse a sentence is to traverse the network, respecting the constraints on each path (for instance, that the next word in the sentence must be a verb). If the parser finds itself unable to proceed, it must backtrack to some previous point and try another alternative.

Often, a sentence will be syntactically ambiguous; that is, the grammar will produce two or more different parse trees. For example, in:

(11) George is seeing John with the telescope,

the prepositional phrase *with the telescope* could describe the seeing, that is, complement the verb, or John, that is, complement the object noun. Deciding which one is intended by the speaker requires considering the meaning and relative plausibility of each. To find this out, the parser will have to ask the semantic interpreter (connected to the general knowledge module) about the meanings of the alternatives; thus, parsing generally alternates with semantic analysis.

4.3 Semantic interpreter and general knowledge base

The ultimate goal of the analysis is to determine the meaning of the sentence. The semantic interpreter needs not wait until the parser has completed its job; usually, it can begin to work on each constituent of the tree as soon as the syntactic analysis of that constituent is complete, regardless of the state of the rest of the analysis. Indeed, many systems rely on this possibility in order that semantics be able to assist syntax.

Meaning is represented in a computer system by means of knowledge representation formalisms or logics. The lexicon gives the meaning of each individual word in such a formalism, and the semantic interpreter must combine these in a manner appropriate to the structure of the sentence and the meanings themselves, either in strict accordance with the principle of compositionality or not. If a word conveys more than one possible meaning, as most words do, then the semantic interpreter must decide which one was intended by the speaker. Usually, this requires determining which makes more sense in the context. The result of this whole process is a logical form that represents the literal meaning of the sentence.

4.4 Pragmatics module

Computers tend to carry out literal interpretations. For example, a computer asked to:

(12) Give me the examination grades of all the mathematics students

might answer with just a list of anonymous marks. Humans often say things obliquely or incompletely, leaving to the intelligence of the listener to determine the intention and fill in the gaps. To be of practical use, a natural language understanding system must follow its literal semantic analysis with a pragmatic analysis, determining what the speaker really meant and how the sentence fits into the conversation.

Imagine that a user says to a travel reservation system:

(13) I've been thinking about going to Hungary.

At the literal level, the user is just only stating a fact that the system could merely take note of. But, at a deeper level, the user is asking the system to provide information about schedules of travels to Hungary. The system must recognize that the user is indirectly asserting that he/she has a goal of finding information about travelling to Hungary, and is asking for help in achieving that goal; that is, the computer has to recognize that the sentence is an indirect speech act. In order to determine the speaker's intention, the system must use not only knowledge of standard linguistic conventions, but also knowledge of how people plan and how their goals can be achieved. Thus, at present the problem of plan inference in natural language systems occupies an outstanding position in computational linguistics.

Just at the pragmatic level the system must also determine how the sentence relates to the preceding conversation or discourse. For example, it may exemplify or elaborate on the previous sentence, or describe the next in a sequence of events, or change the topic of conversation. Sometimes, speakers will make the relationship explicit: *for example* might be used to mark an exemplification, and *by the way* a change of topic. But often the relationship is left implicit and must be determined from the meaning itself of the sentences. This task can be quite complex for computers. Consider, for example, a sentence intended as a conclusion to be drawn from the preceding sentence, as in the following pair:

(14) Nobody likes the new tax system. The government is certain to be defeated.

The system must determine that the second sentence could plausibly be a consequence of the first one.

5 Parallelism in natural language processing

We have seen a decomposition of natural language automatic description into a series of different coordinated levels. Models of sentence processing may or may not refer to this decomposition. Natural language processing systems can be built for quite practical reasons, and therefore efficient performance properties can be much more important than attempting to reflect theoretical ideas coming from linguistics or psychology. Since practical systems do not always have to deal with the full range of natural language sentences –or with an unlimited domain of discourse– , the natural decomposition we have provided does not need to be explicitly present in language processing systems. From a psychological and linguistic point of view, however, computer models of human sentence processing should be consistent with theories developed in those fields. Having a model, it should be possible to simulate phenomena of human sentence processing.

Human sentence processing was initially explained by means of a serial model. This kind of models use a syntactic approach, where the syntactic processing task must be successful before semantic processing can begin, which in turn must precede pragmatic processing. If, in this model of linear interaction between levels of knowledge, higher-level information cannot be used to correct decisions at lower levels, this approach inexorably leads to a combinatorial explosion of all syntactic possibilities. For such reason especially, approaches combining different levels closely interacting at different moments are now preferred.

Models in which this latter type of sentence processing can be displayed are called interactive or parallel. During parsing, a system is capable of using any type of knowledge at any moment it needs. These models may exhibit different appearances. They can take, for instance, the form of a system in which natural language processing tasks are assigned to different processors and in which every knowledge source interacts with every other. In the known blackboard model of interaction, modules can process in parallel and cooperatively by means of a globally accessible blackboard on which they can write and read intermediate results: the modules communicate and interact solely through the blackboard. Some further possibilities exist.

6 Thesis

Assuming the natural decomposition of language we have shown and the parallel type of processing, we regard natural language as the final product of a parallel communicating grammar system (PC) architecture, each one of whose processors simulates one of the modules of natural language we have considered. In addition, each component of the parallel communicating grammar system consists of several subprocessors working as cooperatively distributed grammar systems (CD). We would have, then, a two-levels machinery: a macro-PC-system composed by micro-CD-systems. Its functioning would be as follows. Several processors cooperate distributively in the complex task of producing a syntactically well-formed (grammatical) sentence. Each one of such processors generates one of the levels we can distinguish in the syntactic structure of the sentence. On the other hand, it seems clear that human language is not produced/understood in a serial manner, but in a parallel one: syntax is not strictly generated before semantics can intervene, but in accordance with a complex synchronicity. Different levels and sublevels of each module of language are successively integrated in accordance with a certain protocol of integrative cooperation.

Computer scientists have now the task to formally define such two-levels machinery, and linguists the task to characterize the programme of synchronization of the modules. Both works are strong challenges for the future. If carried out jointly, the forecast is encouraging.

References

[1] Allen, J. (1987), *Natural language understanding.* Addison-Wesley, Reading, Mass.

[2] Csuhaj-Varjú, E. (1994), "Grammar systems: a multi-agent framework for natural language generation", in Gh. Păun, ed., *Mathematical aspects of natural and formal languages*: 63-78. World Scientific, Singapore.

[3] Csuhaj-Varjú, E. & R. Abo Alez (1995), "Multiagent systems in natural language processing", unpublished ms.

[4] Csuhaj-Varjú, E., J. Dassow, J. Kelemen & Gh. Păun (1994), *Grammar systems: a grammatical approach to distribution and cooperation.* Gordon and Breach, London.

[5] Dowty, D., L. Karttunen & A. Zwicky, eds. (1989), *Natural language parsing.* Cambridge University Press, New York.

[6] Grishman, R. (1986), *Computational linguistics.* Cambridge University Press, Cambridge.

[7] Grosz, B., K. Sparck Jones & B. Webber, eds. (1987), *Readings in natural language processing.* Morgan Kaufmann, Los Altos, Ca.

[8] Păun, Gh. (1995), "Generating languages in a distributed way: grammar systems", in C. Martín Vide, ed., *Lenguajes naturales y lenguajes formales, XI*: 45-71. PPU, Barcelona.

[9] Păun, Gh. (1995), "Grammar systems: a grammatical approach to distribution and cooperation", in Z. Fülöp & F. Gécseg, eds., *Proceedings of ICALP'95*: 429-443. Springer, Berlin.

[10] Păun, Gh. (1995), "Parallel communicating grammar systems. A survey", in C. Martín Vide, ed., *Lenguajes naturales y lenguajes formales, XI*: 257-283. PPU, Barcelona.

[11] Smith, G.W. (1991), *Computers and human language.* Oxford University Press, New York.

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Contributions are accepted for review with the understanding that the same work has not been published elsewhere.

Manuscripts must be in English and should be sent in triplicate to any of the Editors. On the first page, the *title* of the paper, the *name(s)* and *affiliation(s)*, together with the *mailing* and *electronic address(es)* of the author(s) must appear. An *abstract* summarizing the results of the paper is also required. References should be listed in alphabetical order at the end of the paper in the form which can be seen in any article already published in the journal. Manuscripts are expected to be made with a great care. If typewritten, they should be typed double-spaced on one side of each sheet. Authors are encouraged to use any available dialect of TEX.

After acceptance, the authors will be asked to send the manuscript's source TEX file, if any, on a diskette to the Managing Editor. Having the TEX file of the paper can speed up the process of the publication considerably. Authors of accepted contributions may be asked to send the original drawings or computer outputs of figures appearing in the paper. In order to make a photographic reproduction possible, drawings of such figures should be on separate sheets, in India ink, and carefully lettered.

There are no page charges. Fifty reprints are supplied for each article published.

Publication information. Acta Cybernetica (ISSN 0324–721X) is published by the Department of Informatics of the József Attila University, Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. For 1996 Numbers 3–4 of Volume 12 are scheduled. Subscription prices are available upon request from the publisher. Issues are sent normally by surface mail except to overseas countries where air delivery is ensured. Claims for missing issues are accepted within six months of our publication date. Please address all requests for subscription information to: Department of Informatics, József Attila University, H-6701 Szeged, P.O.Box 652, Hungary. Tel.: (36)-(62)-311-184, Fax:(36)-(62)-312-292.

URL access. All these information and the contents of the last some issues are available in the Acta Cybernetica home page at http://www.inf.u-szeged.hu/local/acta.

CONTENTS

Sponsored by Zenon Computer Engineering and Trading LTD
A folyóiratot a Zenon Számítástechnikai és Kereskedelmi Kft. szponzorálja

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János
A kézirat a nyomdába érkezett: 1996. december
Terjedelem: 7,12 (B/5) ív